**INFS 740: Final Project (StockSphere)**

Abhishek Jallawaram (G01373042)  
Teja Sajja (G01395837)  
  
Table of Contents

[**Project Overview:** 2](#_Toc164868526)

[**Summary:** 2](#_Toc164868527)

[**Key Objectives:** 2](#_Toc164868528)

[**Project Tech Stack:** 3](#_Toc164868529)

[**React** 3](#_Toc164868530)

[**FastAPIs** 3](#_Toc164868531)

[**MongoDB** 4](#_Toc164868532)

[**System Architecture:** 4](#_Toc164868533)

[**Admins** 4](#_Toc164868534)

[**Customers** 4](#_Toc164868535)

[**Database Schema:** 6](#_Toc164868536)

[**Collections:** 6](#_Toc164868537)

[**Functionalities** 8](#_Toc164868538)

[**CRUD Operations:** 8](#_Toc164868539)

[**Complex Queries:** 10](#_Toc164868540)

[**Stock Queries** 10](#_Toc164868541)

[**Crypto Queries** 12](#_Toc164868542)

[**UI Screenshorts:** 15](#_Toc164868543)

[Customer Access: 15](#_Toc164868544)

[Admin Access: 17](#_Toc164868545)

[**Visualizations:** **Admin Side -** 21](#_Toc164868546)

[**ML Prediction Model:** 24](#_Toc164868547)

[**Member Contributions** 25](#_Toc164868548)

[**Challenges & Future Work** 25](#_Toc164868549)

[**Conclusion:** 26](#_Toc164868550)

GitHub link: <https://github.com/abhishekjallawaram/StockSphere>

## **Project Overview:**

### **Summary:**

The purpose of this stock and crypto management platform is to provide an efficient and user-friendly system for monitoring and managing stock and cryptocurrency transactions. The platform enables users perform buy and sell stocks & cryptocurrencies. The admin can analyze the stocks, cryptocurrencies, agents, customers, and transactions to retain the customers and ensure to maintain a comprehensive tool for financial data analysis and management.

### **Key Objectives:**

1. **Data Management:** Implemented data management capabilities that allow users to access the data. (CRUD operations).
2. **Search Queries:** Advanced search queries implemented that allows users to retrieve data from multiple collections based on specific parameters. This feature aims to facilitate deeper insights into transactions and relationships between different data points.
3. **Data Visualization:** Integrated data visualizations that offer graphical representations of financial data, enhancing the user's ability to analyze trends over time and make informed decisions.
4. **User Interface:** Developed a responsive and intuitive user interface that caters to both novice and experienced users, ensuring ease of navigation and accessibility of features.
5. **Real-Time Updates:** Utilized yfinance API to retrieve real-time data which fetches and updates the data, providing users with the most current information available and enhancing the platform's responsiveness.
6. **ML prediction:** A prediction model to predict the close price for a given stock.

### **Project Tech Stack:**

#### **React**

1. We have utilized the react components to build the frontend components of the application.
2. React has a component-based structure allows for reusable UI components, making the development process more efficient and the codebase more maintainable.
3. React provides virtual DOM which minimizes the number of costly DOM manipulations by batching updates, enhancing the performance of the application.
4. React has a vast community and a rich ecosystem of libraries and tools, providing abundant resources for solving common development challenges.

#### **FastAPIs**

1. We have utilized the fastAPI to build the backed APIs needed to communicate with the MongoDB database and fetch the complex queriers.
2. FastAPI is one of the fastest web frameworks for Python, thanks to its Starlette framework for the web parts and Pydantic for the data parts.
3. Native support for asynchronous request handling, which is essential for handling multiple data-intensive operations like fetching large datasets from a database.
4. FastAPI automatically generates interactive API documentation (using Swagger UI and ReDoc), which simplifies API testing and frontend integration.

#### 

#### **MongoDB**

1. We have utilized the MongoDB as the primary database to store and manage structured and unstructured data of the platform.
2. MongoDB's schema-less nature allows for flexible and dynamic data models, which is beneficial for handling diverse datasets such as stocks and cryptocurrencies.
3. MongoDB offers horizontal scalability through sharding, effectively managing large volumes and high throughput of data.
4. MongoDB provides powerful querying capabilities, making it suitable for executing complex searches and aggregations required by the platform.

## **System Architecture:**

**Admins**a) The admin role has access to all the records and performs all the operations on the resources accessed.

b) Manages the system by performing CRUD (creating, updating, or removing user customers, agents, stocks, crypto and transactions.)

c) Visualize data to retain customers, provide incentives to agents based on performance and offers to premium customers.

**Customers**

Customers can log in to view their own balance, check out the stocks or cryptocurrencies they wish to buy or sell.

A diagram of a data flow

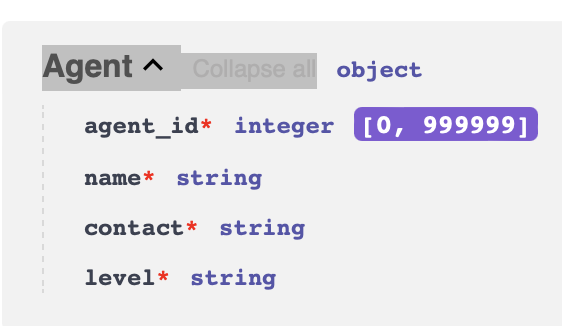
Description automatically generated

Figure1: System Architecture

## **Database Schema:**

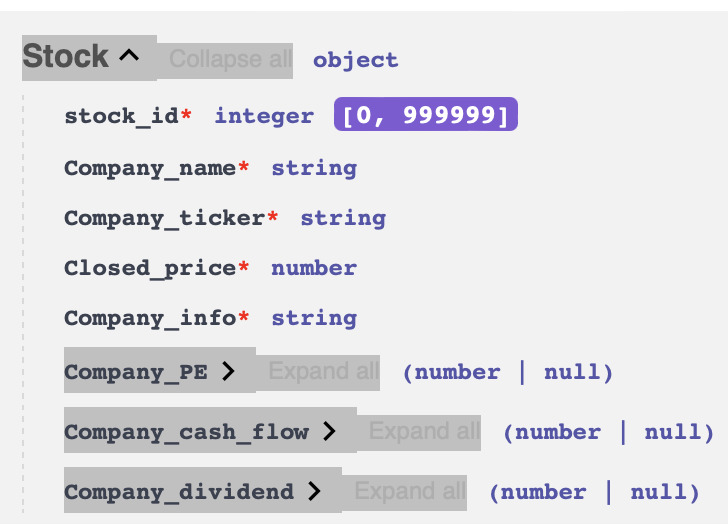
### **Collections:**

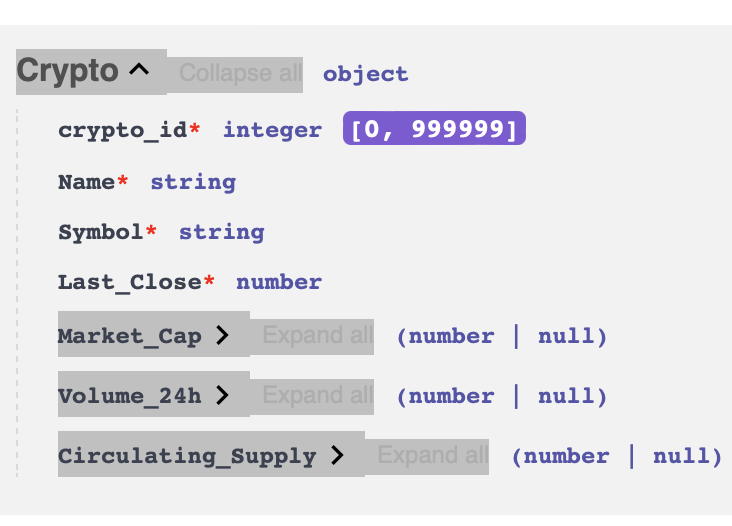
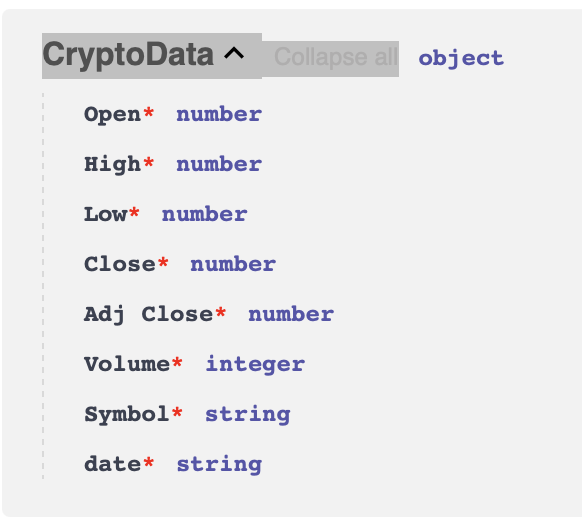
1. Agent (Agent schema)
2. Customer (Customer schema)
3. Stocks (Stock schema)
4. Stock History (Stock Data schema)
5. Cryptocurrencies (Crypto schema)
6. Crypto History (Crypto Data schema)
7. Transactions (Transactions schema)

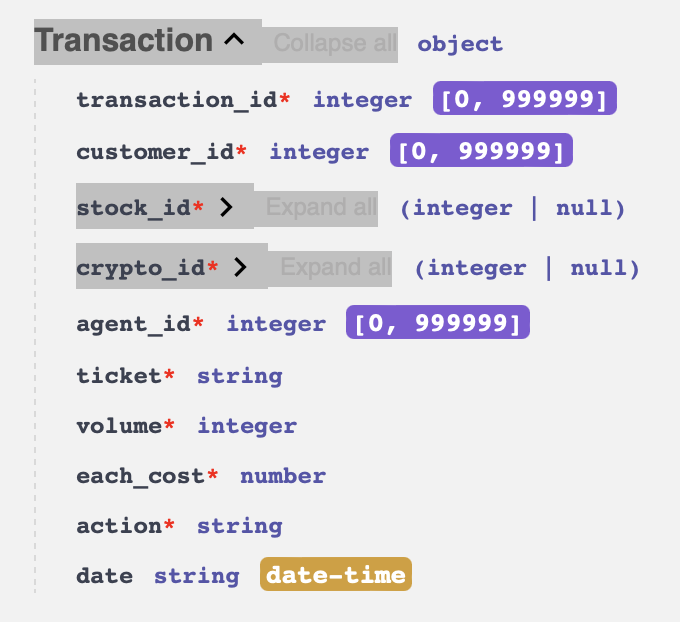
A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated 



