MONITOR STOCK PRICE APPLICATION

DESIGN DOCUMENT

Overview:

Monitor Stock Price is an application to monitor the changes and trend in the stock price of companies. This application exposes Rest apis to perform CRUD operations. There is one more project in repository Monitor-stock-price-ui uses this exposed Rest apis and shows CRUD operations in a web application.

Project Repository - https://github.com/kasturivarun/Monitor-stock-price

Requirements:

R1 – User should be able to add a new company to database. As soon as the company is added its current stock price should be updated.

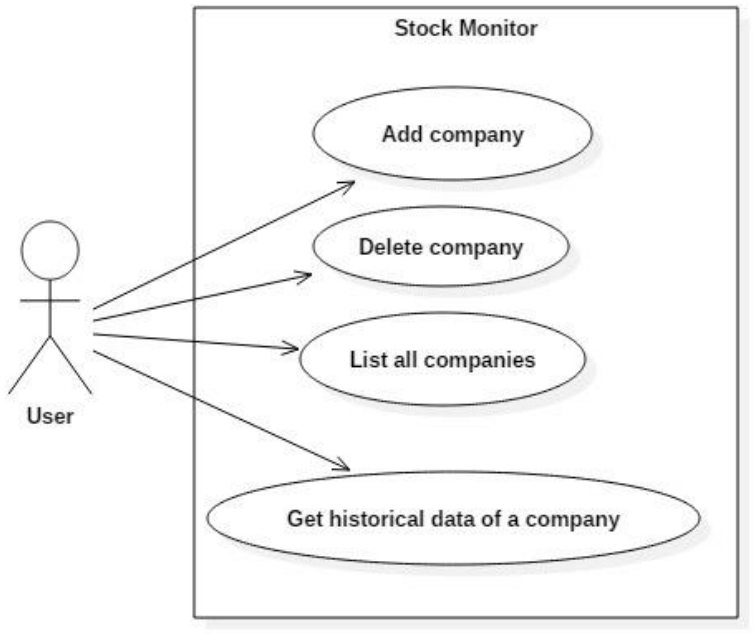
R2 – User can stop monitoring a company and delete the company.

R3 – User should be able to see the list of all the companies' information in the database.

R4 – User should be able to see the historical data for a particular company.

Use case diagram:

The following is the use case diagram for the application.

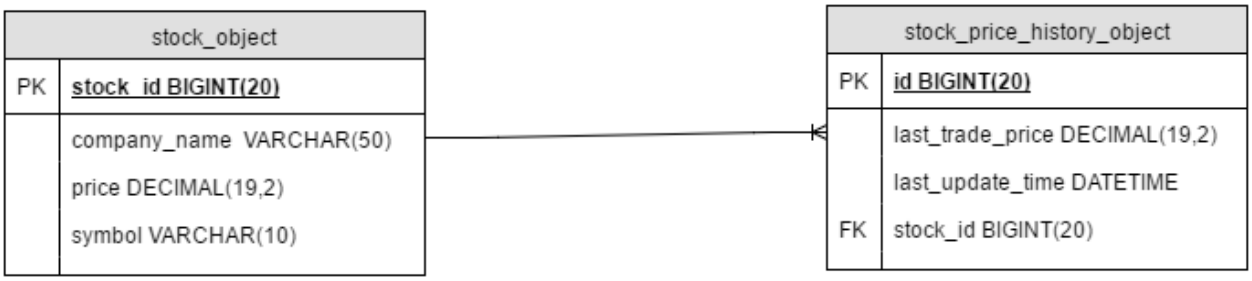


Database design:

The application has two tables,

stock_object – This table stores the details of individual company details with current price.

stock_price_history_object – This table stores the historical stock price for a particular company. It also stores the time that price was recorded.



