Technical Project Report on

Market Place Application

Name: Keyur Kirti Mehta

Professor: Ryan Rybarczyk

Contents

Table of Figures

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.

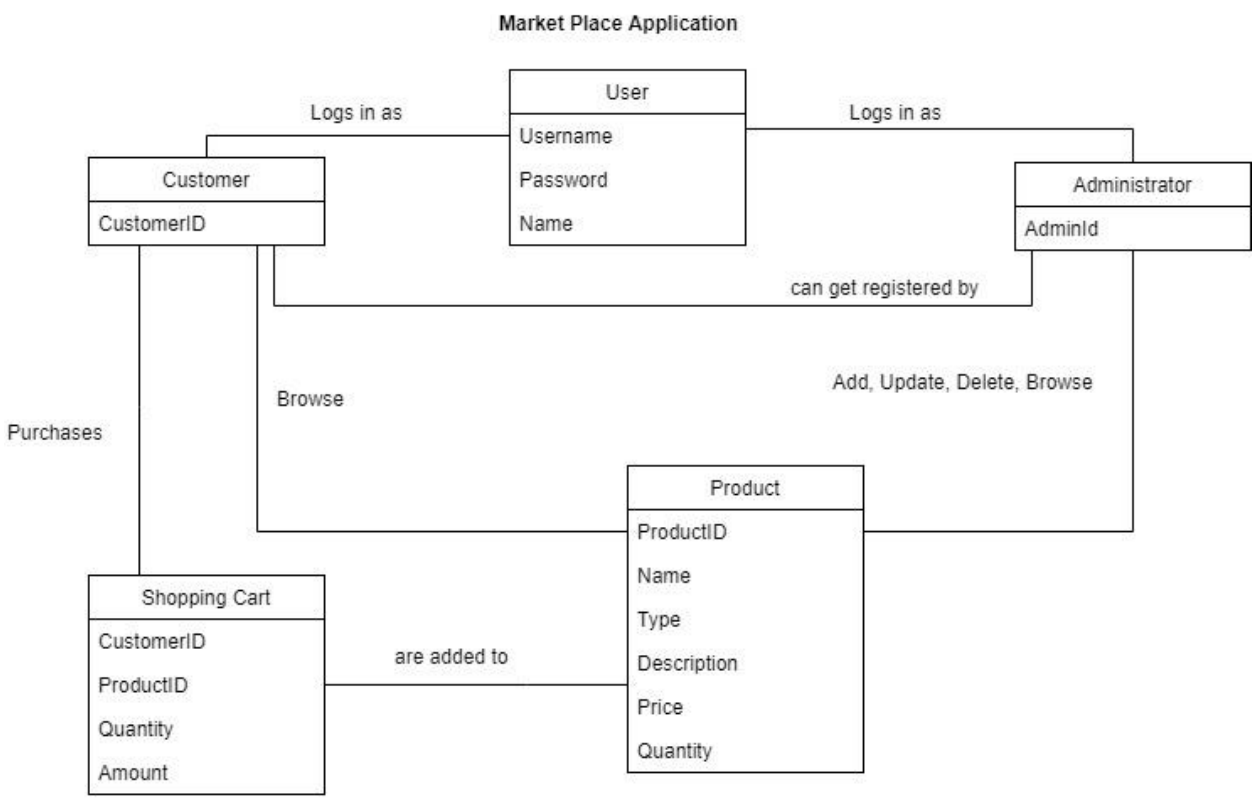


Figure 1 Domain Model

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. [1]

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. [2]

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.

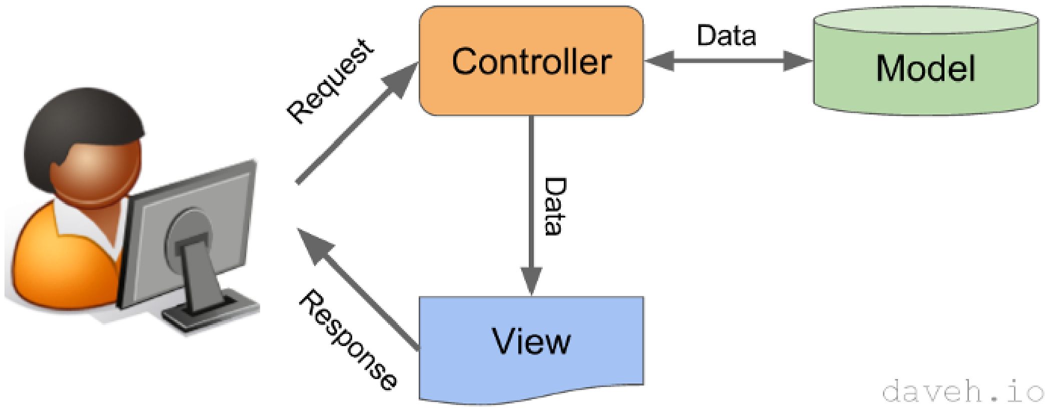


Figure 2 MVC Architecture [3]

Assignment Discussion

In this project, we have MVC architecture in client server environment. The domain model designed is based on the MVC architecture 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.