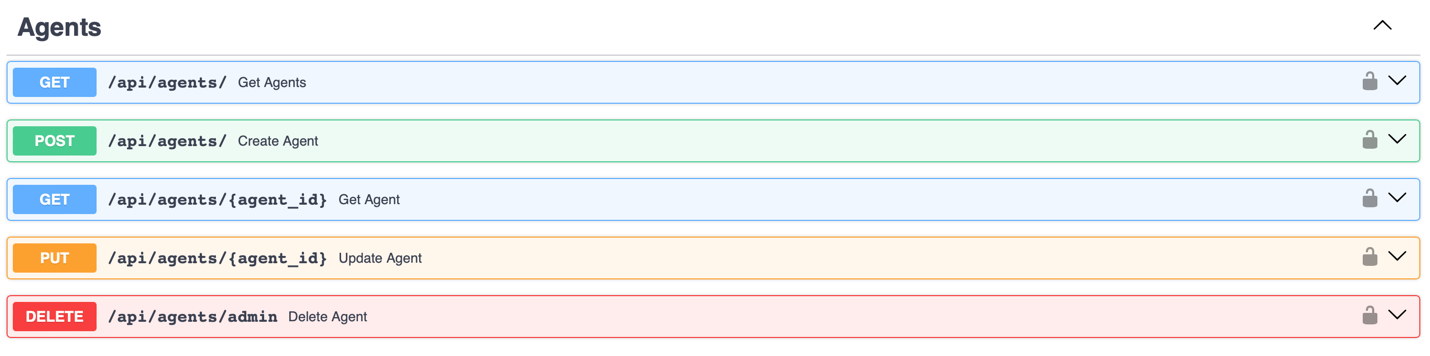
**Relationship between Collections:**

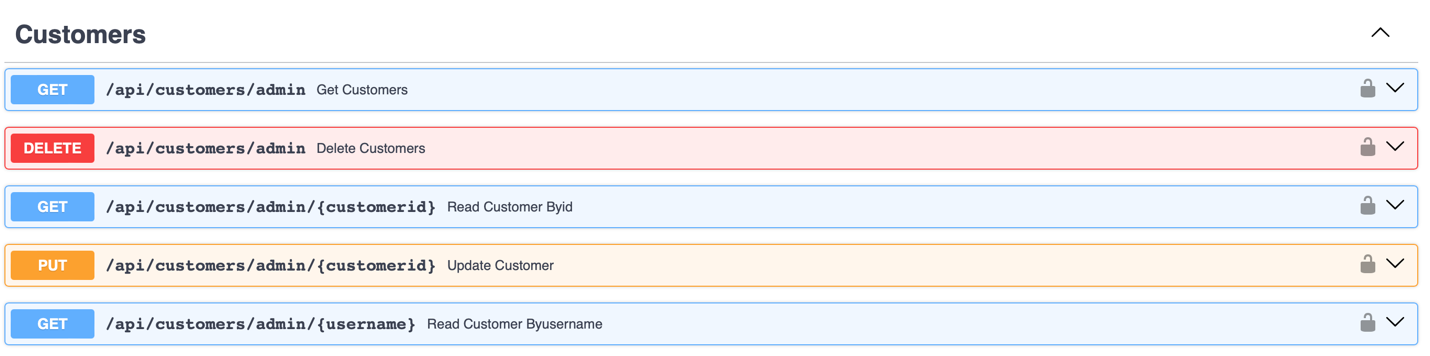
1. A customer would buy or sell stock/cryptocurrency through an valid agent.
2. The stock/cryptocurrency would be validated from stocks and cryptocurrencies by stock\_id or ticket.
3. Transactions are valid only if they have a valid customer\_id, agent\_id, stock\_id or crypto\_id and ticket is validated from stocks and cryptocurrencies.
4. Customer.customer\_id = Transactions.customer\_id
5. Agent.agent\_id = Transactions.agent\_id
6. Stocks.stock\_id = Transactions.stock\_id
7. Stocks.Company\_ticker = Transactions.ticket
8. Cryptocurrencies.crypto\_id = Transactions.crypto\_id
9. Cryptocurrencies.Symbol = Transactions.ticket

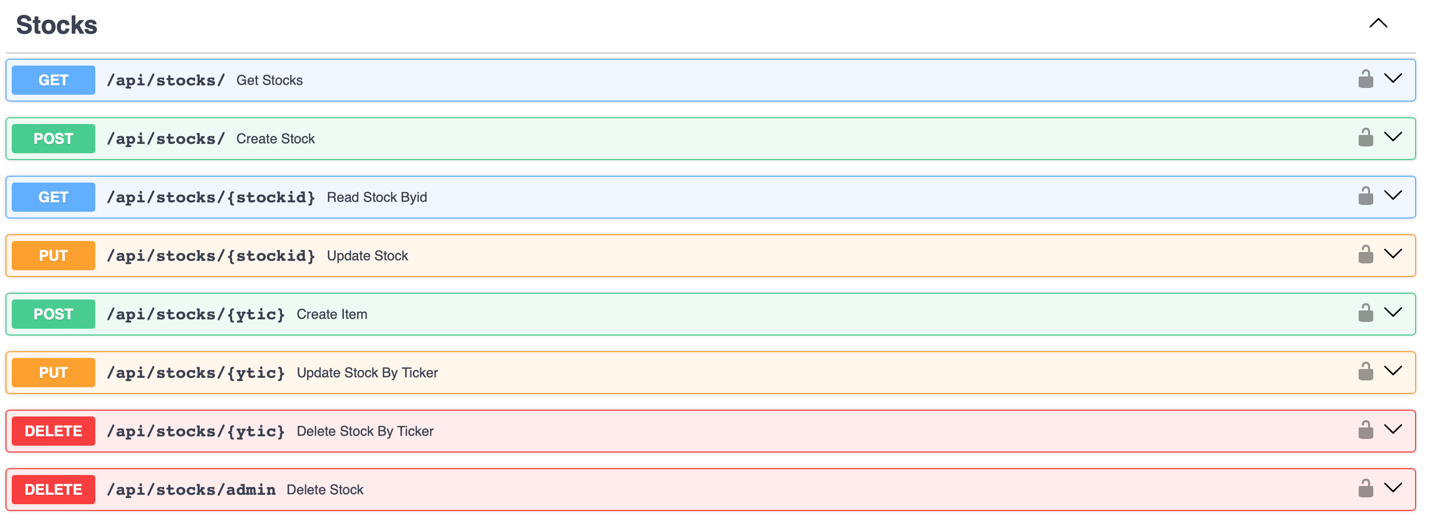
## **Functionalities**

1. We have implemented CRUD operations for all the collections as requested in the project requirements.
2. 6 complex queries from admin end with 2 variations making it a total of 18 complex queries.
3. The agents and customers data were generated using a python script using faker package to generate the valid data based on the schemas. (app.scripts.generate\_data.py)
4. The transactions were generated using a package called faker which generates the synthetic data using the existing stocks/ cryptocurrencies, customers, and agents.  
   (app.scripts.generate\_data.py and app.scripts.generate\_data\_crypto.py)
5. We have written a script to save the collections and load the data for the collections to reproduce the results of the demo. (app.scripts.collections-save.py and app.uitls.insert\_collections\_data.py)
6. The stocks, stock\_history, cryptocurrencies and crypto\_history would be generated on startup using yfinance API populated using the python scripts. (app.utils.yahoo\_finance.py and app.utils.yahoo\_finance\_crypto.py)
7. We have implemented a keras LSTM model to predict the stock ‘Close’ for next 10 days for specific stock for a given a date. We have implemented an endpoint to get the data and visualize the same.
8. We have implemented visualizations for the search queries using the bar charts and line charts.

### **CRUD Operations:**







A row of colorful rectangular objects

Description automatically generated

A screenshot of a computer

Description automatically generated

A group of colorful rectangular objects

Description automatically generated



### **Complex Queries:**

#### **Stock Queries**

All the stock queries have the below variations and used the below 4 collections for each query.  
Total queries (9 : 1 base query & 2 variations \* 3 queries)

**Variations:**  
  
**1) For a specific stock\_id**

**2) For a specific company**

**Collections:**

**1) Transactions**

**2) Stocks**

**3) Customers**

**4) Agents**

Query1:   
**Customers with Most Transactions in the Platform with Stocks Data.**

A screen shot of a computer screen

Description automatically generated

Query2:

**Customers with Top Transactions in the Platform with Stocks Data.**

**A screen shot of a computer program

Description automatically generated**

Query3:

**Agents with Top Transactions in the Platform with Stocks Data.**

**A screen shot of a computer program

Description automatically generated**

#### **Crypto Queries**

All the crypto queries have the below variations and used the below 4 collections for each query. Total queries (9 : 1 base query & 2 variations \* 3 queries)

**Variations:**  
  
**1) For a specific crypto\_id**

**2) For a specific Cryptocurrency**

**Collections:**

**1) Transactions**

**2) Cryptocurrencies**

**3) Customers**

**4) Agents**

Query4:   
**Customers with Most Transactions in the Platform with Crypto Data.  
A screen shot of a computer program

Description automatically generated**

Query5:   
**Customers with Top Transactions in the Platform with Crypto Data.**A screen shot of a computer screen

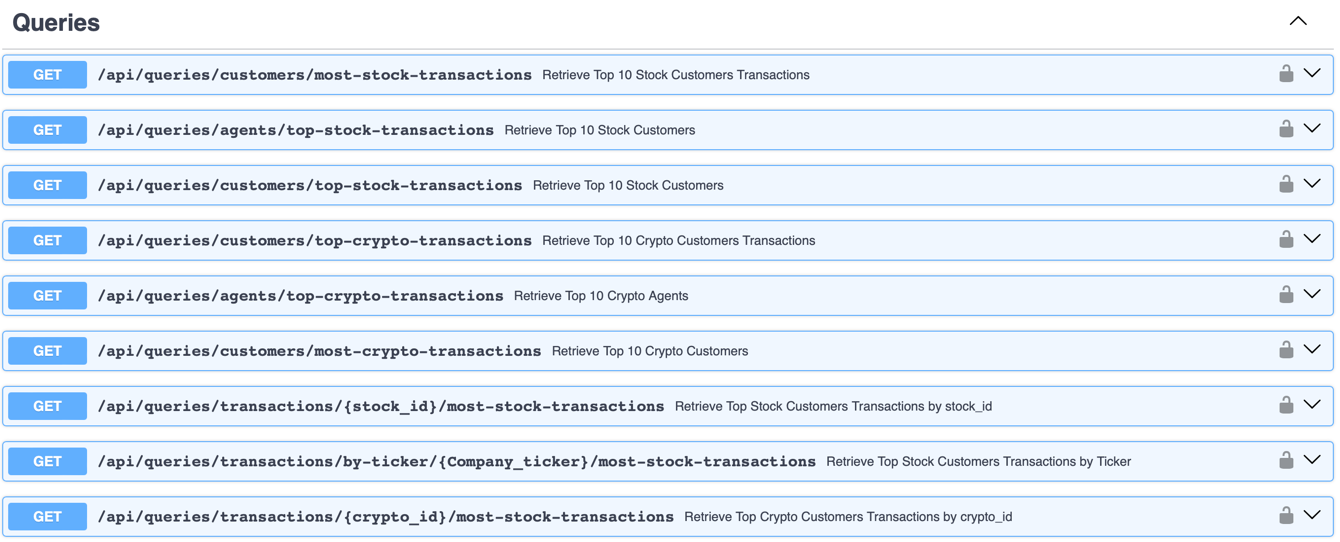
Description automatically generated

Query6:

**Agents with Top Transactions in the Platform with Crypto Data.  
A screen shot of a computer program

Description automatically generated**

All the API calls for the queries have been shown below: They can be access using SWAGGER at  
<http://localhost:8000/docs#/> (Ensure you have the backend running to test this)

