Technical Project Report on

Market Place Application

Name: Keyur Kirti Mehta

Professor: Ryan Rybarczyk

Contents

1. Introduction

Marketplace Application is an e-commerce platform to purchase goods and get it delivered at the door step. This application is hosted on server which makes it accessible to all its user online from anywhere. This application gives user to browse different products based on different categories and will be available for purchase. Also administrator of this application will have authority to control the details of the products.

This document will give detailed technical and general information about the application. Also it will give details about its architecture and its usage. This report will also have screenshot of functionality which will give idea to user about its different modules.

I. Application Functionality

Market Place application will have 2 types of User.

a. Administrator User: One administrator user will be created by default. And these type of users will have following functionalities:
   i. Add other Admin Users
   ii. Add/ Remove Customer users
   iii. Add New products
   iv. Update product description
   v. Delete product
   vi. Browse the products

b. Customer User: This is other type of the user who can purchase the products from the marketplace Application. These kind of user will have following functionalities:
   i. Register for application
   ii. Browse the products
   iii. View Shopping Cart
   iv. Purchase the products

2. Domain Model

Domain model of the application is as per the figure 1. It has one entity as User. User needs to log in to the application to access it. There are 2 types of users in this application. One is Administrator and Customers. Another entity is Product which has all the details about the products like its type, price, description etc. Administrator can add/ update and delete these products. Customers can purchase these products if the desired quantity is available in the stock. Every customer will have its own Shopping Cart where he can add the products which later on can be purchased.

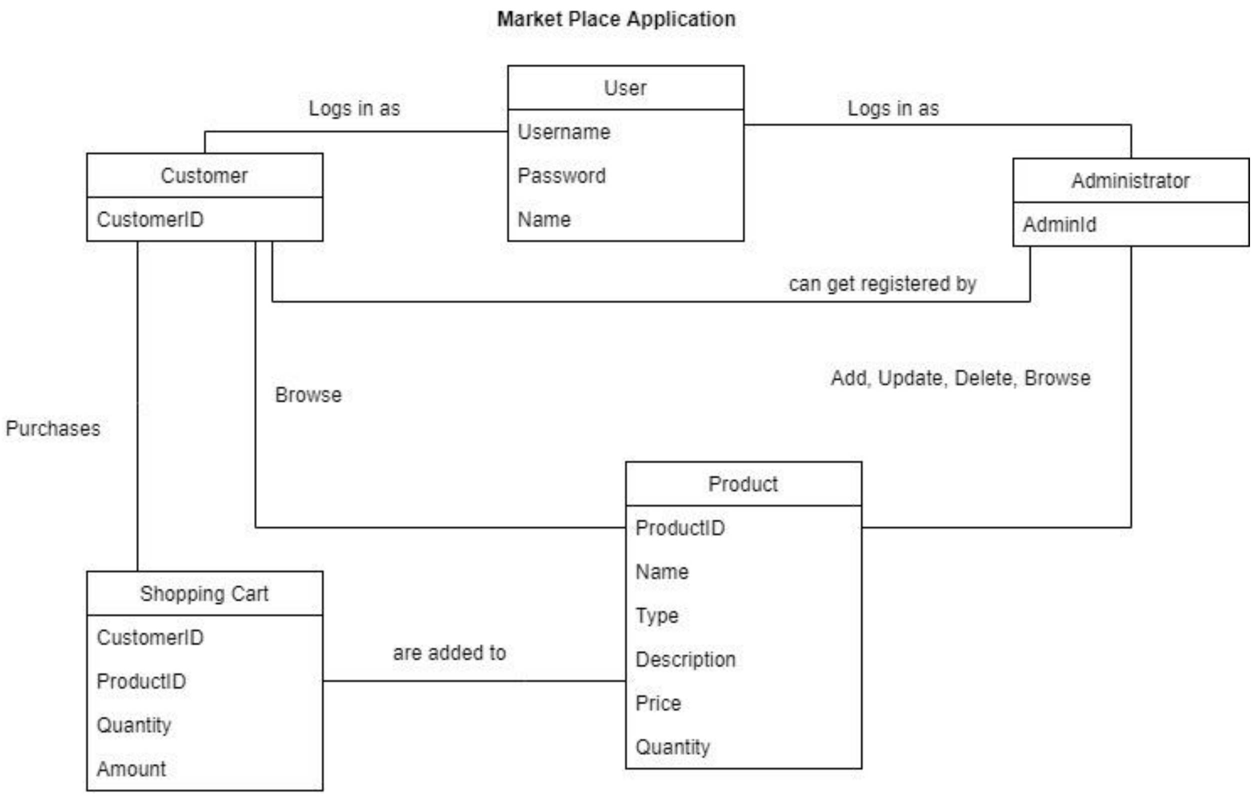


Figure 1 Domain Model

3. Technical Architecture

I. Java RMI

Java RMI is API which is used to communicate between client and server. It uses stub and skeleton framework for the communication. So, basically stub is an object on the client side which initiates the communication with the server. It lookup for the similar string in server as of in the client. Server will rebind with the clients. [2]

Assignment Discussion

In this project, views on the client side will lookup for the connection string on the server. Each view will have different connection string, which will then point to different interfaces on the server side. Server is rebind it with the connection to the client.

II. MVC Architecture

MVC is Model-View-Controller architecture. It is a design pattern used in software designing. This architecture separates the different modules with each other, so as changes in one will not or have little impact on the other. [1]

Model – This component holds to core business logic of the functionality. It processes all the requests received from the views.

View – This is the actual display which user can see. It will have the data along with the information. Views will be different for different type of the users. This can be GUI or the console where user can interact with it.

Controller – It acts as a mediator between the model and views. It takes request from views and send it to the model for processing. And similarly it response (processed request/ result) from server to the client.