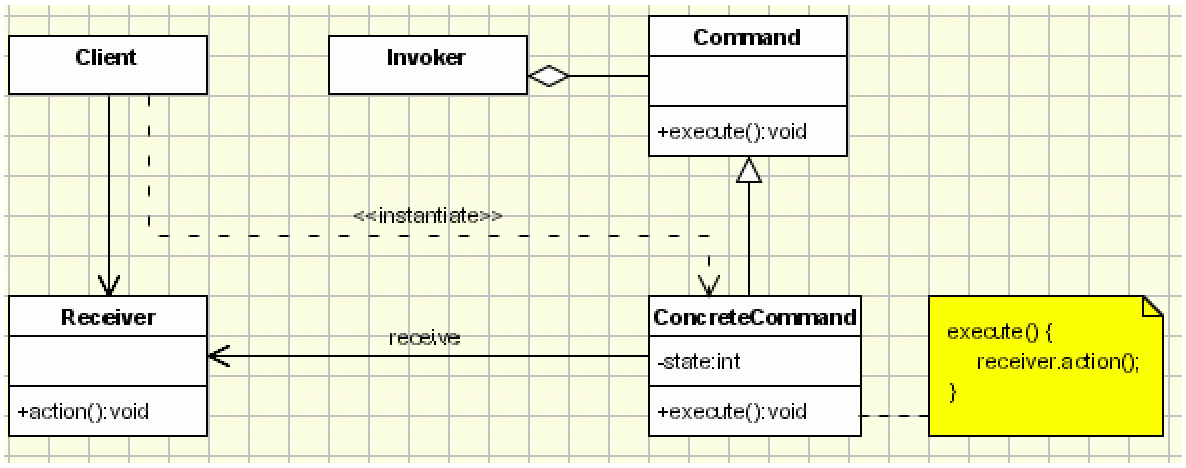


Figure 3 Command Pattern [4]

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.

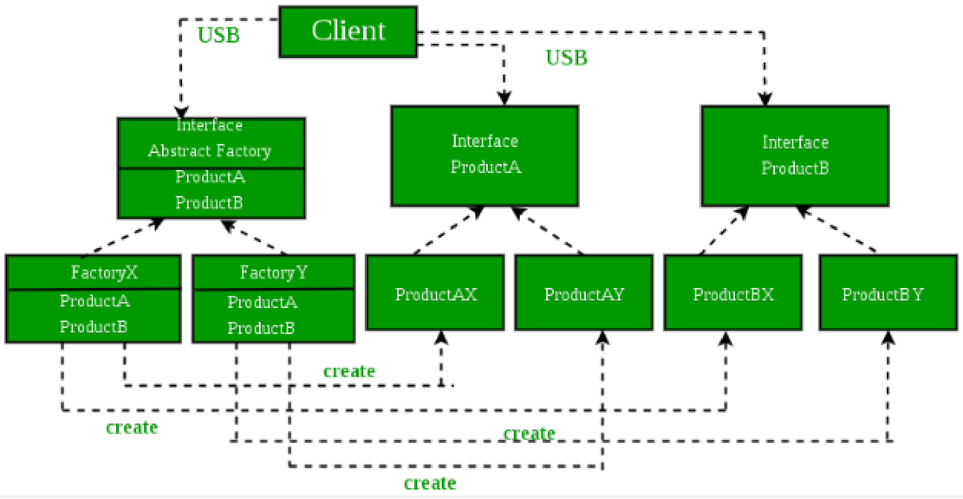


Figure 4 Abstract Factory Pattern [5]

Assignment Discussion

I created 1 abstract factory which has abstract method createView. And this method is override in 2 concrete factories, one for Admin and other for customer. Also one more abstract factory created to display the view. This method is again override in 2 views which will have different menus based on the user roles. Each of the Concrete factory will call respective view.

IV. Authorization Pattern

Authorization is necessary so as only legitimate user can have the access to the application or specific functionality. Once user is authenticated then this user should be able to access the functionality of the application. But implementation of authentication does not provide good security feature to the functionalities. In order to achieve that, we use **Role Based Access Control (RBAC).**

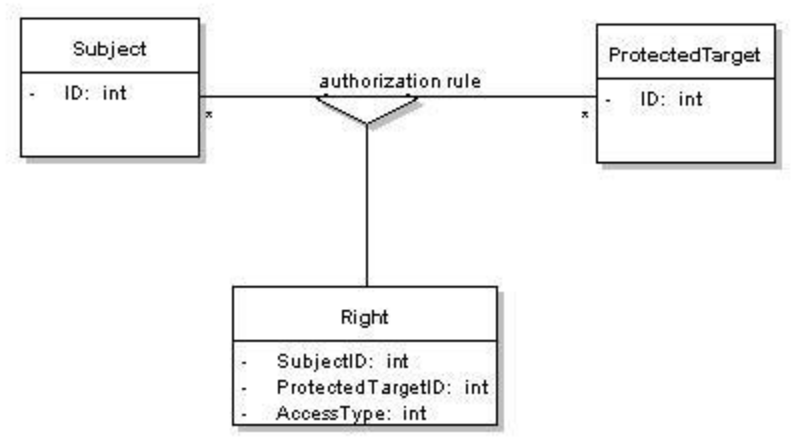


Figure 5 Authorization Pattern [6]

Assignment Discussion

Role Based Access Control (RBAC) through Java Annotation

@RequiresRole("ADMIN")

Each functionality of market place application should not be accessible to the authenticated user. Certain functionality should be available only to certain type of users. This is implemented using Java annotations. We have created custom Java annotation which will define the role type required in order to access the functionality. So, whenever the user object invokes the method, it first checks the user role associated with the objet through annotation. If role is matched then only the object access the method.

