INFS-740 Project

Anjali Sharma asharm32@gmu.edu

Table of Contents

Technology Stake 1
README 2
Presentation 3
MongoDB Database 3
Setup 3
Collections 3
Complex Queries: 7
Get Data for Home page: 7
Get companies with Market Capitalization Over $1 Trillion: 7
Get Highest Price from Historical Data Since 2023-01-01: 8
Get All Price History for Comparison and Verification Against the Summary Report of Highest Prices: 8
Python FLASK RESTful APIs 8
CRUD Operations and Additional Query APIs 9
Python Application for Machine Learning Model 14
Angular Frontend Web Application Setup 16
Testing of APIs - CRUD Operations- get, post, put, delete via postman. 32
Integration of API with Frontend Angular Application 34
Running the project 41
Project Overview Video Recording 43
References 43

Technology Stake

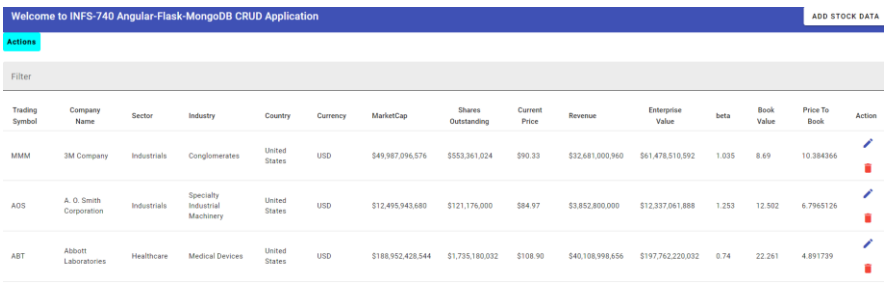
- **Backend Database**: MongoDB

- **REST API**: Python Flask
- **Web User Interface**: Angular Material Web Development Framework
- **Visualization**: D3.js Visualization Libraries
- **Machine Learning**: Scikit-learn, Matplotlib, Seaborn, NumPy, Pandas

README

1. Unzip the File:
   - Unzip the downloaded project file to a desired location.
   - Follow the respective sections of MongoDB, API and Angular Frontend in the document to perform setup.
2. Navigate to the Project Directory:
   - Open a terminal or command prompt.
   - Enter the following command to navigate to the project-app folder:
     - cd project-app

- API Setup:
  - Install Required Dependencies:
    - pip install Flask pymongo pandas numpy scikit-learn matplotlib seaborn
- Start the API:
  - python api.py
- The application will start and produce message
  - (* Running on http://127.0.0.1:5000*)
- At this point, the API server is successfully started.
- Once API started, navigate to the **Web Application Directory**:
  - Change to the web subdirectory within project-app:
    - cd project-app/web
    - ng serve
- After some time, you should see “Compiled successfully”.
- Verify http://localhost:4200/
- You should see stock data loaded on the UI, confirming that the MongoDB, API, and the front end are functioning correctly.



The submission zip contains:

1. A setup instructions document.
2. All source files, including the REST API, Angular Frontend, MongoDB Archive for setting up the database, and PowerPoint presentation.
3. Screenshots of all components of the application's functionality
4. A video recording of the application's functionality, titled "**video1133150885.mp4**".

Presentation

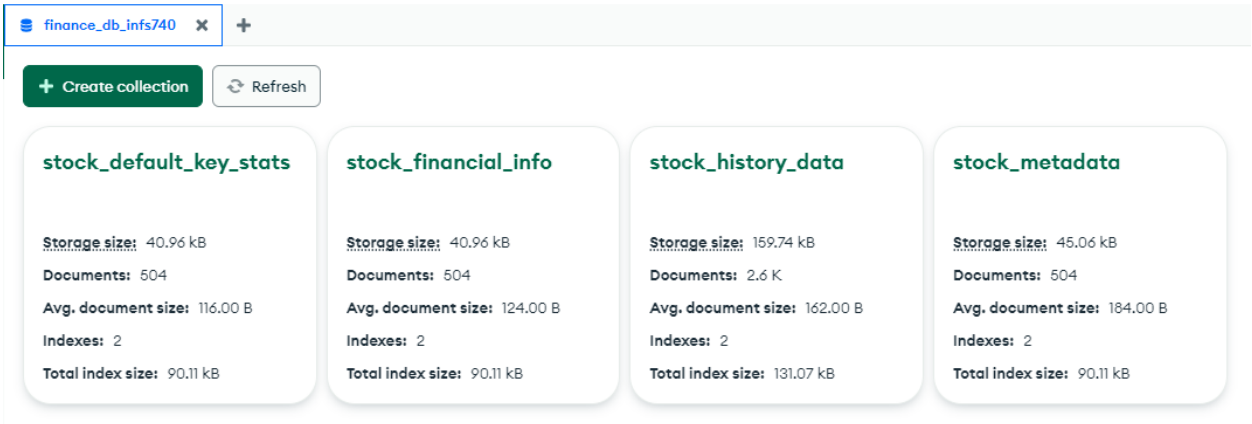
Open the PowerPoint presentation titled **INFS-740 Project Presentation.pptx** included with the submission. This will display all components of the project.

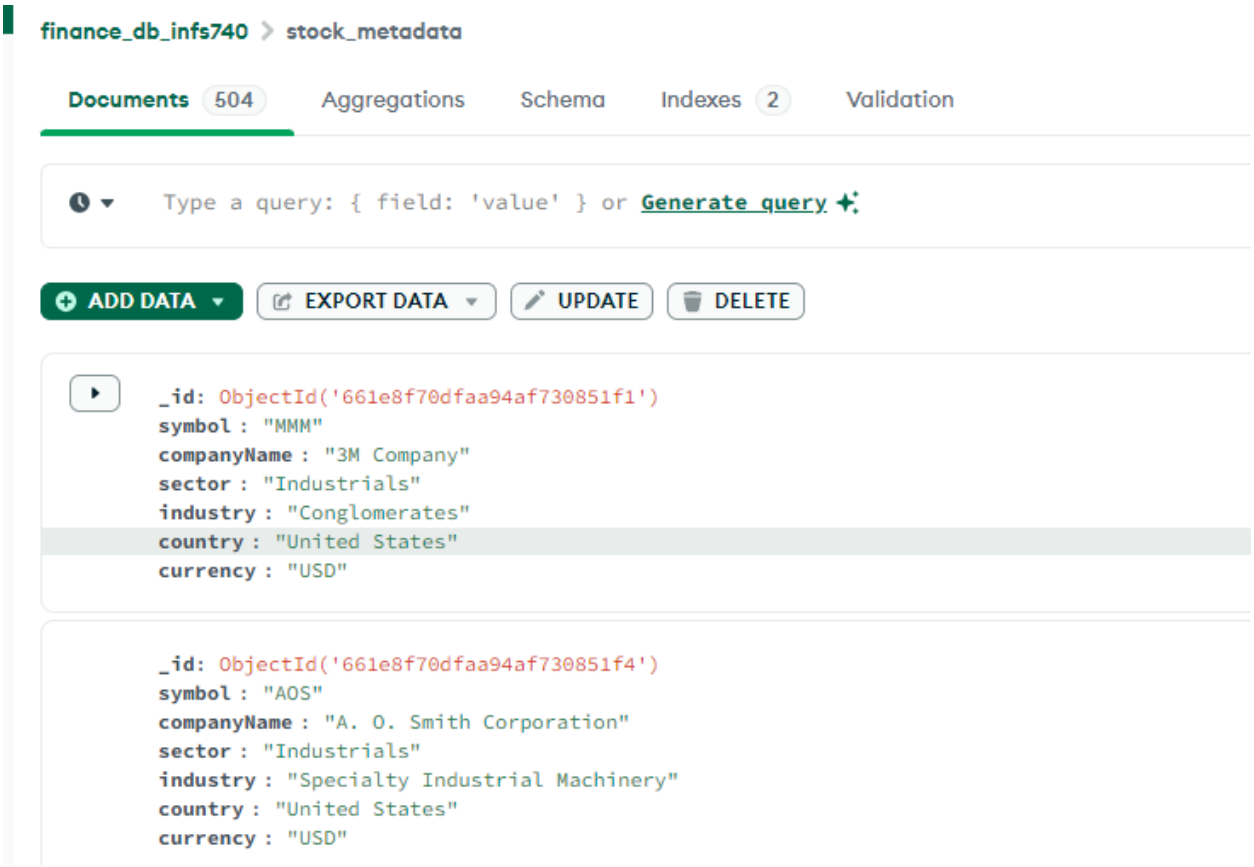
MongoDB Database

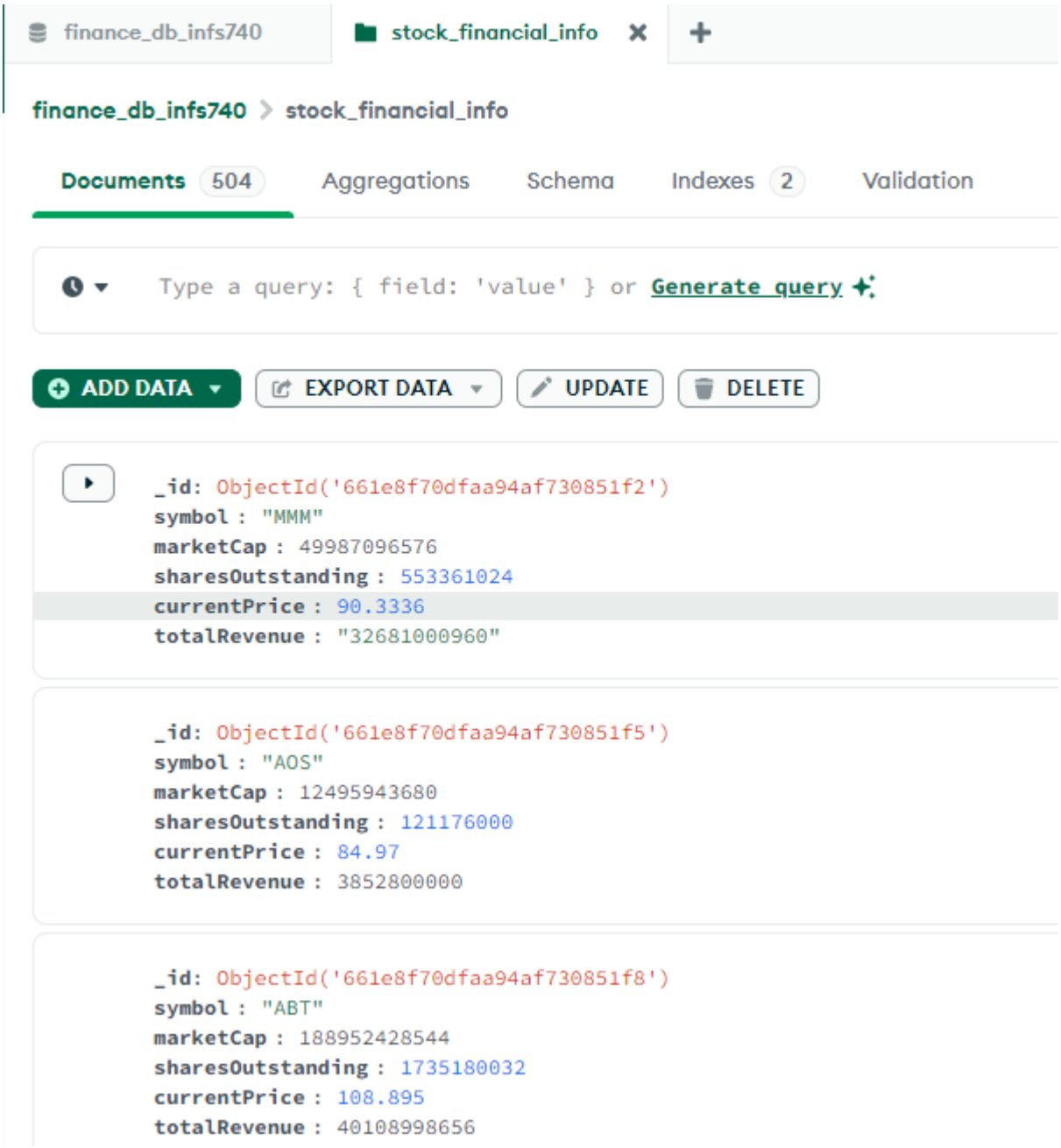
Setup

- Install MongoDB tools, including mongosh. Access the command line to proceed.
- Restore the MongoDB database from the provided archive (finance_db_infs740) located in the submission folder. This operation will recreate the database on localhost with all four collections used in the application.
- Run the following commands:
    - Open mongosh by typing mongosh in your command line.
    - Execute the database restore command
    - mongorestore --archive="finance_db_infs740" --host=localhost --port=27017

