

---

---

# ONLINE MARKETPLACE APPLICATION

---

---

## ASSIGNMENT-6

### FINAL PROJECT REPORT

#### IMPLEMENTING FULL APP FUNCTIONALITY & DATABASE ACCESS LAYER

UNDER GUIDANCE OF

Dr. RYAN RYBARCZYK

BY

ADITHYA MORAMPUDI

## Table of Contents

<b>ASSIGNMENT 6 REPORT .....</b>	<b>3</b>
<b>Assignment Overviews.....</b>	<b>3</b>
Assignment#1 Overview .....	3
Assignment#2 Overview .....	4
Assignment#3 Overview .....	5
Assignment#4 Overview .....	7
Assignment#5 Overview .....	9
<b>Assignment#6 Discussion .....</b>	<b>9</b>
Database Access Layer .....	9
<b>Sample Runs of completed Application .....</b>	<b>10</b>
<b>Conclusion .....</b>	<b>15</b>
<b>References .....</b>	<b>15</b>

# ASSIGNMENT 6 REPORT

## Assignment Overviews

### Assignment#1 Overview

In the first assignment, our main moto was to identify the entities present in the system and to get the **RMI** up and running. I have implemented the **MVC** pattern in the first assignment, where my Model was simple class containing the login access and registration access, the controller was basically to start the application. I had only one controller at that point which was on the server side, and the view had simple login requirements and that was pretty much it for the first assignment. The main problem I had in that assignment was proper implementation of the MVC pattern. In the next assignments, I added a controller on the Client side as well to separate out the concerns between the view and Controller. It was now, I have experimented a lot with the RMI registry, how the discovery service works, how Java RMI abstracts out the underlying complexity of writing socket programming and how it helps us in providing a multithreaded environment. By implementing MVC it helped me to understand the importance of reusability and adaptability in a real-time application.

My class diagram in the first assignment looked like this.

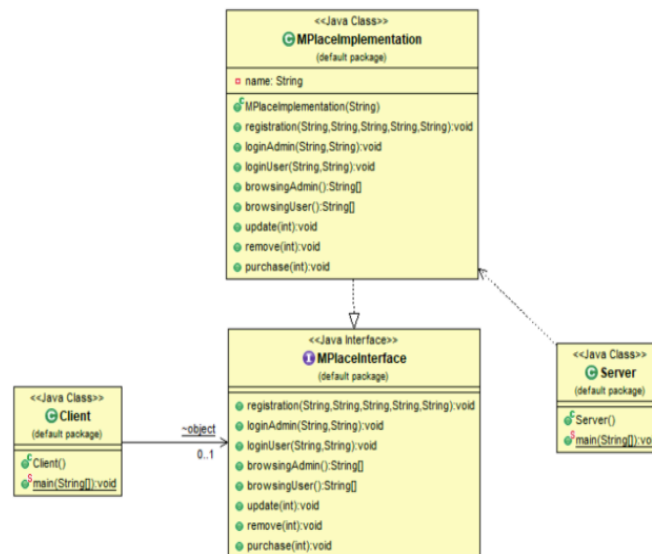


Fig [1] – Initial class Diagram

Again, this wasn't perfect, so I had to change the entire design in next assignment.

## Assignment#2 Overview

In Assignment#2, the main aim was to implement three patterns which were Abstract factory, Front Controller and Command pattern and I also partially implemented the login functionality in this assignment.