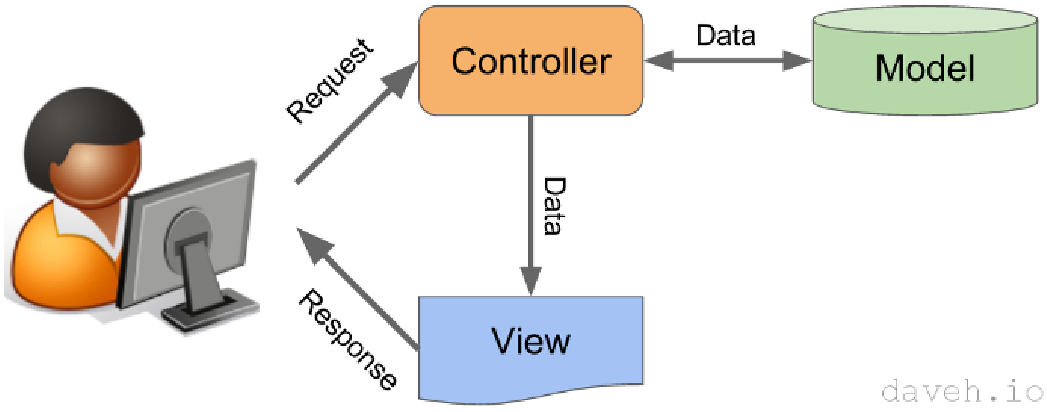


Figure 2 MVC Architecture [4]

Assignment Discussion

In this project, we have MVC architecture in client server environment. The domain model designed is based on the MVC architecture.[3] Where view is on client side and controller and the model are on server side. There can be multiple clients. So, if controller is on client side and is coupled with it, changes in views will result in controller update. So, because of this, the controller is kept on the server side.

Views: Once client starts, user will be displayed with general user view. And there are 2 additional views created which will be displayed based on the type of the customer logged into the application. These are Customer and Administrator views. Any requests from these clients will go to the server through RMI. Views will pass the data (request) through the objects of interfaces. Interfaces will hide the actual information from the client and will provide abstraction layer to the application.

Controller: Controller are designed on the both client and server side. Interfaces for the controllers are implemented on the server side. Any requests (RMI) from views will come to front controller and then forwarded to server controllers where interface methods are implemented from server class where RMI connection is bind. Controller than will forward the request to the models through the objects of models. Each view has its own controller that will handle its request.

Model: Similar to controller model are designed on the server side. And for controller have its own model which will process the request forwarded by the controller. Here request will be processed and the result will be sent back to views through controller. And views will display / take necessary action based on the response.

In this designed all the components are loosely coupled with each other. So, the changes in one will have no or littler effect on the other component. i.e. changes in any 1 particular model will have no effect on other models or the controller or views. Similarly for changes in view will not make model to update its own logic.

4. Designing Patterns

I. Front Controller Pattern

This pattern have 3 component in it whose working is mentioned as below:
Controller: The front controller is the point of contact for handling all the requests. This controller will perform basic authentication and then the request will move forward to the server. This request then processed at the server end. And send back the result the front controller.

Dispatcher: Once the request is received from the server, it will be navigated to the dispatcher. Dispatcher is responsible for navigating the response and display of view. Based on the response received, the dispatcher will render the view and display the corresponding view to the user.

View: Views are the displayed to the user. It will contain all the text and takes the input from users which then will be transferred to the front controller to process and based on the received response displays the new information.

Assignment Discussion

We have implemented the front controller pattern in market place application, where once the application starts user will see the user view. And this view will take username and password and send it to the front controller. Front controller will perform basic validation on it and send it to model via RMI for authentication. And once the response is received, it will display the corresponding view based on the role (Admin or Customer) with the help of dispatcher.

II. Command Pattern

In command pattern, command is passed from client which will perform the desired action. Here the action is performed by other than the client. A request or command is enclosed in an object and then pass to the command Invoker. This command invoker will perform the desired action (method) based on the command received to him. It allows parameterization of methods with different requests. Also can be used to follow the sequence in which commands needs to be executed.

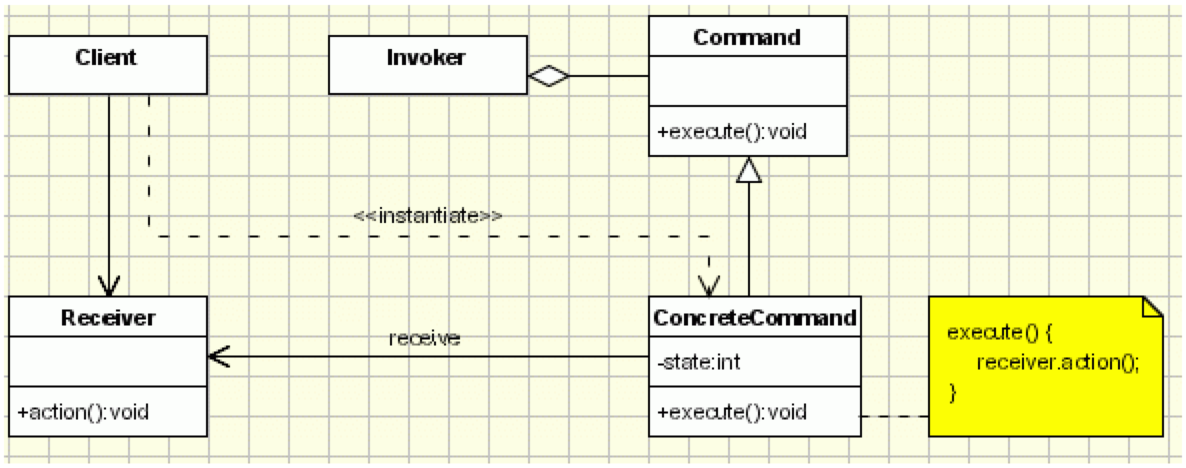


Figure 3 Command Pattern [5]

Assignment Discussion

In this application, I have implemented command pattern on Administrator and Customer View. Where command performed by the admin, customers is sent to command invoker and then based on the received command, invoker will create an object of the class where interface is implemented. One interface is implemented to many different classes which individually performs the operation. I.e. sent the request to server via RMI.

