WPL Final Project Report

Team: Elkmé

Authors: Zongzhe Li, Tianhang Zhu, Yuncheng Gao

## Problem Description

The goal for this project is to implement two web applications: the stock broker application and the stock exchange web application. The stock broker web application provides functionality and user interface for stock brokers with the ability to buy/sell stocks, search stocks, transfer of money from and to their web broker application wallet, and the ability to manage their own stocks and user info. In order to achieve the above functionality requirements, we used web programming standards, various protocols and techniques learned in class, including Ajax, asynchronous services, HTTP requests, APIs, CSS styling, HTML5, distributed caching, authentication services, data compression, database queries and dynamic data tables. Additionally, to achieve the idea of service-oriented-architecture, we separated the stock broker’s web application into two servers, with an additional isolated stock exchange service. The detailed implementation and design of these three web applications will be discussed in this report.

## Web Application Architecture Design

## 

## Techniques And Technologies Used

At first, we thought of using Python Django as database and html files to render the front end. However, as we research through the Internet, we found more abundant resources on teaching how to use MySQL as database for the backend and using Tomcat to hold web pages and hosting other servlets for web services. The first approach for connecting database to our development environment (Eclipse) was to connect MySQL using “Data Connections” feature inside Eclipse. Although the connection between Eclipse and database was easy to set up, it had no effects on connecting database to the web application. Therefore, we decided to use those servlets to make MySQL and application server in Eclipse communicate with each other instead of figuring out a new way of connection in using Python Django.

We used Apache Tomcat v9.0.29 web server, Eclipse IDE 2019-09 for Java EE as the programming platform, MySQL community server as our database and MySQLWorkbench as our display tool for the database. We used jsp files to define the UI of the webpages, and the Java servlet files to handle Post and Get requests along with other service functionalities to redirect users to different pages, accessing databases and so on. We used the default port 8080 provided by Tomcat to host our frontend pages. We used local instance 3306 to connect to our database. There is a MySQL database that can start on each local machine. A dataset of four tables can be downloaded among team members to their local machines. MySQLWorkbench connects to MySQL and gets our imported dataset “latest.sql”. Tomcat server is set up in Eclipse as the web server for hosting web pages.

Three web application projects are implemented using Eclipse. Each jsp file is for a certain web page in the web application. Users can access all the web pages starting with the login page at “https://[localhost:8443/stocky](https://localhost:8080/stocky/index.jsp)”. In each project, Java files inside “Java Resources” serve as servlets to redirect pages, get cached data, retrieve data from database, or record user edits to the database.

Application server “stocky” uses HTTP port 8080, SSL port 8443, Tomcat admin port 8005, AJP port 8009, is where we implement the responsive real time brokerage web application. Most of the UI jsp files are located in that project, including add bank page, member home page, log in page, password retrieval page, display password page, registration page, personal profile page and bank account money transfer page.

Application server “webApplication” using HTTP port 8180, SSL port 8543, Tomcat admin port 8105 and AJP port 8109, connects to the database tables. It retrieves data and memcached them. “stocky” makes web service calls by using database object created in “webApplication”. The information in front end pages involving user data is identified based on user email addresses. Most of the Java files are located in this project,including features AddBankAccount, CheckPassword, DBConnection, EditProfile, JspFilter, LoginCheck, Logout, MemberAuthentication, Registration, TransferService, TransferServiceAction, Utilities and View Profiles. Java files either help navigate browser redirect to different pages based on the logic, update data changes from user input to database, or retrieve data from database and send them to the front end.

Application server “stockExchange” using HTTP port 8280, SSL port 8643, Tomcat admin port 8205, AJP port 8209, contains ViewStockMarket, getStockPrice, Utils and DBConnection Java files. ViewStockMarket.java uses JSON Object to store data, idSTOCKS, STOCK\_NAME, Company, Price, Shares and TimeStamp. And store them into JSON Array.

“mysql-connector-java-8.0.18.jar” files are imported in all three application servers for connecting application servers Stock Exchange and webApplication using Java files to the database. JSON jar files are imported to all application servers for parsing requests. The jar file is called “java-json.jar”. Importing it into the Java Library References in our dynamic web projects in Eclipse, we are able to parse requests sent among web application for doing queries or making updates to the database.