V. Proxy Pattern

In proxy pattern, one class represent the functionality of the other class. An object of the class is instantiate when the object of the original class is invoked. The proxy class will handles the functionality of other class. Instead of calling the method of the original class, first proxy class method will be called and executed.

Assignment Discussion

In our assignment, Role Based Access Control is implemented using Java annotation and proxy pattern. When client sends the request to controller to access the method, it will redirect it to proxy class which will check the role assigned to the method (annotation). If user role matches with the role required in annotation, then the corresponding method is called.

VI. Reflection Pattern

Sometimes we must issue a request to an object without knowing much about that object. In this type of scenario, Reflection Pattern comes to the picture. In Reflection pattern, we create a reflection image of the class and we can then using the object, we access the information of that class.

Assignment Discussion

In our assignment, Reflection pattern is used in the implementation of the proxy pattern where invocationHandler where it gets the details of the annotation used in class.

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 controller classes 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.

Views are designed using Abstract Factory pattern where dispatcher will create the object of the concrete factories (Admin or customer) based on the role received to him from server. And then subsequent admin or customer view will be displayed to the user.

When user is authenticated, a new session object is created for him which contains the user role. This object is pass through each transaction carried out by the user. Role based access control is provided in this application using the user role. Whenever admin or customer user asks for any operation, before executing the concern request,   AuthorizationInvocationHandler class will check the user role is matching with the access required to perform that operation. For this we have used custom annotation (RequiresRole). This checking is carried out using proxy object on the server side.

