INFS 740: Final Project (StockSphere)

Abhishek Jallawaram (G01373042)
Teja Sajja (G01395837)

Table of Contents

Project Overview: 2
- **Summary:** 2
- **Key Objectives:** 2
- **Project Tech Stack:** 3
  - **React** 3
  - **FastAPIs** 3
  - **MongoDB** 4

System Architecture: 4
- **Admins** 4
- **Customers** 4

Database Schema: 6
- **Collections:** 6

Functionalities 8
- **CRUD Operations:** 8
- **Complex Queries:** 10
  - **Stock Queries** 10
  - **Crypto Queries** 12
- **UI Screenshorts:** 15
- Customer Access: 15
- Admin Access: 17
- **Visualizations: Admin Side -** 21
- **ML Prediction Model:** 24

Member Contributions 25

Challenges & Future Work 25

Conclusion: 26

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

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

FastAPIs

a) We have utilized the fastAPI to build the backed APIs needed to communicate with the MongoDB database and fetch the complex queriers.
b) 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.
c) Native support for asynchronous request handling, which is essential for handling multiple data-intensive operations like fetching large datasets from a database.
d) FastAPI automatically generates interactive API documentation (using Swagger UI and ReDoc), which simplifies API testing and frontend integration.

MongoDB

a) We have utilized the MongoDB as the primary database to store and manage structured and unstructured data of the platform.
b) MongoDB's schema-less nature allows for flexible and dynamic data models, which is beneficial for handling diverse datasets such as stocks and cryptocurrencies.
c) MongoDB offers horizontal scalability through sharding, effectively managing large volumes and high throughput of data.
d) 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.

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)

```
Agent ^ Collapse all object
    agent_id* integer [0, 999999]
    name* string
    contact* string
    level* string
```

```
Customer ^ Collapse all object
    customer_id* integer [0, 999999]
    username* string
    email* string email
    hashed_password* string
    balance > Expand all (number | null)
    net_stock > Expand all number
    role > Expand all string
```

```
StockData ^ Collapse all object
    Open* number
    High* number
    Low* number
    Close* number
    Adj Close* number
    Volume* integer
    Company_ticker* string
    date* string
```

```
Stock ^ Collapse all object
    stock_id* integer [0, 999999]
    Company_name* string
    Company_ticker* string
    Closed_price* number
    Company_info* string
    Company_PE > Expand all (number | null)
    Company_cash_flow > Expand all (number | null)
    Company_dividend > Expand all (number | null)
```

```
Crypto ^ Collapse all object
  crypto_id* integer [0, 999999]
  Name* string
  Symbol* string
  Last_Close* number
  Market_Cap > Expand all (number | null)
  Volume_24h > Expand all (number | null)
  Circulating_Supply > Expand all (number | null)
```

```
CryptoData ^ Collapse all object
  Open* number
  High* number
  Low* number
  Close* number
  Adj Close* number
  Volume* integer
  Symbol* string
  date* string
```

```
Transaction ^ Collapse all object
  transaction_id* integer [0, 999999]
  customer_id* integer [0, 999999]
  stock_id* > Expand all (integer | null)
  crypto_id* > Expand all (integer | null)
  agent_id* integer [0, 999999]
  ticket* string
  volume* integer
  each_cost* number
  action* string
  date string date-time
```

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.

    a) Customer.customer_id = Transactions.customer_id
    b) Agent.agent_id = Transactions.agent_id
    c) Stocks.stock_id = Transactions.stock_id
    d) Stocks.Company_ticker = Transactions.ticket
    e) Cryptocurrencies.crypto_id = Transactions.crypto_id
    f) 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:

Agents

GET	/api/agents/	Get Agents
POST	/api/agents/	Create Agent
GET	/api/agents/{agent_id}	Get Agent
PUT	/api/agents/{agent_id}	Update Agent
DELETE	/api/agents/admin	Delete Agent

Customers

GET	/api/customers/admin	Get Customers
DELETE	/api/customers/admin	Delete Customers
GET	/api/customers/admin/{customerid}	Read Customer Byid
PUT	/api/customers/admin/{customerid}	Update Customer
GET	/api/customers/admin/{username}	Read Customer Byusernamе