III. Abstract Factory Pattern

Abstract factory pattern will work on factories of factory method. An interface will create a factory of similar related objects. Factory creates the objects for the clients and then the implemented classes will be executed using created objects. Abstract factory will be interface which will create abstract products. And the concrete factories implements the interface.

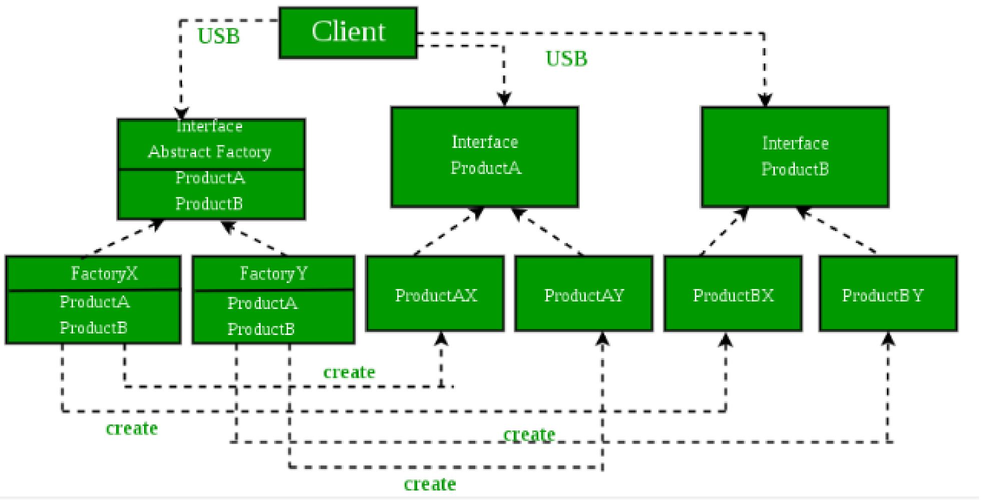


Figure 4 Abstract Factory Pattern [6]

Assignment Discussion

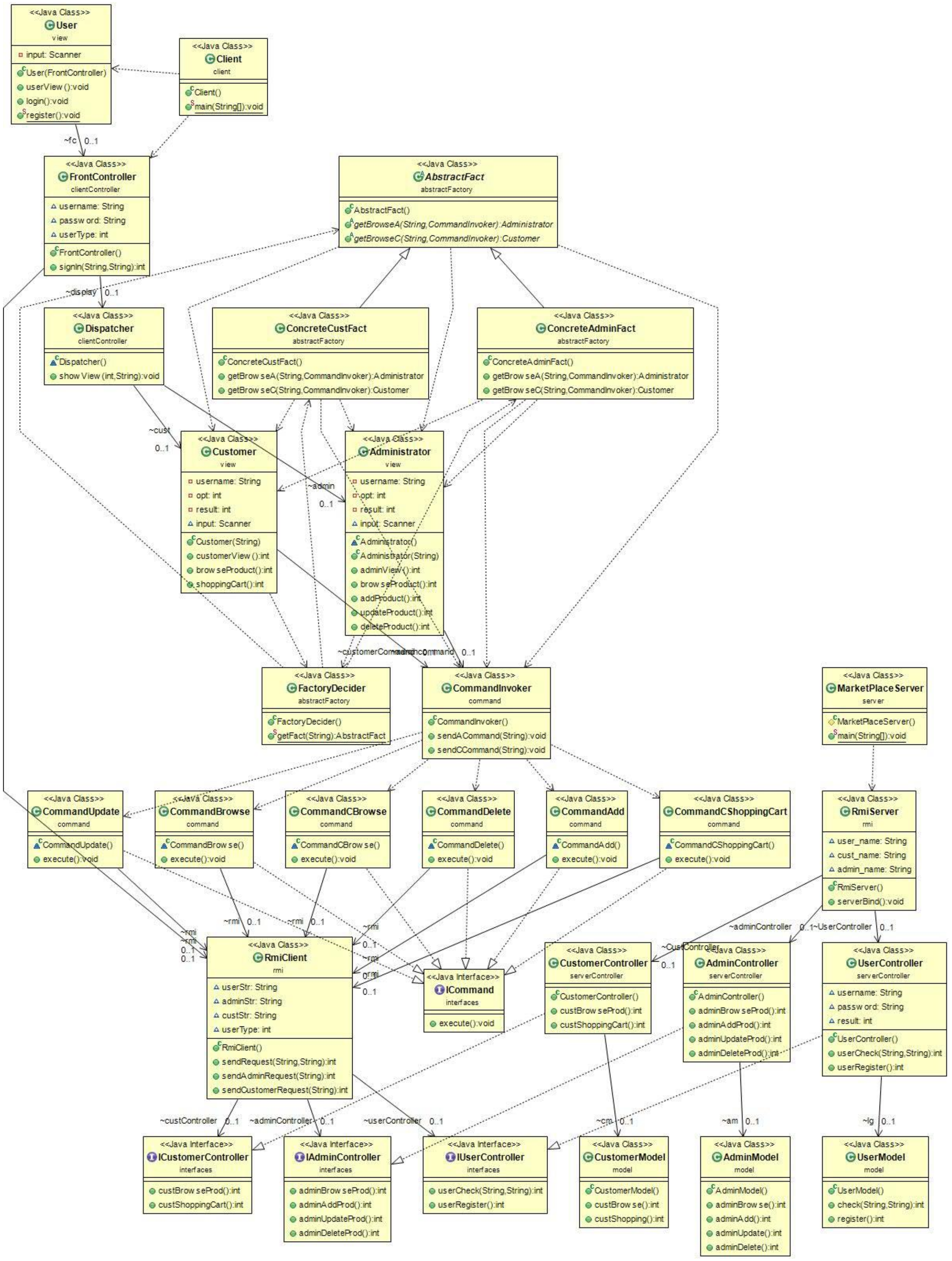
I have created 1 abstract factory which is interface having browse method. And is implemented for 2 concrete factories. Now, there is one factory designer class which is creating the object of any of the concrete factory based on the request. And the corresponding method is executed.