## Functionality Details

* Users are able to log in with their emails and passwords, and log out to the login page.
* They can also click on “Forgot my password” to proceed to answer a security question to get their password revealed to them.
* New users can register by clicking the “Sign Up” button. They will be transferred to Sign Up page to enter their emails, user ids, passwords and physical addresses.
* Once users finish logging in, they enter the member home page where they can determine whether to view personal profile, add another bank, transfer money. Once logged, the menu bar shows five options, either to go to the login page, member home page, current stock market, buy/sell stock page, or personal stock history page.
* A personal profile page where user can edit his or her bank information, physical address and email address. Users can add one or more banks on personal profile page.
* Among users’ bank accounts, users are able to transfer money from one bank account to another directly on the website. By clicking on one of the radio buttons, users can select one of their accounts to transfer money.
* User can set the amount of stocks to be purchased under a user selected schedule from a particular company. Or user can buy a number of stocks from a company as a one-time purchase. Buying stocks is done on “Trade Stocks” page rendered by ‘stocky’ application server. On that page, user first type the stock name to view its availability. Then enter an amount that does not exceed the limit or below. Invalid input amount would be rejected. Below that, there is a search feature for searching for stock history. Users can view the history of a certain stock for a desired time period by typing stock name and select the specific start date and end date.
* User can delete, add or modify any row of the stock table. There is a page listing all the stocks owned by the user called “My Stocks”, in which 4 columns, “ID”, “Stock Name”, “Shares” and “Buy Time” are sortable to sort the entries on the current data table.
* An error 404 page has been designed for display when user is trying to access any unavailable content. Click on “Home” in menu bar to go back to the Login Page.
* MySQL runs separately from web application in Eclipse. A schema named “stocky” that stores three tables, “BANK\_ACCOUNT”, “STOCKS”, “USER\_STOCKS” and “USERS” for the web applications.
* From weekdays 8am CDT to 5pm CDT, a table of 100 companies' stock prices is shown to users in stock brokerage web application. A display message on every page indicating whether stock market is open at the current time.
* Current stock prices change every 30 secs based on the random number generator.
* Users are able to type down the date to see the current day, last business day, current week, previous week, certain month in the year upto today, certain year upto today, or type down to see the stock in the past 5 years. They can select any time period
* Stocks are only allowed to be purchased based on current prices.
* For distributed cache, we chose memcached option and used spymemcached jar file in Eclipse Java Build Path. One feature is that a string “Have not edited profile recently” for profile editing status is cached in login page to memcache server as “status” on port: 11211. When user reaches profile page, the string is got from cache and displays to user. After user finishes editing and clicks on “save” button, the string “status” on server is replaced with “Edited profile recently”. Another feature that took advantage of memcached is a date variable. When user enters password retrieval page, a date variable is initialized with the current date. When user gets to show password page, not only its password is shown, but also the process start date got from memcache server displays to user.
* Data tables are used for displaying current stock market table and personal purchased stocks “My Stocks”. In “My Stocks”, stock purchase time, stock id, amount of shares and stock name. Users can input amount to sell and click on “Sell” button to sell. Excessive data on those tables can be loaded by clicking on indexes. Users can also select how many entries to view on the current data table. Current stock market is implemented in Stock Exchange Application Server.
* There is a specific page for searching for a particular stock. Type the stock name and click on “View” to view the stock information consisting of stockID, stock name, company, price, shares and last updated.
* In the database, the newest dataset we created is called “latest.sql”. There are four tables created to support our web applications.

In “BANK\_ACCOUNT” table, there are four columns representing, User\_ID, Bank\_Account\_num, Bank\_Routing\_Num, and Balance.

In “STOCKS”, there are six columns, idSTOCKS, Stock\_Name, Company, Price, TimeStamp and Shares. Shares store the amount of stocks available for purchase for that particular stock. TimeStamp indicates the time that stock price was updated. idSTOCKS indexes them in an order in the list.

In “USERS” table, there are 6 columns, “Email\_Address”, “First\_Name”, “Last\_Name”, “Password”, “Address” and “Wallet\_Balance”.

In “USER\_STOCKS”, there are 6 columns, “id”, “USER”, “Stock\_Name”, “Shares”, “Buy\_Time” and “Sell\_Time”.

## Web Services

1. The stock brokerage website “stocky” makes a Web Service call to the stock brokerage web application “webApplication” for user request as to retrieve data from database, or update changes made by the user to the database through data retrieval or update requests.

2. All calls from the stock brokerage “webApplication” to the stock exchange application server are through a web service. We use JSON jar files to parse requests sent among web applications to call certain functions to do certain things with the database. Requests sent from “Stock Exchange” or “stocky” to “webApplication” are used to update database or pull data from database through “webApplication”.