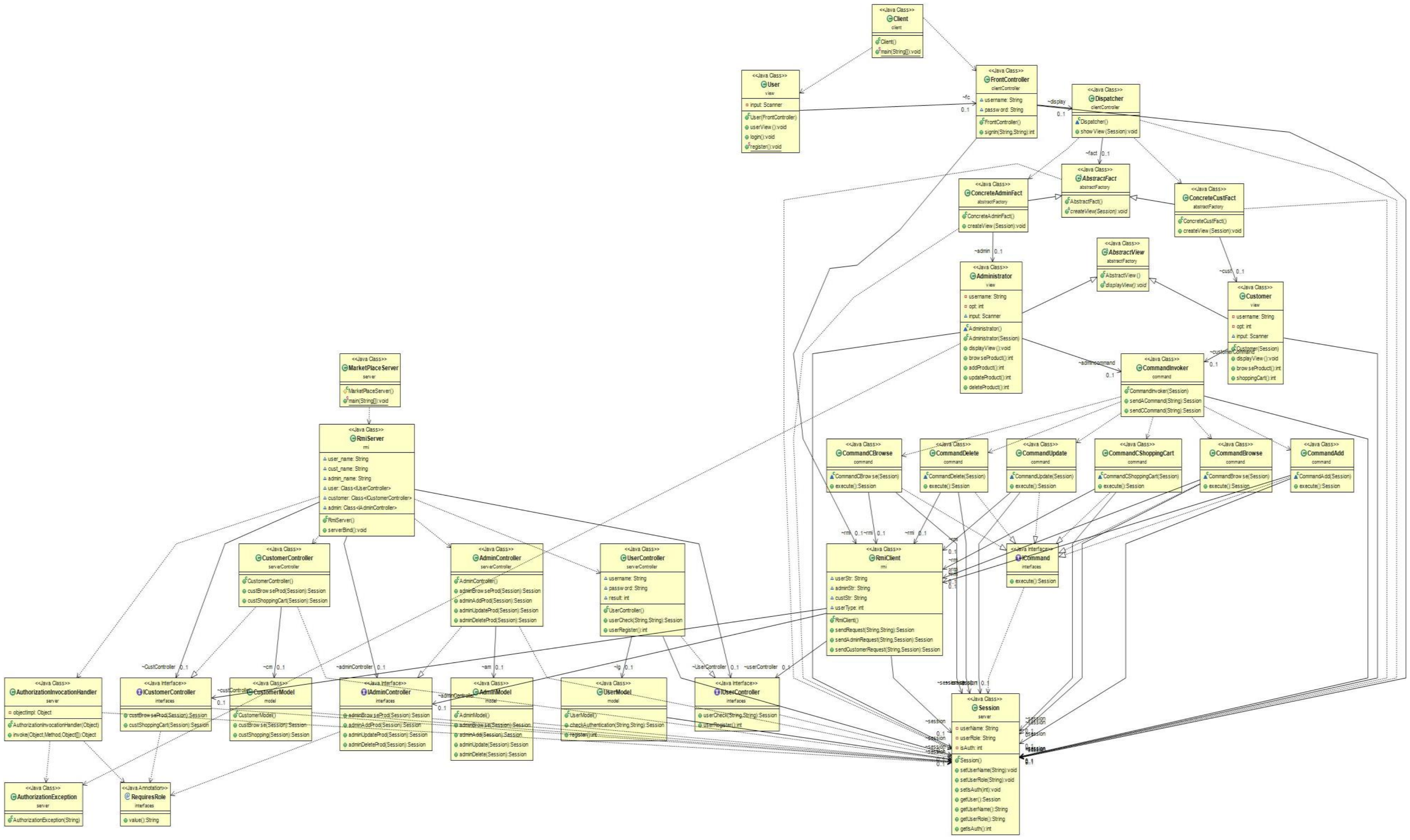


Figure 6 Class Diagram

II. Sequence Diagram

1. Login Activity Sequence

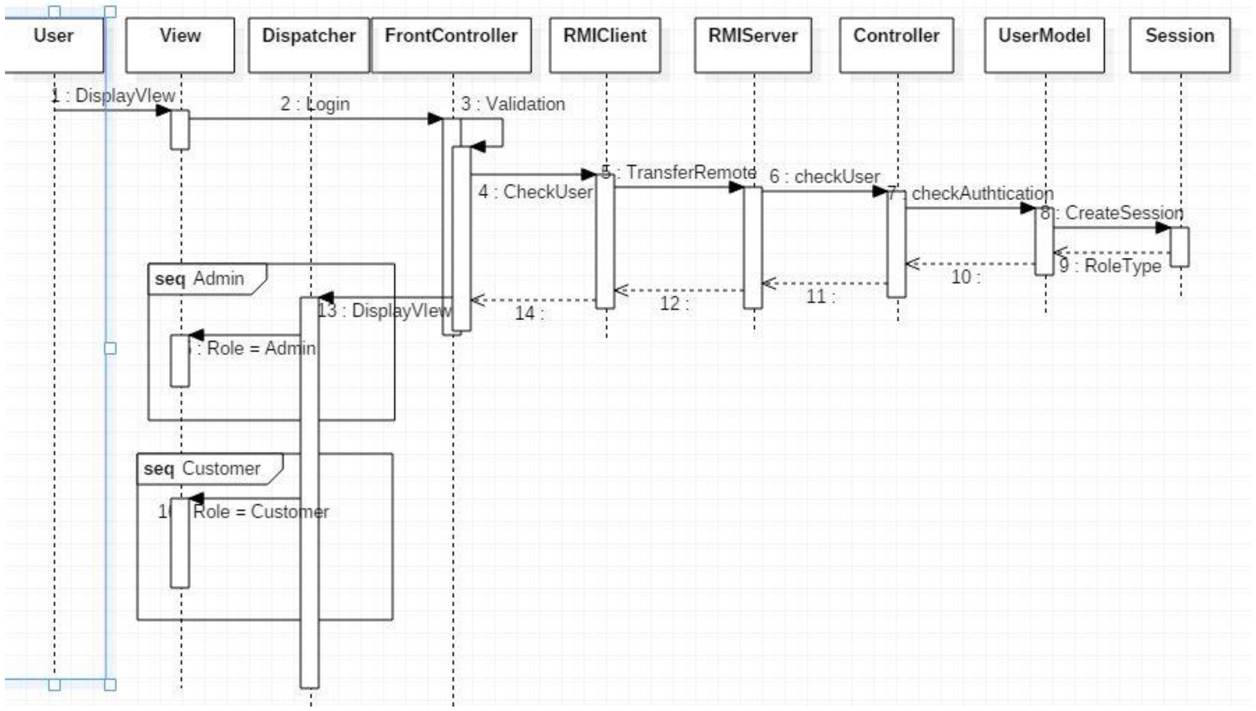


Figure 7 Login Sequence Diagram

Whenever user wants to log in to the application, his credentials are will go to front controller and then sent to server through RMI. On server, Usermodel will validate the credentials and if authenticated then creates a new session for the user along with its user role. This session object is then passed back to the front controller on client and based on the user role dispatcher will show the respective view to the user.

2. Admin/ Customer Command Activity Sequence

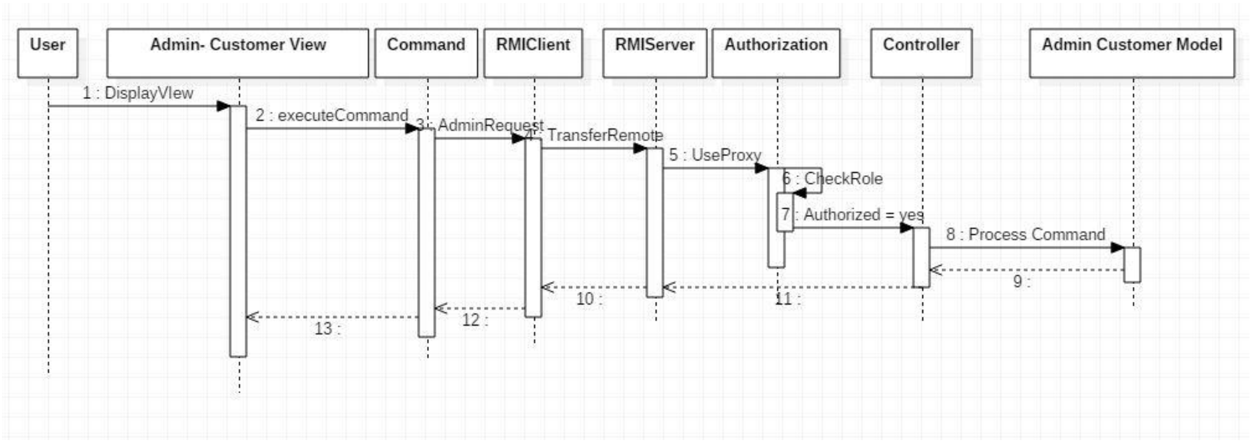


Figure 8 Admin/ Customer Sequence Diagram

This sequence diagram shows the sequence flow for both Admin and Customer's any request operation. When customer or Admin view request for any operation, then it is send as command to invoker and based on the command, it will call the desired method and sent the request to the server through RMI. Then on server side, whether the user has desired role to perform the requested operation is checked through proxy pattern. If he is authorized then the respective method is call and response is sent back to the client view.