Collections




finance_db_infs740 > stock_metadata
Documents 504
Aggregations
Schema
Indexes 2
Validation
Type a query: { field: 'value' } or Generate query
ADD DATA
EXPORT DATA
UPDATE
DELETE
_id: ObjectId('661e8f70dfaa94af730851f1')
symbol : "MMM"
companyName : "3M Company"
sector : "Industrials"
industry : "Conglomerates"
country : "United States"
currency : "USD"
_id: ObjectId('661e8f70dfaa94af730851f4')
symbol : "AOS"
companyName : "A. O. Smith Corporation"
sector : "Industrials"
industry : "Specialty Industrial Machinery"
country : "United States"
currency : "USD"


finance_db_infs740
stock_financial_info
finance_db_infs740 > stock_financial_info
Documents 504
Aggregations
Schema
Indexes 2
Validation
Type a query: { field: 'value' } or Generate query
ADD DATA
EXPORT DATA
UPDATE
DELETE
_id: ObjectId('661e8f70dfaa94af730851f2')
symbol : "MMM"
marketCap : 49987096576
sharesOutstanding : 553361024
currentPrice : 90.3336
totalRevenue : "32681000960"
_id: ObjectId('661e8f70dfaa94af730851f5')
symbol : "AOS"
marketCap : 12495943680
sharesOutstanding : 121176000
currentPrice : 84.97
totalRevenue : 3852800000
_id: ObjectId('661e8f70dfaa94af730851f8')
symbol : "ABT"
marketCap : 188952428544
sharesOutstanding : 1735180032
currentPrice : 108.895
totalRevenue : 40108998656

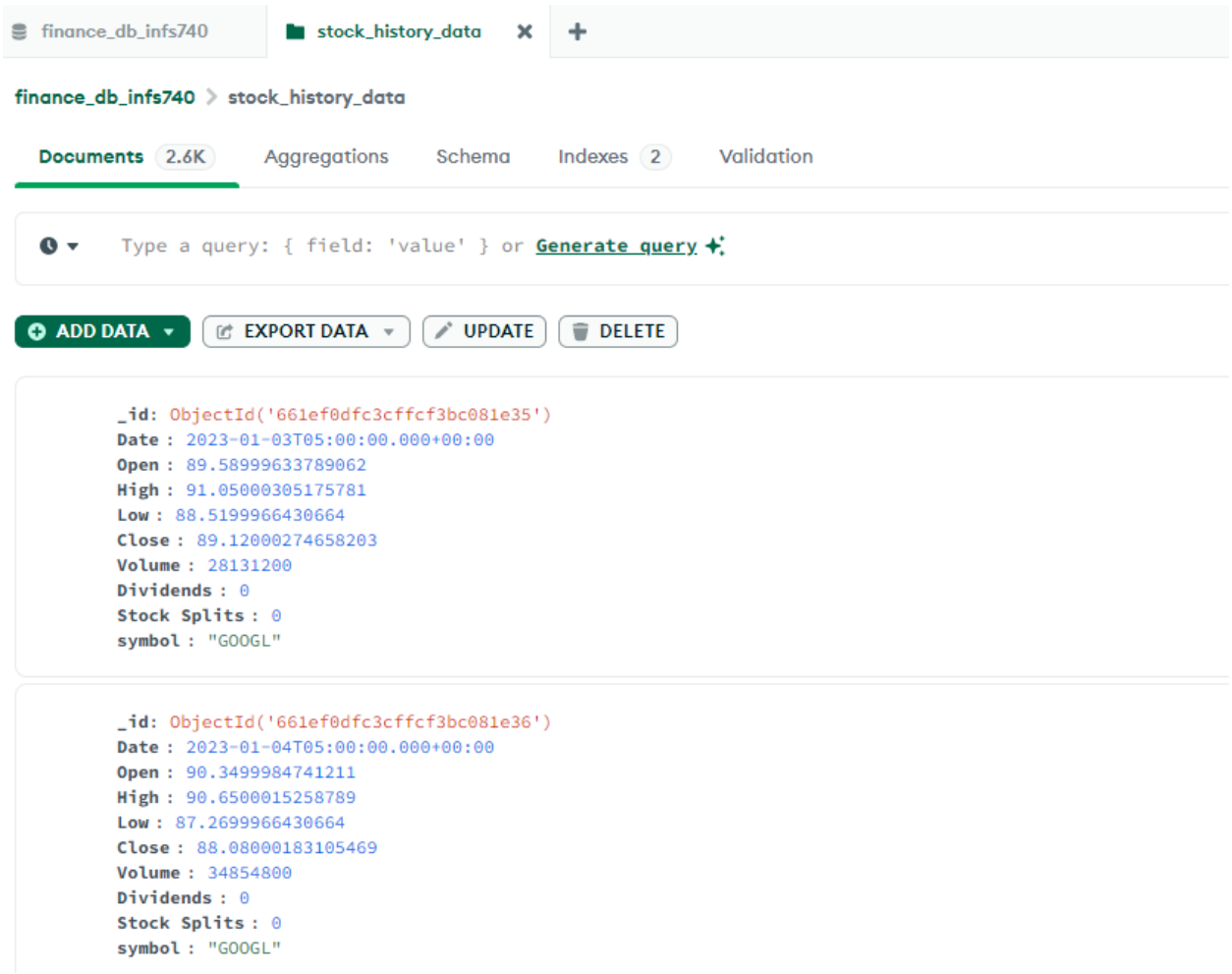
finance_db_infs740 > stock_default_key_stats

Documents 504 | Aggregations | Schema | Indexes 2 | Validation

Type a query: { field: 'value' } or Generate query

ADD DATA | EXPORT DATA | UPDATE | DELETE


```
_id: ObjectId('661e8f70dfaa94af730851f3')
symbol : "MMM"
enterpriseValue : 61478510592
beta : 1.035
bookValue : "8.69"
priceToBook : 10.384366

_id: ObjectId('661e8f70dfaa94af730851f6')
symbol : "AOS"
enterpriseValue : 12337061888
beta : 1.253
bookValue : 12.502
priceToBook : 6.7965126

_id: ObjectId('661e8f70dfaa94af730851f9')
symbol : "ABT"
enterpriseValue : 197762220032
beta : 0.74
bookValue : 22.261
priceToBook : 4.891739
```



Complex Queries:

Get Data for Home page:

Access data by querying the following three collections:

- stock_metadata
- stock_financial_info
- stock_default_key_stats

Get companies with Market Capitalization Over $1 Trillion:

Start by using the stock_financial_info collection as the primary source to filter trading symbols representing companies with a market capitalization exceeding $1 trillion. Then, join this data with the stock_metadata and stock_default_key_stats collections to gather comprehensive data for the user interface.

Collections involved:

- stock_financial_info
- stock_metadata
- stock_default_key_stats

Get Highest Price from Historical Data Since 2023-01-01:

Use the stock_history_data collection as the primary source to aggregate and summarize data. Group the data by symbol, and then join it with the stock_metadata and stock_financial_info collections to compile the necessary information for the user interface.

Collections involved:

- stock_history_data
- stock_metadata
- stock_financial_info

Get All Price History for Comparison and Verification Against the Summary Report of Highest Prices:

Access data from four collections to compare and check results with the summary report of the highest price page:

- stock_metadata
- stock_history_data
- stock_financial_info
- stock_default_key_stats

Python FLASK RESTful APIs

I have developed a Python Flask REST API application for fetching, adding, deleting, and updating data through an Angular Material web interface. The file **api.py** is included in the submission zip. Please ensure to install Flask, PyMongo, pandas, NumPy, scikit-learn, matplotlib, and seaborn by using the following pip install command:

pip install Flask pymongo pandas numpy scikit-learn matplotlib seaborn

```python
import yfinance as yf
from pymongo import MongoClient
from yahooquery import Ticker
from flask import Flask, jsonify, request
from flask_cors import CORS
from sklearn.linear_model import LinearRegression

...

def getDB():
    client = MongoClient('mongodb://localhost:27017/')
    db = client['finance_db_infs740']
    return db
```

CRUD Operations and Additional Query APIs

Get Data

```python
@app.route('/getData')
def get_data():
    """
        Get data for home page
    """
    db = getDB()
    stock_metadata = [s for s in db["stock_metadata"].find({}, {"_id": 0})]
    stock_financial_info = [s for s in db["stock_financial_info"].find({}, {"_id": 0})]
    stock_default_key_stats = [s for s in db["stock_default_key_stats"].find({}, {"_id": 0})]
    result = [
        {"symbol": sm["symbol"], "companyName": sm["companyName"], "sector": sm["sector"], "industry": sm["industry"],
         "country": sm["country"], "currency": sm["currency"]
            , "marketCap": sf["marketCap"], "sharesOutstanding": sf["sharesOutstanding"],
         "currentPrice": sf["currentPrice"], "totalRevenue": sf["totalRevenue"]
            , "enterpriseValue": sd["enterpriseValue"], "beta": sd["beta"], "bookValue": sd["bookValue"],
         "priceToBook": sd["priceToBook"]
         }
        for sm in stock_metadata
        for sf in stock_financial_info if sm["symbol"] == sf["symbol"]
        for sd in stock_default_key_stats if sm["symbol"] == sd["symbol"]
    ]

    return jsonify(result)
```

Update Data by trading symbol

```
@app.route('/updateData/<string:symbol>', methods=['PUT'])
def updateFinanceData(symbol: str):
    """
            API to update existing data for trading symbol
      """
    data = request.get_json()
    db = getDB()
    stock_metadata = db["stock_metadata"]
    stock_financial_info = db["stock_financial_info"]
    stock_default_key_stats = db["stock_default_key_stats"]
    if (stock_metadata.find_one({"symbol": symbol})) is not None:
        stock_metadata.update_one({"symbol": symbol},
                                  {"$set": {"companyName": data["companyName"], "sector": data["sector"],
                                            "industry": data["industry"]
                                     , "country": data["country"], "currency": data["currency"]}})
        stock_financial_info.update_one({"symbol": symbol},
                                        {"$set": {"marketCap": data["marketCap"],
                                                  "sharesOutstanding": data["sharesOutstanding"],
                                                  "currentPrice": data["currentPrice"],
                                                  "totalRevenue": data["totalRevenue"]}})
        stock_default_key_stats.update_one({"symbol": symbol},
                                           {"$set": {"enterpriseValue": data["enterpriseValue"], "beta": data["beta"],
                                                     "bookValue": data["bookValue"],
                                                     "priceToBook": data["priceToBook"]}})
    return get_data()  # jsonify([a for a in getDB()["test1"].find({}, {"_id": 0})])  # get_data()
```

Adding Data

```
@app.route('/addData', methods=['POST'])
def addFinanceData():
    """
              API to add new data
       """
    data = request.get_json()
    db = getDB()
    stock_metadata = db["stock_metadata"]
    stock_financial_info = db["stock_financial_info"]
    stock_default_key_stats = db["stock_default_key_stats"]
    stock_metadata.insert_one({"symbol": data["symbol"], "companyName": data["companyName"], "sector": data["sector"],
                               "industry": data["industry"], "country": data["country"], "currency": data["currency"]})
    stock_financial_info.insert_one(
        {"symbol": data["symbol"], "marketCap": data["marketCap"], "sharesOutstanding": data["sharesOutstanding"],
         "currentPrice": data["currentPrice"], "totalRevenue": data["totalRevenue"]})
    stock_default_key_stats.insert_one(
        {"symbol": data["symbol"], "enterpriseValue": data["enterpriseValue"], "beta": data["beta"],
         "bookValue": data["bookValue"], "priceToBook": data["priceToBook"]})
    return get_data()
```

Deleting Data

```python
@app.route('/deleteData/<string:symbol>', methods=['DELETE'])
def deleteFinanceData(symbol: str):
    """
                API to delete record for given symbol
    """
    db = getDB()
    stock_metadata = db["stock_metadata"]
    stock_financial_info = db["stock_financial_info"]
    stock_default_key_stats = db["stock_default_key_stats"]
    stock_metadata.delete_one({"symbol": symbol})
    stock_financial_info.delete_one({"symbol": symbol})
    stock_default_key_stats.delete_one({"symbol": symbol})
    return get_data()
```

Get Companies with Market Capitalization Over $1 Trillion

```python
@app.route('/get_marketCap_over_1T')
def get_marketCap_Over_1T():
    """
       Get companies with marketCap over $1 Trillion Dollors
    """
    db = getDB()
    stock_metadata = [s for s in db["stock_metadata"].find({}, {"_id": 0})]
    stock_default_key_stats = [s for s in db["stock_default_key_stats"].find({}, {"_id": 0})]
    stock_financial_info = [sf for sf in
                            db["stock_financial_info"].find({"marketCap": {"$gt": 1000000000000}}, {"_id": 0})]
    result = [
        {"symbol": sm["symbol"], "companyName": sm["companyName"], "sector": sm["sector"], "industry": sm["industry"],
         "country": sm["country"], "currency": sm["currency"]
            , "marketCap": sf["marketCap"], "sharesOutstanding": sf["sharesOutstanding"],
         "currentPrice": sf["currentPrice"], "totalRevenue": sf["totalRevenue"]
            , "enterpriseValue": sd["enterpriseValue"], "beta": sd["beta"], "bookValue": sd["bookValue"],
         "priceToBook": sd["priceToBook"]
         }
        for sm in stock_metadata
        for sf in stock_financial_info
        for sd in stock_default_key_stats
        if sm["symbol"] == sf["symbol"] and sm["symbol"] == sd["symbol"]
    ]
    return jsonify(result)
```

