Project Name: Virtual Stock Transaction Platform
Team Members: Hao Wu, Mengyu Wu, Yutaka Urakami
Source code: here

Final State of System Statement

Sell stock

A user can sell less than or equal to the number of owing stocks. If a user selects more than the number, the error page will be shown.

Buy stock

A user can buy eight companies' stocks within his/her balance. If a user selects more than the upper limit of stock, the error page will be shown. We limited the number of companies because there is an API limitation. The limitation is 120 API requests per minute.

Change a user's profile

A user can modify the first name, last name, email.

Ranking

A user can see the ranking which is the sum of the remaining balance and the owing stock's current prices.

Login

A user can log in.

Create an account

A new user can make a new account to use this service.

Publish the program on the Web.

We deployed it by AWS Elastic Beanstalk here.

Final Class Diagram and Comparison Statement

Final UML Class Diagram

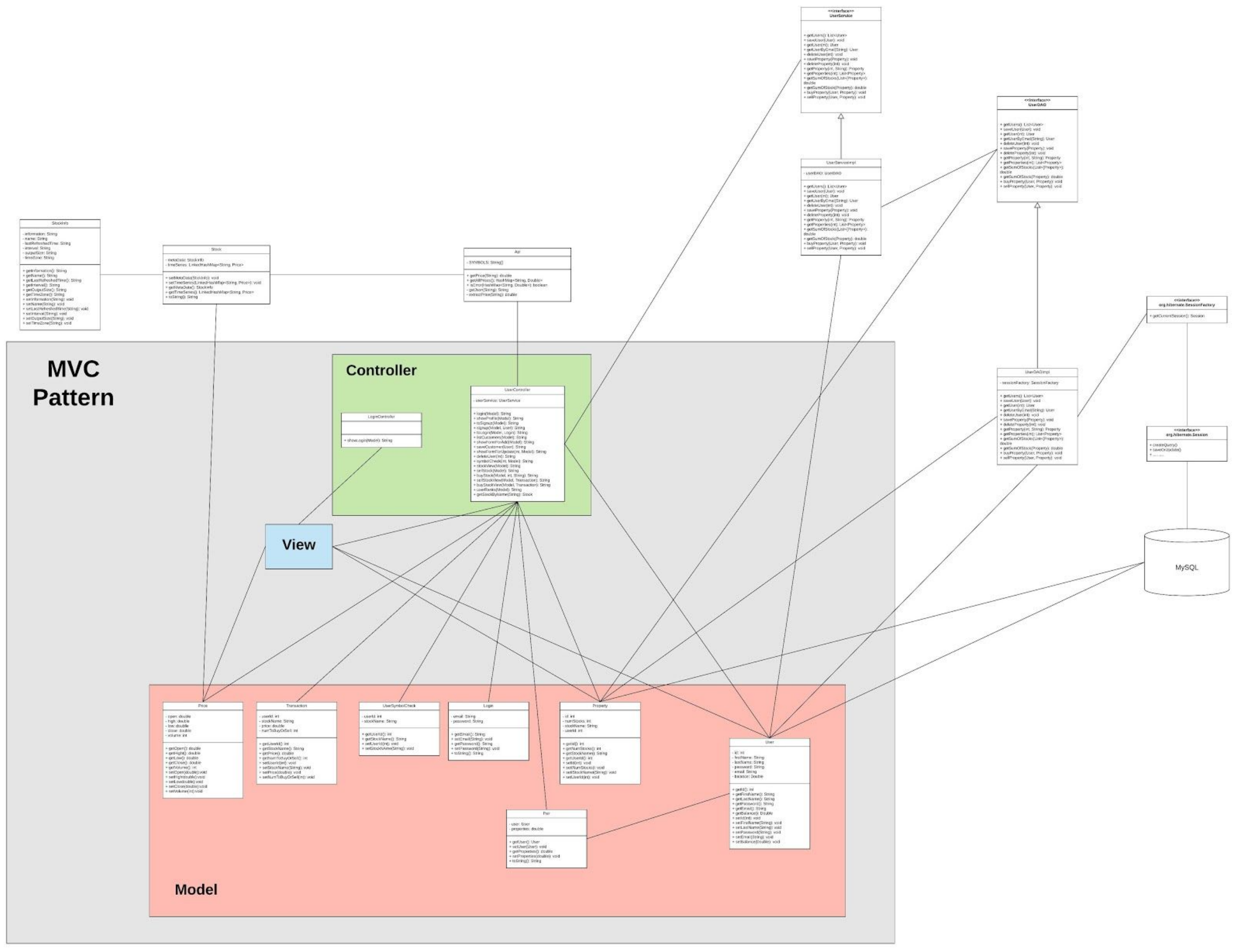
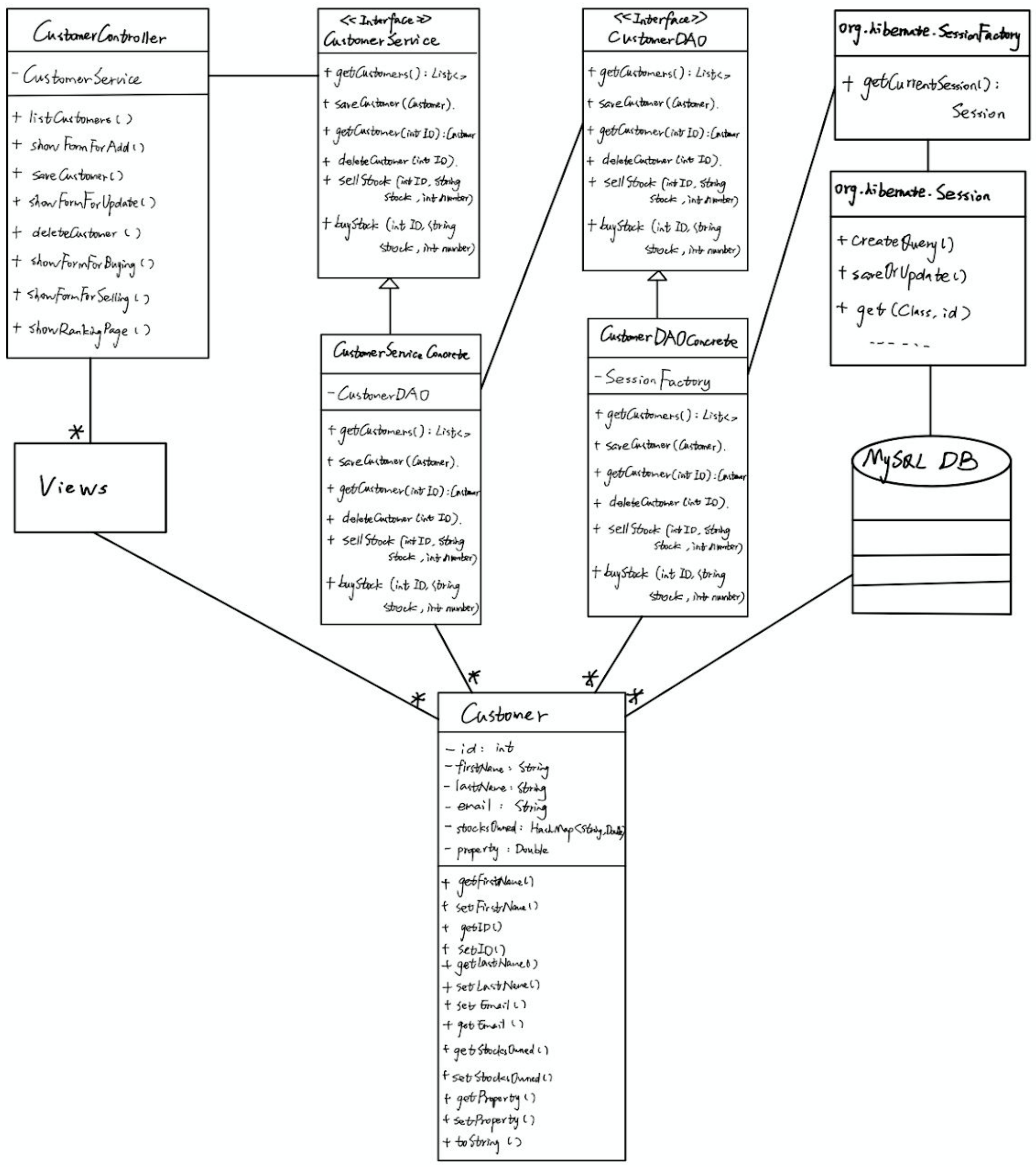