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.

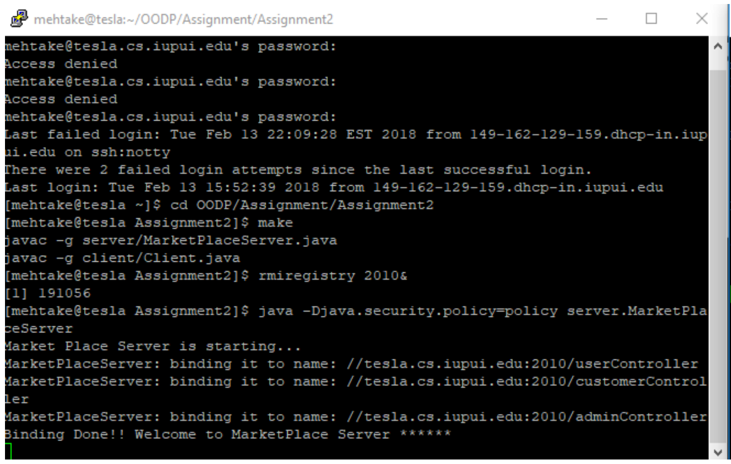
II. Assignment 2 to Assignment 3

As I received the comment on abstract factory pattern, I redesigned and implemented the Abstract Factory Pattern where each of the concrete factory will call the respective view based on the user role. And concrete factories will override the method of abstract factory. The detail explanation is mentioned in the section 4.3

As a part of this assignment, I have implemented Authorization pattern with Role Based Access Control which is implemented using Java annotation. Proxy and Reflection pattern is also implemented in the assignment to check the user has desired role to access the particular functionality. Proxy object will invoke and check the user role whenever admin/ customers operation is called by their respective objects. Annotation name and whether it's present or not on those operation is obtained by using the reflection pattern. Sessions objects are created for each successful login of the user. This session objects will be pass through each function call done by that particular user. Session objects holds the username, its role and isAuthenticate information. This information is used to check the RBAC and display desired view. In future, session object will also be used to track the transaction carried out by that particular user during that login session.

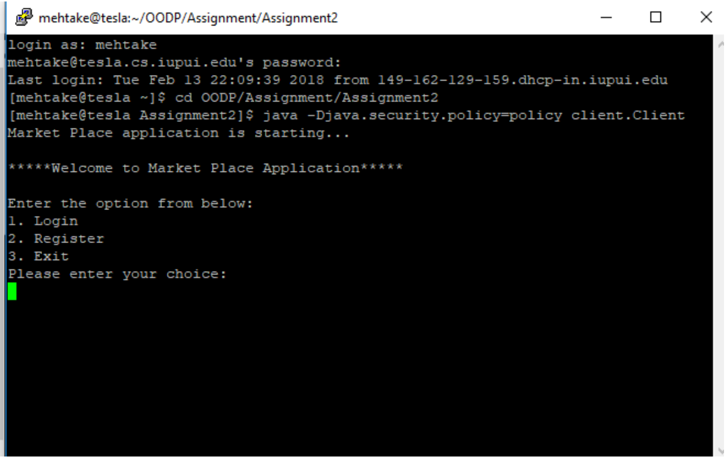
7. Sample Runs

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 ******
```

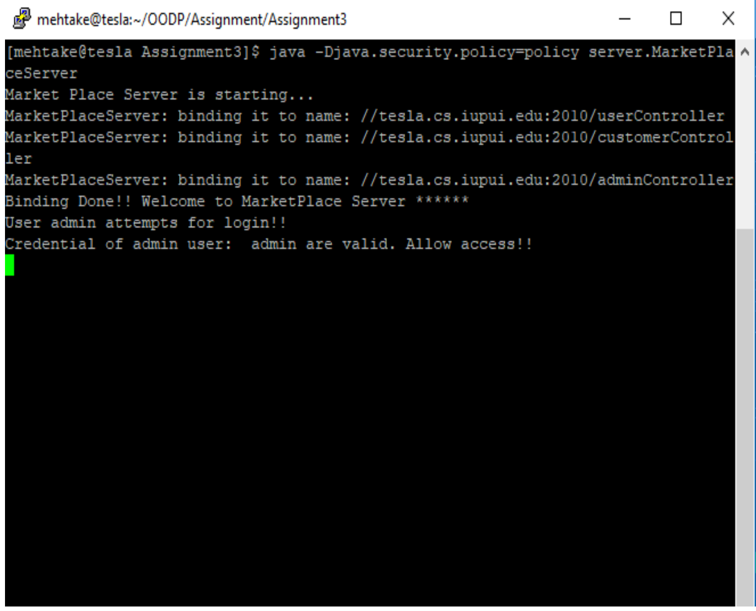


mehtake@tesla:~/OODP/Assignment/Assignment2