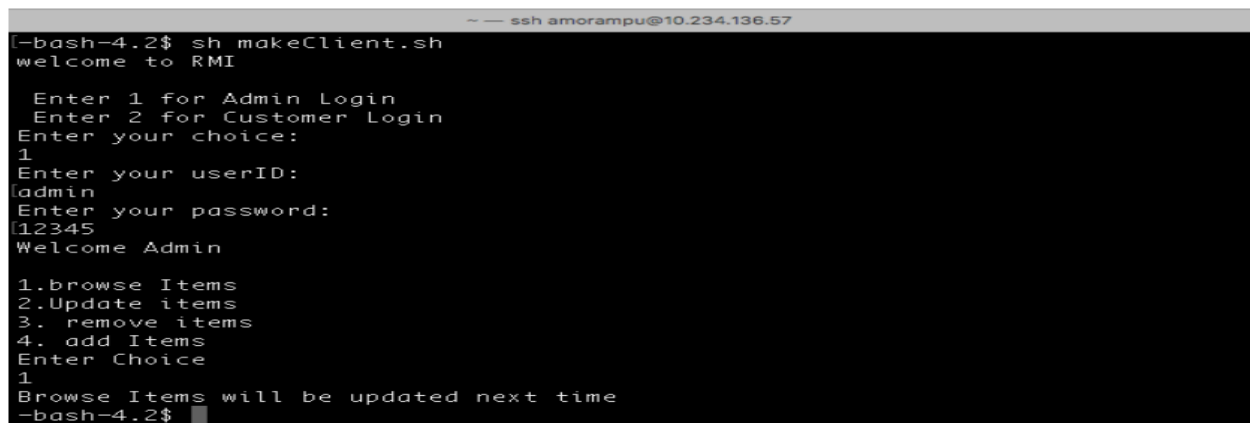
I implemented the **Front controller** pattern at the client side, which served as the main entry into the application, every user must pass through this point to enter the application, this had two consequences, the positive consequence being **centralized control**, -ve being a **bottleneck** of single entry point, if that point is compromised then there is no way to stop an attacker from accessing sensitive information.

The Second Pattern being the **Abstract Factory pattern**, I implemented this pattern on the client side as well, whenever a user logs into the system, based on his role (Administrator or Customer), we display the corresponding view to the user. The Dispatcher of the Front Controller pattern is responsible for calling the Abstract Factory pattern. The Abstract Factory then displays all this information based on the role of the User.

The Third Pattern was **Command pattern** which was implemented on the client side as well, with the help of this pattern, I could take the commands from the user and then pass them onto the server side.

After this Assignment, I got a clear sense of abstraction and access control in a real-world application. Since I haven't properly implemented the MVC pattern in the first assignment, I have done it in this assignment and this helped me in furthering the application.

This is how the skeleton looked in second Assignment



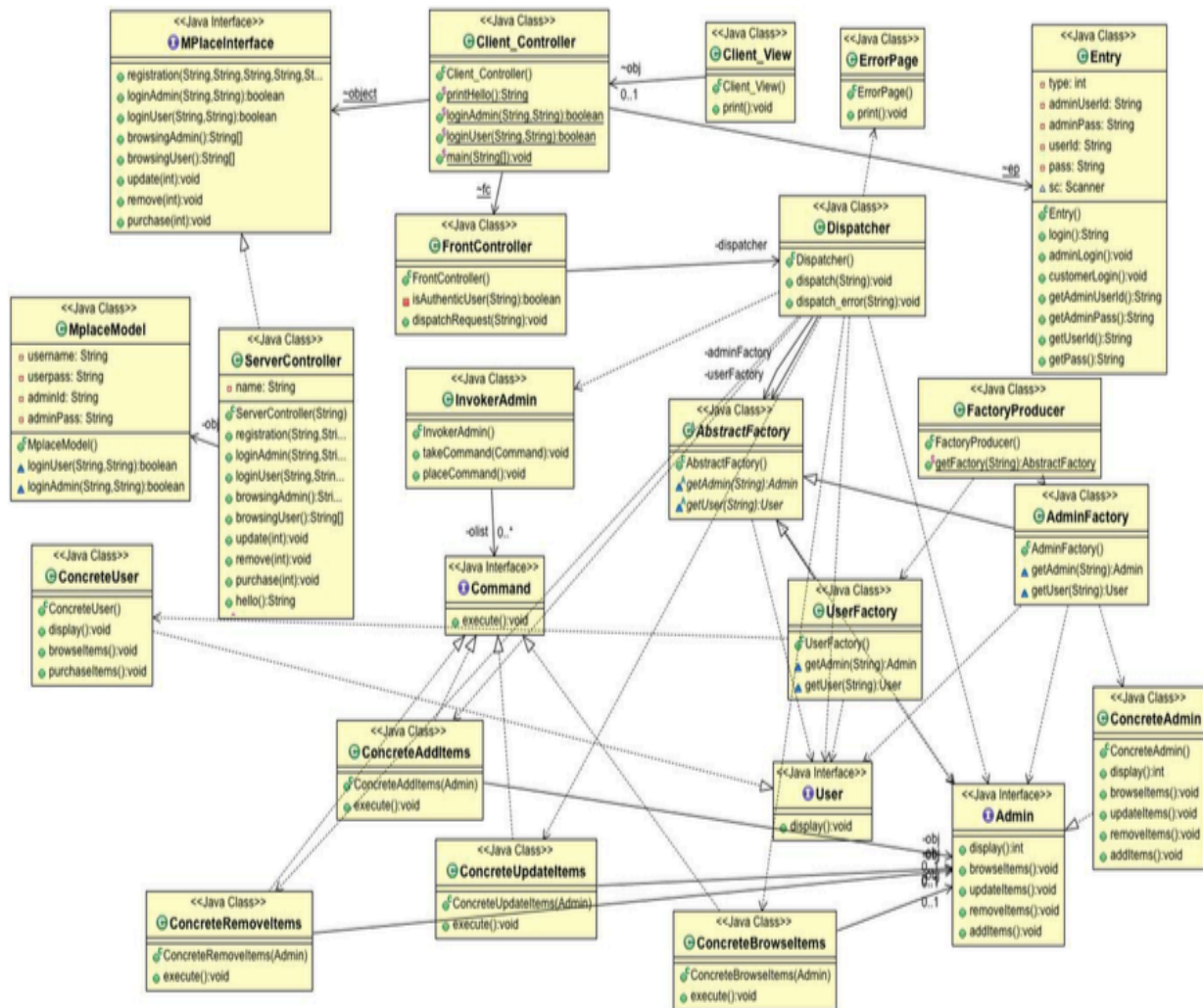
```
[~bash-4.2$] sh makeClient.sh
welcome to RMI

Enter 1 for Admin Login
Enter 2 for Customer Login
Enter your choice:
1
Enter your userID:
admin
Enter your password:
12345
Welcome Admin

1.browse Items
2.Update items
3. remove items
4. add Items
Enter Choice
1
Browse Items will be updated next time
[~bash-4.2$]
```

