**Stock Price Monitor**

This is a small system used to monitor the stock price using Yahoo Finance API. The system contains 3 parts.

1. **MySQL Database**

I am using MySQL Database for storing all the necessary data related to this system. It has one database named **stock\_db** which contains single table named **stockdata** to store all the stock related data. **stockdata** table consists of few of the important values of each stock. The schema of the table is as follows:

`stockdata` (

`symbol` varchar(255) NOT NULL,

`name` varchar(255) DEFAULT NULL,

`price` decimal(10,2) DEFAULT NULL,

`changeValue` decimal(6,2) DEFAULT NULL,

`changePercent` decimal(6,2) DEFAULT NULL,

`volume` bigint(20) DEFAULT NULL,

`prevClose` decimal(10,2) DEFAULT NULL,

`open` decimal(10,2) DEFAULT NULL,

`dayHigh` decimal(10,2) DEFAULT NULL,

`dayLow` decimal(10,2) DEFAULT NULL,

PRIMARY KEY (`symbol`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

1. **StockQuoteDump**

This is standalone java program to create the **stockdata** table and to initially insert stock details of few stocks. The design of this java application is as below:

JDBC Driver

StockDatDump

MySQL Database

I am using Yahoo Finance API to get the stock details based on the stock symbol. I have initially inserted the required stock details according to the schema for the below 10 stocks in the database table.

* Amazon (AMZN)
* Bank Of America (BAC)
* HP (HPQ)
* ICICI Bank (IBN)
* Citi Bank (C)
* Facebook (FB)
* Apple (AAPL)
* Groupon (GRPN)
* Microsoft (MSFT)
* Ebay (EBAY)

1. **StockMonitor**

This is the RESTful web application developed using Java Spring MVC and Hibernate. Spring MVC web framework build on notion of a DispatcherServlet, which is responsible for dispatching each request to appropriate controller (StockController in this project) and returns the response. Design of the system is as below:

StockController

(Controller)

Request

Response

Client

Dispatcher Servlet

(Front Controller)

StockDao

(Persistence layer)

StockService

(Service Layer)

MySQL Database

(stock\_db.stockdata)

Upon receiving a request from the client, the dispatcher Servlet of Spring handles the request and calls the respective controller based on the Request URL and Request Type. Here, I have only one controller named **stockController** to perform all the CRUD operation on stock data. So, based on the Request type and Parameters respective methods is called inside service layer. All the business logic is handled by **stockService**, which in turn calls the persistance layer taken care by the **StockDao** for the data related operations. I have one entity class named **StockData** mapped with my database table **stockdata** using hibernate.

The different features provided by this RESTful API are:

1. **List All Companies**

This is basically a GET request to provide the list of all the existing stocks in system along with their latest stock details.

1. **Add a Company**

This is a POST request which accept stock symbol from the request body and gets the stock details of that respective stock and insert in the database.

1. **Delete a Company**

This is http DELETE request which accepts stock symbol as a path parameter and deletes the stock associated with the symbol from the database.

1. **Company History**

This is HTTP GET request which accepts stock symbol as a path parameter and provides the last one year history details of the stock.

**Steps to Execute the project**

1. Download both the projects my GITHub repository <https://github.com/satishanumolu/StockPriceMonitor> to your local machine.
2. Open MySQL and create a database named **stock\_db**.
3. Now, open the **StockDataDump** standalone java project and update the mysql.properties file with the following details appropriate to your system.

**DB.url**=<url of your database>

**DB.username**=<database username>

**DB.password**=<database password>

1. Run the java program and it should create a table named stockdata in stock\_db database and insert 10 stock details.
2. Now, open the StockMonitor spring MVC application and also update the app.properties file accordingly.

**DB.url**=<url of your database>

**DB.username**=<database username>

**DB.password**=<database password>

1. Now, run the project using run on server option which will load the index.html page as follows.
2. You can add any stock by entering the stock symbol in the input text field and clicking the Add Stock button.
3. You can delete any system using the respective delete button in the table by just clicking it. It will delete the system
4. You can view the Line chart representation of the stock history details using the respective History button in the table by just clicking it. It will show last one year History details of the stock. It displays Date on the x-axis and stock Open Value on the y-axis.

**I can integrated swagger to my RESTful API. You can also test the API features by using the below URL**

**http:<servername>:<port>/StockMonitor/api/swagger-ui.html**

**It will provide the details of all the services as below.**

1. **Find All Stocks**

**Request URL :** http:<servername>:<port>/StockMonitor/api/stocks

**Request Type :** GET

**Sample Response :** [

{

"symbol": "AAPL",

"name": "Apple Inc.",

"price": 96.91,

"changeValue": 0.15,

"changePercent": 0.16,

"volume": 28991131,

"prevClose": 96.76,

"open": 97.2,

"dayHigh": 98.02,

"dayLow": 96.58

},

{

"symbol": "AMZN",

"name": "Amazon.com, Inc.",

"price": 555.23,

"changeValue": 0.08,

"changePercent": 0.01,

"volume": 4877039,

"prevClose": 555.15,

"open": 560.12,

"dayHigh": 562.5,

"dayLow": 553.17

},

……………

]

1. **Add Stock**

**Request URL :** http:<servername>:<port>/StockMonitor/api/stocks

**Request Type :** POST

**RequestBody :** Stock Symbol (ex. FB)

1. **Delete a Stock**

**Request URL :** http://<servername>:<port>/StockMonitor/api/stocks/{symbol}

**Request Type :** DELETE

**Path Parameter :** Stock Symbol (ex. FB)

1. **Get History of a Stock**

**Request URL :** http://<servername>:<port>/StockMonitor/api/stocks/{symbol}

**Request Type :** GET

**Path Parameter :** Stock Symbol (ex. FB)

**Sample Response :** [

{

"date": "2016-02-01",

"open": 7.1

},

{

"date": "2016-01-04",

"open": 9.17

},

{

"date": "2015-12-01",

"open": 7.96

},

{

"date": "2015-11-02",

"open": 7.29

},

{

"date": "2015-10-01",

"open": 6.19

}

…………..

]