```
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:
```

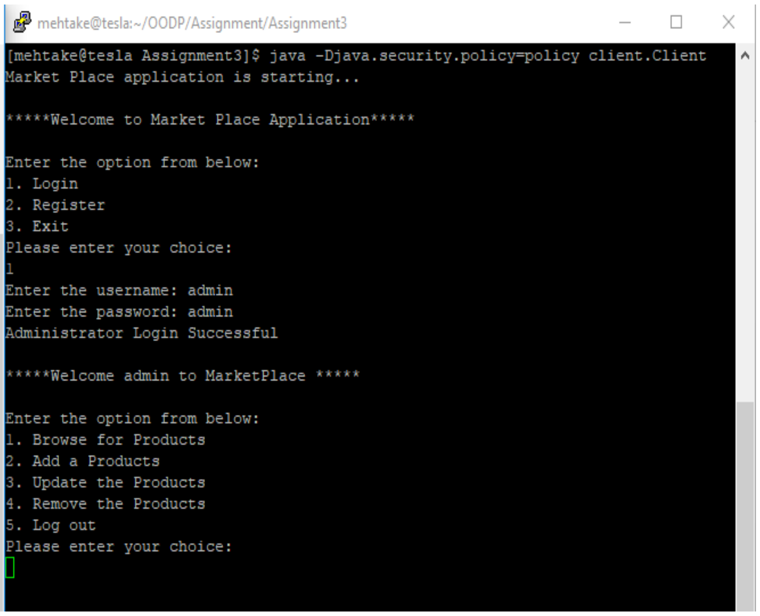
Once the server start, it will bind with the view on the client. And server will display start message. And on the other console, client will display initial user view with option to login and register.

II. Admin Login



mehtake@tesla:~/OODP/Assignment/Assignment3

```
[mehtake@tesla Assignment3]$ 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!!
```

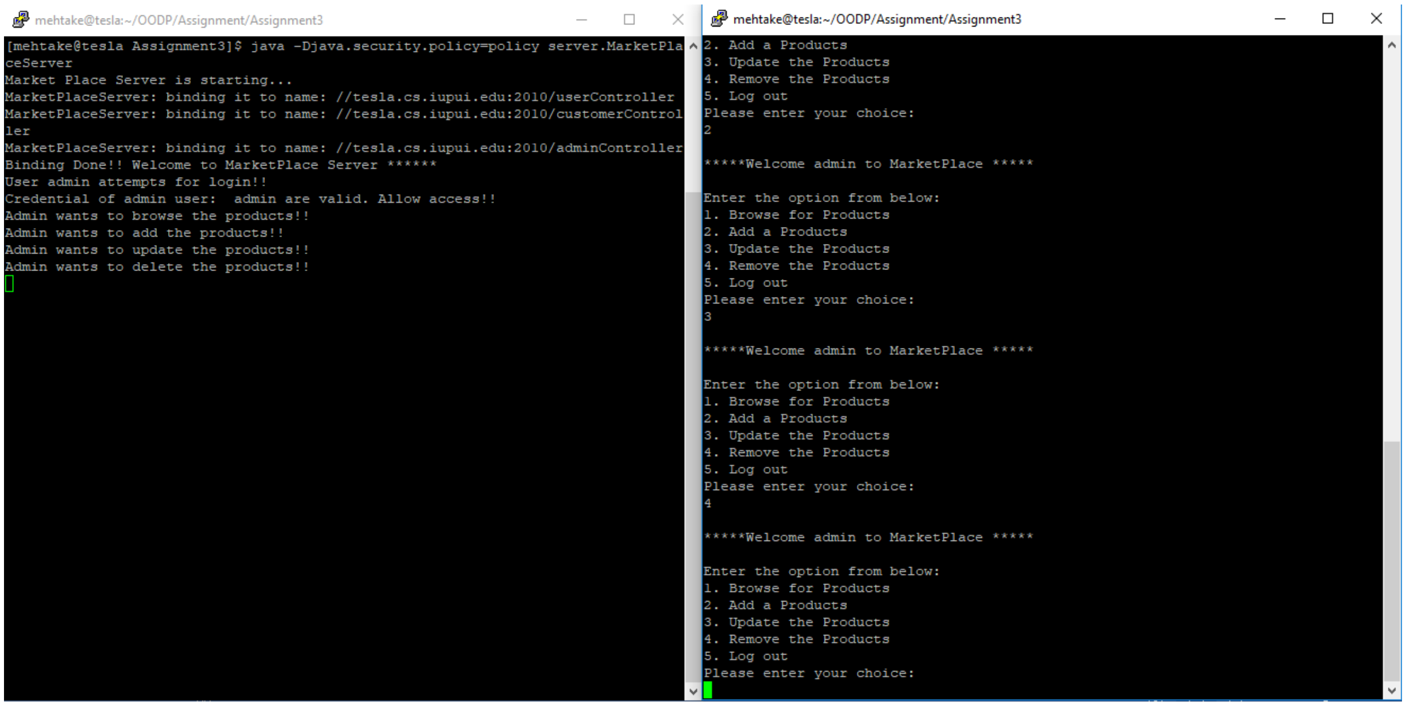


mehtake@tesla:~/OODP/Assignment/Assignment3

```
[mehtake@tesla Assignment3]$ 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:
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