Fig. Class Diagram in Project 4



CustomerController
- CustomerService
+ listCustomers ( )
+ showFormForAdd ( )
+ saveCustomer ( )
+ showFormForUpdate ( )
+ deleteCustomer ( )
+ showFormForBuying ( )
+ showFormForSelling ( )
+ showRankingPage ( )
<<Interface>>
CustomerService
+ getCustomers ( ) : List<>
+ saveCustomer (Customer).
+ getCustomer (int ID) : Customer
+ deleteCustomer (int ID).
+ sellStock (int ID, String stock, int number)
+ buyStock (int ID, String stock, int number)
<<Interface>>
CustomerDAO
+ getCustomers ( ) : List<>
+ saveCustomer (Customer).
+ getCustomer (int ID) : Customer
+ deleteCustomer (int ID).
+ sellStock (int ID, String stock, int number)
+ buyStock (int ID, String stock, int number)
org.hibernate.SessionFactory
+ getCurrentSession ( ) : Session
org.hibernate.Session
+ createQuery ( )
+ saveOrUpdate ( )
+ get (Class, id)
CustomerServiceConcrete
- CustomerDAO
+ getCustomers ( ) : List<>
+ saveCustomer (Customer).
+ getCustomer (int ID) : Customer
+ deleteCustomer (int ID).
+ sellStock (int ID, String stock, int number)
+ buyStock (int ID, String stock, int number)
CustomerDAOConcrete
- SessionFactory
+ getCustomers ( ) : List<>
+ saveCustomer (Customer).
+ getCustomer (int ID) : Customer
+ deleteCustomer (int ID).
+ sellStock (int ID, String stock, int number)
+ buyStock (int ID, String stock, int number)
Views
MySQL DB
*
*
*
*
*
Customer
- id : int
- firstName : String
- lastName : String
- email : String
- stocksOwned : HashMap<String, Double>
- property : Double
+ getFirstName ( )
+ setFirstName ( )
+ getID ( )
+ setID ( )
+ getLastName ( )
+ setLastName ( )
+ setEmail ( )
+ getEmail ( )
+ getStocksOwned ( )
+ setStocksOwned ( )
+ getProperty ( )
+ setProperty ( )
+ toString ( )

Changes

In project 4 and 5, we missed login feature, storing properties, handling of the API for getting a stock price. Therefore, we added several classes for them. Also, we used "Customer" as class names in project 4, but we changed it to "User." We create a page

"Stock view" for showing more stock details and in-time stock chart. Finally, we redesign the layout and activity flow for the whole web app.

Third-Party code vs. Original code Statement

Setting files

We used some setting files on this online tutorial.
WebContent/WEB-INF/web.xml
WebContent/WEB-INF/virtual-stock-platform-servlet.xml

Modules

org.springframework:spring-context
org.springframework:spring-webmvc
org.springframework:spring-web
org.springframework:spring-tx
org.springframework:spring-orm
org.springframework:spring-core
org.hibernate:hibernate-core
org.springframework.security:spring-security-web
org.springframework.security:spring-security-config
org.springframework.security:spring-security-taglibs
com.mastfrog:jackson
mysql:mysql-connector-java
com.mchange:c3p0
javax.servlet:javax.servlet-api
javax.servlet.jsp:javax.servlet.jsp-api
javax.servlet:jstl
javax.xml.bind:jaxb-api
junit:junit

Statement on the OOAD process

1. **Positive: When we design this whole project, we found many useful frameworks, like Hibernate, Jackson, etc. They can work with the Spring framework really well and each of them has a clear responsibility. For example, the Hibernate framework is an ORM pattern which will work with MySQL database really well, and all our entity can be in a Java Object way, it can make this project's code style to be an object-oriented programming style. On the other hand, Jackson can map JSON data we got from API to POJOs in Java. It can decouple the whole project well so that the whole project is well-organized.**
2. **Negative -> Positive: As we can see from above, we have a lot of dependencies for our project. So, in the beginning, we need to use a lot of .jar files to store locally as the dependencies. Then, we found it is very inconvenient as every team member needs to download all .jar files which are horrible. So, after that, we take this thing into consideration and transfer it to a MAVEN project, therefore,**

everyone just needs to import the MAVEN project and all dependencies will be set up automatically in the backend.

3. **Negative -> Positive: All team members in the team have no former experience to deploy a java web project online, so it is a challenge for us. We spend some amounts of time to do research on this. And finally, we did it on AWS, and now it works! The website is**
   www.virtualstockplatform.com
4. **Negative: We used the online API to get a stock price. Some errors sometimes happen because of the limit of the API or the internet connection error. In our program, when these errors happen, the program cannot handle it by a normal way like showing an error message.**