Stock Monitor: Project Report

1. Introduction

This project implements a small stock monitoring system which allows a user to follow specific stock items and receive the associated live stock price updates. The user is able to add, remove the stock symbols she wants to follow; further, through the web interface, the user is able to get the newest prices of followed symbols, as well as keep track of the history of prices of any specific followed symbol.

2. System Architecture

The implemented system consists of four main components, shown in Figure 1:

- A MySQL database to store necessary information
- A stand-alone daemon process which is able to contact several different content providers (e.g., Google, Yahoo) to retrieve stock prices in fixed intervals (e.g., 5 minutes) and store information to MySQL
- A RESTful web service which provides CRUD operations on stock information. A user can add (follow), delete (unfollow) stock symbols, get updated prices of followed symbols and review the history of stock prices of a specific symbol
- A java-based web application that allows users to interact with the RESTful web service.

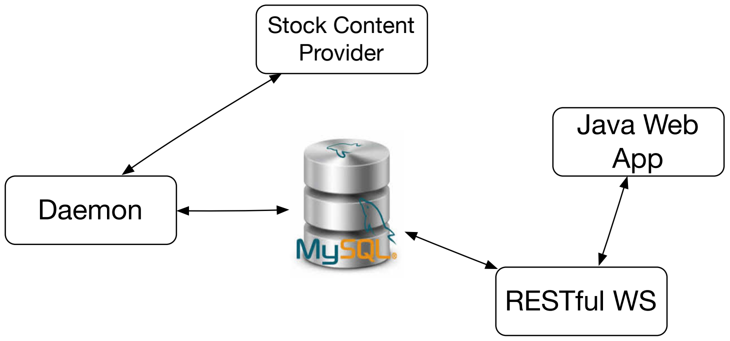


Figure 1 System Architecture

2.1. MySQL database

The SQL database is used to store important information about stock symbols and users in the system. The database consists of 4 tables; the primary keys are underlined.

User (<u>userid</u>, password, name): stores information about users

Field	Data type	Description	Note
userid	nvarchar	Stores user id	Primary key
Password	Nvarchar	User password	Should be hashed (not implemented)
Name	Nvarchar	User real name, or additional information (etc. email address)	Optional

Stock_symbol(symbol,name): stores information about stock symbols

Field	Data type	Description	Note
Symbol	nvarchar	Stock symbol/id, e.g. GOOG for Google	Primary key
Name	Nvarchar	Company name	Optional

Follow(userid, symbol): stores which stock items a user follows

Field	Data type	Description	Note
userid	nvarchar	Stores user id	Foreign key to table user(userid), on delete cascade
symbol	Nvarchar	Stores stock symbol the user follows	Foreign key to table stock_symbol(symbol), on delete cascade

Stock(symbol, last_query, price, lastTrade): information about stock prices

Field	Data type	Description	Note
Symbol	nvarchar	Stock symbol	Foreign key to table stock_symbol(symbol), on delete cascade
Last_query	Bigint	The time the system retrieved stock price for this symbol	Stored in epoch format for easily compared
Price	Decimal	Price at queried time	
lastTrade	String	Last trade time (UTC) of this symbol	Stored as a string, only for presentation.

2.2. Daemon process

Runs in background, uses TimerTask to retrieve stock price information from a content provider at fixed intervals for all stored stock symbols (of all users) in the system, and stores the retrieved information in MySQL.

2.2.1. Stock content provider

Our system is able to retrieve stock information from several different content providers.

- **Yahoo Finance**: Used as suggested in the assignment description. However, in many cases, Yahoo provides inconsistent and stale information, e.g., one query provides information about the price at 2:11 pm, the following query provides information about the price at 4:00 pm the day before. Thus, I do not recommend using Yahoo service
- **Google Finance**: Google finance service is said to be discontinued; however, it still provides up to date, accurate and consistent information about the stock prices and thus is used at the primary source

- **MIT** (**M**ark**it** on Demand): Is used as a secondary and failover provider; specifically, it is queried whenever a query sent to Google fails. The MIT service, however, only allows querying 1 specific item as a time, and has a rate limit on a second basis; and thus, should not be used as the primary source.

2.2.2. Connection Pool

The interval length (5 minutes, by default) is customizable through a configuration file. In order to prevent time consuming and expensive connection creation to MySQL if the interval is short (e.g., a few seconds), the daemon process use a connection pool to MySQL provided by Apache DBCP.

2.3. RESTful web service

The RESTful web service provides CRUD operations on stock and user information. Specifically, it allows a user to add/delete followed stock symbol, get all followed symbols and associated newest stock prices, and query the history about stock price of a specific symbol. The RESTful service is implemented using the Jersey 2.0 library.