Get data for the Highest Price

```
@app.route('/get_pricing_history_highest_open')
def get_pricing_history_highest_open():
    db = getDB()
    """
       Get Highesr opening price
    """
    stock_history_data = [sh for sh in db["stock_history_data"].aggregate(
        [{"$group": {"_id": "$symbol", "maxOpenHigh": {"$max": "$High"}}}])]
    stock_metadata = [s for s in db["stock_metadata"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    stock_financial_info = [s for s in db["stock_financial_info"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    stock_default_key_stats = [s for s in db["stock_default_key_stats"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    result = [
        {"symbol": sm["symbol"], "companyName": sm["companyName"], "sector": sm["sector"], "industry": sm["industry"],
         "country": sm["country"], "currency": sm["currency"]
            , "marketCap": sf["marketCap"], "sharesOutstanding": sf["sharesOutstanding"],
         "totalRevenue": sf["totalRevenue"],
         "priceToBook": sd["priceToBook"]
            , "maxOpenHigh": sh["maxOpenHigh"]
         }
        for sm in stock_metadata
        for sf in stock_financial_info
        for sd in stock_default_key_stats
        for sh in stock_history_data
        if sm["symbol"] == sf["symbol"] and sm["symbol"] == sd["symbol"] and sm["symbol"] == sh["_id"]
    ]
    return jsonify(result)
```

Get Complete Price History

```python
@app.route('/get_all_pricing_history')
def get_all_pricing_history():
    db = getDB()
    '''
    All Pricing History
    '''
    stock_history_data = [sh for sh in db["stock_history_data"].find({}, {"_id": 0}).sort(
        {"High": -1, "Date": -1, "symbol": 1, "High": -1})]
    stock_metadata = [s for s in db["stock_metadata"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    stock_financial_info = [s for s in db["stock_financial_info"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    stock_default_key_stats = [s for s in db["stock_default_key_stats"].find(
        {"symbol": {"$in": ["GOOGL", "GOOG", "AMZN", "AAPL", "META", "MSFT", "NVDA", "TSLA"]}}, {"_id": 0})]
    result = [
        {"symbol": sm["symbol"], "companyName": sm["companyName"], "sector": sm["sector"], "industry": sm["industry"],
         "country": sm["country"], "currency": sm["currency"]
            , "marketCap": sf["marketCap"], "sharesOutstanding": sf["sharesOutstanding"],
         "priceToBook": sd["priceToBook"]
            , "Date": sh["Date"], "Open": sh["Open"], "High": sh["High"], "Low": sh["Low"], "Close": sh["Close"],
         "Volume": sh["Volume"]
         }
        for sm in stock_metadata
        for sf in stock_financial_info
        for sd in stock_default_key_stats
        for sh in stock_history_data
        if sm["symbol"] == sf["symbol"] and sm["symbol"] == sd["symbol"] and sm["symbol"] == sh["symbol"]
    ]
    return jsonify(result)
```

Get Full Price History by Trading Symbol

```python
@app.route('/get_all_pricing_history/<string:symbol>')
def get_all_pricing_history_by_symbol(symbol: str):
    """
       All Pricing History by trading symbol order by High, Date, Symbol
    """
    db = getDB()
    stock_history_data = [sh for sh in db["stock_history_data"].find({"symbol": symbol}, {"_id": 0}).sort(
        {"High": -1, "Date": -1, "symbol": 1})]
    #  db.stock_history_data.aggregate([{$group:{_id:"$symbol",maxOpenHigh:{$max:"$Open"},Date:{$max:"$Date"},Volume:{$max:"$Volume"}}}])
    stock_metadata = [s for s in db["stock_metadata"].find({"symbol": symbol}, {"_id": 0})]
    stock_financial_info = [s for s in db["stock_financial_info"].find({"symbol": symbol}, {"_id": 0})]
    stock_default_key_stats = [s for s in db["stock_default_key_stats"].find({"symbol": symbol}, {"_id": 0})]
    result = [
        {"symbol": sm["symbol"], "companyName": sm["companyName"], "sector": sm["sector"], "industry": sm["industry"],
         "country": sm["country"], "currency": sm["currency"]
            , "marketCap": sf["marketCap"], "sharesOutstanding": sf["sharesOutstanding"],
         "priceToBook": sd["priceToBook"]
            , "Date": sh["Date"], "Open": sh["Open"], "High": sh["High"], "Low": sh["Low"], "Close": sh["Close"],
         "Volume": sh["Volume"]
         }
        for sm in stock_metadata
        for sf in stock_financial_info
        for sd in stock_default_key_stats
        for sh in stock_history_data
        if sm["symbol"] == sf["symbol"] and sm["symbol"] == sd["symbol"] and sm["symbol"] == sh["symbol"]
    ]
    return jsonify(result)
```

Get Data for Visualization

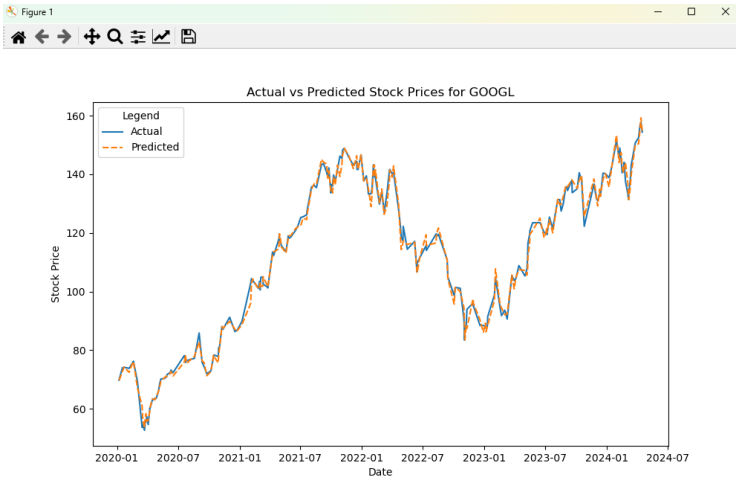
```
@app.route('/get_data_to_visualize')
def getDataToDisplay():
    """
        API to get data for Visualization - symbol, High Open Price
     """
    db = getDB()
    stock_history_data = [sh for sh in db["stock_history_data"].aggregate(
        [{"$group": {"_id": "$symbol", "maxOpenHigh": {"$max": "$High"}}}])]
    result = [{"symbol": d["_id"], "High": format(d["maxOpenHigh"], ".2f")} for d in stock_history_data]
    return jsonify(result)
```

Get Predictions for the Next Day's Closing Price

```
@app.route('/get_predictions')
def get_predictions():
    """
            API to get prediction for next day closing price
    """
    # Fetch stock data
    data = yf.download('AAPL', start='2010-01-01', end='2020-01-01')
    data['Prev Close'] = data['Close'].shift(1)
    data.dropna(inplace=True)

    # Prepare data for prediction
    X = data[['Prev Close']]
    y = data['Close']
    model = LinearRegression()
    model.fit(X, y)

    # Make a prediction for the next day
    prediction = model.predict([[data.iloc[-1]['Close']]])

    # Return the prediction
    return jsonify({'prediction': prediction[0]})
```

Python Application for Machine Learning Model

The file **infs_740_ml_linear_regression.py** is included in the submission. In this application, I have developed a Linear Regression model and calculated the Root Mean Square Error (RMSE). Based on the specified symbol, the application dynamically loads data, trains the model, makes predictions, calculates RMSE, and generates plots using the Matplotlib and Seaborn libraries




```python
import pandas as pd
import numpy as np
import yfinance as yf
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error

def predict_next_closing_price_symbol():
    # Fetch stock data
    ticker_symbol = 'AAPL'  ##use any symbol
    ticker_symbol = 'GOOGL'
    data = yf.download(ticker_symbol, start='2020-01-01', end='2025-01-01')

    # Prepare data, ensure it remains as DataFrame
    data = pd.DataFrame(data['Close'])
    data['Prev Close'] = data['Close'].shift(1)
    data.dropna(inplace=True)  # Remove any rows with NaN values
    # Setup features and target
    X = data[['Prev Close']]
    y = data['Close']
    data.sort_index(inplace=True)

    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # Model training using LinearRegression
    model = LinearRegression()
```

```
model.fit(X_train, y_train)
# Predicting the stock prices
predictions = model.predict(X_test)
predictions = pd.Series(predictions, index=X_test.index)  # Convert predictions to a pandas Series with proper index
# Calculate the RMSE
'''
The root-mean-square deviation (RMSD) or root-mean-square error (RMSE)
is one of frequently used measures of the differences between true or predicted values on the one hand and observed values or an estimator on the other.
'''
rmse = np.sqrt(mean_squared_error(y_test, predictions))

print("Root Mean Squared Error:", rmse)
# Create a DataFrame for plotting to handle indices seamlessly
results = pd.DataFrame({
  'Actual': y_test,
  'Predicted': predictions
})
# Plotting using Seaborn and Matplotlib
plt.figure(figsize=(10, 6))
sns.lineplot(data=results)
plt.title(f'Actual vs Predicted Stock Prices for {ticker_symbol}')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend(title='Legend', labels=['Actual', 'Predicted'])
plt.show()
```

```
if __name__=='__main__':
  predict_next_closing_price_symbol()
```

Angular Frontend Web Application Setup

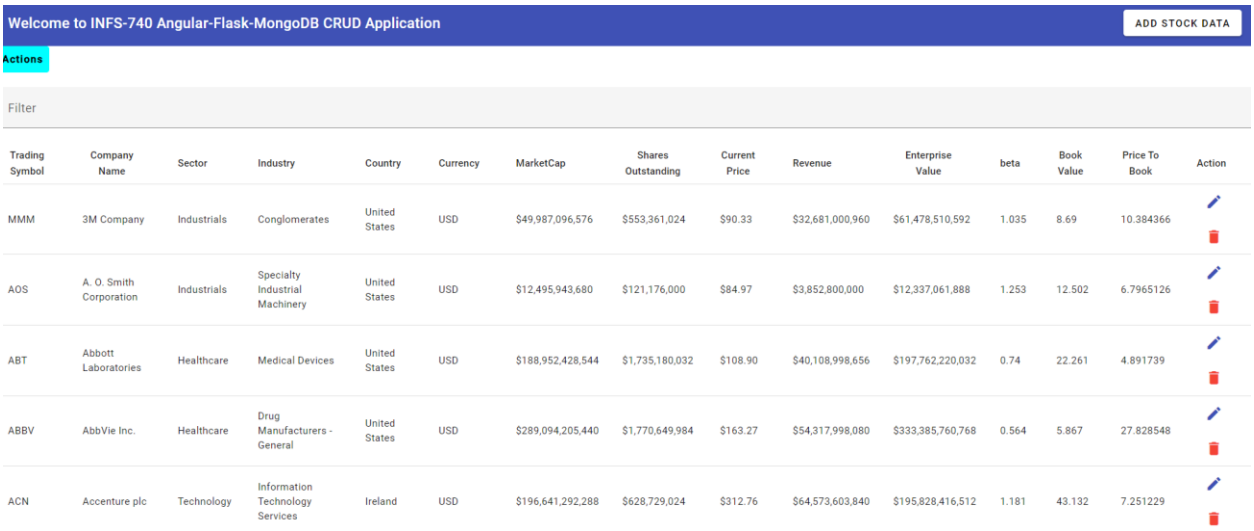
After successfully setting up the MongoDB database and starting and verifying the REST API application by following the steps provided in their respective sections, proceed with these steps to setup the Angular web application:

- Navigate to the web folder:
  - cd project-app/web
- Add Angular Material to your project:
  - ng add @angular/material
- Install the necessary dependencies from package.json:
  - npm install (it will install the dependencies from package. json)
- Start the Angular server:
  - ng serve
- After a short wait, you should see the application compile and start serving at localhost:4200.

```
Application bundle generation complete. [14.219 seconds]

Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:4200/
```

- Verify the Application:
    - Access the application by visiting:
    - http://localhost:4200/

Home Page:



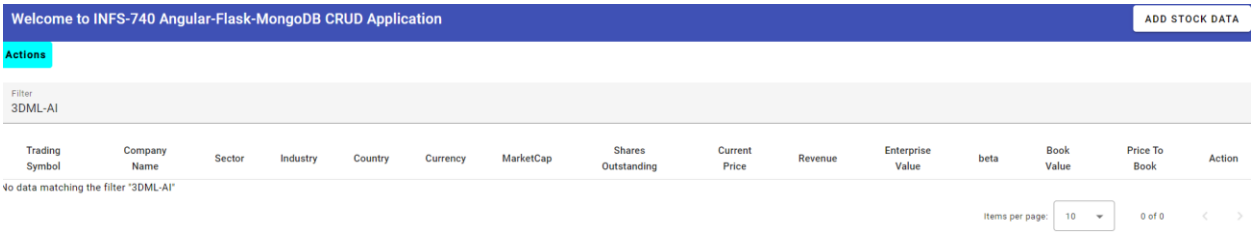
Trading Symbol	Company Name	Sector	Industry	Country	Currency	MarketCap	Shares Outstanding	Current Price	Revenue	Enterprise Value	beta	Book Value	Price To Book	Action
MMM	3M Company	Industrials	Conglomerates	United States	USD	$49,987,096,576	$553,361,024	$90.33	$32,681,000,960	$61,478,510,592	1.035	8.69	10.384366	
AOS	A. O. Smith Corporation	Industrials	Specialty Industrial Machinery	United States	USD	$12,495,943,680	$121,176,000	$84.97	$3,852,800,000	$12,337,061,888	1.253	12.502	6.7965126	
ABT	Abbott Laboratories	Healthcare	Medical Devices	United States	USD	$188,952,428,544	$1,735,180,032	$108.90	$40,108,998,656	$197,762,220,032	0.74	22.261	4.891739	
ABBV	AbbVie Inc.	Healthcare	Drug Manufacturers - General	United States	USD	$289,094,205,440	$1,770,649,984	$163.27	$54,317,998,080	$333,385,760,768	0.564	5.867	27.828548	
ACN	Accenture plc	Technology	Information Technology Services	Ireland	USD	$196,641,292,288	$628,729,024	$312.76	$64,573,603,840	$195,828,416,512	1.181	43.132	7.251229	

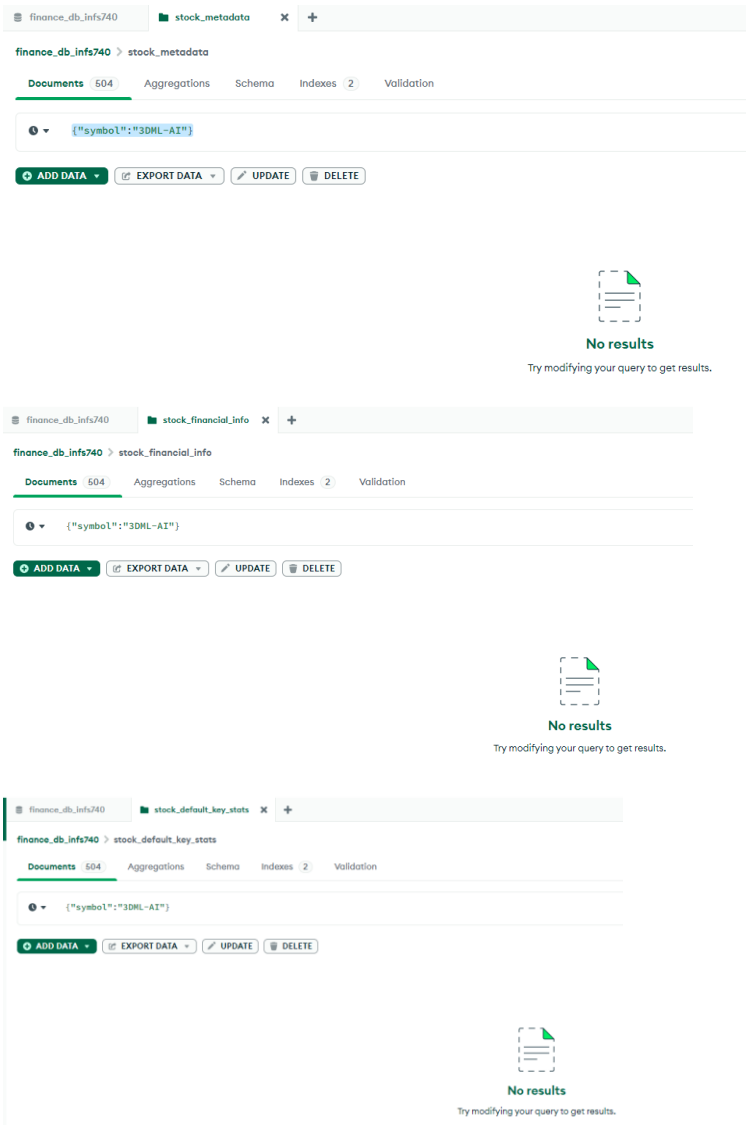
The web application's homepage offers sorting, filtering, and pagination features to enhance analysis. Additionally, Angular Material design ensures responsiveness for a seamless user experience across devices.

Add, Update, Delete from Web User Interface

Let's use the same symbol that does not currently exist to do full cycle testing of Add, Update, and Delete operations.

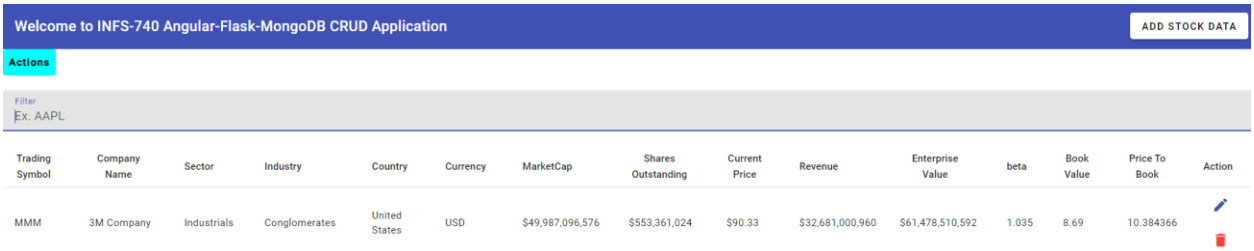
Verify first symbol **3DML-AI** is not present in the UI filter search and across all three MongoDB collections that serve as the data source for the frontend home page display:

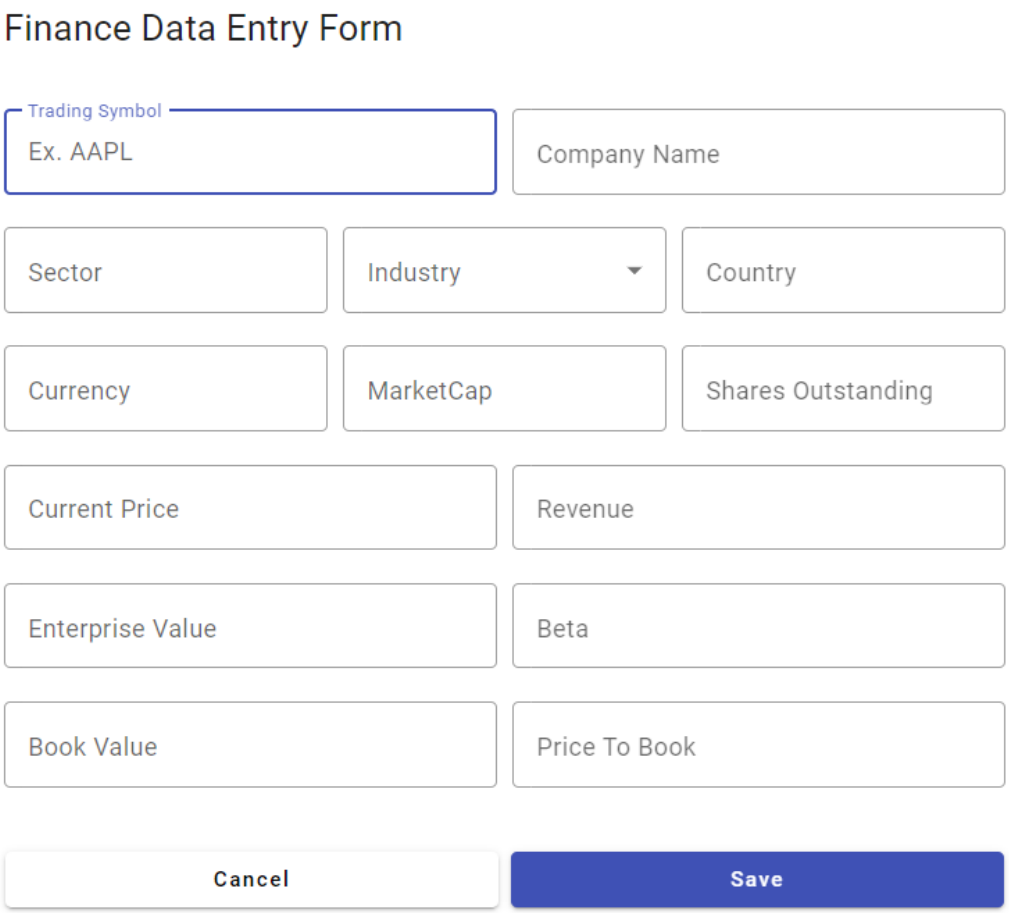




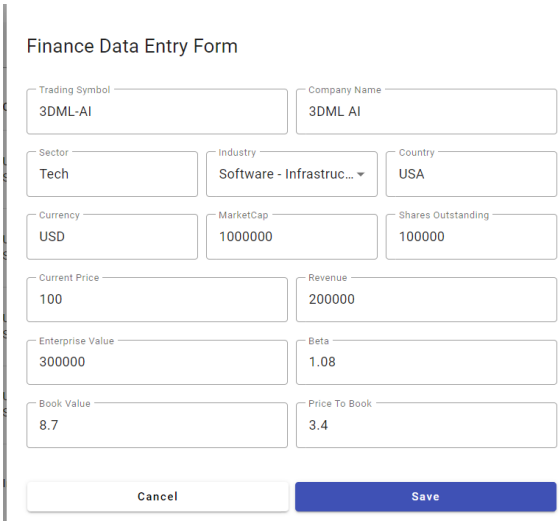
Now let's add the symbol **3DML-AI** from the Web User Interface.

- Click on **ADD STOCK DATA** on the right side of the tool bar. The application will present an empty stock entry data form.
- As this is a new entry form, the submit button will display **Save** as text.

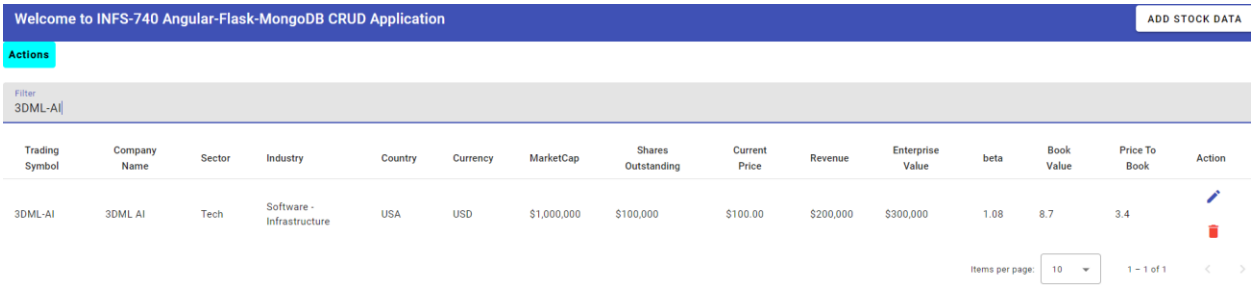




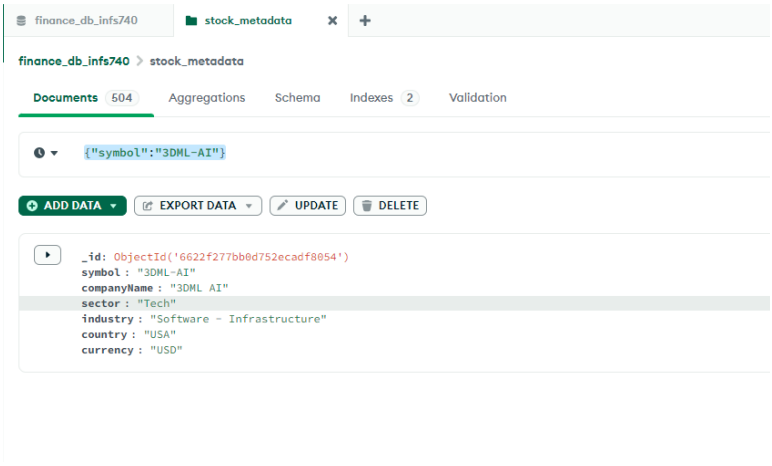
- Go ahead and enter data for symbol **3DML-AI** and click on **Save**.
- **Cancel** Operation - If you decide to cancel the operation, click on the **Cancel** button. This action will close the entry form and return the user to the home page.