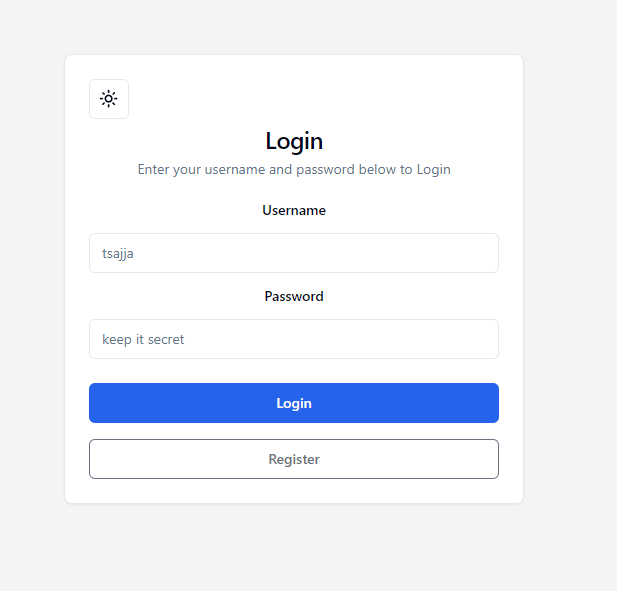
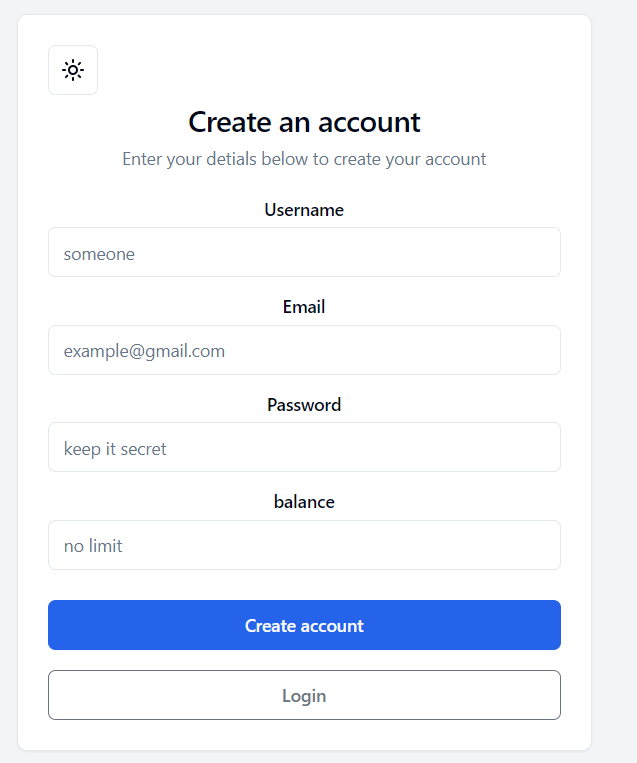


A screenshot of a computer

Description automatically generated

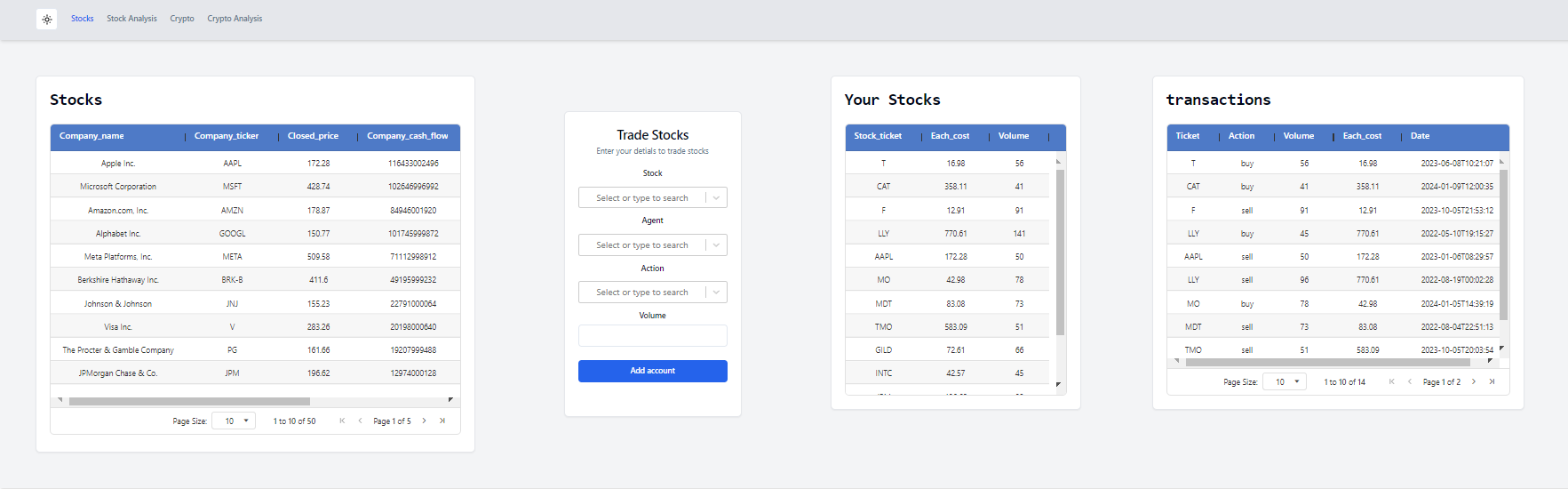
### **UI Screenshorts:**

Login Page Registration:

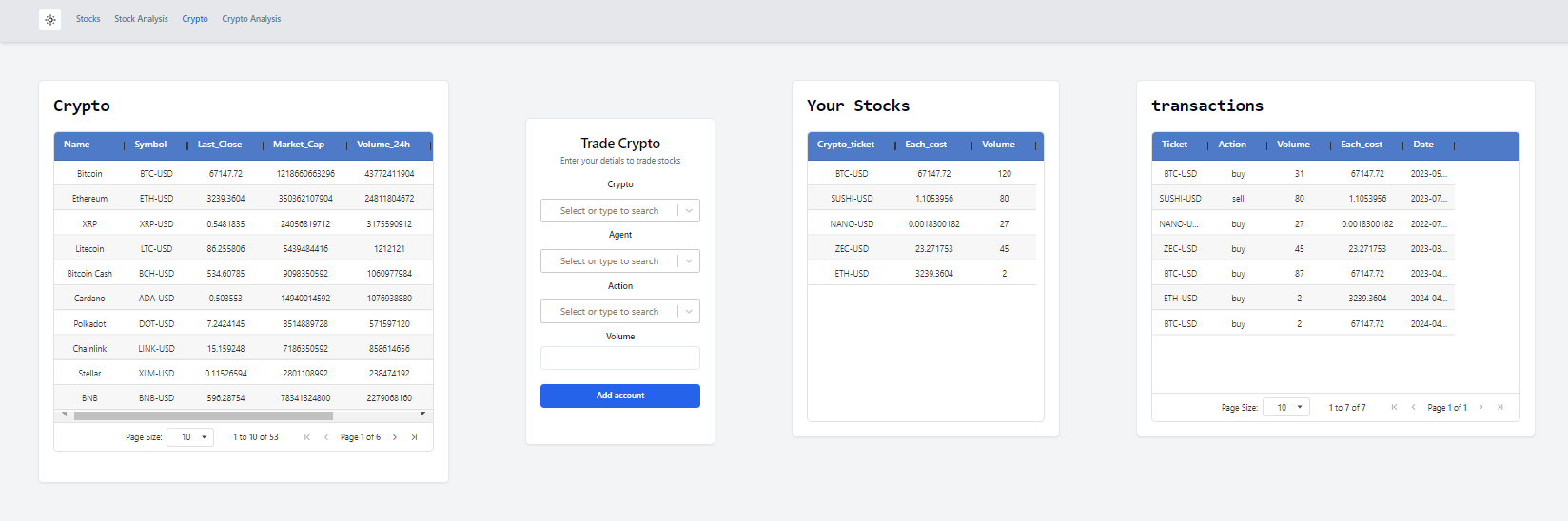
 

### Customer Access:

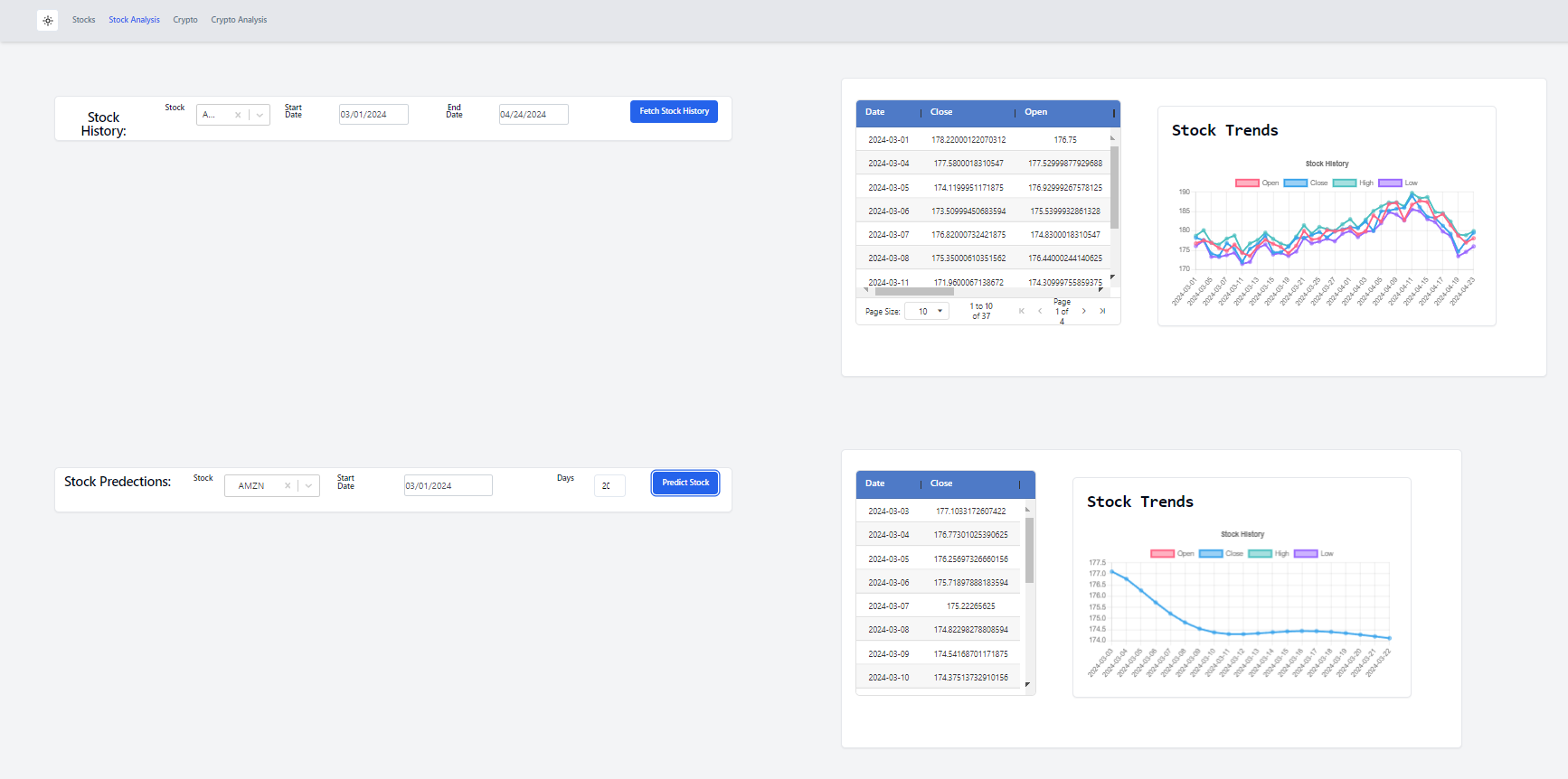
Stock dashboard:

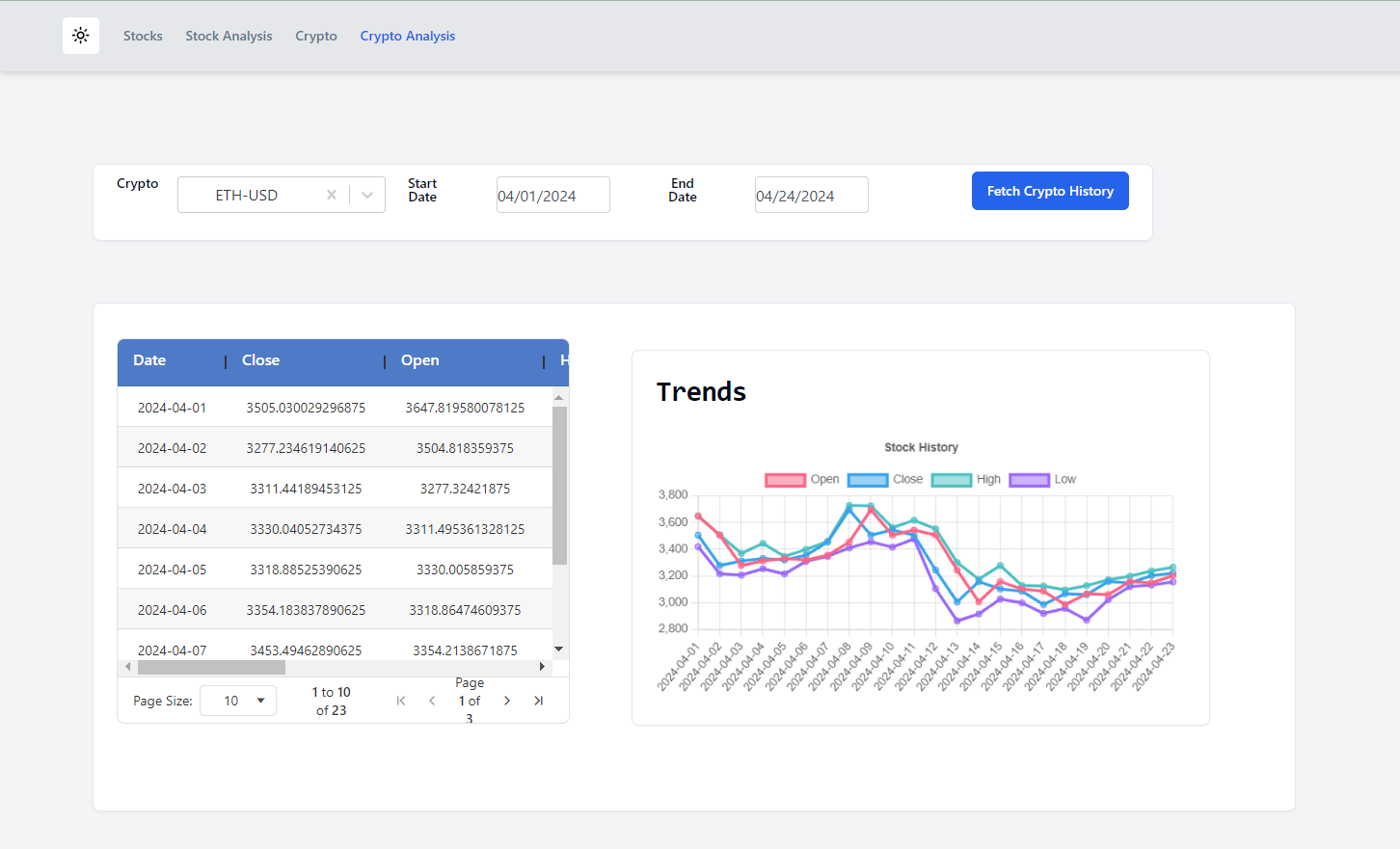


Crypto Dashboard:

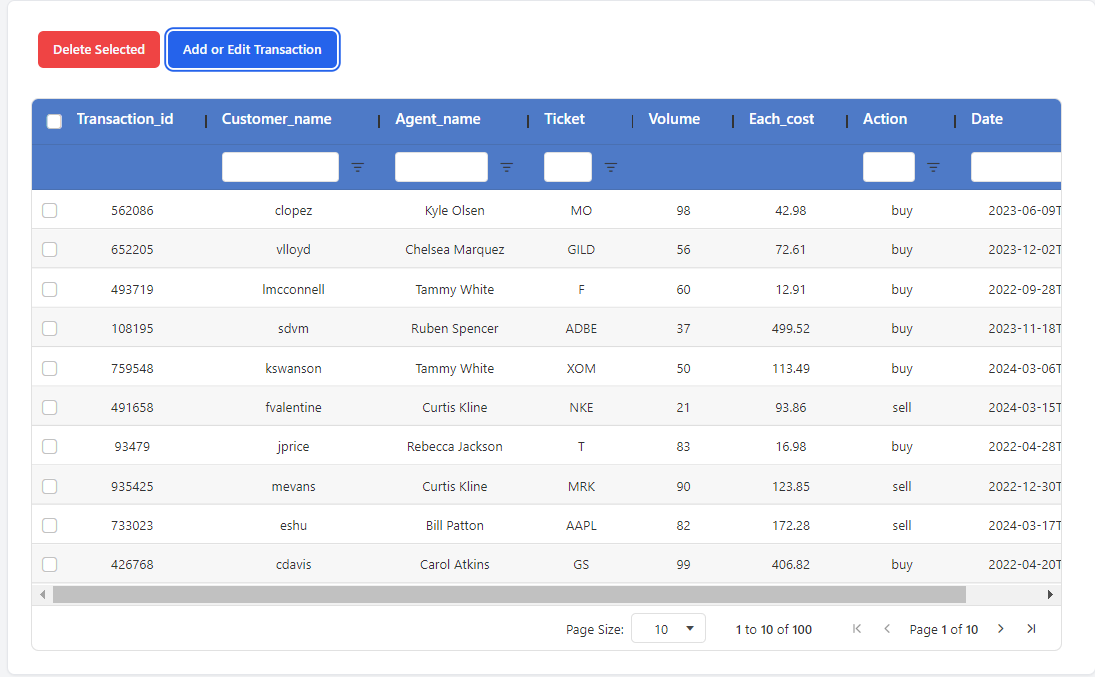


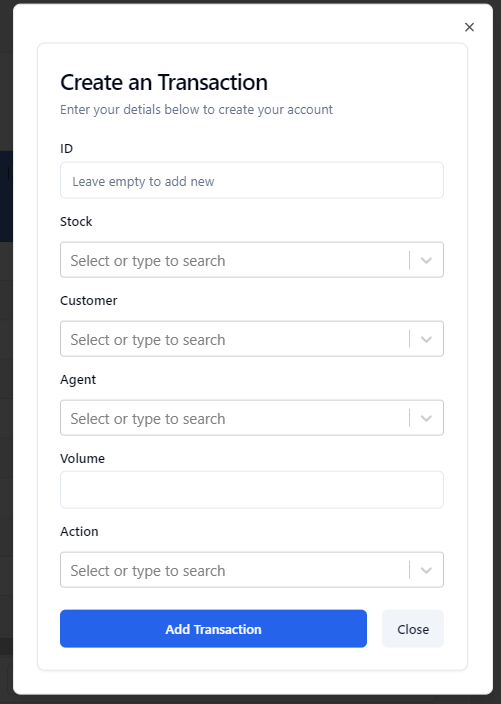
Stock Analysis:



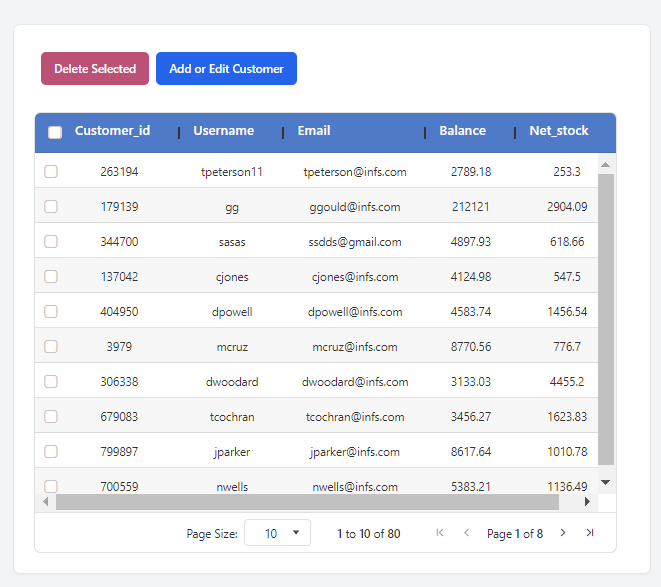
Crypto Analysis:  
  


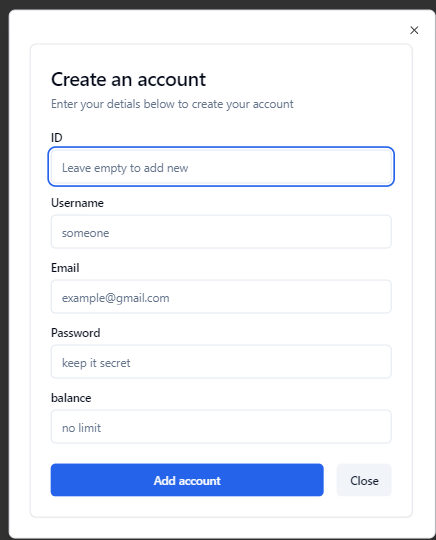
### Admin Access:

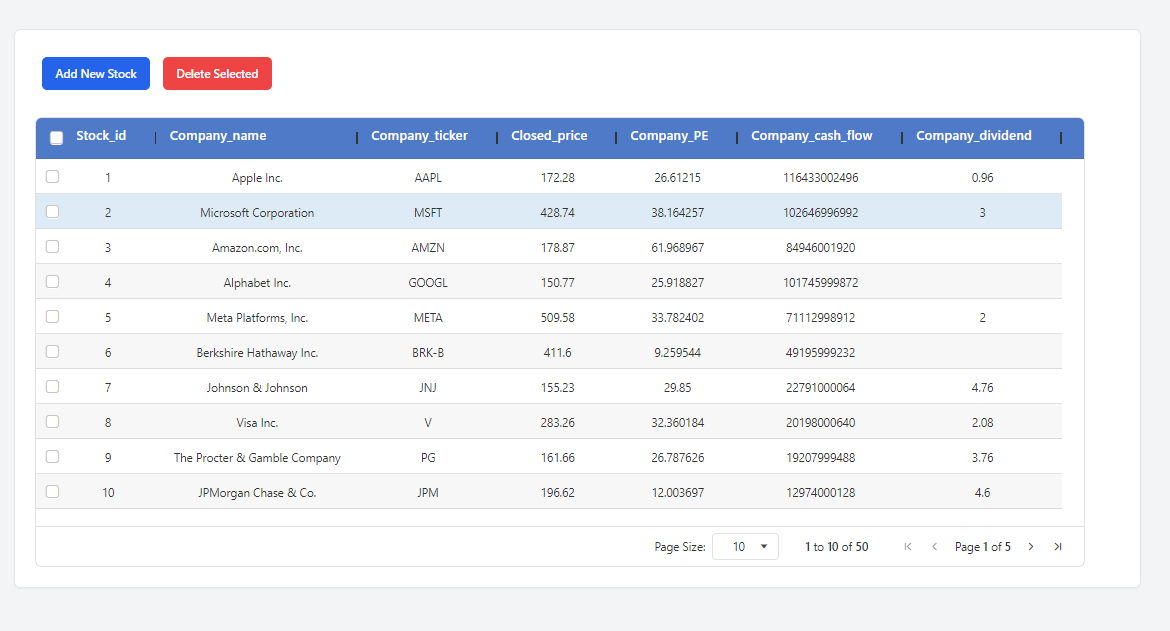
Transaction: (CRUD)  
  


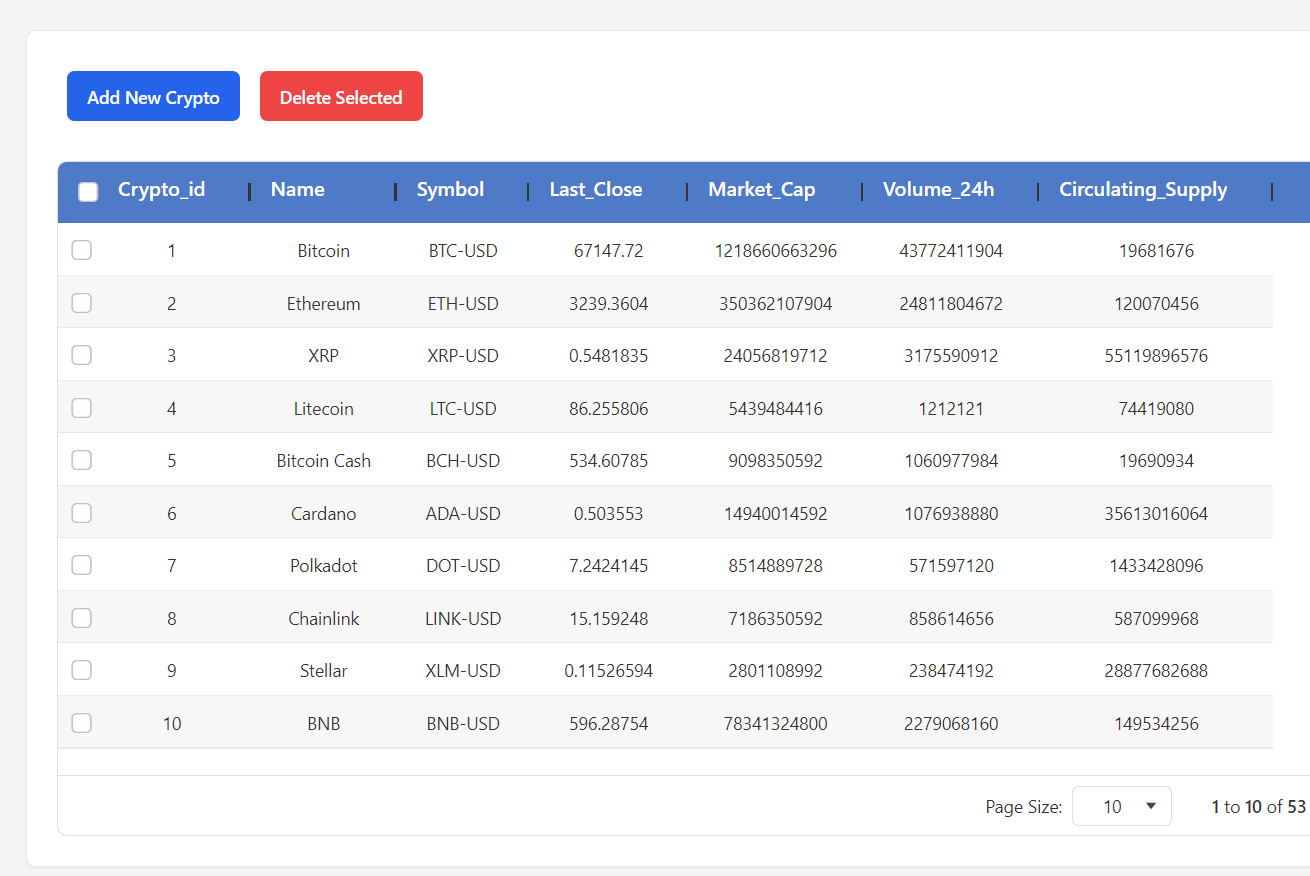


Customer: (CRUD)

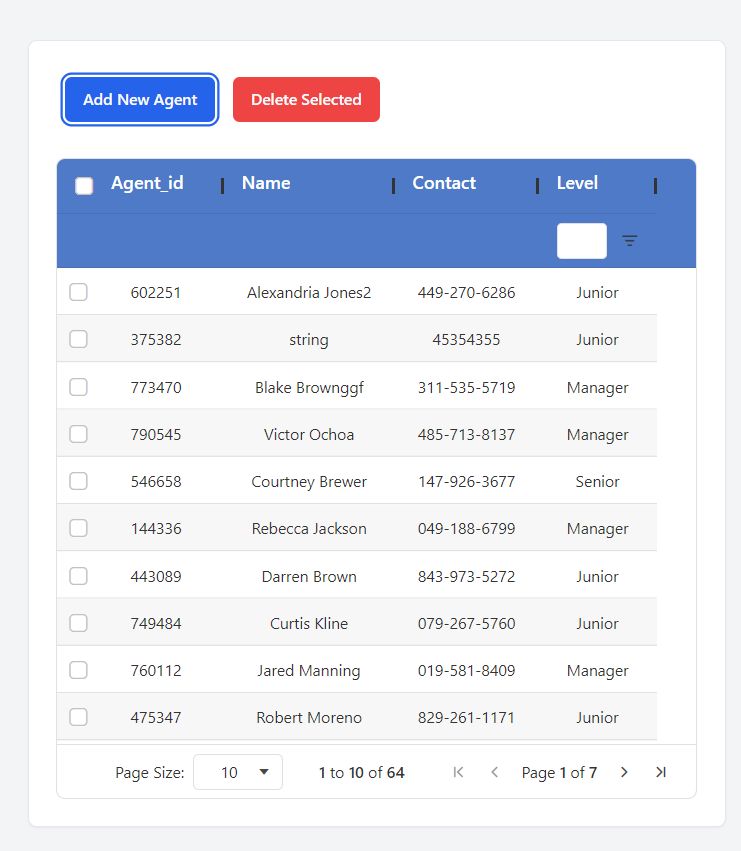




Stocks: (CRUD) 