Stocks

GET	/api/stocks/	Get Stocks
POST	/api/stocks/	Create Stock
GET	/api/stocks/{stockid}	Read Stock Byid
PUT	/api/stocks/{stockid}	Update Stock
POST	/api/stocks/{ytic}	Create Item
PUT	/api/stocks/{ytic}	Update Stock By Ticker
DELETE	/api/stocks/{ytic}	Delete Stock By Ticker
DELETE	/api/stocks/admin	Delete Stock

Stock-History

GET	/api/stock-history/	Get Stocksdata
POST	/api/stock-history/	Create Stockdata
POST	/api/stock-history/range	Get Stocks Data In Range
PUT	/api/stock-history/{stock_id}	Update Stockdata
DELETE	/api/stock-history/{stock_id}	Delete Stockdata

Cryptocurrencies

GET	/api/cryptocurrencies/	Get Cryptos
POST	/api/cryptocurrencies/	Create Crypto
GET	/api/cryptocurrencies/{cryptoid}	Read Crypto Byid
PUT	/api/cryptocurrencies/{cryptoid}	Update Crypto
POST	/api/cryptocurrencies/{ytic}	Create Crypto By Ticker
PUT	/api/cryptocurrencies/{ytic}	Update Crypto By Ticker
DELETE	/api/cryptocurrencies/{ytic}	Delete Crypto By Ticker
DELETE	/api/cryptocurrencies/admin	Delete Crypto

Crypto_History

GET	/api/crypto_history/	Get Cryptos Data
POST	/api/crypto_history/	Create Crypto Data
POST	/api/crypto_history/range	Get Stocks Data In Range
PUT	/api/crypto_history/{crypto_id}	Update Crypto Data
DELETE	/api/crypto_history/{crypto_id}	Delete Crypto Data

Transactions

GET	/api/transactions/	Get Stocks
POST	/api/transactions/	Create Stock
GET	/api/transactions/customer/stocks	Get Customer Stock Transactions
GET	/api/transactions/customer/cryptos	Get Customer Crypto Transactions
GET	/api/transactions/customer-stocks	Get Customer Stocks
GET	/api/transactions/customer-cryptos	Get Customer Cryptos
GET	/api/transactions/admin/	Get Stocks
GET	/api/transactions/adminpro/	Get Transactions
GET	/api/transactions/admin/{transcationid}	Read Transaction Byid
PUT	/api/transactions/admin/{transcationid}	Update Transaction
POST	/api/transactions/admin/[transactions]	Get Transactions
POST	/api/transactions/admin	Create Stock
DELETE	/api/transactions/admin	Delete Transaction

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.

```python
@router.get("/customers/most-stock-transactions", response_model=List[CustomerTransactionDetail], summary="Retrieve Top 10 Stock Customers Transactions")
Pieces: Comment | Pieces: Explain
async def get_customers_with_most_stock_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"crypto_id": {"$eq": 0}}).to_list(length=None)
    customer_ids = {t["customer_id"] for t in transactions if "customer_id" in t}
    agent_ids = {t["agent_id"] for t in transactions if "agent_id" in t}
    customers = {c["customer_id"]: c for c in await collections["customers"].find({"customer_id": {"$in": list(customer_ids)}}).to_list(length=None)}
    agents = {a["agent_id"]: a for a in await collections["agents"].find({"agent_id": {"$in": list(agent_ids)}}).to_list(length=None)}

    customer_aggregates = {}
    for transaction in transactions:
        customer_id = transaction.get("customer_id")
        if customer_id:
            if customer_id not in customer_aggregates:
                customer_aggregates[customer_id] = {
                    "total_transactions": 0,
                    "customer_info": customers.get(customer_id),
                    "agent_info": agents.get(transaction.get("agent_id"))
                }
            customer_aggregates[customer_id]["total_transactions"] += 1

    sorted_customers = sorted(customer_aggregates.items(), key=lambda x: x[1]["total_transactions"], reverse=True)[:10]
    results = [
        CustomerTransactionDetail(
            customer_id=(key),

            username=val["customer_info"]["username"] if val["customer_info"] else None,
            email=val["customer_info"]["email"] if val["customer_info"] else None,
            total_transactions=val["total_transactions"],
            agent_name=val["agent_info"]["name"] if val["agent_info"] else None,
            agent_level=val["agent_info"]["level"] if val["agent_info"] else None
        ) for key, val in sorted_customers
    ]
    return results
```

Query2:

Customers with Top Transactions in the Platform with Stocks Data.

```python
@router.get("/customers/top-stock-transactions", response_model=List[CustomerStockTransactionDetail], summary="Retrieve Top 10 Stock Customers")
async def get_customers_with_top_stock_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"crypto_id": {"$eq": 0}}).to_list(length=None)
    stocks_dict = {stock['Company_ticker']: stock for stock in await collections["stocks"].find().to_list(length=None)}
    customers_dict = {customer['customer_id']: customer for customer in await collections["customers"].find().to_list(length=None)}
    agents_dict = {agent['agent_id']: agent for agent in await collections["agents"].find().to_list(length=None)}
    customer_aggregates = {}

    for transaction in transactions:
        stock_info = stocks_dict.get(transaction.get('ticket'))
        customer_info = customers_dict.get(transaction.get('customer_id'))
        agent_info = agents_dict.get(transaction.get('agent_id'))

        if customer_info and agent_info:
            customer_id = transaction['customer_id']
            if customer_id not in customer_aggregates:
                customer_aggregates[customer_id] = {
                    'total_cost': 0,
                    'customer_info': customer_info,
                    'agent_info': agent_info
                }

            volume = transaction.get('volume', 0)
            each_cost = transaction.get('each_cost', 0)
            action_multiplier = 1 if transaction.get('action') == 'buy' else -1
            total_cost = volume * each_cost * action_multiplier

            customer_aggregates[customer_id]['total_cost'] += total_cost
    limit = 10
    sorted_customers = sorted(customer_aggregates.items(), key=lambda x: x[1]['total_cost'], reverse=True)[:limit]

    results = [
        CustomerStockTransactionDetail(
            customer_id=(cust_id),
            username=cust['customer_info']['username'],
            email=cust['customer_info']['email'],
            total_cost=cust['total_cost'],
            agent_name=cust['agent_info']['name'] if cust['agent_info'] else None,
            agent_level=cust['agent_info']['level'] if cust['agent_info'] else None
        ) for cust_id, cust in sorted_customers
    ]
    return results
```

Query3:

Agents with Top Transactions in the Platform with Stocks Data.

```python
@router.get("/agents/top-stock-transactions", response_model=List[AgentTransactionDetail], summary="Retrieve Top 10 Stock Customers")
async def get_agents_with_top_stock_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"crypto_id": {"$eq": 0}}).to_list(length=None)
    agent_ids = {t["agent_id"] for t in transactions if "agent_id" in t}
    agents = {a["agent_id"]: a for a in await collections["agents"].find({"agent_id": {"$in": list(agent_ids)}}).to_list(length=None)}

    agent_totals = {}
    for transaction in transactions:
        agent_id = transaction.get("agent_id")
        if agent_id:
            if agent_id not in agent_totals:
                agent_totals[agent_id] = {
                    "total_cost": 0,
                    "agent_info": agents.get(agent_id)
                }
            volume = transaction.get("volume", 0)
            each_cost = transaction.get("each_cost", 0)
            action = transaction.get("action", "")
            multiplier = 1 if action == "buy" else -1
            total_cost_contribution = volume * each_cost * multiplier

            agent_totals[agent_id]['total_cost'] += total_cost_contribution
    sorted_agents = sorted(agent_totals.items(), key=lambda x: x[1]["total_cost"], reverse=True)[:10]

    results = [
        AgentTransactionDetail(
            agent_id=(agent_id),
            agent_name=info["agent_info"]["name"] if info["agent_info"] else None,
            agent_level=info["agent_info"]["level"] if info["agent_info"] else None,
            total_cost=info["total_cost"]
        ) for agent_id, info in sorted_agents
    ]

    return results
```