3. The stock brokerage website “stocky”, stock brokerage web application “webApplication”, and stock exchange web application “exchange” are implemented separately and run on three tomcat servers via different ports. Eventually, the ports are encrypted using SSL for web security. URL addresses connected via 8080, 8180, 8280 ports are automatically redirected to secure ports: 8443, 8543 and 8643, respectively.

4. The stock brokerage website “stocky”, stock brokerage web application “webApplication”, and stock exchange web application “exchange” can all run on the same local machine and be built and run on the same platform, Eclipse, and run on three tomcat servers with secure ports.

5. All Web Services contain authorization, which is based on cookies that store user email addresses.

## Problems Encountered

* One problem we encountered was setting up Tomcat server on Eclipse. There were problems when we tried to open a different project, Tomcat servers wouldn’t switch to run the new project. Then we learned that we hadn’t deleted old Tomcat server configuration on the same port. Eclipse would still use the old Eclipse environment. We can remove previous Tomcat settings by going to Eclipse → Preferences → Servers/Runtime Environments and remove the old runtime environment over there.
* Also when we opened project from each other, Eclipse gave lots of “errors” in Java files. The cause could be that Java Build Path is not properly set up. Tim found the solution as to change build path. Specifically, right click on the targeted project, select “Properties”, search for “Java Build Path”, select “Module” and click on “Add external Jar files”, add mysql-connector-java jar file into the Java Build Path library and apply the connector in deployment assembly.
* A problem Zongzhe encountered was that localhost failed to start. Many factors could lead to such error. Eventually, Zongzhe looked up the error “an illegal reflective access operation has occurred” and found that JRE team were not planning to fix such issue anytime soon. However, restarting Eclipse happen to solve such issue that Eclipse is not connecting to Tomcat servers properly.
* Not all SSL encryption tutorials are accountable to do the job of redirecting our websites to secure ports. But find the solution was not difficult. Eventually, we found one block of code containing commands to redirect all our 8080, 8180, 8280 ports to 8443, 8543, and 8643, which are the secure ports. We needed to insert the block of code in server.xml in all Tomcat servers we added in Eclipse.
* We thought database was supposed to connect to Eclipse in order to be used. But we found out that was not necessary. To make interactions between web applications and the database, web services need to be implemented to retrieve data and record changes. Those web services are located in servlets as Java files.
* One common issue with Tomcat connecting to Eclipse is “Localhost:8080 server is still running”. For Mac or other Unix system, type command “pkill -f catalina” or “pkill -9 -f catalina” to kill all Tomcat server processes running in the system. After that, restart Tomcat servers in Eclipse to run the web applications, the error message should be gone.
* Sometimes, projects do not seem to have any errors, but the browser could not display correct output. We encountered such problem several times. One reason could be that Eclipse is not refreshed. Restart Eclipse would solve the problem.
* Opening each other’s projects is also inconvenient. Eclipse user could only create a new dynamic project for each shared project and copy paste the files to the new project file locations, set up each server’s connection and make sure tomcat web servers’ ports are not occupied.
* Eclipse was an inconvenient programming platform for our project. It periodically fails for loading desired content, or give out “XX is not found” type of errors without indicating the actual cause or providing any solutions. We googled about all kinds of those “XX is not found” or “Not able to access” problems caused by Eclipse and tried out redesigning our code, or looking back again at certain parts of the code. In most cases, restarting Eclipse or tomcat servers within Eclipse would solve the problems. But for several times, we received errors about data tables not found. After spending hours debugging and restarting servers, database and Eclipse, we found that restarting the entire operating system was the solution to resolve the problem with Eclipse not recognizing our database tables and causing the “Data Table not found” error.

## Problem And Study References

Apache Tomcat Download: <https://tomcat.apache.org/download-90.cgi>

MySQL Download: <https://dev.mysql.com/downloads/>

MySQL Workbench Download: <https://dev.mysql.com/downloads/workbench/>

Eclipse Packages Download: <https://www.eclipse.org/downloads/packages/>

Fixed The Issue That Eclipse Gives Errors in Java files: <https://stackoverflow.com/questions/1585811/classnotfoundexception-com-mysql-jdbc-driver/32936812#32936812>

Illegal reflective access when I stop SpringBoot web application with Tomcat 9 and Java10: <https://stackoverflow.com/questions/52185626/illegal-reflective-access-when-i-stop-springboot-web-application-with-tomcat-9-a?noredirect=1&lq=1>

Memcached Set Data: <https://www.tutorialspoint.com/memcached/memcached_set_data.htm>

How To Install Memcached On Mac With Homebrew And Lunchy. <https://gist.github.com/tomysmile/ba6c0ba4488ea51e6423d492985a7953>