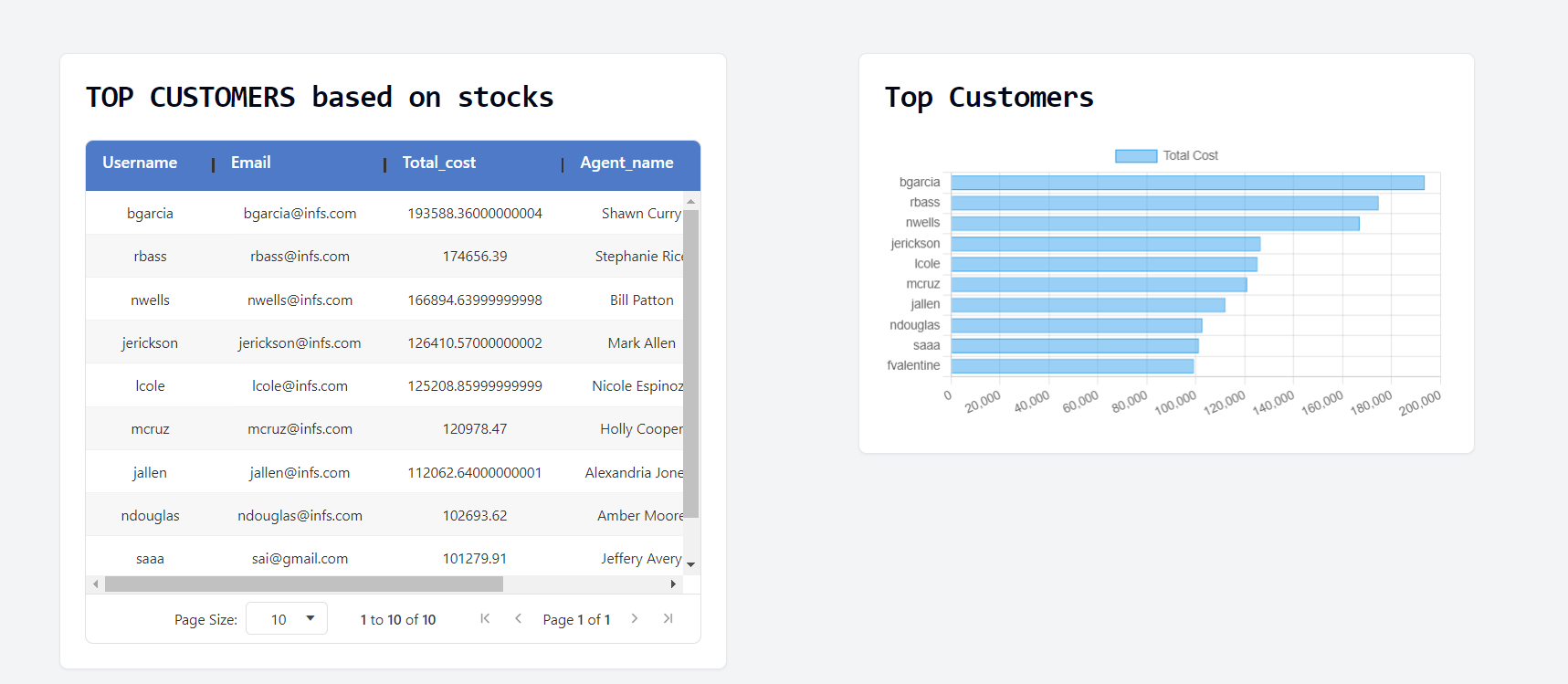
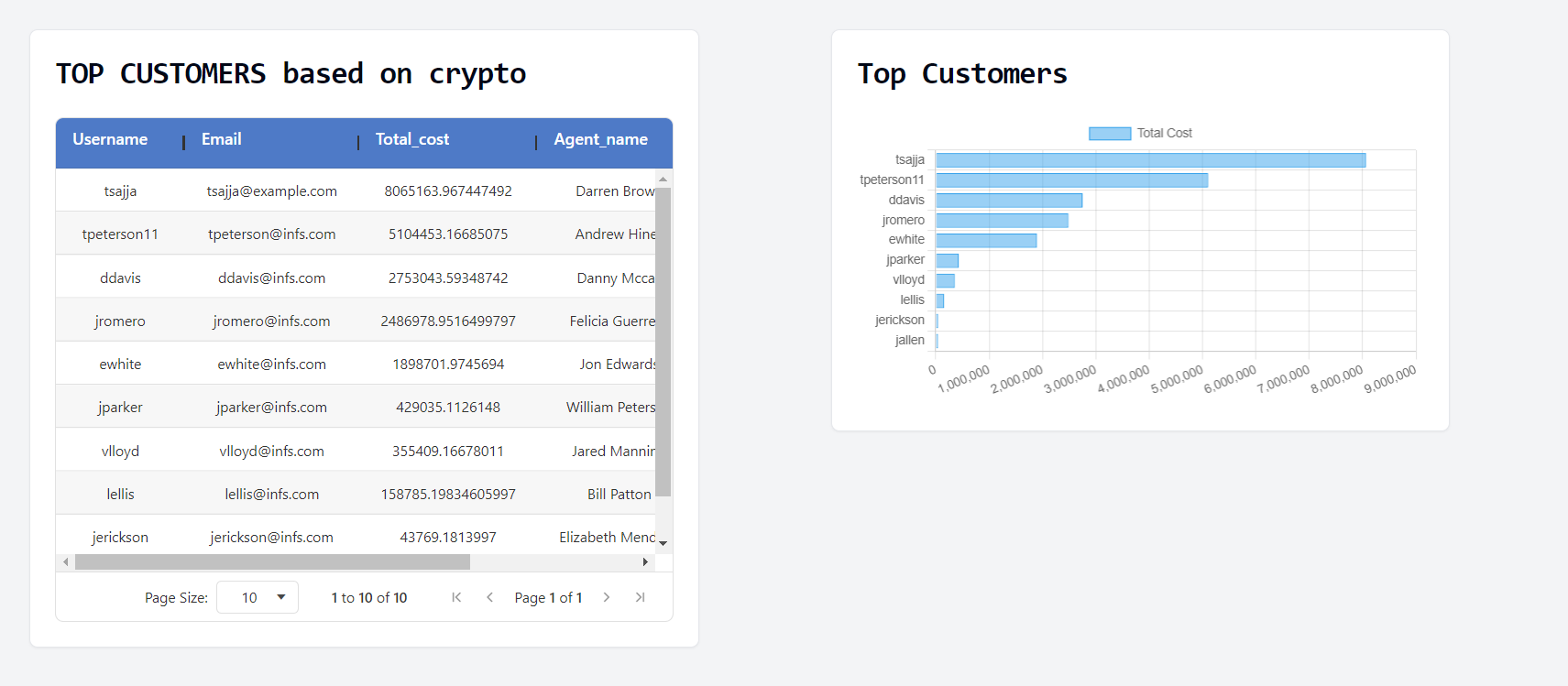
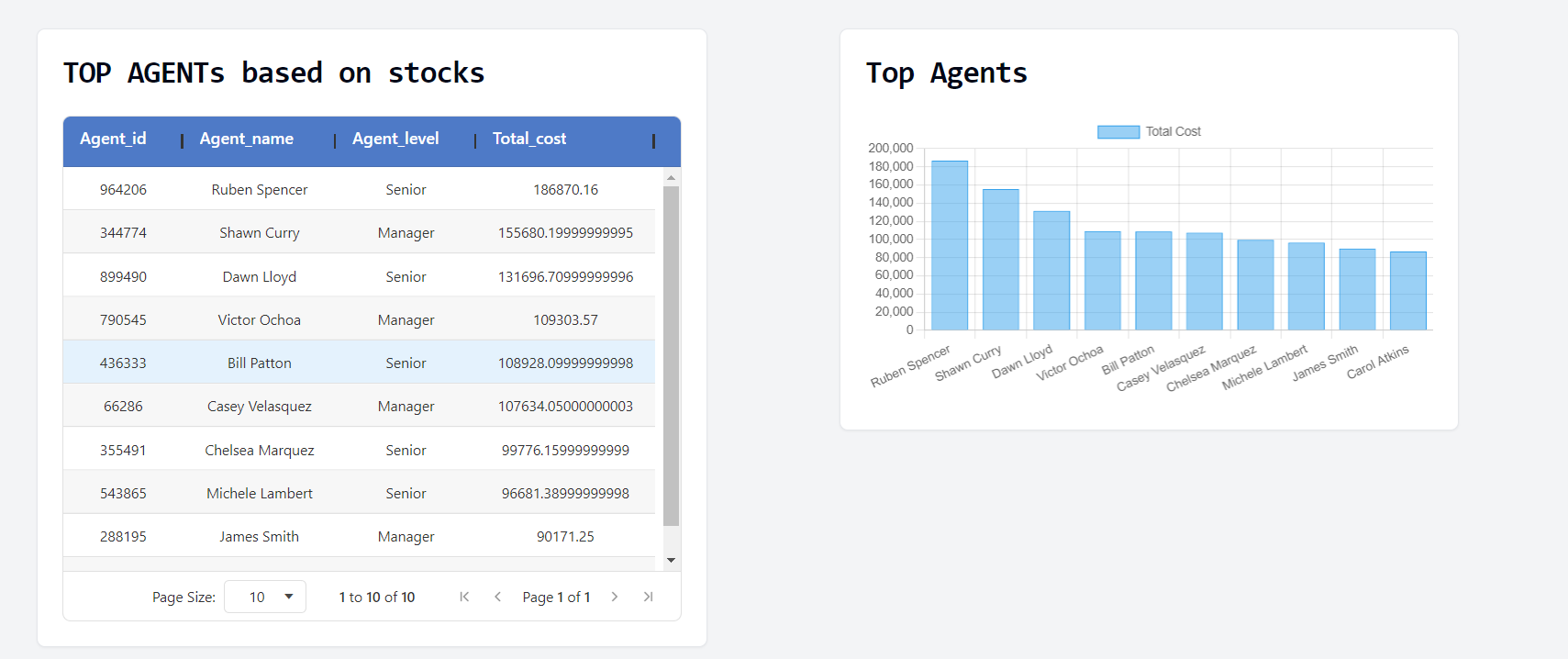
Crypto: (CRUD)  
  


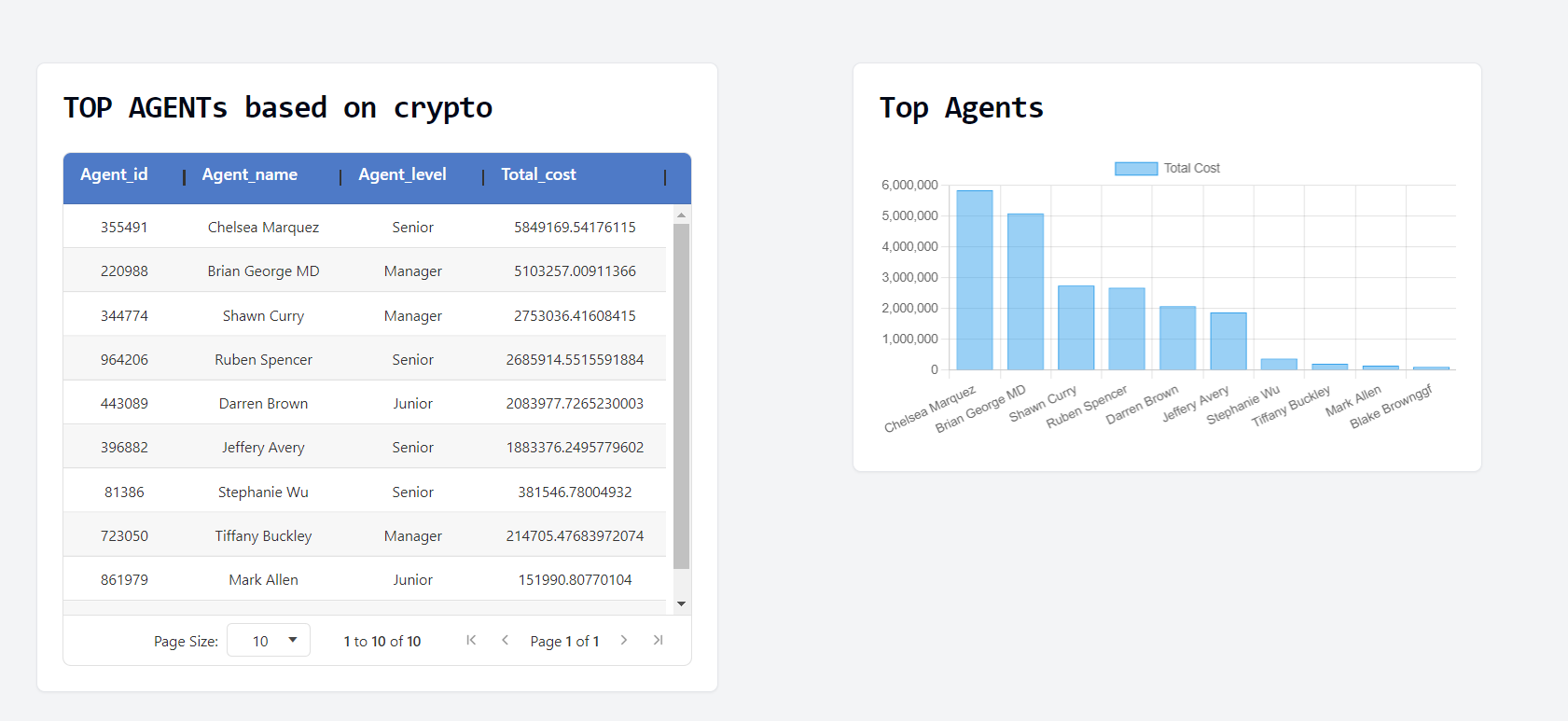
Agents:

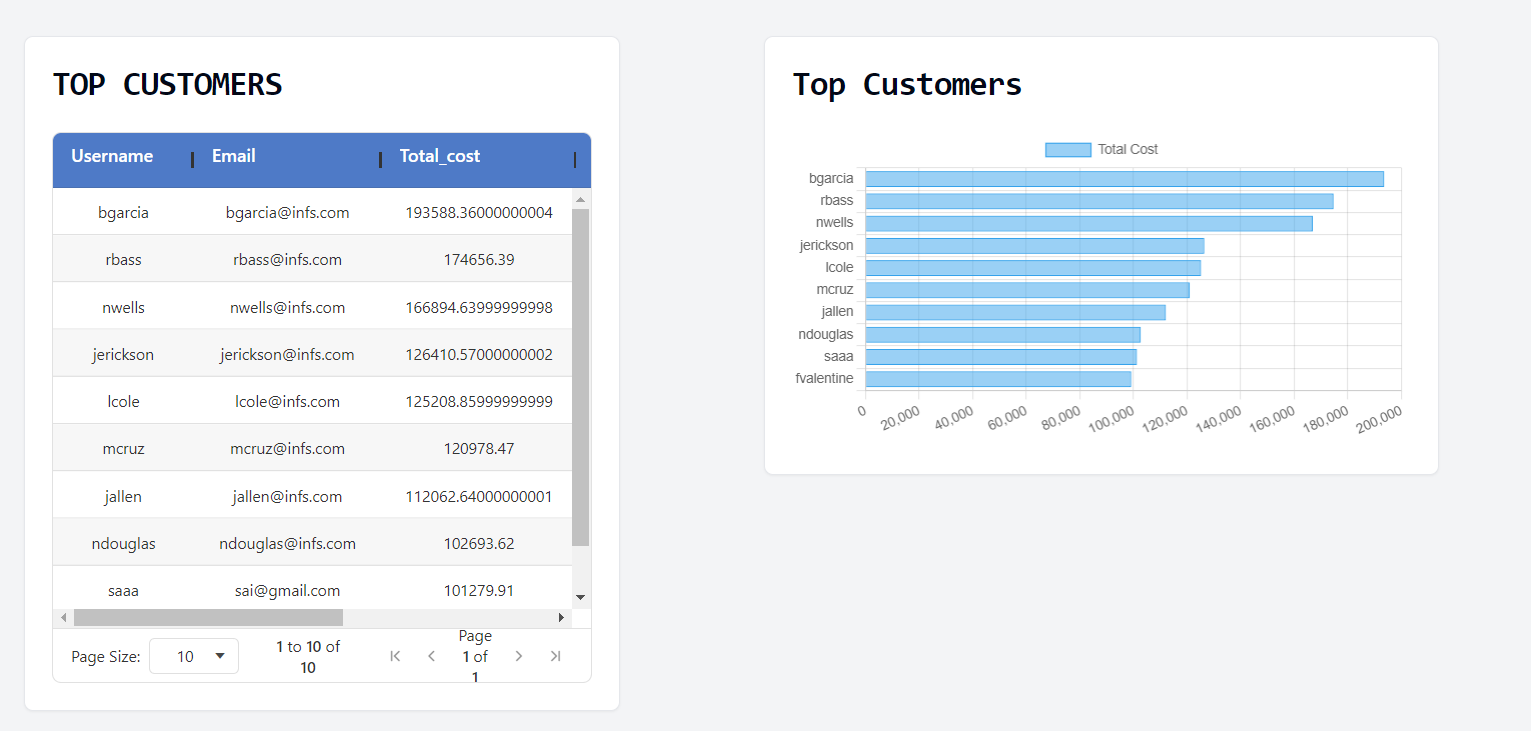


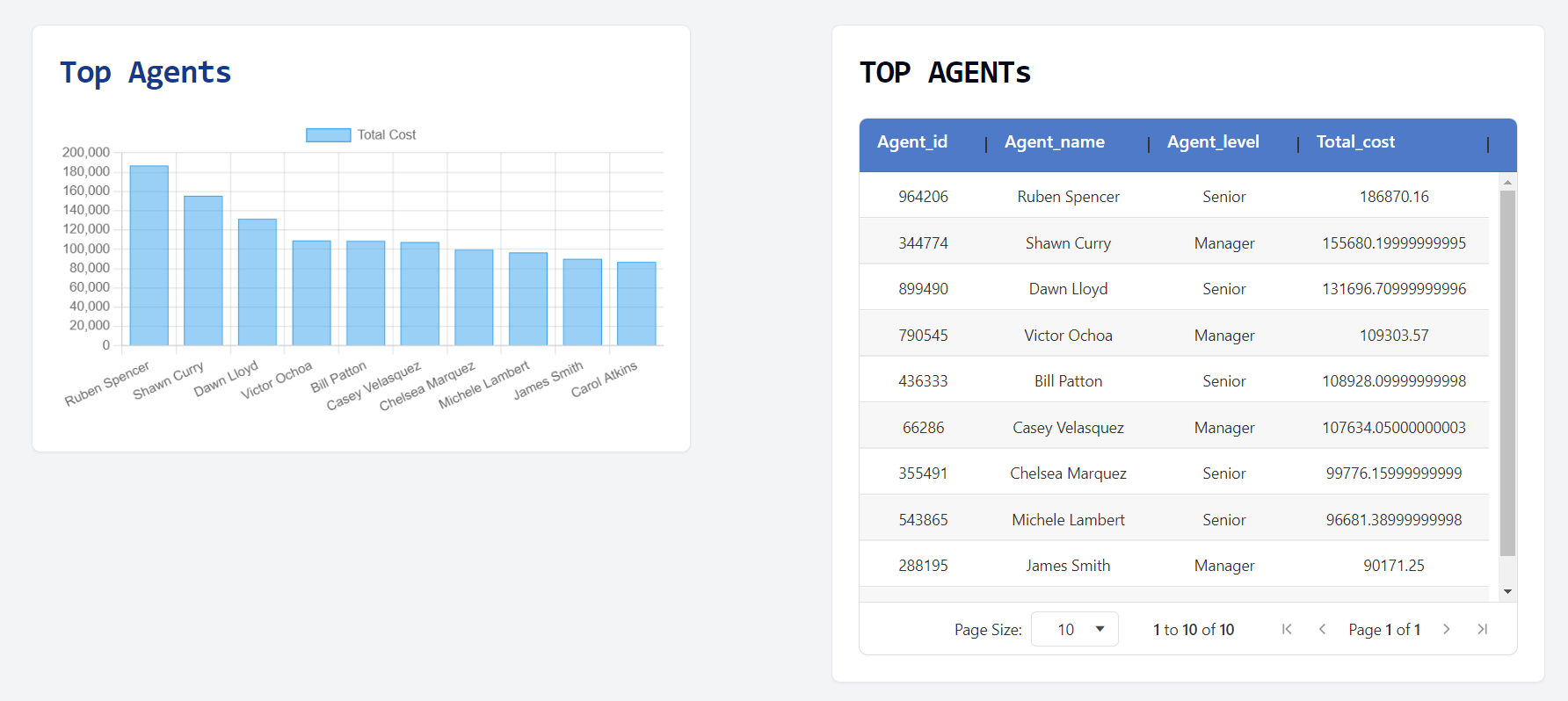
### **Visualizations:** **Admin Side -**

Transactions:



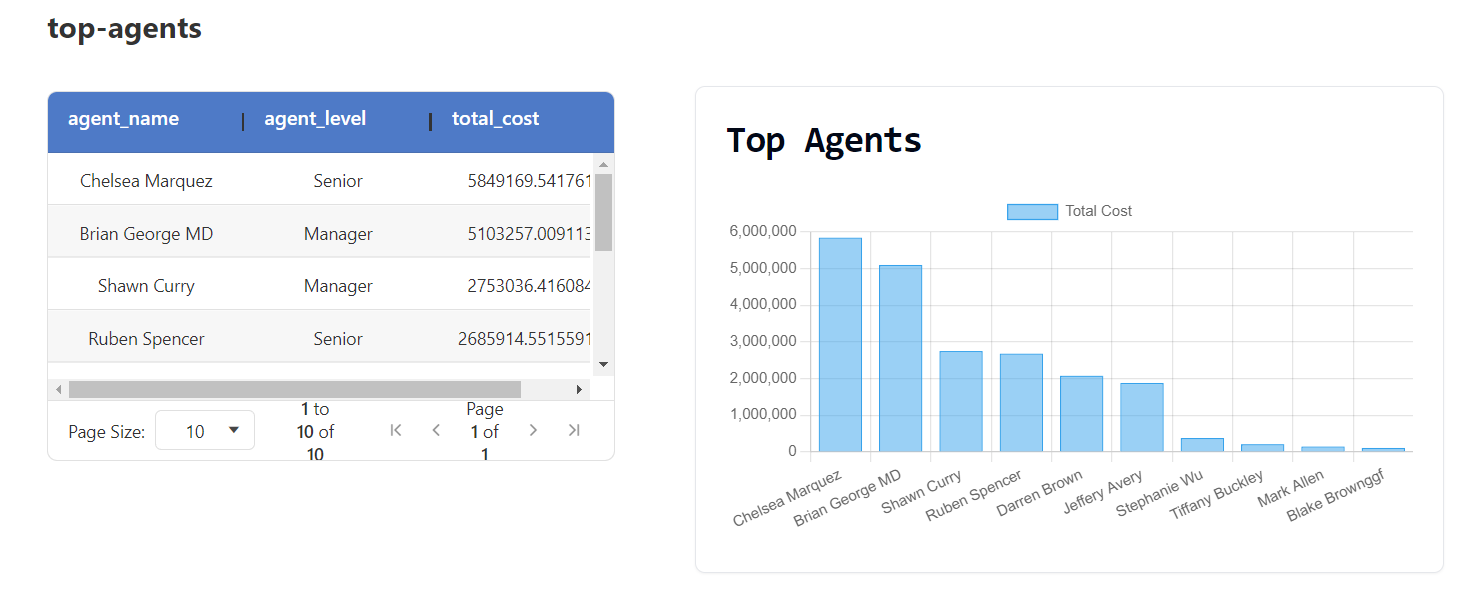


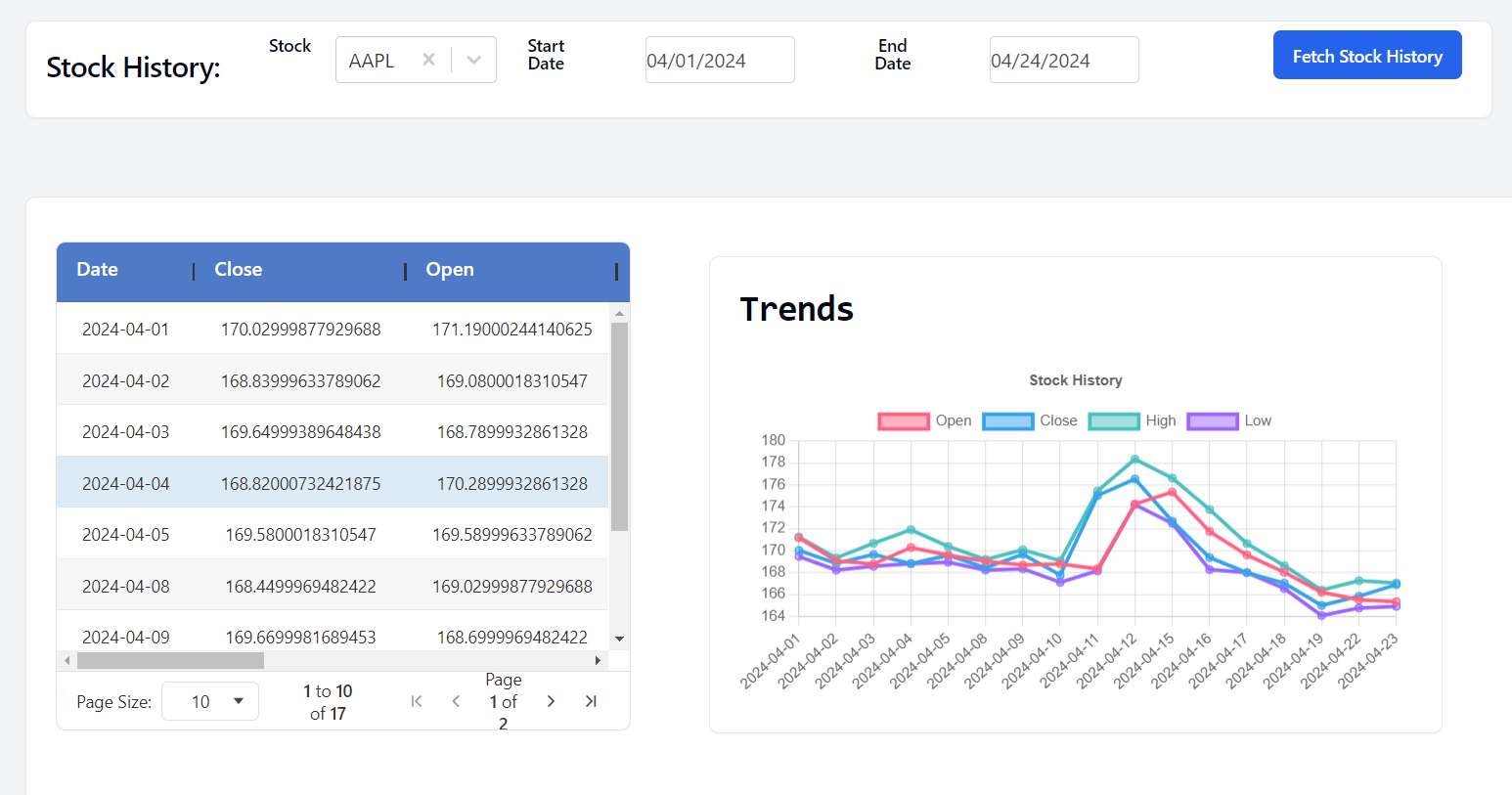
Customer:  


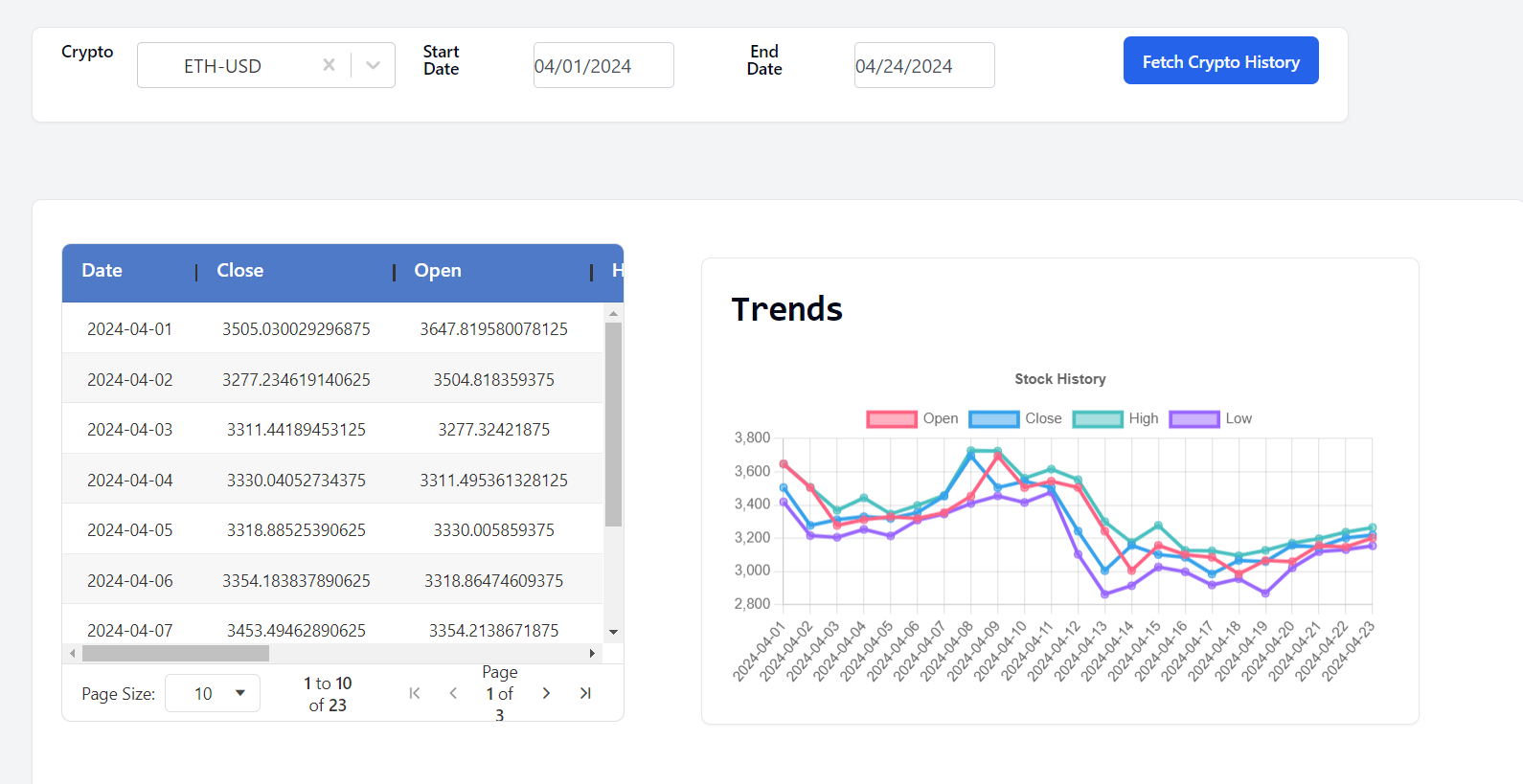
Agents:   


Stock:  
based on particular stock   


Crypto:   
based on particular crypto

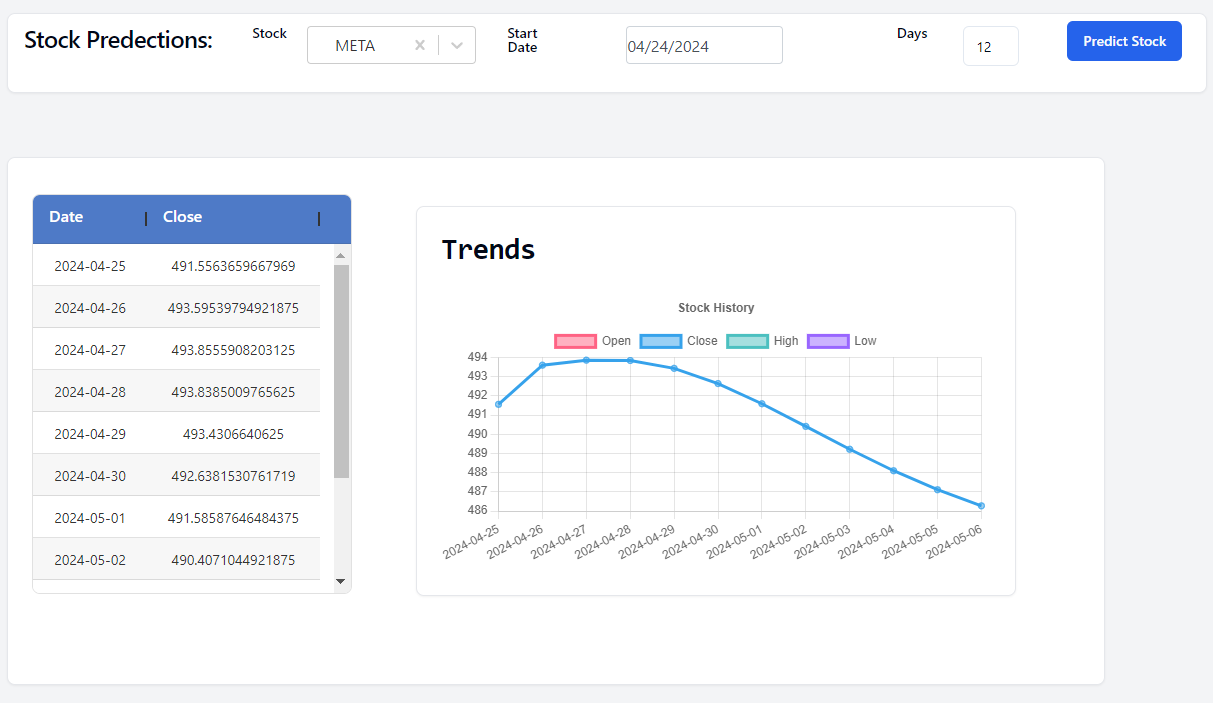


Customer Side StockAnalyisis :  


Customer Side CryptoAnaysis:   


### **ML Prediction Model:**

We have built a keras LSTM models for each stock and created an end point to predict the `Close’ given a stock based on the start date.



## **Member Contributions**

We have split the task into three phrases and divided the work in the below format to ensure we do not have any code conflicts. We have only provided the major contributions below. Both of us have worked on the all the components. The planning was done as below:

1. Phase - I
2. Backend Endpoints & python scripts to generate synthetic data for collections (Customer, Agents & Transactions) (Abhishek)
3. Frontend Endpoints and python scripts for yfinance API to fetch stocks and crypto data (Teja)
4. Phase – II
5. 3 Queries and 2 variations each for Stock Data (Teja) (Total: 11)
6. 3 Queries and 2 variation each for Crypto Data (Abhishek) (Total :9)
7. Phase – III
8. Visualizations for the end points. (Teja)
9. ML Model for predictions and attempted recommendation system for stocks (Abhishek)

Any issues with the tasks were troubleshooted from both of us and can we observed from the github repo commits and pull requests.

GitHub link: <https://github.com/abhishekjallawaram/StockSphere>

## **Challenges & Future Work**

We have attempted to build a recommendation system for the stock data and faced challenges in getting the final ratings for the stock to provide the best action [“Sell”, “Buy”, “Hold”]. We would need more data and a better algorithmic approach to implement the recommendation system. We have provided a base script we attempted in the tests folder for reference.

Due to the time constraints, we were unable to implement more visualizations and analysis end points we had in mind (Implemented queries: 6 with 2 other variations totaling 18). We other ideas we had were as below:

1. Implement stock and cryptocurrencies on active data as we were constrained with API calls. We have only taken a snap for a fixed timeframe.
2. Visualizations for stocks and crypto data for each stock/ crypto. We have focused more on a platform perspective to analyze the customers and agents emphasizing on the transactions.

## **Conclusion:**

This project has provided a platform in working efferently with NoSQL databases and an application which leverages the front-end(React) and back-end(FastAPI) components to build a Stocks and Crypto Management platform. The complex search queries and visualizations were simulating and essential to gain required hand-on on the industrial usage of NoSQL database and gain an understanding on the implementation of MongoDB in real life scenarios. Implementing the ML models and the attempt on recommendation system help us gain knowledge needed to further our careers in software industry. Overall the course and the project would be a substantial boost in pursuing a job in this domain.