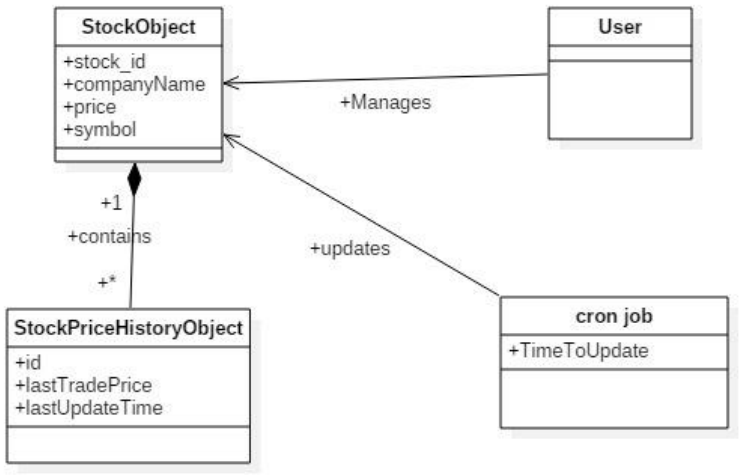
Above diagram shows the database schema. The following are the schema details,

1) stock_id is the primary key for stock_object.
2) id is the primary key for stock_price_history_object
3) stock_object table has one to many relation with stock_price_history_object table.
4) stock_id acts as a foreign key to stock_price_history_object table.

I used object relational mapping in this application. If the tables are not created then it will be created by the application. stock_id and id in the tables are auto incremented.

Domain Model for the application:

There are only four objects in the system.



A user can manage a StockObject.

A cron job runs every 5 minutes which updates the StockObject and adds the old record to StockPriceHistoryObject.

Application Usage:

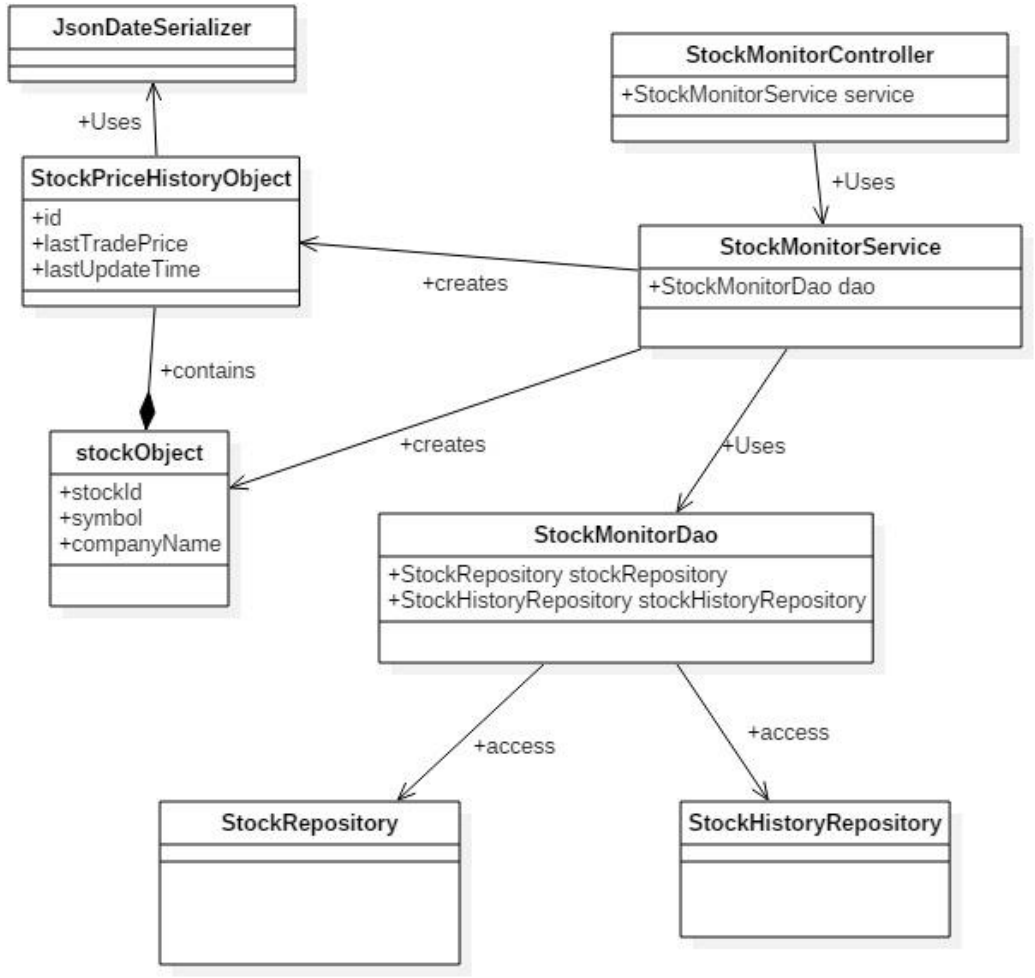
The specifications of the rest API is as follows,