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.

```python
@router.get("/agents/top-crypto-transactions", response_model=List[AgentCryptoTransactionDetail],summary="Retrieve Top 10 Crypto Agents")
async def get_agents_with_top_crypto_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"stock_id": {"$eq": 0}}).to_list(length=None)
    agent_ids = {t["agent_id"] for t in transactions if "agent_id" in t}
    agents = {a["agent_id"]: a for a in await collections["agents"].find({"agent_id": {"$in": list(agent_ids)}}).to_list(length=None)}
    agent_totals = {}
    for transaction in transactions:
        agent_id = transaction.get("agent_id")
        if agent_id:
            if agent_id not in agent_totals:
                agent_totals[agent_id] = {
                    "total_cost": 0,
                    "agent_info": agents.get(agent_id)
                }
            volume = transaction.get("volume", 0)
            each_cost = transaction.get("each_cost", 0)
            action = transaction.get("action", "")
            multiplier = 1 if action == "buy" else -1
            total_cost_contribution = volume * each_cost * multiplier
            agent_totals[agent_id]['total_cost'] += total_cost_contribution
    limit = 10
    sorted_agents = sorted(agent_totals.items(), key=lambda x: x[1]["total_cost"], reverse=True)[:limit]
    results = [
        AgentCryptoTransactionDetail(
            agent_id=(agent_id),
            agent_name=info["agent_info"]["name"] if info["agent_info"] else None,
            agent_level=info["agent_info"]["level"] if info["agent_info"] else None,
            total_cost=info["total_cost"]
        ) for agent_id, info in sorted_agents
    ]
    return results
```

Query5:

Customers with Top Transactions in the Platform with Crypto Data.

```python
@router.get("/customers/top-crypto-transactions", response_model=List[CustomerCryptoTransactionDetail], summary="Retrieve Top 10 Crypto Customers Transactions")
async def get_customers_with_top_crypto_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"stock_id": {"$eq": 0}}).to_list(length=None)
    crypto_dict = {crypto['Symbol']: crypto for crypto in await collections["cryptocurrencies"].find().to_list(length=None)}
    customers_dict = {customer['customer_id']: customer for customer in await collections["customers"].find().to_list(length=None)}
    agents_dict = {agent['agent_id']: agent for agent in await collections["agents"].find().to_list(length=None)}
    customer_aggregates = {}
    for transaction in transactions:
        crypto_info = crypto_dict.get(transaction.get('ticket'))
        customer_info = customers_dict.get(transaction.get('customer_id'))
        agent_info = agents_dict.get(transaction.get('agent_id'))
        if customer_info and agent_info:
            customer_id = transaction['customer_id']
            if customer_id not in customer_aggregates:
                customer_aggregates[customer_id] = {
                    'total_cost': 0,
                    'customer_info': customer_info,
                    'agent_info': agent_info
                }
            volume = transaction.get('volume', 0)
            each_cost = transaction.get('each_cost', 0)
            action_multiplier = 1 if transaction.get('action') == 'buy' else -1
            total_cost = volume * each_cost * action_multiplier

            customer_aggregates[customer_id]['total_cost'] += total_cost
    limit = 10
    sorted_customers = sorted(customer_aggregates.items(), key=lambda x: x[1]['total_cost'], reverse=True)[:limit]
    results = [
        CustomerCryptoTransactionDetail(
            customer_id=(cust_id),
            username=cust['customer_info']['username'],
            email=cust['customer_info']['email'],
            total_cost=cust['total_cost'],
            agent_name=cust['agent_info']['name'] if cust['agent_info'] else None,
            agent_level=cust['agent_info']['level'] if cust['agent_info'] else None
        ) for cust_id, cust in sorted_customers
    ]
    return results
```

Query6:

Agents with Top Transactions in the Platform with Crypto Data.

```python
@router.get("/agents/top-crypto-transactions", response_model=List[AgentCryptoTransactionDetail],summary="Retrieve Top 10 Crypto Agents")
async def get_agents_with_top_crypto_transactions(user: Customer = Depends(authutils.get_current_admin)):
    collections = await get_collections()
    transactions = await collections["transactions"].find({"stock_id": {"$eq": 0}}).to_list(length=None)
    agent_ids = {t["agent_id"] for t in transactions if "agent_id" in t}
    agents = {a["agent_id"]: a for a in await collections["agents"].find({"agent_id": {"$in": list(agent_ids)}}).to_list(length=None)}
    agent_totals = {}
    for transaction in transactions:
        agent_id = transaction.get("agent_id")
        if agent_id:
            if agent_id not in agent_totals:
                agent_totals[agent_id] = {
                    "total_cost": 0,
                    "agent_info": agents.get(agent_id)
                }
            volume = transaction.get("volume", 0)
            each_cost = transaction.get("each_cost", 0)
            action = transaction.get("action", "")
            multiplier = 1 if action == "buy" else -1
            total_cost_contribution = volume * each_cost * multiplier
            agent_totals[agent_id]['total_cost'] += total_cost_contribution
    limit = 10
    sorted_agents = sorted(agent_totals.items(), key=lambda x: x[1]["total_cost"], reverse=True)[:limit]
    results = [
        AgentCryptoTransactionDetail(
            agent_id=(agent_id),
            agent_name=info["agent_info"]["name"] if info["agent_info"] else None,
            agent_level=info["agent_info"]["level"] if info["agent_info"] else None,
            total_cost=info["total_cost"]
        ) for agent_id, info in sorted_agents
    ]
    return results
```