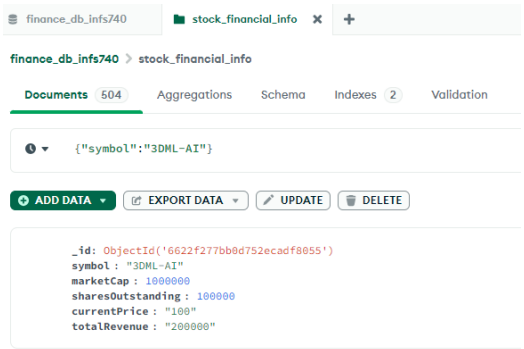


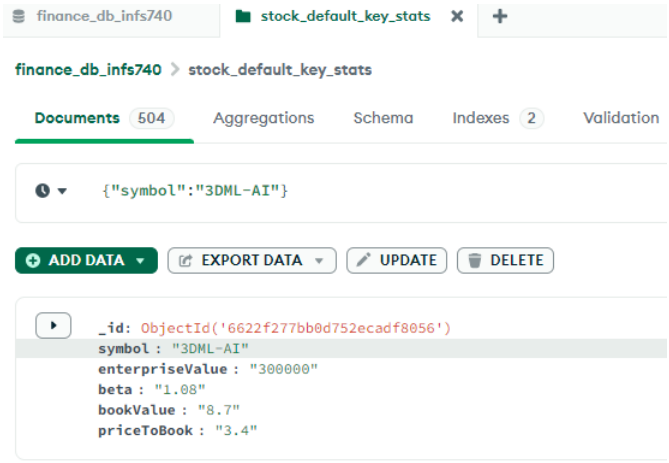
- Now, let's verify that the symbol **3DML-AI** is available in the UI and in all three collections in the MongoDB, which are the sources of the home page information.
- Enter 3DML-AI in the Filter on the home page and verify:



- Verify from the backend:





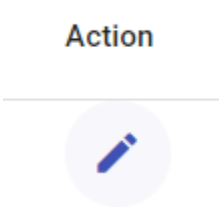


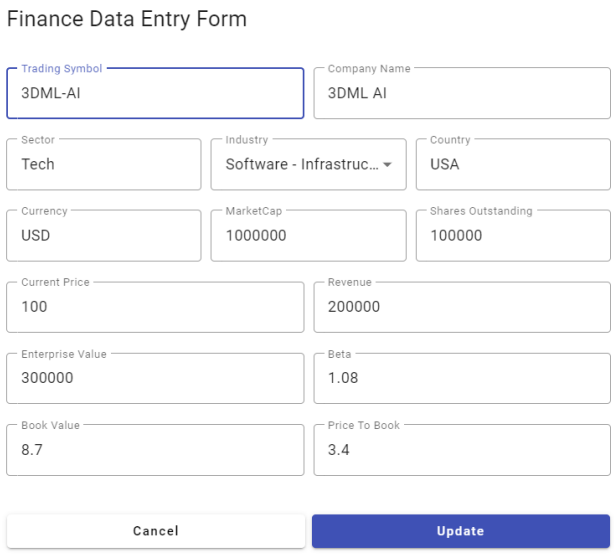
Data is entered to all respective collections.

Now, we will use the same symbol **3DML-AI** for our Update and Delete operations.

Filter the result for **3DML-AI** by entering the text in Filter field. Once the record is displayed, click on the pencil icon to edit the information for the trading symbol. The application will present the form to edit the information.

This time, the submit button text will show **Update** as we are editing the existing record.





Let's go ahead and click on the "**Update**" to edit the information belonging to all source collections so that we can verify data is simultaneously updated to all collections.

We changed the Company Name, Currency, MarketCap and Book Value fields. Let's go ahead and click on **Update** and verify updated information.

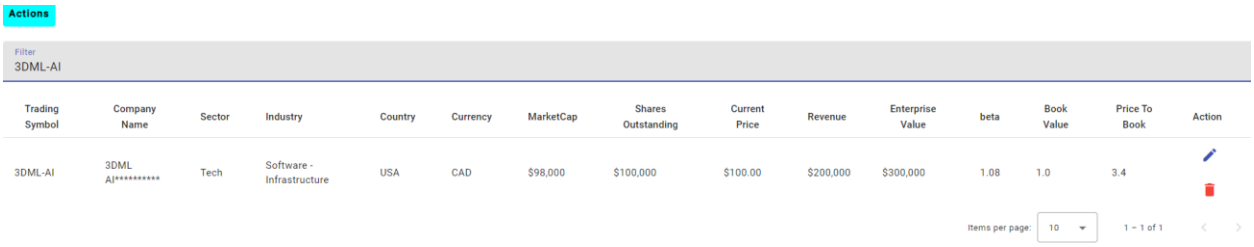
Cancel Operation - If you decide to cancel the operation, you can click on the **Cancel** button. This action will close the entry form and return the user to the home page, leaving the original data unchanged.

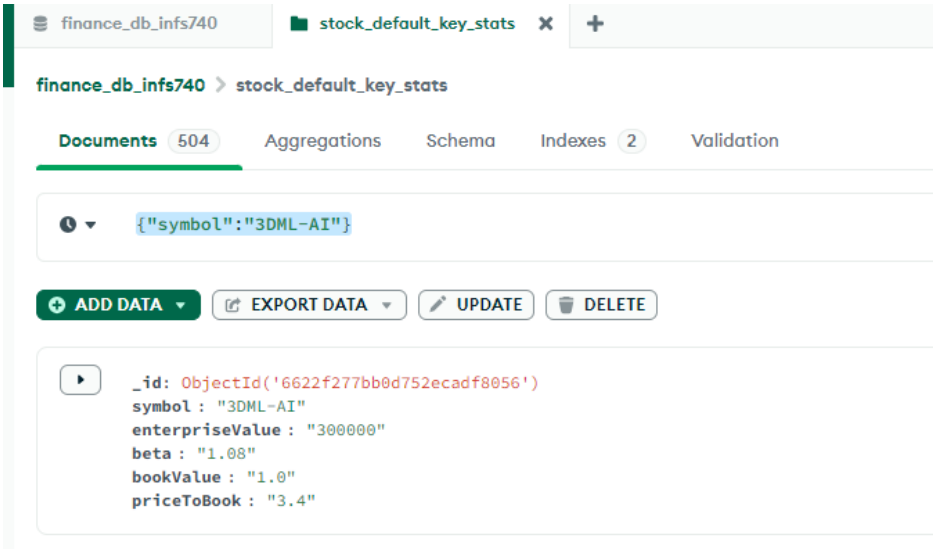
Finance Data Entry Form

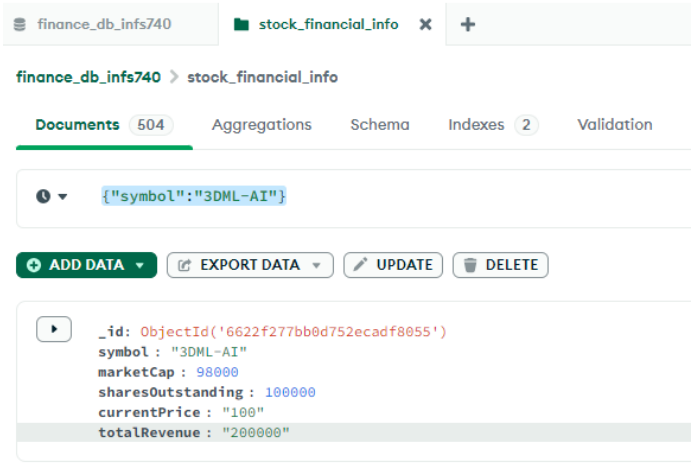
Trading Symbol: 3DML-AI
Company Name: 3DML AI**********
Sector: Tech
Industry: Software - Infrastruc...
Country: USA
Currency: CAD
MarketCap: 98000
Shares Outstanding: 100000
Current Price: 100
Revenue: 200000
Enterprise Value: 300000
Beta: 1.08
Book Value: 1.0
Price To Book: 3.4

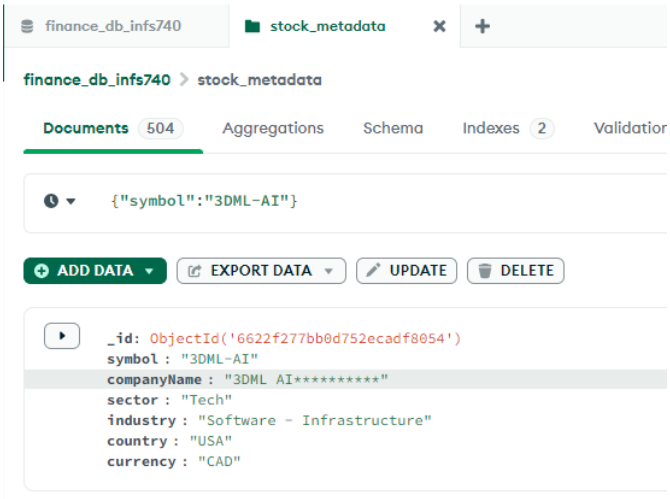
Cancel | Update

To ensure our edits have been applied correctly, let's perform verifications by filtering for the symbol. Check the edit screen, UI record, and MongoDB database to confirm that the updates have been successfully propagated across all relevant collections.









The data is updated to UI and all three collections.

Now, let’s perform Delete operation on **3DML-AI** and verify the results. The information for symbol **3DML-AI** should be deleted from all places.

Locate the Record: Filter and find the record for the symbol "3DML-AI" in the system.

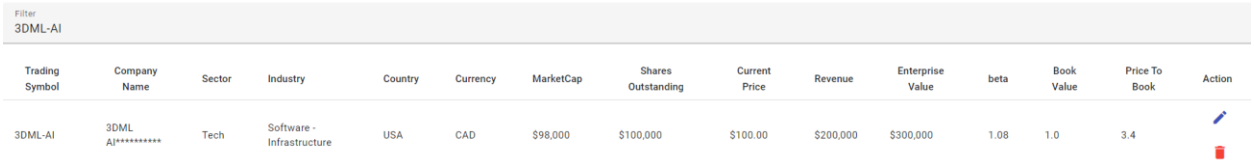
Perform the Delete Operation: Click on the trashcan icon corresponding to the row of "3DML-AI". This should trigger the deletion process.

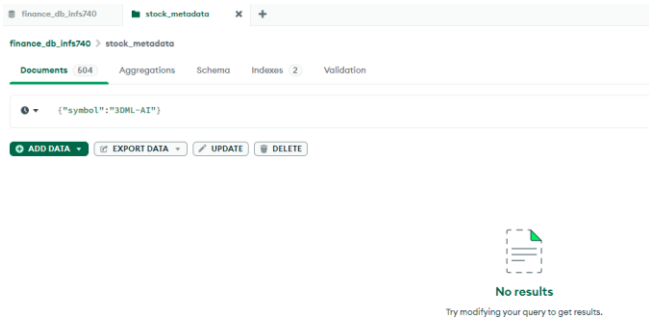
Observe the Success Message: Immediately after deletion, watch for the success message to appear, confirming the operation was successful.

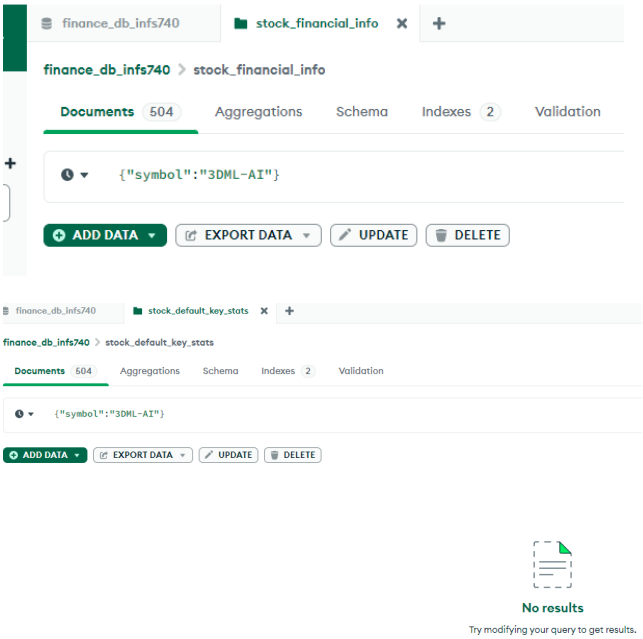
Message Disappearance: Note that the success message should automatically fade away or disappear after 2 seconds, ensuring that it doesn't obstruct the user interface or require any user interaction to close.

Verify the Results: After deleting, verify that the information for "3DML-AI" has been completely removed from all places in the system. This includes checking the UI, and backend MongoDB collections.

The following screenshots show that the symbol **3DML-AI** has been successfully deleted from all relevant parts of the system, including both the UI and the backend database.



