

Web Stock Application

V1.0

Author: Lifeng Xiang

Creation Date: August 26, 2019

Content

1, Overview…………………………………………… 3

2, Database……………………………………………. 4

3, Backend…………………………………………….. 6

4, Frontend…………………………………………….. 9

5, Results……………………………………………… 11

6, Future Improvements……………………..…………13

**Overview**

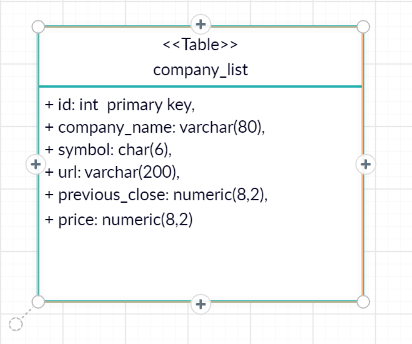
This purpose of this task is to load the live stock price together with the related company information from database, and display into a webpage. Therefore, it contains three parts for the project:

* Database - it saves the list of company information, as well as the their stocks’ trading prices history;
* Backend - it is responsible to provide the detailed information to the frontend client whenever such information is requested;
* Frontend – Once the website is opened, frontend send the request to backend, retrieve the response data, and display into the webpage.

**Database**

It is required to use RDBMS for the project. Therefore, MySQL is being selected for the task.

As a starting point, I use a very simple model to begin with, the plan is to make everything working first. The model I started with just use one table:



The database setting for the backend is configured at “/src/main/resources/application.properties” file.

The sql used for the create database, table, and insert data are as follows:

create DATABASE webstock;

use webstock;

DROP TABLE IF EXISTS companylist;

create table companylist (id int primary key, symbol char(6), company\_name varchar(80), url varchar(200), previous\_close numeric(8,2), price numeric(8,2));

INSERT INTO companylist VALUES (1, 'AAPL', 'Apple Inc', 'http://www.apple.com/', 202.64, 206.64);

insert into companylist values (2, 'BIDU', 'Baidu Inc (ADR)', 'http://www.baidu.com.cn/', 101.85, 103.4);

insert into companylist values (3, 'NFLX', 'Netflix Inc', 'https://www.netflix.com', 291.44, 295.70);

insert into companylist values (4, 'MSFT', 'Microsoft Corpor', 'http://www.microsoft.com/', 133.89, 134.78);

insert into companylist values (5, 'EBAY', 'eBay Inc', 'http://www.ebay.com/', 38.70, 39.16);

insert into companylist values (6, 'AMZN', 'Amazon Inc', 'http://www.amazon.com/', 1749.62, 1771.56);

insert into companylist values (7, 'GOOGL', 'Google Inc', 'http://www.google.com/', 1153.92, 1170.00);

insert into companylist values (8, 'FB', 'Facebook Inc', 'http://www.facebook.com/', 177.75, 179.8);

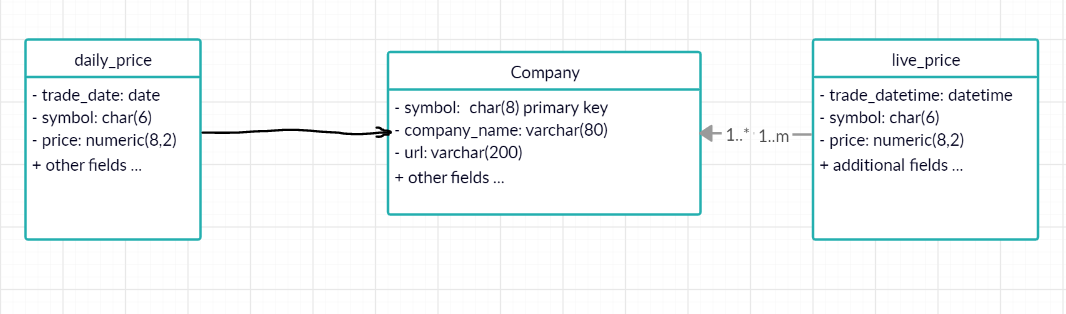
insert into companylist values (9, 'PM', 'Philip Morris International Inc.', 'https://www.pmiscience.com/', 80.89, 77.73);