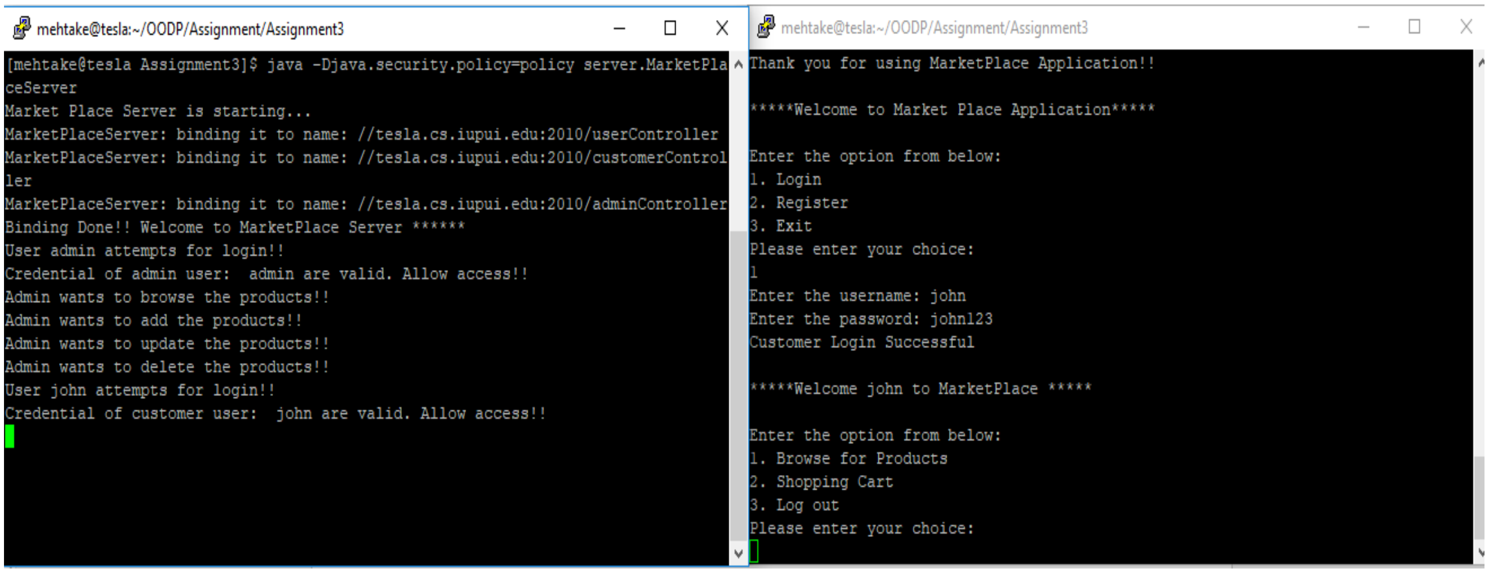

mehtake@tesla:~/OODP/Assignment/Assignment3
mehtake@tesla:~/OODP/Assignment/Assignment3

```
[mehtake@tesla Assignment3]$ 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!!
```

```
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 yet.

IV. Customer Login

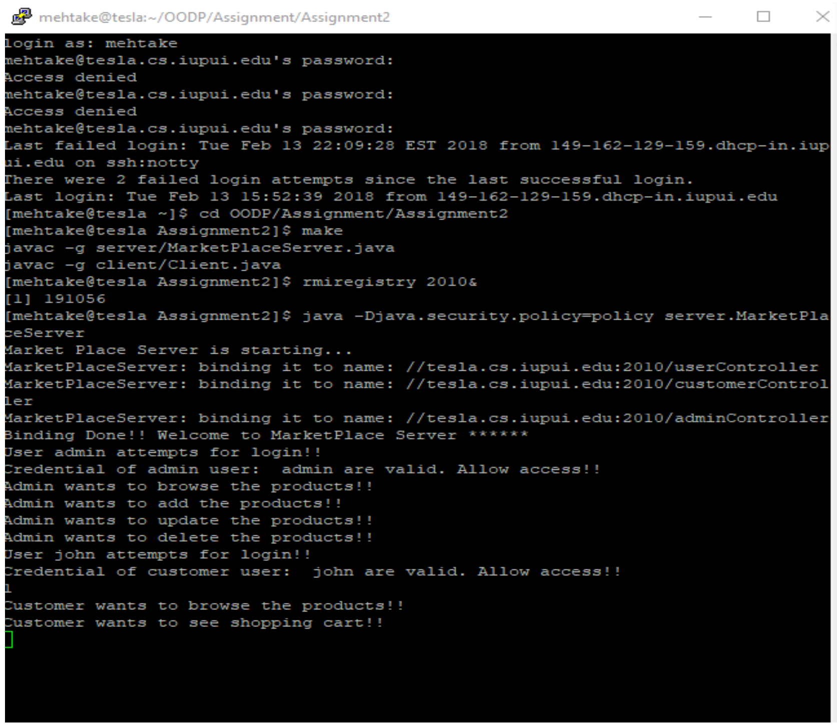

mehtake@tesla:~/OODP/Assignment/Assignment3
mehtake@tesla:~/OODP/Assignment/Assignment3