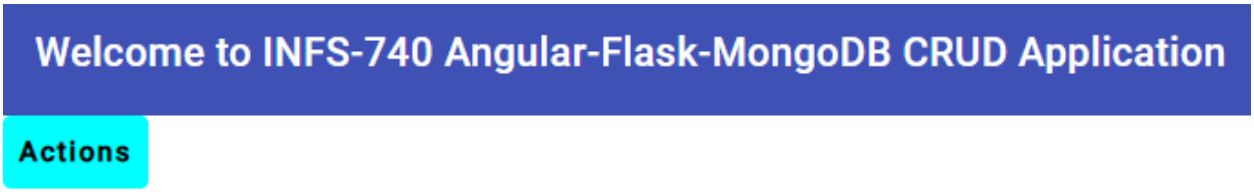


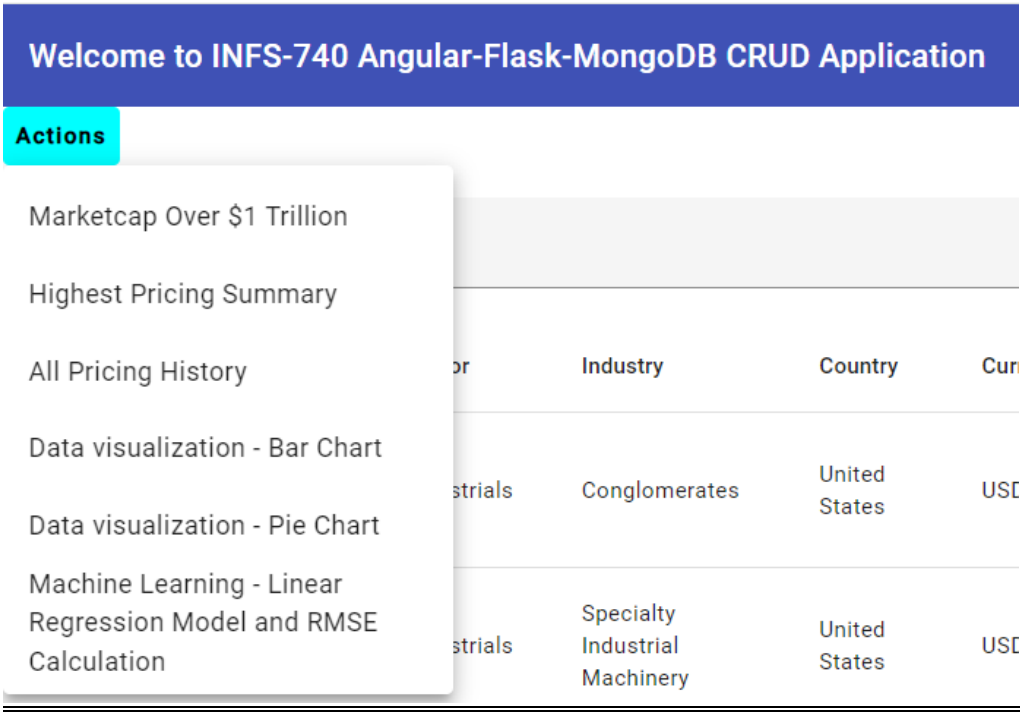


Please note that after every successful **Add, Update, or Delete** operation, the application automatically displays a message confirming the action. This message is designed to disappear after 2 seconds without requiring any user interaction. This is a useful feature for enhancing user experience by providing immediate feedback while keeping the interface clean and uncluttered.

Actions from UI:

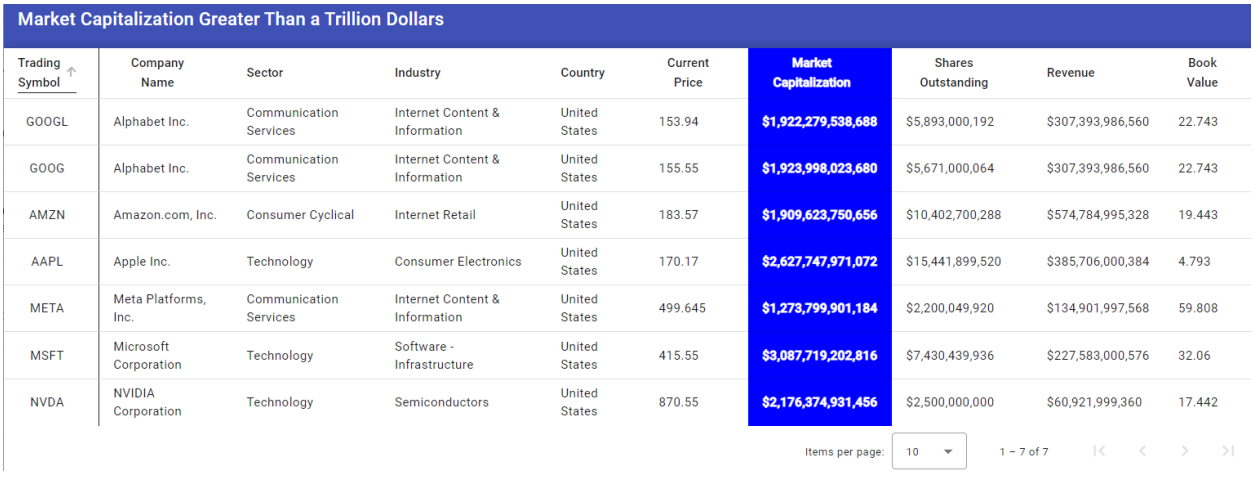
Click on the **Actions** button to see a menu that lists different actions:





Market Capitalization Over $1 Trillion Dollars

Click on the **'MarketCap Over $1 Trillion**' action item. The application will then query the 'stock_financial_info' collection, filter the results to include only those companies with a market capitalization over $1 trillion, and join this data with information from the 'stock_metadata' and 'stock_default_key_stats' collections to populate the page.



Highest Pricing Summary

Select the **'Highest Pricing Summary'** menu item. The application will query the 'stock_history_data' collection, aggregate the data, perform grouping by symbol, and get the maximum price for each trading symbol over the past year. It will then join this data with

'stock_metadata' and 'stock_financial_info' collections to compile and display the relevant information on the page.

Summary - Highest Price for GOOGL, GOOG, AMZN, AAPL, META, MSFT, NVDA, TSLA Since 2023-01-01

Filter

Trading Symbol	Company Name	Highest Opening Price	Revenue
AAPL	Apple Inc.	**$199.37**	$385,706,000,384
AMZN	Amazon.com, Inc.	**$189.77**	$574,784,995,328
GOOG	Alphabet Inc.	**$161.7**	$307,393,986,560
GOOGL	Alphabet Inc.	**$160.22**	$307,393,986,560
META	Meta Platforms, Inc.	**$531.49**	$134,901,997,568
MSFT	Microsoft Corporation	**$430.82**	$227,583,000,576
NVDA	NVIDIA Corporation	**$974**	$60,921,999,360
TSLA	Tesla, Inc.	**$299.29**	$96,772,997,120

Items per page: 10 1 – 8 of 8

All Pricing History

Select the '**All Pricing History'** menu item, the application will bring data from four collections stock_history_data, stock_metadata, stock_financial_info, and stock_default_key_stats.

This feature displays the complete pricing history for a selected symbol, sorted in descending order by the High price. For example, if you filter for 'AAPL' on this page, you will see that Apple has 323 records. The highest price listed will match the numbers from the **'Highest Pricing Summary**' page.

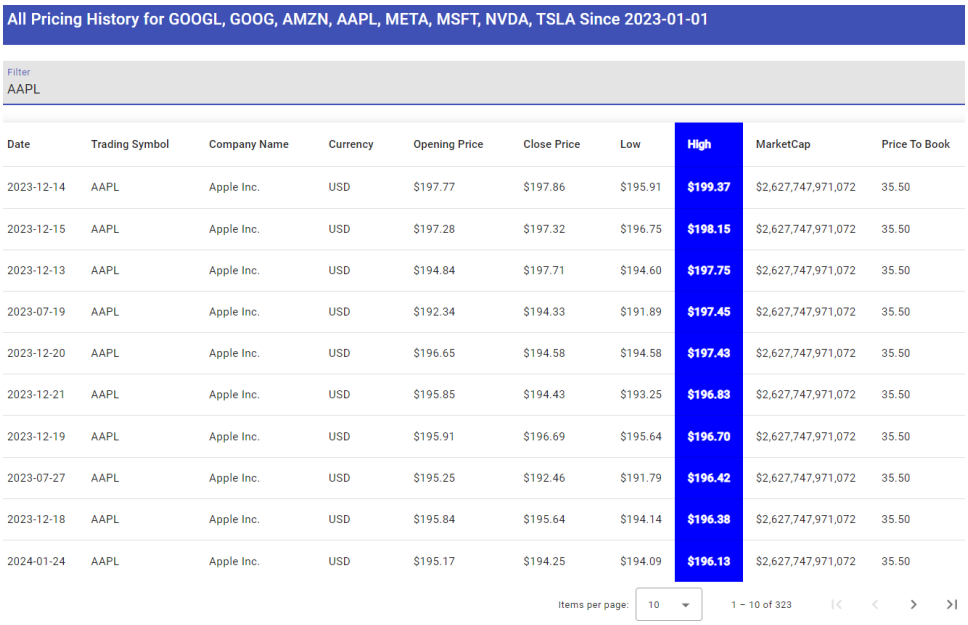
All Pricing History for GOOGL, GOOG, AMZN, AAPL, META, MSFT, NVDA, TSLA Since 2023-01-01

Filter

Date	Trading Symbol	Company Name	Currency	Opening Price	Close Price	Low	High	MarketCap	Price To Book
2023-12-14	AAPL	Apple Inc.	USD	$197.77	$197.86	$195.91	**$199.37**	$2,627,747,971,072	35.50
2023-12-15	AAPL	Apple Inc.	USD	$197.28	$197.32	$196.75	**$198.15**	$2,627,747,971,072	35.50
2023-12-13	AAPL	Apple Inc.	USD	$194.84	$197.71	$194.60	**$197.75**	$2,627,747,971,072	35.50
2023-07-19	AAPL	Apple Inc.	USD	$192.34	$194.33	$191.89	**$197.45**	$2,627,747,971,072	35.50
2023-12-20	AAPL	Apple Inc.	USD	$196.65	$194.58	$194.58	**$197.43**	$2,627,747,971,072	35.50
2023-12-21	AAPL	Apple Inc.	USD	$195.85	$194.43	$193.25	**$196.83**	$2,627,747,971,072	35.50
2023-12-19	AAPL	Apple Inc.	USD	$195.91	$196.69	$195.64	**$196.70**	$2,627,747,971,072	35.50
2023-07-27	AAPL	Apple Inc.	USD	$195.25	$192.46	$191.79	**$196.42**	$2,627,747,971,072	35.50
2023-12-18	AAPL	Apple Inc.	USD	$195.84	$195.64	$194.14	**$196.38**	$2,627,747,971,072	35.50
2024-01-24	AAPL	Apple Inc.	USD	$195.17	$194.25	$194.09	**$196.13**	$2,627,747,971,072	35.50

Items per page: 10 1 – 10 of 2584

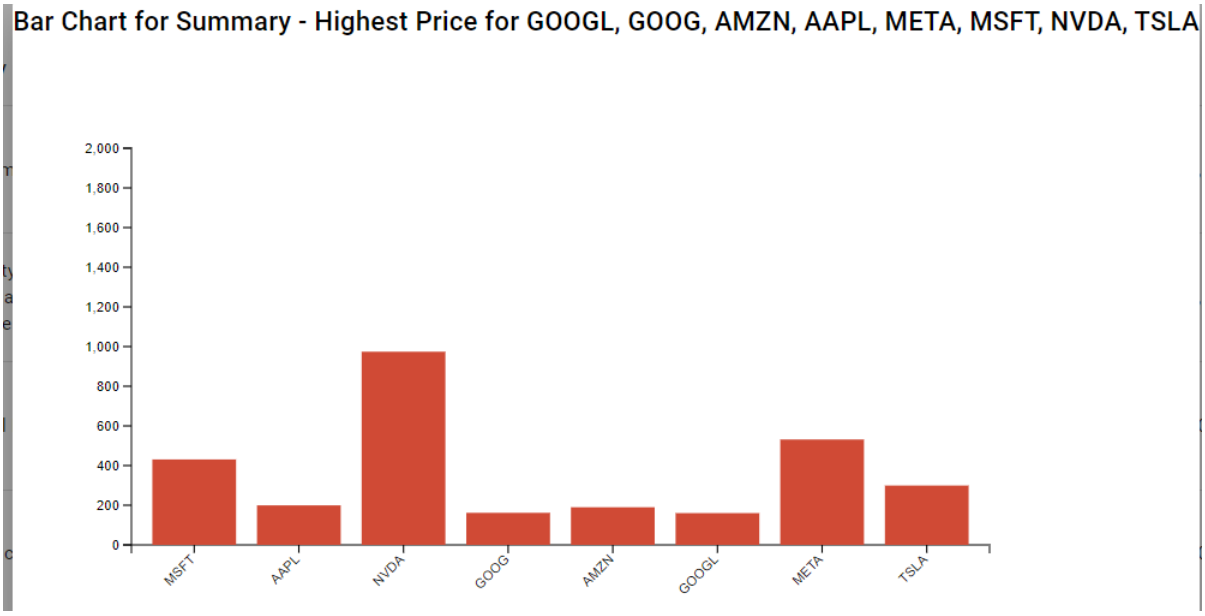
Filter for AAPL on this screen, you will see 323 records for pricing history.