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)

Queries

GET	/api/queries/customers/most-stock-transactions	Retrieve Top 10 Stock Customers Transactions
GET	/api/queries/agents/top-stock-transactions	Retrieve Top 10 Stock Customers
GET	/api/queries/customers/top-stock-transactions	Retrieve Top 10 Stock Customers
GET	/api/queries/customers/top-crypto-transactions	Retrieve Top 10 Crypto Customers Transactions
GET	/api/queries/agents/top-crypto-transactions	Retrieve Top 10 Crypto Agents
GET	/api/queries/customers/most-crypto-transactions	Retrieve Top 10 Crypto Customers
GET	/api/queries/transactions/{stock_id}/most-stock-transactions	Retrieve Top Stock Customers Transactions by stock_id
GET	/api/queries/transactions/by-ticker/{Company_ticker}/most-stock-transactions	Retrieve Top Stock Customers Transactions by Ticker
GET	/api/queries/transactions/{crypto_id}/most-stock-transactions	Retrieve Top Crypto Customers Transactions by crypto_id

GET	/api/queries/transactions/by-symbol/{Symbol}/most-stock-transactions	Retrieve Top Crypto Customers Transactions by Symbol
GET	/api/queries/transactions/{stock_id}/top-agents	Retrieve Top Stock Agents by stock_id
GET	/api/queries/transactions/{crypto_id}/top-agent	Retrieve Top Crypto Agents by crypto_id
GET	/api/queries/transactions/by-ticker/{Company_ticker}/top-agents	Retrieve Top Stock Agents by Ticker
GET	/api/queries/transactions/by-Symbol/{Symbol}/top-agents	Retrieve Top Crypto Agents by Symbol
GET	/api/queries/transactions/{stock_id}/top-customers	Retrieve Top Stock Customers by stock_id
GET	/api/queries/transactions/{crypto_id}/top-customer	Retrieve Top Crypto Customers by crypto_id
GET	/api/queries/transactions/by-ticker/{Company_ticker}/top-customers	Retrieve Top Stock Customers by Ticker
GET	/api/queries/transactions/by-Symbol/{Symbol}/top-customers	Retrieve Top Crypto Customers by Symbol

UI Screenshorts:

Login Page

Registration:

Customer Access:

Stock dashboard:

Crypto Dashboard:

Stock Analysis:

Crypto Analysis:

Admin Access:

Transaction: (CRUD)

Transaction_id	Customer_name	Agent_name	Ticket	Volume	Each_cost	Action	Date
562086	clopez	Kyle Olsen	MO	98	42.98	buy	2023-06-09T
652205	vlloyd	Chelsea Marquez	GILD	56	72.61	buy	2023-12-02T
493719	lmcconnell	Tammy White	F	60	12.91	buy	2022-09-28T
108195	sdvm	Ruben Spencer	ADBE	37	499.52	buy	2023-11-18T
759548	kswanson	Tammy White	XOM	50	113.49	buy	2024-03-06T
491658	fvalentine	Curtis Kline	NKE	21	93.86	sell	2024-03-15T
93479	jprice	Rebecca Jackson	T	83	16.98	buy	2022-04-28T
935425	mevans	Curtis Kline	MRK	90	123.85	sell	2022-12-30T
733023	eshu	Bill Patton	AAPL	82	172.28	sell	2024-03-17T
426768	cdavis	Carol Atkins	GS	99	406.82	buy	2022-04-20T

Customer: (CRUD)

Customer_id	Username	Email	Balance	Net_stock
263194	tpeterson11	tpeterson@infs.com	2789.18	253.3
179139	gg	ggould@infs.com	212121	2904.09
344700	sasas	ssdds@gmail.com	4897.93	618.66
137042	cjones	cjones@infs.com	4124.98	547.5
404950	dpowell	dpowell@infs.com	4583.74	1456.54
3979	mcruz	mcruz@infs.com	8770.56	776.7
306338	dwoodard	dwoodard@infs.com	3133.03	4455.2
679083	tcochran	tcochran@infs.com	3456.27	1623.83
799897	jparker	jparker@infs.com	8617.64	1010.78
700559	nwells	nwells@infs.com	5383.21	1136.49