5. UML Diagrams

I. Class Diagram

Figure 5 shows the class diagram created for the market place application. It has one client and server classes which are the entry point of the application on client and server respectively. There are 3 views User, Administrator and Customer view. Based on the roles these views are displayed. Initial request will be sent to front controller from User view and once it is processed, and response will go back to dispatcher class which will display the corresponding view. On a client side command pattern is implemented where classes have implemented the command Interface and that will be called by command Invoker class. Similarly abstract factory class is implemented to 2 concrete factories. Communication between client and server is handled by RMI client and RMI server classes.

On a server side, different controllers are designed for different functionality. I.e. controller for each User, Admin and customer functionality. These controllers class are then pass the requests to the concern model classes which have the actual business logic of the application. Once they perform the business logic return the response back to the views using similar path.



<<Java Class>> User view
input: Scanner
User(FrontController)
userView():void
login():void
register():void
<<Java Class>> Client client
Client()
main(String[]):void
~fc 0..1
<<Java Class>> FrontController clientController
username: String
password: String
userType: int
FrontController()
signIn(String,String):int
~display 0..1
<<Java Class>> Dispatcher clientController
Dispatcher()
showView(int,String):void
<<Java Class>> AbstractFact abstractFactory
AbstractFact()
getBrowseA(String,CommandInvoker):Administrator
getBrowseC(String,CommandInvoker):Customer
<<Java Class>> ConcreteCustFact abstractFactory
ConcreteCustFact()
getBrowseA(String,CommandInvoker):Administrator
getBrowseC(String,CommandInvoker):Customer
<<Java Class>> ConcreteAdminFact abstractFactory
ConcreteAdminFact()
getBrowseA(String,CommandInvoker):Administrator
getBrowseC(String,CommandInvoker):Customer
~cust 0..1
<<Java Class>> Customer view
username: String
opt: int
result: int
input: Scanner
Customer(String)
customerView():int
browseProduct():int
shoppingCart():int
~admin 0..1
<<Java Class>> Administrator view
username: String
opt: int
result: int
input: Scanner
Administrator()
Administrator(String)
adminView():int
browseProduct():int
addProduct():int
updateProduct():int
deleteProduct():int
~customerCommand ~adminCommand 0..1
<<Java Class>> FactoryDecider abstractFactory
FactoryDecider()
getFact(String):AbstractFact
<<Java Class>> CommandInvoker command
CommandInvoker()
sendACommand(String):void
sendCCommand(String):void
<<Java Class>> MarketPlaceServer server
MarketPlaceServer()
main(String[]):void
<<Java Class>> CommandUpdate command
CommandUpdate()
execute():void
<<Java Class>> CommandBrowse command
CommandBrowse()
execute():void
<<Java Class>> CommandCBrowse command
CommandCBrowse()
execute():void
<<Java Class>> CommandDelete command
CommandDelete()
execute():void
<<Java Class>> CommandAdd command
CommandAdd()
execute():void
<<Java Class>> CommandCShoppingCart command
CommandCShoppingCart()
execute():void
<<Java Class>> RmiServer rmi
user_name: String
cust_name: String
admin_name: String
RmiServer()
serverBind():void
~rmi 0..1
~adminController 0..1 ~UserController 0..1
~CustController 0..1
<<Java Class>> RmiClient rmi
userStr: String
adminStr: String
custStr: String
userType: int
RmiClient()
sendRequest(String,String):int
sendAdminRequest(String):int
sendCustomerRequest(String):int
<<Java Interface>> ICommand interfaces
execute():void
<<Java Class>> CustomerController serverController
CustomerController()
custBrowseProd():int
custShoppingCart():int
<<Java Class>> AdminController serverController
AdminController()
adminBrowseProd():int
adminAddProd():int
adminUpdateProd():int
adminDeleteProd():int
<<Java Class>> UserController serverController
username: String
password: String
result: int
UserController()
userCheck(String,String):int
userRegister():int
~custController 0..1
~adminController 0..1
~userController 0..1
~cm 0..1
~am 0..1
~lg 0..1
<<Java Interface>> ICustomerController interfaces
custBrowseProd():int
custShoppingCart():int
<<Java Interface>> IAdminController interfaces
adminBrowseProd():int
adminAddProd():int
adminUpdateProd():int
adminDeleteProd():int
<<Java Interface>> IUserController interfaces
userCheck(String,String):int
userRegister():int
<<Java Class>> CustomerModel model
CustomerModel()
custBrowse():int
custShopping():int
<<Java Class>> AdminModel model
AdminModel()
adminBrowse():int
adminAdd():int
adminUpdate():int
adminDelete():int
<<Java Class>> UserModel model
UserModel()
check(String,String):int
register():int