2.3.1. HTTP basic authentication

As the system is designed with a multi-user perspective, retrieving user (and password) information in the RESTful web service is necessary. Further, as a RESTful service, the service needs to be able to process each request independently; thus, username/password information needs to be carried on each request. An alternate approach would be storing authentication information in the web session; however, it can be considered as a violation to the RESTful attribute, and thus, should be avoided.

Basic HTTP authentication can be leveraged to store user information in each request, assuming HTTPS is used to protect data integrity and privacy. Jersey provides "pre-matching" content filter, in which all requests are first sent over a filter class, which can be used for authentication check. Any request without username/password information is denied. An alternate approach is OAuth, but is much more complicated.

Note that I made use of some piece of codes from other people for decoding the authentication information in the HTTP requests, specifically the BasicAuth and AuthFilter classes in the restapi package.

2.3.2. Connection Pool

Similar to the daemon process, the web service makes use of the connection pool feature provided by Tomcat web server. Connection pool is implemented in the dataaccess.util package.

2.3.3. Stock information retrieval

As described in the assignment, when a user retrieves the list of followed stock symbols, the RESTful service needs to provide updated associated stock prices; thus, in these cases, the web service also contacts the content provider for stock prices and then stores the information in MySQL, as the information retrieved by the stock daemon process might be to old (every minute matters in stock exchange).

2.3.4. Data cleanup process

The daemon process queries stock price information about followed stock symbols of all users. Thus, when a user deletes a stock symbol, the symbol should not be deleted, since other users might still follow that symbol. However, if no users follow a stock symbol, it would be unnecessary for the daemon to continue querying that symbol. Hence, when the last user that follows a symbol delete it, the symbol is deleted from the stock_symbol table, and all the associated information in other tables will also be deleted through cascade delete.

However, this is feature is still experimental and is commented out in the code.

2.3.5. The web application

The web application provides the users a convenient way to interact with the web service for CRUD operations. The web application itself is also implemented in Java, using JSP/Servlet. Currently, the system only support 1 user (no login page); however, it is easy to extend it to support multiple users as the design is scalability in other modules.

2.3.6. The home page:

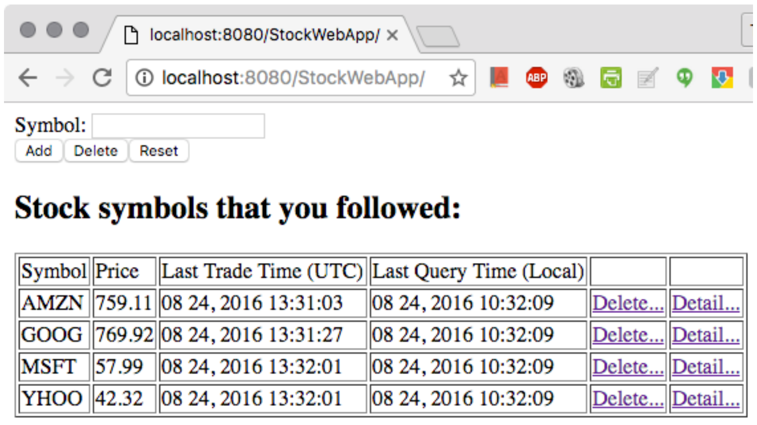
In the home page, the user can see all the items she is currently following, with associated latest prices. The prices are updated when the page is refresh, as show in Figure 2.

Here, the user can input a stock symbol (e.g., GOOG for Google, MSFT for Microsoft) and add or delete that symbol. The symbol id will be sent to a servlet, and then forwarded to the web service for processing.

One simple note, when a symbol is added, its name is checked against Google for validity. For example, if the user wants to add the symbol "xyzt", the system let the user know the symbol code is not valid and denies to add the item.

When a user follows a new stock symbol, previous retrieved information of the stock is used for the user also.

The user can also click the "Detail…" link to see history information about the symbol



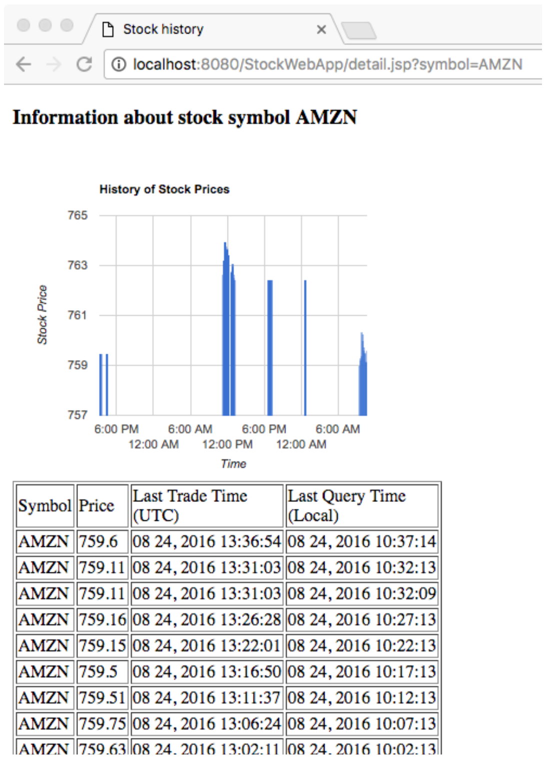
Stock symbols that you followed:

Symbol	Price	Last Trade Time (UTC)	Last Query Time (Local)		
AMZN	759.11	08 24, 2016 13:31:03	08 24, 2016 10:32:09	Delete...	Detail...
GOOG	769.92	08 24, 2016 13:31:27	08 24, 2016 10:32:09	Delete...	Detail...
MSFT	57.99	08 24, 2016 13:32:01	08 24, 2016 10:32:09	Delete...	Detail...
YHOO	42.32	08 24, 2016 13:32:01	08 24, 2016 10:32:09	Delete...	Detail...

Figure 2 Web homepage

2.3.7. The symbol detail page

When a user clicks on a detail link for a symbol, e.g. AMZN, the history information about the link is shown, as in Figure 3. The page shows detail prices via a table, as well as a graph. Google chart API is leveraged for drawing the charts from data provided by the web service api. Note that, the graph is not continuous since I did not collect information about stock prices continuously.



Information about stock symbol AMZN

Symbol	Price	Last Trade Time (UTC)	Last Query Time (Local)
AMZN	759.6	08 24, 2016 13:36:54	08 24, 2016 10:37:14
AMZN	759.11	08 24, 2016 13:31:03	08 24, 2016 10:32:13
AMZN	759.11	08 24, 2016 13:31:03	08 24, 2016 10:32:09
AMZN	759.16	08 24, 2016 13:26:28	08 24, 2016 10:27:13
AMZN	759.15	08 24, 2016 13:22:01	08 24, 2016 10:22:13
AMZN	759.5	08 24, 2016 13:16:50	08 24, 2016 10:17:13
AMZN	759.51	08 24, 2016 13:11:37	08 24, 2016 10:12:13
AMZN	759.75	08 24, 2016 13:06:24	08 24, 2016 10:07:13
AMZN	759.63	08 24, 2016 13:02:11	08 24, 2016 10:02:13

Figure 3 Stock detail

2.4. Building the system

2.4.1. Requirement:

- MySQL
- Tomcat: 8.0+, has not tested with other web server
- Maven: 3.3.9+
- MacOS or Linux system: has not tested on Windows

2.4.2. Running sql script

First and most important, please manually run the sql script (in the sqlscript folder) on the SQL database, it cannot be done automatically by the provided shell script.

2.4.3. Build script

The build.sh is provided for building the system. This file contains some very important configuration settings for the system, **please change those settings to reflect the setup at your location before running the script.** Further, please do not change the location of the build script or other folders, as the script use relative paths to build the system.

Important Settings:

```
#define configurations
#sql server <host>:<port>
```

```
dbhost=localhost:3306
#database name, default=stock
dbname=stock
#username to access db
dbuser=root
#password
dbpass=tuandao
#path to tomcat webapp folder
tomcatPath=/Users/tuandao/tomcat/apache-tomcat-8.0.36/webapps
#path to web server, for JUnit test the rest api, by default http://localhost:8080/StockWebApp/
weburl=http://localhost:8080/StockWebApp/
```

The build script use ‘sed’ to automatically change the settings in other files based on provided information.

The build process:

- Change setting files
- Build the Daemon process
- Install the daemon process to the local maven repository: the web app will reuse some part of the daemon source code, so this step is necessary
- Build the web app, copy the web app to the tomcat webapps folder.
- Run Junit tests

2.4.4. Test cases

I provided the following 6 Junit test case to test the rest api:

- Test connection to MySQL: use “select 1” and get result from MySQL
- Accessing the rest api withouth authentication, HTTP code 401 is expected.
- Four mock test to add, delete, query information about a symbol, query all followed symbols to test the CRUD operations of the rest api

Note: Do not manually run the test case with “mvn test”, since it requires an input configuration file. Please use the build script, or refer to the final part of the script, for running the test cases.

2.4.5. Running the stand alone daemon

The build script compiles and packages but does not execute the stand alone daemon process. To execute the daemon, use the following command:

“java -jar StockDaemon/target/stockdaemon-1.0-SNAPSHOT-jar-with-dependencies.jar **daemon.conf**”

with daemon.conf is the configuration file for the process

2.5. Contact

There might be undiscovered bugs in the system, and the code is not very carefully commented I as wanted because of the time limitation. If you have any questions regarding the source code or the building process, please contact me at tdao006@cs.ucr.edu

Source code of this project is also available at https://github.com/datuan/stockmonitor