insert into companylist values (10, 'QCOM', 'QUALCOMM, Inc.', 'https://www.qualcomm.com/', 73.52, 72.91);

insert into companylist values (11, 'DD', 'DuPont de Nemours Inc', 'https://www.dupont.com', 65.99, 64.26);

insert into companylist values (12, 'GE', 'General Electric Company', 'http://www.ge.com/', 7.97, 8.05);

insert into companylist values (13, 'MMM', '3M Company', 'http://www.facebook.com/', 156.35, 156.60);

I plan to update it to a more proper model if time permits. 

**Backend**

Spring Boot 2 is chosen as the development tool for the backend, comparing with Spring framework, Spring Boot has drastically reduced the configuration and setup time for spring projects. Web, JPA, MySQL are the dependencies from Spring Boot. The directory structure of the web stock project is showing in the next page.

It includes the following packages:

* spring.project.stockModel
* spring.project.stockModel.controller
* spring.project.stockModel.exception
* spring.project.stockModel.model
* spring.project.stockModel.repository

**spring.project.stockModel**

This is the main entry point of the Spring Boot backend application:

package spring.project.stockModel;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing

public class StockModelApplication {

public static void main(String[] args) {

SpringApplication.run(StockModelApplication.class, args);

}

}

The simplified @SpringBootApplication annotation tells Spring to automatically configure the application based on the dependencies defined in pom.xml file and the other tasks necessary to start the application. The application is launched by calling the Spring Boot’s SpringApplication.run() method.

**spring.project.stockModel.model**

This package defines the data model of the application. It has the following fields:

**private** Integer id;

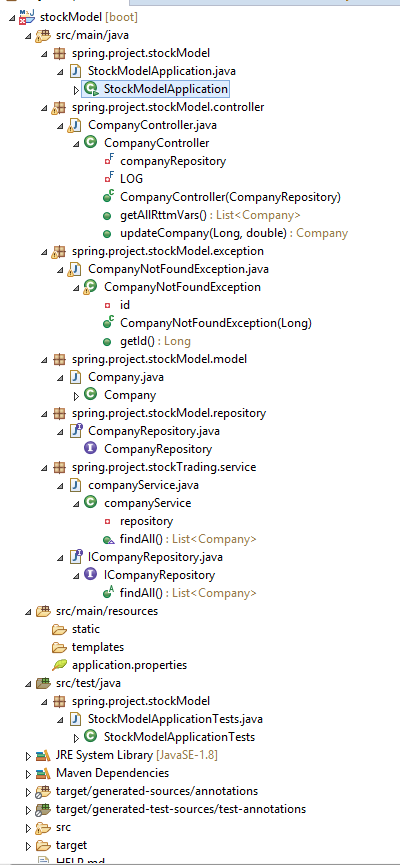
**private** String symbol;

**private** String companyName;

**private** String url;

**private** **double** previousClose;

**private** **double** price;



**spring.project.stockModel.repository**

This repository package allows the application to access data from database:

**package** spring.project.stockModel.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** spring.project.stockModel.model.Company;

@Repository

**public** **interface** CompanyRepository **extends** JpaRepository<Company, Long> {

}

The JpaRepository interface defines all the CRUD operations on the entity. The @Repository

annotation tells Spring to bootstrap the repository during component scan.