Figure 5 Class Diagram

6. Assignments Transition

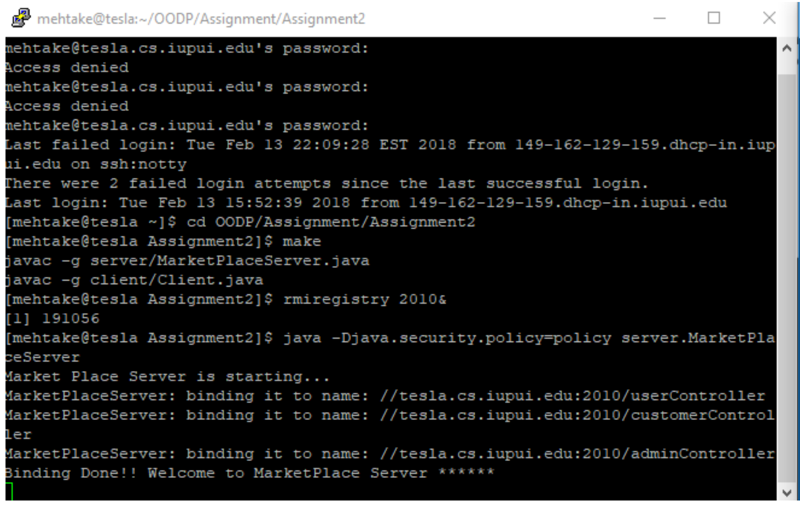
I. Assignment 1 to Assignment 2

Initially, I worked on the comments received in assignment # 1. I need to separate the framework (RMI communication) from the view. Views are displayed to the user, so those cannot have the internal framework of how communication take place between client and server. So, I created new class which will act as RMI controller on both side and will responsible only for communication. And all the requests from views are now forwarded to RMI client through front controller and command pattern. On a server side, controller will receive the request through RMI server and then sent it forward.

Also in this assignment 2, three patterns front controller, command and abstract factory patterns are implemented whose details are mentioned in above sections.

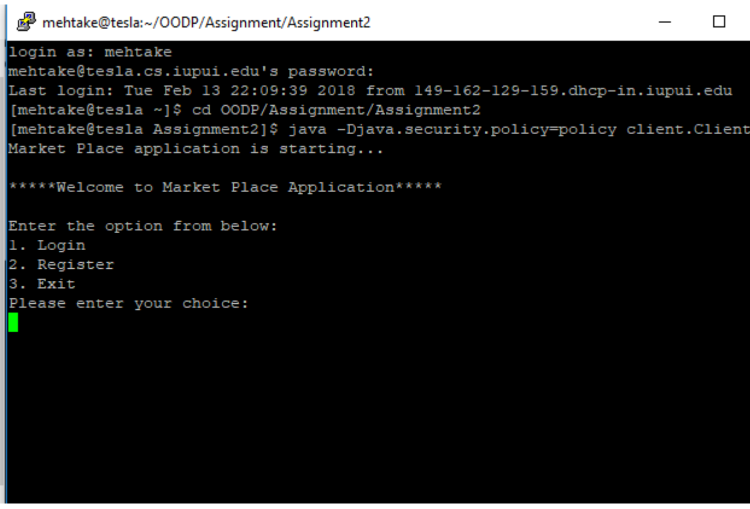
7. Screenshots

I. Server start and Login view



mehtake@tesla:~/OODP/Assignment/Assignment2

```
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Feb 13 15:52:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ make
javac -g server/MarketPlaceServer.java
javac -g client/Client.java
[mehtake@tesla Assignment2]$ rmiregistry 2010&
[1] 191056
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy server.MarketPla
ceServer
Market Place Server is starting...
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/userController
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/customerControl
ler
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/adminController
Binding Done!! Welcome to MarketPlace Server ******
```

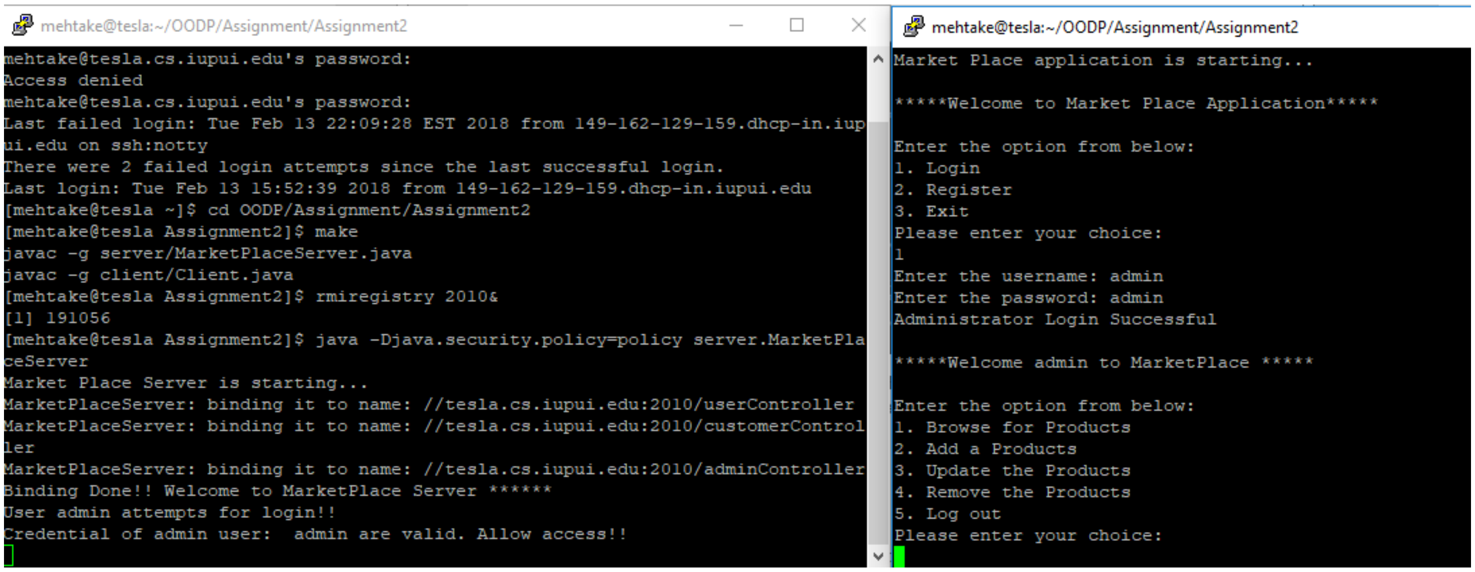


mehtake@tesla:~/OODP/Assignment/Assignment2

```
login as: mehtake
mehtake@tesla.cs.iupui.edu's password:
Last login: Tue Feb 13 22:09:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy client.Client
Market Place application is starting...

*****Welcome to Market Place Application*****

Enter the option from below:
1. Login
2. Register
3. Exit
Please enter your choice:
```

Once the server start, it will bind with the view on the client. And server will display start message. And on the other console, client will display initial user view with option to login and register.