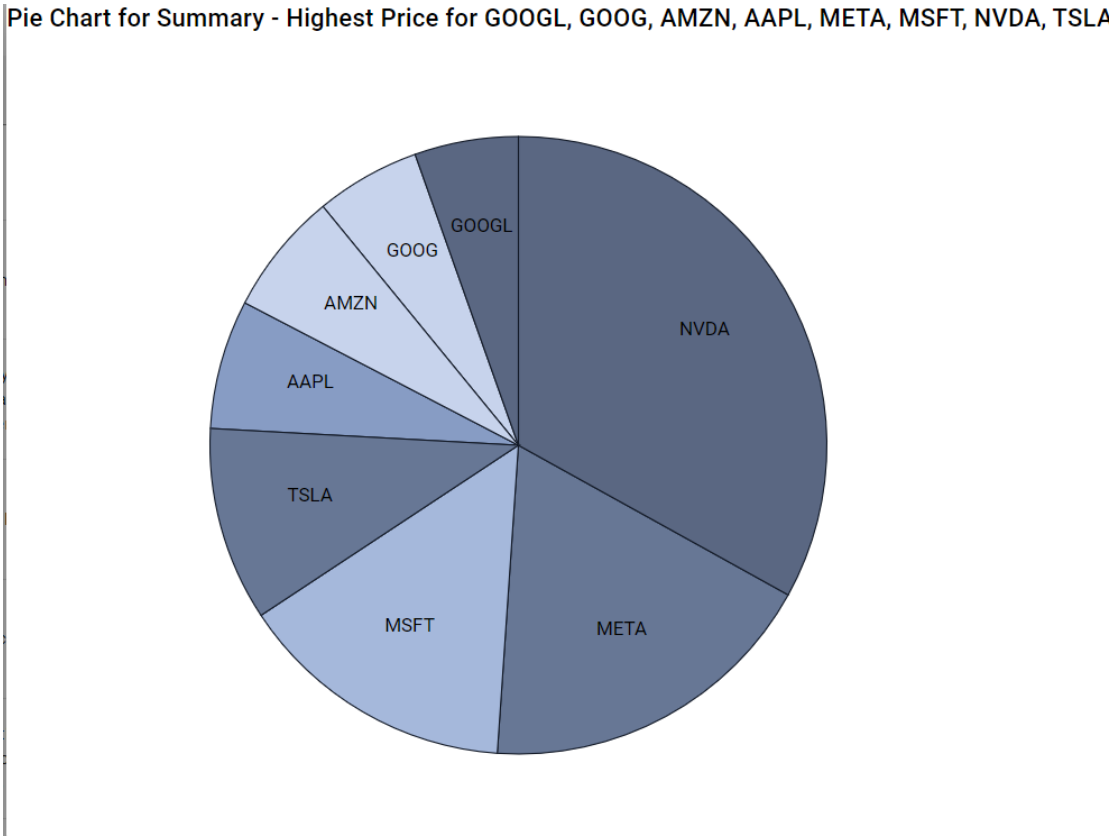
Data Visualization

Data Visualization is performed on the Highest Pricing Data.

Click on **Data Visualization - Bar Chart** to view a comparison of the highest pricing standings of various companies. This visualization helps illustrate the highest prices reached by each company, providing a clear and comparative insight into their financial performance.



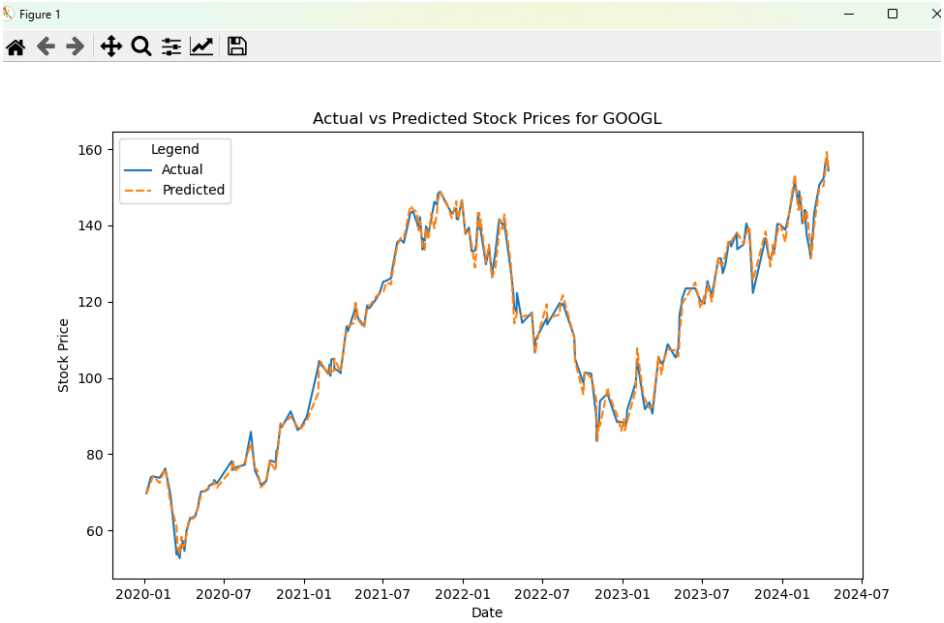
Click on the **Data Visualization – Pie Chart** to view the highest pricing standings of various companies in a different visual format. This pie chart offers a marketing representation of each company's highest price relative to others, providing an intuitive and comparative insight into their financial performance.



Machine Learning – Linear Regression Model and RMSE Calculation

Execute the python application file **infs_740_ml_linear_regression.py** that is included with the submission to see the Root Mean Squared Error calculation, Linear Regression model training, predictions and plotting using matplotlib and seaborn libraries.

```
[**********100%%**********]  1 of 1 completed
Root Mean Squared Error: 2.1229981447781348

Process finished with exit code 0
```

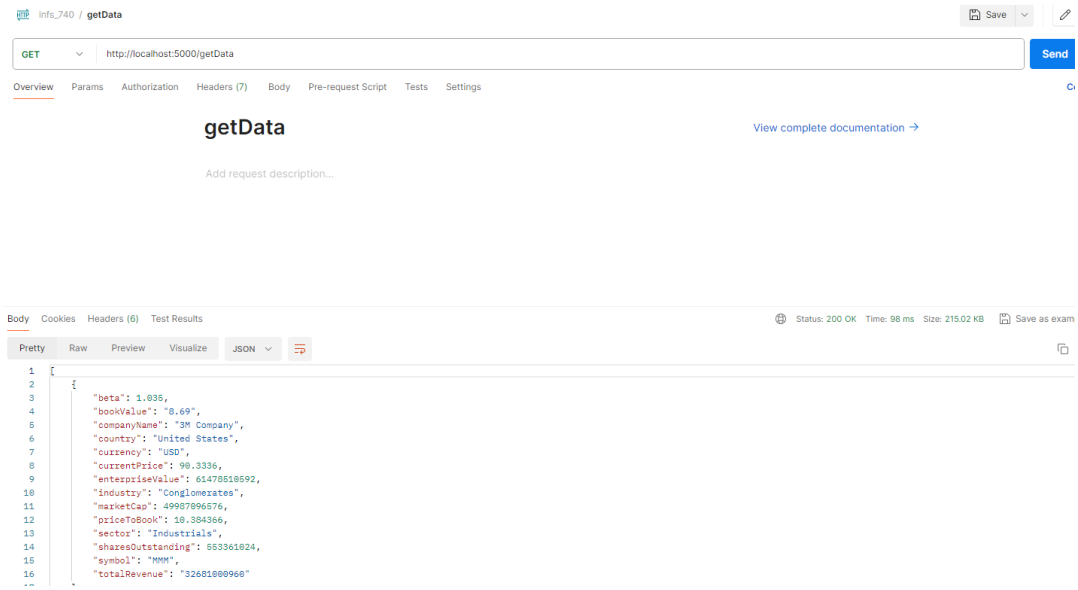


Testing of APIs - CRUD Operations- get, post, put, delete via postman.

Postman API testing tool is used to verify the functionality of APIs for CRUD (Create, Read, Update, Delete) and other UI operations. The Flask REST API is deployed at http://127.0.0.1:5000/

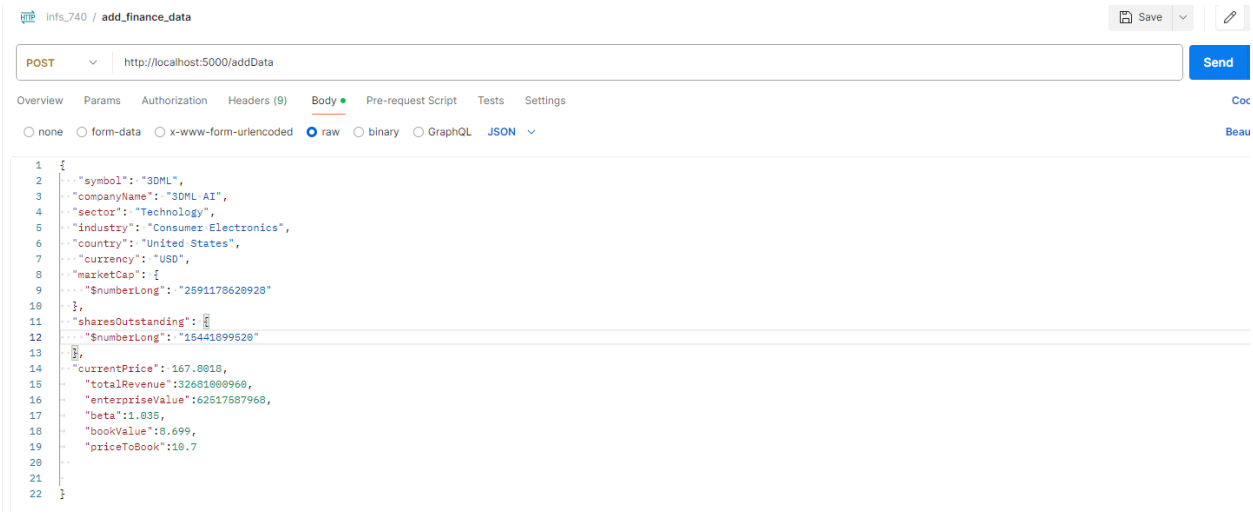
Get Data

http://localhost:5000/getData



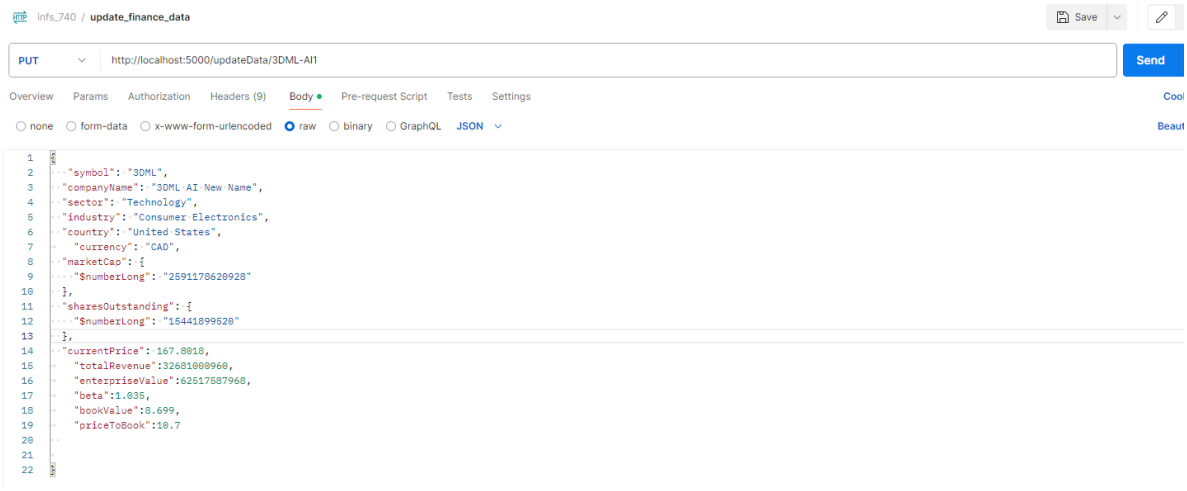
Add Data

http://localhost:5000/addData



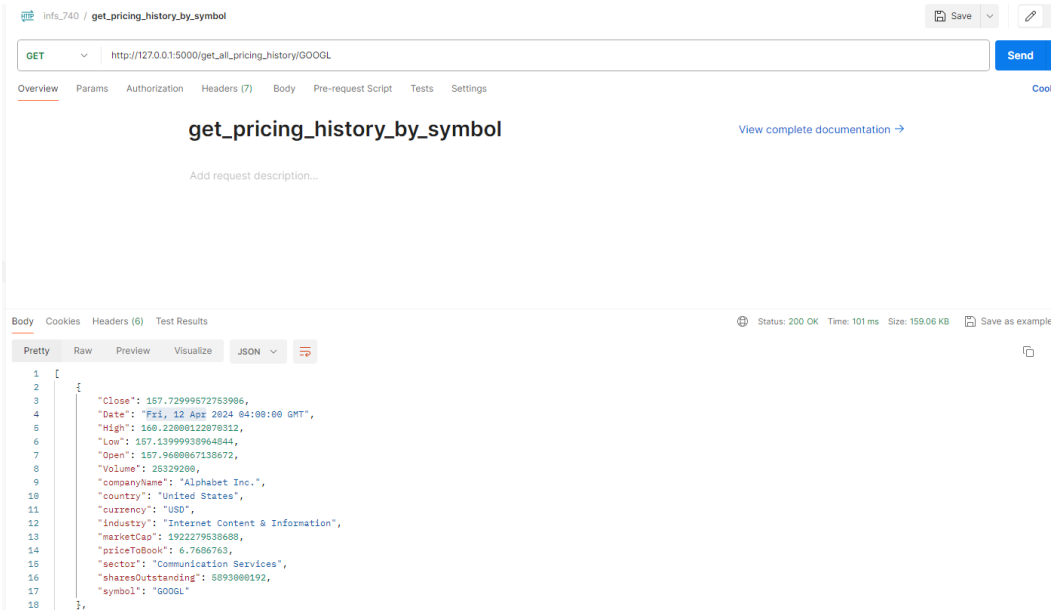
Update Data

http://localhost:5000/updateData/3DML



Get Pricing History for a given trading symbol

http://127.0.0.1:5000/get_all_pricing_history/GOOGL



Delete Data for a given trading symbol

http://localhost:5000/deleteData/3DML where 3DML is a trading symbol

Integration of API with Frontend Angular Application

The Angular frontend application interacts with the backend by calling REST API methods (GET, POST, PUT, DELETE) mapped to specific URLs. Angular controllers utilize the global **StockService** service to perform CRUD operations—getting, posting, updating, and deleting data from/to the MongoDB database via RESTful Flask APIs. Below are several screenshots that illustrate these interactions within the Angular front-end application.