**spring.project.stockModel.exception**

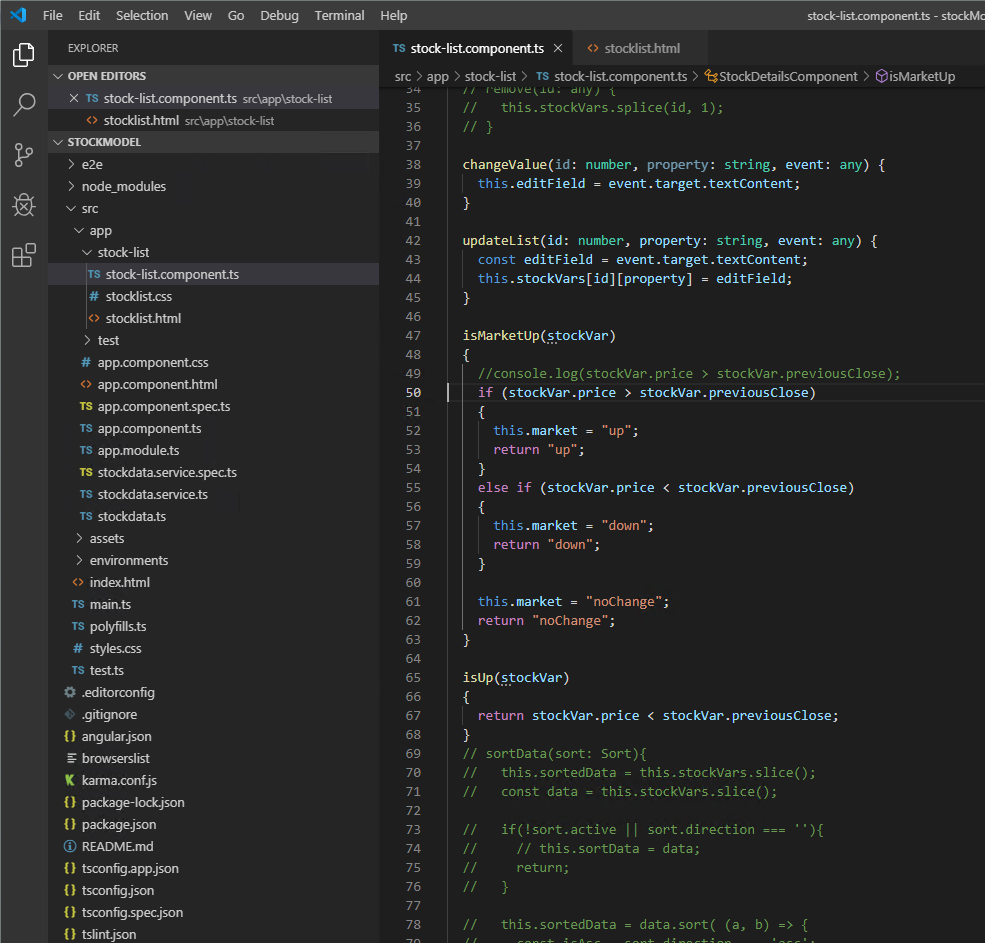
This package defines the custom runtime exception during the database update operation.

**spring.project.stockModel.controller**

The controller package defines the REST APIs for the service to do the CRUD operation with entity.

**Frontend**

The frontend uses html5, css inside the angular framework, here is the directory structure:



HTML call structure:

1, The index.html is simply the entry point the web page:

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Intro</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html>

2, The <app-root></app-root> block defines the entry point for app.component.html:

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

</div>

<stock-detail></stock-detail>

3, Here the <stock-detail></stock-detail> block defines the access of stocklist.html, the main part of the html, which shows the details of the table display.

The StockDataService class in stockdata.service.ts makes the http request to retrieve the detailed information from web-service in the backend.

While the stockdata.ts defines the interface of the data structure of the http response data retrieved:

export interface StockData {

symbol: string,

companyName: string,

previousClose: number,

price: number,

url: string

}

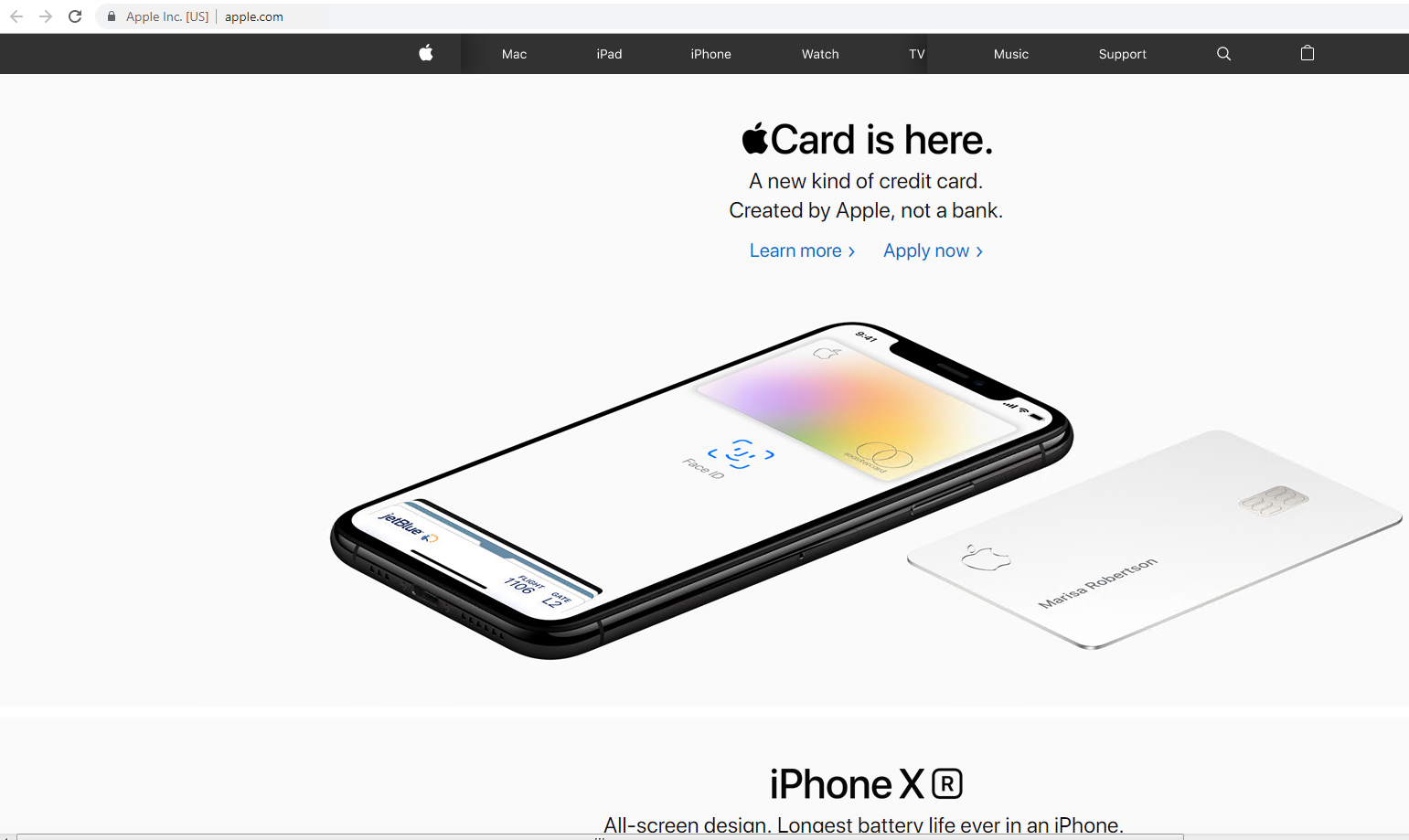
**Results**

The data in the MySQL was able to be retrieved successfully and displayed in the webpage as showing below:



Here are the functionalities implemented in the table:

* The Change and Change Percent columns were calculated based on the data retrieved;
* The table is showing different colors in even and old rows.
* The ‘+’ is displayed when the price change is positive, i.e. the market is up.
* These two calculated columns are showing in different colors when market is up, down or flat.
* All the numeric numbers are showing with 2 decimal points;
* The company name is left aligned, while the rest of columns are much shorter, so these are center aligned;
* When the company name is clicked, a separated tab is displayed for the company. For example, when Apple is clicked, the following page opened in a new tab.



**Future Improvements**

* Live update in the webpage when the market data is updated
* Sorting column in the table is not working yet
* A better database table design model is more preferred