II. Admin Login

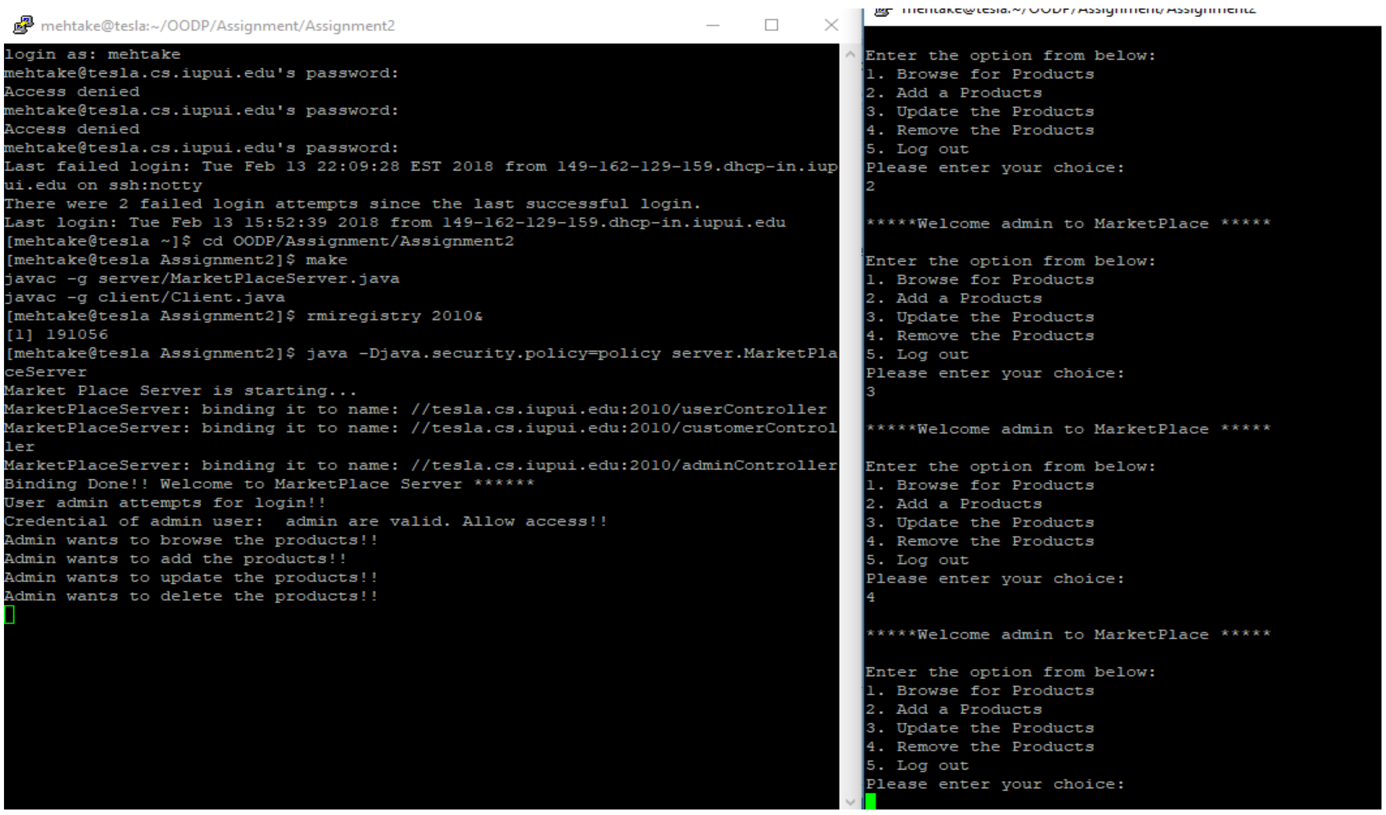


```
mehtake@tesla:~/OODP/Assignment/Assignment2
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Feb 13 15:52:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ make
javac -g server/MarketPlaceServer.java
javac -g client/Client.java
[mehtake@tesla Assignment2]$ rmiregistry 2010&
[1] 191056
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy server.MarketPla
ceServer
Market Place Server is starting...
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/userController
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/customerControl
ler
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/adminController
Binding Done!! Welcome to MarketPlace Server ******
User admin attempts for login!!
Credential of admin user:  admin are valid. Allow access!!
```

```
mehtake@tesla:~/OODP/Assignment/Assignment2
Market Place application is starting...

*****Welcome to Market Place Application*****

Enter the option from below:
1. Login
2. Register
3. Exit
Please enter your choice:
1
Enter the username: admin
Enter the password: admin
Administrator Login Successful

*****Welcome admin to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
```

There are 2 kinds of login provided in this application. One is admin and other is Customer user. On entering valid credentials of admin, it displays the admin view. And on server side, will get message showing access.