StockService

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class StockService {
  constructor(private _http: HttpClient) {}

  addFinanceData(data: any): Observable<any> {
    return this._http.post('http://localhost:5000/addData', data);
  }

  updateFinanceData(symbol: string, data: any): Observable<any> {
    return this._http.put(`http://localhost:5000/updateData/${symbol}`, data);
  }

  getFinanceDataList(): Observable<any> {
    return this._http.get('http://localhost:5000/getData');
  }

  deleteFinanceData(symbol: string): Observable<any> {
    return this._http.delete(`http://localhost:5000/deleteData/${symbol}`);
  }

  getFinanceOver1T(): Observable<any> {
    return this._http.get('http://127.0.0.1:5000/get_marketCap_over_1T');
  }
  getHighestOpenPricingHistory(): Observable<any> {
    return this._http.get(
      'http://127.0.0.1:5000/get_pricing_history_highest_open'
    );
  }
  getAllPricingHistory(): Observable<any> {
    return this._http.get('http://127.0.0.1:5000/get_all_pricing_history');
  }
  getDataToVisualize(): Observable<any> {
    return this._http.get('http://127.0.0.1:5000/get_data_to_visualize');
  }
```

Components:

```ts
import { Component, OnInit, ViewChild } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { StockAddEditComponent } from './stock-add-edit/stock-add-edit.component';
import { StockService } from './services/stock.service';
import { MatPaginator } from '@angular/material/paginator';
import { MatSort } from '@angular/material/sort';
import { MatTableDataSource } from '@angular/material/table';
import { CoreService } from './core/core.service';
import { StockInfoComponent } from './stock-info/stock-info.component';
import { StockPriceComponent } from './stock-price/stock-price.component';
import { StockPriceAllComponent } from './stock-price-all/stock-price-all.component';
import { VisualComponent } from './visual/visual.component';
import { BarComponent } from './bar/bar.component';
import { PieComponent } from './pie/pie.component';

You, 4 days ago | 1 author (You)
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss',
})
export class AppComponent implements OnInit {      You, 4 days ago • wip
  title = 'infs740_project_app';
```

```
displayedColumns: string[] = [
  'symbol',
  'companyName',
  'sector',
  'industry',
  'country',
  'currency',
  'marketCap',
  'sharesOutstanding',
  'currentPrice',
  'totalRevenue',
  'enterpriseValue',
  'beta',
  'bookValue',
  'priceToBook',

  'action',
];
dataSource!: MatTableDataSource<any>;

@ViewChild(MatPaginator) paginator!: MatPaginator;
@ViewChild(MatSort) sort!: MatSort;

constructor(
  private _dialog: MatDialog,
  private _stockService: StockService,
  private _coreService: CoreService
) {}

ngOnInit(): void {
  this.getFinanceData();
}
```

```
openAddEditFinanceForm() {
  const dialogRef = this._dialog.open(StockAddEditComponent);
  dialogRef.afterClosed().subscribe({
    next: (val) => {
      if (val) {
        this.getFinanceData();
      }
    },
  });
}

getFinanceData() {
  this._stockService.getFinanceDataList().subscribe({
    next: (res) => {
      // console.log(res);
      this.dataSource = new MatTableDataSource(res);
      this.dataSource.sort = this.sort;
      this.dataSource.paginator = this.paginator;
    },
    error: console.log,
  });
}
getFinanceOver1T() {
  this._stockService.getFinanceOver1T().subscribe({
    next: (res) => {
      // console.log(res);
      this._dialog.open(StockInfoComponent);
    },
    error: console.log,
  });
}

getAllPricingHistory() {
  this._stockService.getAllPricingHistory().subscribe({
    next: (res) => {
      this._dialog.open(StockPriceAllComponent);
    },
    error: console.log,
  });
}

applyFilter(event: Event) {
  const filterValue = (event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.trim().toLowerCase();

  if (this.dataSource.paginator) {
    this.dataSource.paginator.firstPage();
  }
}
```

```
deleteFinanceData(symbol: string) {
  this._stockService.deleteFinanceData(symbol).subscribe({
    next: (res) => {
      this._coreService.openSnackBar('Finance Data Deleted!', 'done');
      this.getFinanceData();
    },
    error: console.log,
  });
}

openEditForm(data: any) {
  const dialogRef = this._dialog.open(StockAddEditComponent, {
    data,
  });

  dialogRef.afterClosed().subscribe({
    next: (val) => {
      if (val) {
        this.getFinanceData();
      }
    },
  });
}
```

```
  displayBarChart() {
    // this._dialog.open(VisualComponent);
    this._dialog.open(BarComponent);
  }
  displayPieChart() {
    this._dialog.open(PieComponent);
  }
}
```

```ts
import { AfterViewInit, Component, OnInit, ViewChild } from '@angular/core';
import { MatSort, Sort } from '@angular/material/sort';
import { MatTableDataSource } from '@angular/material/table';
import { StockService } from '../services/stock.service';
import { LiveAnnouncer } from '@angular/cdk/a11y';
import { MatPaginator } from '@angular/material/paginator';

You, 3 days ago | 1 author (You)
@Component({
  selector: 'app-stock-info',
  templateUrl: './stock-info.component.html',
  styleUrl: './stock-info.component.scss',
})
export class StockInfoComponent implements OnInit, AfterViewInit {
  ngOnInit(): void {
    this.getData();
  }

  displayedColumns: string[] = [
    'symbol',
    'companyName',
    'sector',
    'industry',
    'country',
    'currentPrice',
    'marketCap',
    'sharesOutstanding',
    'totalRevenue',
    'bookValue',
  ];

  dataSource!: MatTableDataSource<any>;
  @ViewChild(MatSort) sort!: MatSort;
  @ViewChild(MatPaginator) paginator!: MatPaginator;

  constructor(
    private _stockService: StockService,
    private _liveAnnouncer: LiveAnnouncer
  ) {}
```

```
announceSortChange(sortState: Sort) {
  if (sortState.direction) {      You, 4 days ago • wip
    this._liveAnnouncer.announce(`Sorted ${sortState.direction}ending`);
  } else {
    this._liveAnnouncer.announce('Sorting cleared');
  }
}

ngAfterViewInit(): void {
  this.dataSource.sort = this.sort;
  this.dataSource.paginator = this.paginator;
}
getData() {
  this._stockService.getFinanceOver1T().subscribe({
    next: (res) => {
      console.log(res);
      this.dataSource = new MatTableDataSource(res);
      this.dataSource.sort = this.sort;
      this.dataSource.paginator = this.paginator;
    },
    error: console.log,
  });
}
}
```

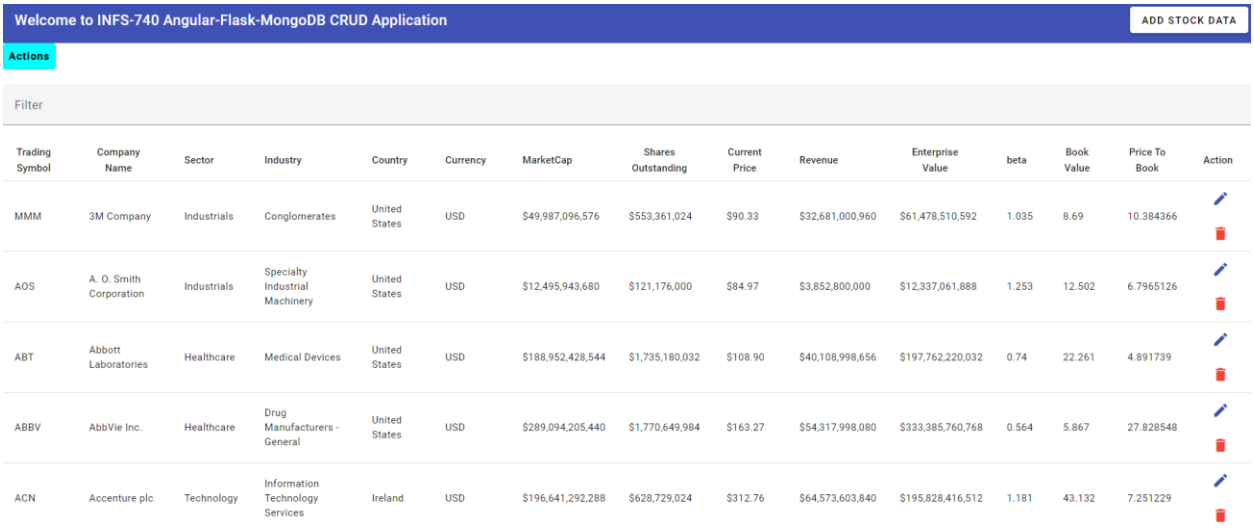
Running the project

Please follow the steps below in the specified order to ensure all components of the project are operational:

1. **MongoDB Setup**: Ensure the Mongo database is set up using the archive included with the submission, **finance_db_infs740**. Refer to the MongoDB database setup instructions to restore the database with this archive.
2. **API Verification**: Ensure the API is actively running and serving endpoints at http://localhost:5000/.
3. After successfully starting and verifying MongoDB database and API (steps #1 and #2), perform the steps below.
4. **Web Project Application Setup**:
   - Navigate to the 'web' folder within the **project-app** directory.
   - Open a terminal or VS Code and run the following commands:
     - cd project-app/web — Changes the directory to the web folder.
     - npm install — Installs all required dependencies first.
     - ng serve — Compiles the application and provides a URL to access it.

```
$ ng serve
- Building...

Browser bundles
Initial chunk files     | Names               |  Raw size
styles.css              | styles              |  93.20 kB |
main.js                 | main                |  89.99 kB |
polyfills.js            | polyfills           |  83.60 kB |

                        | Initial total       | 266.79 kB


Server bundles
Initial chunk files     | Names               |  Raw size
main.server.mjs         | main.server         |   2.11 MB |
chunk-YH62REDW.mjs      | -                   |   1.70 MB |
polyfills.server.mjs    | polyfills.server    | 555.05 kB |
chunk-VPSODEBW.mjs      | -                   |   2.51 kB |
render-utils.server.mjs | render-utils.server | 423 bytes |

Lazy chunk files        | Names               |  Raw size
chunk-OTT6LQ5K.mjs      | xhr2                |  39.10 kB |

Application bundle generation complete. [14.219 seconds]

Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:4200/
```

- **Access the Application**: After successful compilation, follow the URL provided to access the home page of the application. This page will display financial information. http://localhost:4200/



Validation

- **CRUD Operations**: Perform Add, Update, and Delete operations within the web application to ensure these functionalities are working correctly.

- **Menu Items**: Click on the various menu items under the **'Actions'** menu to verify that different queries are executed properly, and that the application is serving respective pages as expected.

Project Overview Video Recording

Play the video **video1133150885.mp4** included in the submission folder to view a demonstration of all components of the INFS-740 web application. The video will start after a few seconds and provide a comprehensive explanation of how each part of the application functions.

References

- INFS-740 Lectures and Support Material
- Angular documentation: Angular Official Site
- Angular Material Getting Started Guide: Material Design for Angular
- MongoDB Documentation: MongoDB Official Documentation