Stocks: (CRUD)

Stock_id	Company_name	Company_ticker	Closed_price	Company_PE	Company_cash_flow	Company_dividend
1	Apple Inc.	AAPL	172.28	26.61215	116433002496	0.96
2	Microsoft Corporation	MSFT	428.74	38.164257	102646996992	3
3	Amazon.com, Inc.	AMZN	178.87	61.968967	84946001920	
4	Alphabet Inc.	GOOGL	150.77	25.918827	101745999872	
5	Meta Platforms, Inc.	META	509.58	33.782402	71112998912	2
6	Berkshire Hathaway Inc.	BRK-B	411.6	9.259544	49195999232	
7	Johnson & Johnson	JNJ	155.23	29.85	22791000064	4.76
8	Visa Inc.	V	283.26	32.360184	20198000640	2.08
9	The Procter & Gamble Company	PG	161.66	26.787626	19207999488	3.76
10	JPMorgan Chase & Co.	JPM	196.62	12.003697	12974000128	4.6

Crypto: (CRUD)

Add New Crypto | Delete Selected

Crypto_id	Name	Symbol	Last_Close	Market_Cap	Volume_24h	Circulating_Supply
1	Bitcoin	BTC-USD	67147.72	1218660663296	43772411904	19681676
2	Ethereum	ETH-USD	3239.3604	350362107904	24811804672	120070456
3	XRP	XRP-USD	0.5481835	24056819712	3175590912	55119896576
4	Litecoin	LTC-USD	86.255806	5439484416	1212121	74419080
5	Bitcoin Cash	BCH-USD	534.60785	9098350592	1060977984	19690934
6	Cardano	ADA-USD	0.503553	14940014592	1076938880	35613016064
7	Polkadot	DOT-USD	7.2424145	8514889728	571597120	1433428096
8	Chainlink	LINK-USD	15.159248	7186350592	858614656	587099968
9	Stellar	XLM-USD	0.11526594	2801108992	238474192	28877682688
10	BNB	BNB-USD	596.28754	78341324800	2279068160	149534256

Page Size: 10 1 to 10 of 53

Agents:

Add New Agent | Delete Selected

Agent_id	Name	Contact	Level
602251	Alexandria Jones2	449-270-6286	Junior
375382	string	45354355	Junior
773470	Blake Browngf	311-535-5719	Manager
790545	Victor Ochoa	485-713-8137	Manager
546658	Courtney Brewer	147-926-3677	Senior
144336	Rebecca Jackson	049-188-6799	Manager
443089	Darren Brown	843-973-5272	Junior
749484	Curtis Kline	079-267-5760	Junior
760112	Jared Manning	019-581-8409	Manager
475347	Robert Moreno	829-261-1171	Junior

Page Size: 10 1 to 10 of 64 Page 1 of 7

Visualizations:
Admin Side -

Transactions:

TOP AGENTs based on stocks

Agent_id	Agent_name	Agent_level	Total_cost
964206	Ruben Spencer	Senior	186870.16
344774	Shawn Curry	Manager	155680.19999999995
899490	Dawn Lloyd	Senior	131696.70999999996
790545	Victor Ochoa	Manager	109303.57
436333	Bill Patton	Senior	108928.09999999998
66286	Casey Velasquez	Manager	107634.05000000003
355491	Chelsea Marquez	Senior	99776.15999999999
543865	Michele Lambert	Senior	96681.38999999998
288195	James Smith	Manager	90171.25

Page Size: 10 1 to 10 of 10 Page 1 of 1

Top Agents

TOP CUSTOMERS based on crypto

Username	Email	Total_cost	Agent_name
tsajja	tsajja@example.com	8065163.967447492	Darren Brow
tpeterson11	tpeterson@infs.com	5104453.16685075	Andrew Hine
ddavis	ddavis@infs.com	2753043.59348742	Danny Mcca
jromero	jromero@infs.com	2486978.9516499797	Felicia Guerre
ewhite	ewhite@infs.com	1898701.9745694	Jon Edwards
jparker	jparker@infs.com	429035.1126148	William Peters
vlloyd	vlloyd@infs.com	355409.16678011	Jared Mannir
lellis	lellis@infs.com	158785.19834605997	Bill Patton
jerickson	jerickson@infs.com	43769.1813997	Elizabeth Menc