```
[mehtake@tesla Assignment3]$ 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!!
```

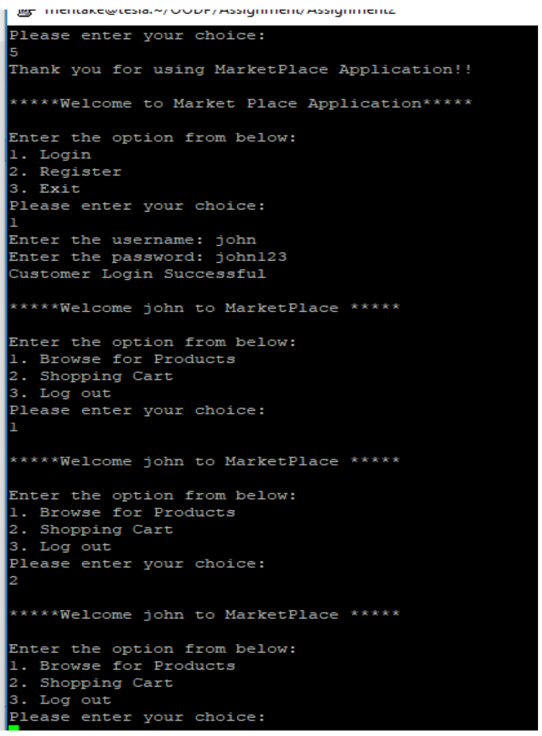
```
Thank you for using MarketPlace Application!!

*****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



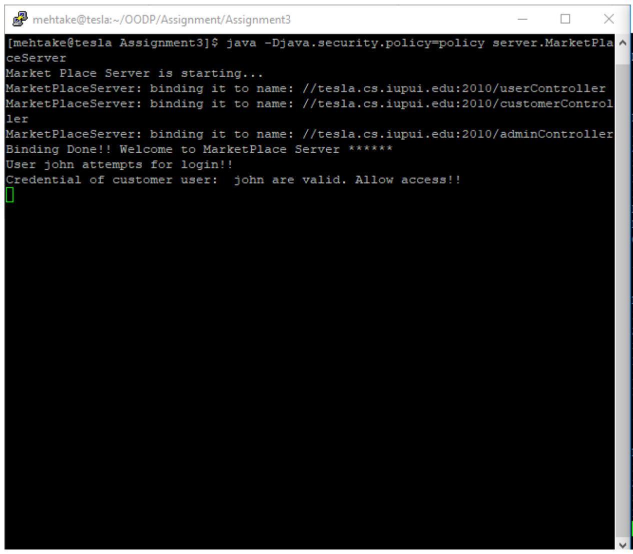
```
login as: mehtake
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Access denied
mehtake@tesla.cs.iupui.edu's password:
Last failed login: Tue Feb 13 22:09:28 EST 2018 from 149-162-129-159.dhcp-in.iup
ui.edu on ssh:notty
There were 2 failed login attempts 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!!
```



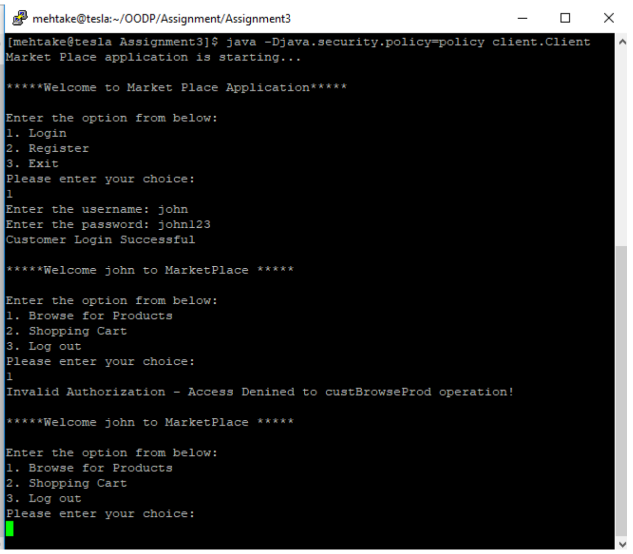
```
Please enter your choice:
5
Thank you for using MarketPlace Application!!

*****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.

VI. Role Based Access Control



```
[mehtake@tesla Assignment3]$ 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 john attempts for login!!
Credential of customer user:  john are valid. Allow access!!
```



```
[mehtake@tesla Assignment3]$ 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:
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
Invalid Authorization - Access Denied to custBrowseProd operation!

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

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

Currently, I have separate admin and customer view which are accessed by admin and customer users only. So, ideally authorization error won't occur. But just to display in sample run, I have hard coded the ADMIN role in customer user. And when the operation of customer user are invoked, it throws user defined Authorization exception (RBAC) through proxy object.

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.

In assignment 3, we implemented authorization along with Role based Access Control, reflection and proxy pattern in our market place application. Also we created session objects which will be used in all transaction carried out by the user during his 1 login session.

References

[1] [Online]. Available: https://www.javatpoint.com/RMI.

[2] [Online]. Available: https://developer.chrome.com/apps/app_frameworks.

[3] [Online]. Available: https://daveh.io/img/the-mvc-pattern.png.

[4] [Online]. Available: http://www.oodesign.com/command-pattern.html.

[5] [Online]. Available: https://www.geeksforgeeks.org/abstract-factory-pattern/.

[6] [Online]. Available: http://www.cnblogs.com/callwangxiang/archive/2009/02/15/1391043.html.