III. Admin View

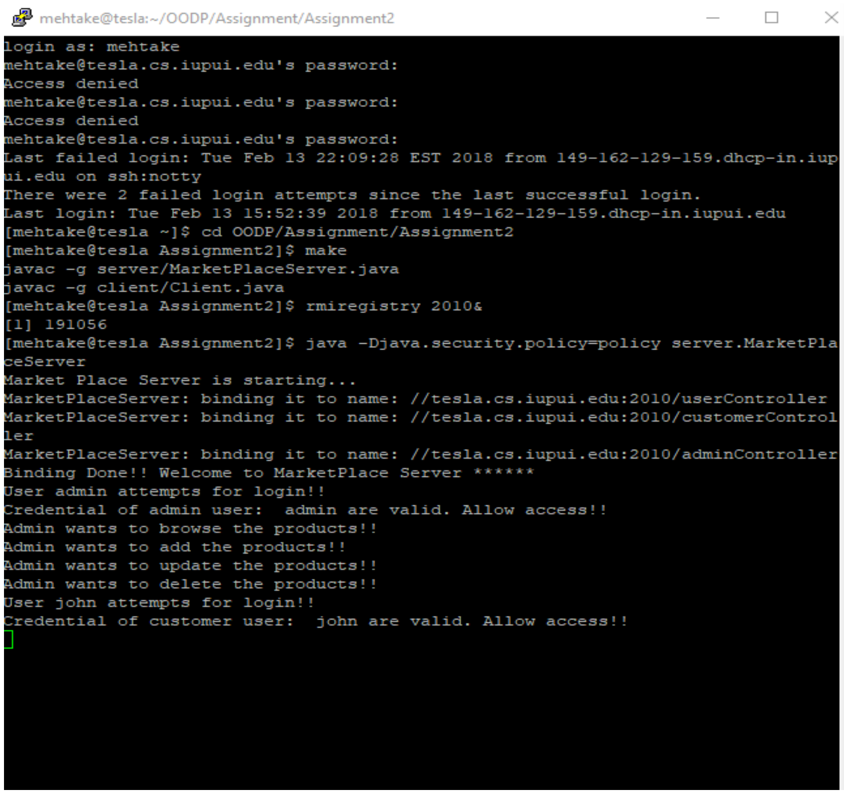


```
mehtake@tesla:~/OODP/Assignment/Assignment2
login as: mehtake
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Feb 13 15:52:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ make
javac -g server/MarketPlaceServer.java
javac -g client/Client.java
[mehtake@tesla Assignment2]$ rmiregistry 2010&
[1] 191056
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy server.MarketPla
ceServer
Market Place Server is starting...
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/userController
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/customerControl
ler
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/adminController
Binding Done!! Welcome to MarketPlace Server ******
User admin attempts for login!!
Credential of admin user:  admin are valid. Allow access!!
Admin wants to browse the products!!
Admin wants to add the products!!
Admin wants to update the products!!
Admin wants to delete the products!!
```

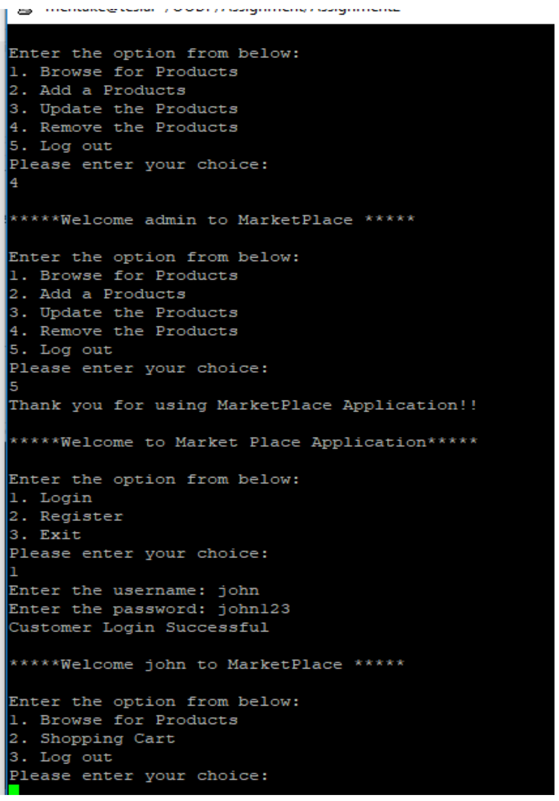
```
mehtake@tesla:~/OODP/Assignment/Assignment2

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
2

*****Welcome admin to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
3

*****Welcome admin to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
4

*****Welcome admin to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
```

Admin view will display all the options of admin have in MarketPlace Application. Based on the selection, model will display respective message. Functionality is not implemented.

IV. Customer Login



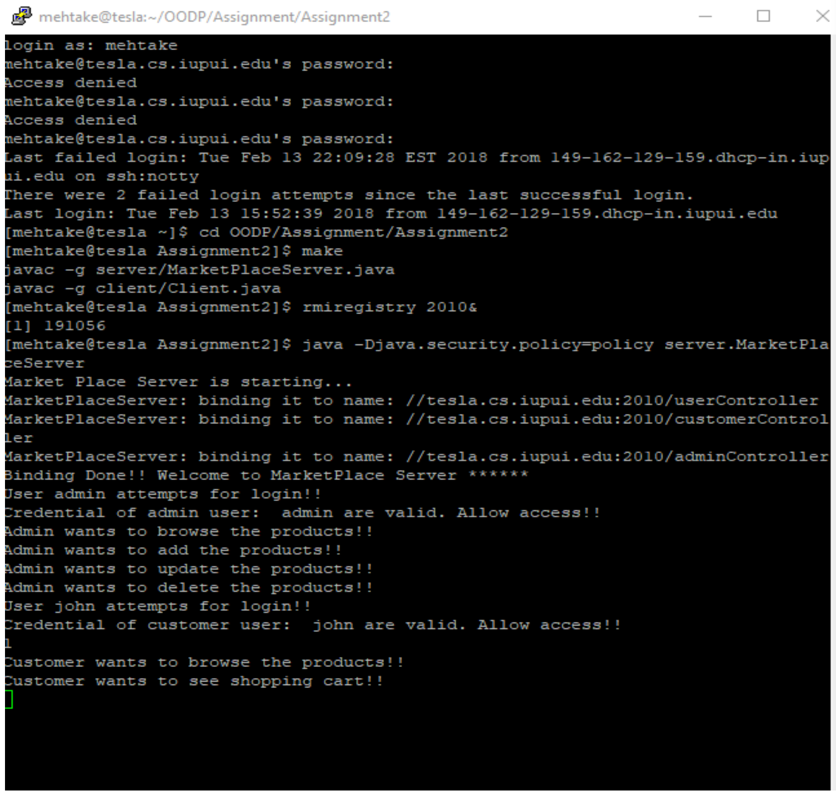
```
login as: mehtake
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Feb 13 15:52:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ make
javac -g server/MarketPlaceServer.java
javac -g client/Client.java
[mehtake@tesla Assignment2]$ rmiregistry 2010&
[1] 191056
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy server.MarketPla
ceServer
Market Place Server is starting...
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/userController
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/customerControl
ler
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/adminController
Binding Done!! Welcome to MarketPlace Server ******
User admin attempts for login!!
Credential of admin user:  admin are valid. Allow access!!
Admin wants to browse the products!!
Admin wants to add the products!!
Admin wants to update the products!!
Admin wants to delete the products!!
User john attempts for login!!
Credential of customer user:  john are valid. Allow access!!
```