Page Size: 10 1 to 10 of 10 Page 1 of 1

Top Customers

TOP CUSTOMERS based on stocks

Username	Email	Total_cost	Agent_name
bgarcia	bgarcia@infs.com	193588.36000000004	Shawn Curry
rbass	rbass@infs.com	174656.39	Stephanie Ric
nwells	nwells@infs.com	166894.63999999998	Bill Patton
jerickson	jerickson@infs.com	126410.57000000002	Mark Allen
lcole	lcole@infs.com	125208.85999999999	Nicole Espinoz
mcruz	mcruz@infs.com	120978.47	Holly Cooper
jallen	jallen@infs.com	112062.64000000001	Alexandria Jone
ndouglas	ndouglas@infs.com	102693.62	Amber Moore
saaa	sai@gmail.com	101279.91	Jeffery Avery

Page Size: 10 1 to 10 of 10 Page 1 of 1

Top Customers

TOP AGENTs based on crypto

Agent_id	Agent_name	Agent_level	Total_cost
355491	Chelsea Marquez	Senior	5849169.54176115
220988	Brian George MD	Manager	5103257.00911366
344774	Shawn Curry	Manager	2753036.41608415
964206	Ruben Spencer	Senior	2685914.5515591884
443089	Darren Brown	Junior	2083977.7265230003
396882	Jeffery Avery	Senior	1883376.2495779602
81386	Stephanie Wu	Senior	381546.78004932
723050	Tiffany Buckley	Manager	214705.47683972074
861979	Mark Allen	Junior	151990.80770104

Page Size: 10 1 to 10 of 10 Page 1 of 1

Top Agents

Customer:

TOP CUSTOMERS

Username	Email	Total_cost
bgarcia	bgarcia@infs.com	193588.36000000004
rbass	rbass@infs.com	174656.39
nwells	nwells@infs.com	166894.63999999998
jerickson	jerickson@infs.com	126410.57000000002
lcole	lcole@infs.com	125208.85999999999
mcruz	mcruz@infs.com	120978.47
jallen	jallen@infs.com	112062.64000000001
ndouglas	ndouglas@infs.com	102693.62
saaa	sai@gmail.com	101279.91

Page Size: 10 1 to 10 of 10 Page 1 of 1

Top Customers

Agents:

Top Agents

TOP AGENTs

Agent_id	Agent_name	Agent_level	Total_cost
964206	Ruben Spencer	Senior	186870.16
344774	Shawn Curry	Manager	155680.19999999995
899490	Dawn Lloyd	Senior	131696.70999999996
790545	Victor Ochoa	Manager	109303.57
436333	Bill Patton	Senior	108928.09999999998
66286	Casey Velasquez	Manager	107634.05000000003
355491	Chelsea Marquez	Senior	99776.15999999999
543865	Michele Lambert	Senior	96681.38999999998
288195	James Smith	Manager	90171.25

Page Size: 10 1 to 10 of 10 Page 1 of 1

Stock:
based on particular stock

top-agents :

agent_name	agent_level	total_cost
Ruben Spencer	Senior	186870.16
Shawn Curry	Manager	155680.199999999
Dawn Lloyd	Senior	131696.709999999
Victor Ochoa	Manager	109303.57

Page Size: 10 | 1 to 10 of 10 | Page 1 of 1

Crypto:
based on particular crypto

top-agents

agent_name	agent_level	total_cost
Chelsea Marquez	Senior	5849169.541761
Brian George MD	Manager	5103257.009113
Shawn Curry	Manager	2753036.416084
Ruben Spencer	Senior	2685914.5515591

Page Size: 10 | 1 to 10 of 10 | Page 1 of 1

Customer Side StockAnalysis :

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:

a) Phase - I
   1) Backend Endpoints & python scripts to generate synthetic data for collections (Customer, Agents & Transactions) (Abhishek)
   2) Frontend Endpoints and python scripts for yfinance API to fetch stocks and crypto data (Teja)
b) Phase – II
   1) 3 Queries and 2 variations each for Stock Data (Teja) (Total: 11)
   2) 3 Queries and 2 variation each for Crypto Data (Abhishek) (Total :9)
c) Phase – III
   1) Visualizations for the end points. (Teja)
   2) 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.