API route	GET	POST	DELETE
/company/AAPL	This request will return historical stock data for Apple Inc.		If Apple Inc is present in the database then it will delete the record and return true else it returns false.
/company/AAPL		This request will create a new entry in the database for Apple Inc. If Apple Inc. is already present in the database then it returns false else it will return true	
/company	This request will return the list of all companies in database with latest stock price.		

Class diagram for the application:

I followed Model View Controller (MVC) architecture in designing the application. Therefore the code maintenance and reuse is easy as there is less dependency between business, view and database layer. MVC makes the application easily scalable. The model objects are directly coupled with database tables.

Below is the class diagram for the application.



StockMonitorController is the main controller for the application. This controller calls appropriate service methods depending on the rest API routes through StockMonitorService class. The service calls include `getSymbol()`, `addSymbol()`, `deleteSymbol()` and `getAllCompanies()`.

The entire business logic for the application is written in StockMonitorService. The service calls inside StockMonitorService class will create StockObject and StockPriceHistoryObject whenever required.

The service calls access the database through StockMonitorDao. This is a database access object whose functionality is to use StockRepository or StockHistoryRepository object to connect to database and perform necessary functionality. In this way the business logic is separated from database logic.

I used a util class called as JsonDateSerializer as the output date format from this rest apis are unreadable. This classes change the format of the date format in the output.

In addition to above functionalities, there is cron job in application which runs every 5 minutes and updates the companies with new stock price. This cron job is executed through `updateStockHistoryPriceRecord()` in StockMonitorService class. @scheduled annotation is used for this cron job. This job updates the stock_object table with new price and add one more entry to stock_price_history_object table.

Design features of the application:

1) I used Singleton pattern that restricts the instantiation of a class to one object.
2) I used MVC architecture which makes code maintenance and reuse easy.
3) Object relational mapping is used in this application which carries the updates of the model to database.
4) My application has the advantages of spring such as Dependency Injection and Inversion of control. Therefore whenever a StockObject is created then StockPriceHistoryObject is also created.
5) Unit test are written using Junit in the application which runs when a new build is kicked off.

Build and Deployment:

The following must be present to run the project:

- JDK 1.8
- MySql server. (You can download workbench also from MySql website which makes visualization easy) (http://dev.mysql.com/downloads/mysql/)
- Maven 3.0+

Technologies used

- Java
- Maven
- MySql
- Spring framework
- Junit, Mockito
- Github - https://github.com/kasturivarun/Monitor-stock-price

Instructions to run the application:

1) Install Java, Maven, MySql
2) Create a database in MySql.

3) You clone the application by,
   $ git clone https://github.com/kasturivarun/Monitor-stock-price.git
4) Change the database details in src/main/resources/application.poperties. The changes should be made to following fields,

```
spring.datasource.url=jdbc:mysql://localhost:3306/{database name}
spring.datasource.username={username}
spring.datasource.password={password}
```

   Now, the perquisites are finished.

5) Open command and traverse to project folder.
6) Type the command “mvn clean package”. This command will create a jar in target folder in project folder.
7) Traverse to the jar folder and type the command “java –jar {JAR NAME}.jar”.
8) This command will start the server now and you can access the rest API from the browser.