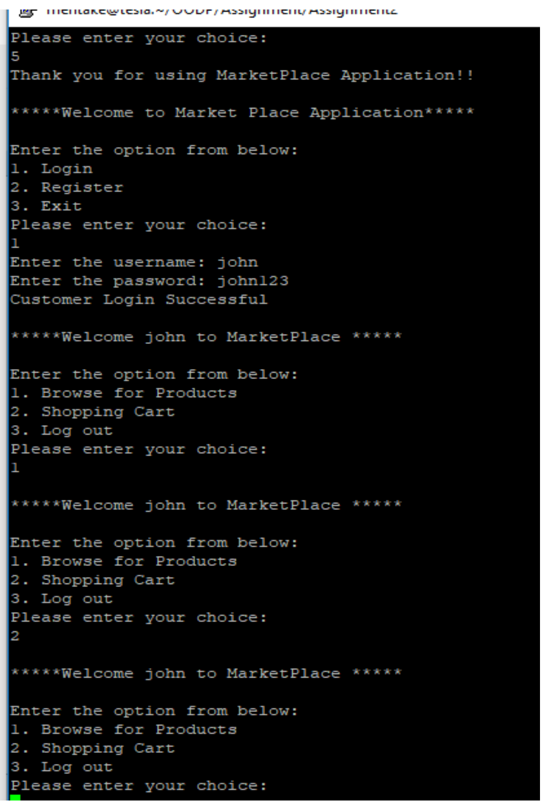
```
Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
4

*****Welcome admin to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Add a Products
3. Update the Products
4. Remove the Products
5. Log out
Please enter your choice:
5
Thank you for using MarketPlace Application!!

*****Welcome to Market Place Application*****

Enter the option from below:
1. Login
2. Register
3. Exit
Please enter your choice:
1
Enter the username: john
Enter the password: john123
Customer Login Successful

*****Welcome john to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Shopping Cart
3. Log out
Please enter your choice:
```

On entering valid credentials of customer, it displays the customer view. And on server side, will get message showing access.

V. Customer View



```
login as: mehtake
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Feb 13 15:52:39 2018 from 149-162-129-159.dhcp-in.iupui.edu
[mehtake@tesla ~]$ cd OODP/Assignment/Assignment2
[mehtake@tesla Assignment2]$ make
javac -g server/MarketPlaceServer.java
javac -g client/Client.java
[mehtake@tesla Assignment2]$ rmiregistry 2010&
[1] 191056
[mehtake@tesla Assignment2]$ java -Djava.security.policy=policy server.MarketPla
ceServer
Market Place Server is starting...
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/userController
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/customerControl
ler
MarketPlaceServer: binding it to name: //tesla.cs.iupui.edu:2010/adminController
Binding Done!! Welcome to MarketPlace Server ******
User admin attempts for login!!
Credential of admin user:  admin are valid. Allow access!!
Admin wants to browse the products!!
Admin wants to add the products!!
Admin wants to update the products!!
Admin wants to delete the products!!
User john attempts for login!!
Credential of customer user:  john are valid. Allow access!!
1
Customer wants to browse the products!!
Customer wants to see shopping cart!!
```



```
Please enter your choice:
5
Thank you for using MarketPlace Application!!

*****Welcome to Market Place Application*****

Enter the option from below:
1. Login
2. Register
3. Exit
Please enter your choice:
1
Enter the username: john
Enter the password: john123
Customer Login Successful

*****Welcome john to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Shopping Cart
3. Log out
Please enter your choice:
1

*****Welcome john to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Shopping Cart
3. Log out
Please enter your choice:
2

*****Welcome john to MarketPlace *****

Enter the option from below:
1. Browse for Products
2. Shopping Cart
3. Log out
Please enter your choice:
```

Customer view will display all the options customers have in MarketPlace Application. Based on the selection, model will display respective message. Functionality is not implemented.

8. Conclusion

The agile methodology of the designing and developing the MarketPlace Application will result in good and flawless design. Also this first iteration created the basic structure of the application using MVC architecture and the communication between different models are setup through RMI.

As a part of assignment 2, I have implemented 3 design pattern which results in good design of the market place application. Also corrected the feedback received from the assignment 1. This assignments made the application independent of different modules creating more layers.

9. References

1. [website] https://developer.chrome.com/apps/app_frameworks
2. [website] https://www.javatpoint.com/RMI
3. [website] https://www.smartdraw.com/class-diagram/
4. [image -website] https://daveh.io/img/the-mvc-pattern.png
5. [image -website] http://www.oodesign.com/command-pattern.html
6. [image –website] https://www.geeksforgeeks.org/abstract-factory-pattern/
7. [website] https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/