Fig [2]-Skeleton of the application

In third Assignment, I implemented the **Role Based Access Control** with the help of Java **Annotations, Proxy pattern, Authorization Pattern** and **Reflection**. These are implemented on the Server side.



In third Assignment, I implemented the **Role Based Access Control** with the help of Java **Annotations, Proxy pattern, Authorization Pattern** and **Reflection**. These are implemented on the Server side.

I started the Assignment by creating an **Annotation** called `@RequiresRole("value")`, the value in here in the scope of my application are "Admin" and "User". Annotations are creating in java using interfaces. The main reason behind creating this Annotation is for the implementation of Role Based Access Control(RBAC).

**Authorization Pattern** is used to assign access rights to each user who is accessing the security-sensitive system and checks the Access rights before executing any functionality. we can have any number of roles in the application as we can extend the number of roles in the application without the need to refactor the entire application.

The **proxy pattern** provides our application with a proxy Interface, which is being created once a user requests any method, this interface acts as proxy between the user and the main application without giving direct access to the application. This helps us in protecting the methods which are not accessible by the current role.

**Reflection** is used to find out the annotations present on the interface at runtime. It can be used to inspect the class and find out what's in there during runtime, but only at the cost of an extra overhead.

**AuthorizationException** Created a custom exception class, this exception is thrown whenever the role required by the method doesn't match with the role of the user. This exception needs to be caught at runtime.

I have also created a **Session** for each user This class helps us in creating a Session object of the user, this session can be used in accessing the methods present on the server.

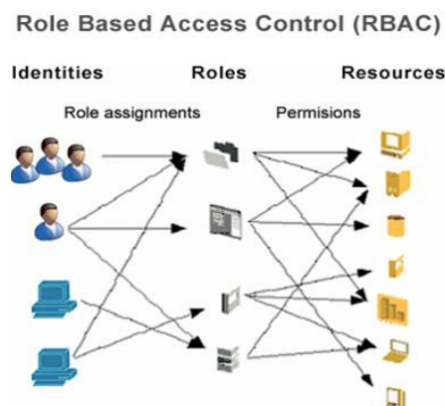


Fig [4] - RBAC

This is how the class diagram of third assignment looks like:

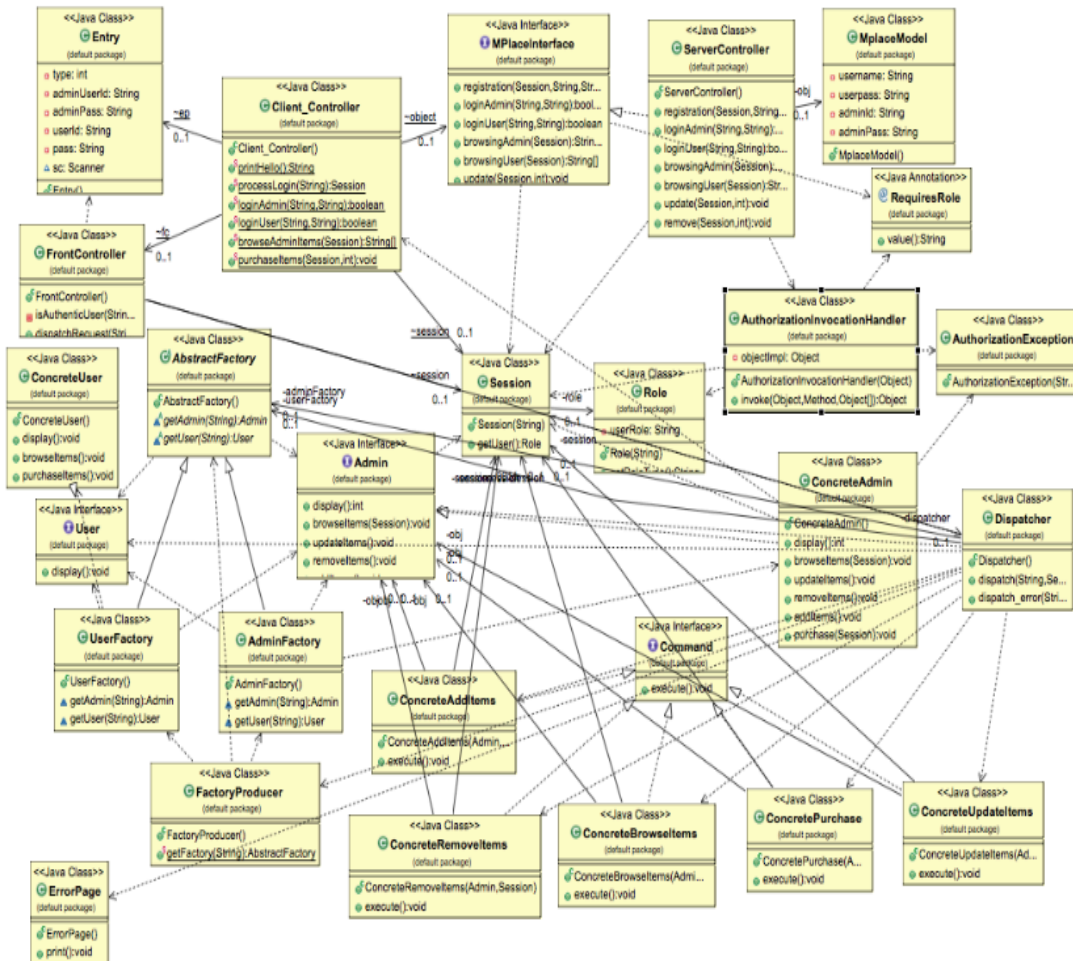


Fig [5] – Assignment#3 Class diagram

## Assignment#4 Overview

In this Assignment, we are asked to fully implement the Add items, Browse Items, Purchase Items functionality, and we are asked to simulate the java RMI concurrency feature using all the 6 network machines provided to us, in this assignment I also implemented the database functionality, just that it makes things easier for the next assignments.







## Assignment#5 Overview

In this Assignment, I have fully implement the application functionality and examined how java implements synchronization and concurrency patterns, and made the system to efficiently handle concurrent requests at the same time.

- Monitor Object Pattern
- Future
- Guarded Suspension
- Scoped Locking
- Thread safe interface
- Synchronization.
- Database Access Layer.

The main aim of this assignment is on concurrency control and synchronization of the application, which was done mainly using the synchronized keyword in java and Volatile keyword.

The class diagram hasn't changed from 5<sup>th</sup> to 6<sup>th</sup> assignment, so I have kept it on my readme file in GitHub as it is difficult to place it in here.

## Assignment#6 Discussion

### Database Access Layer

I have implemented the database access layer in 5<sup>th</sup> assignment itself, as I was concerned about the design choice I had to make at that time, since we always needed separation of concerns in the application.

By eliminating the database access code from the model and placing it in a separate class it gives the flexibility to reuse and modify code without effecting the entire working of the system.

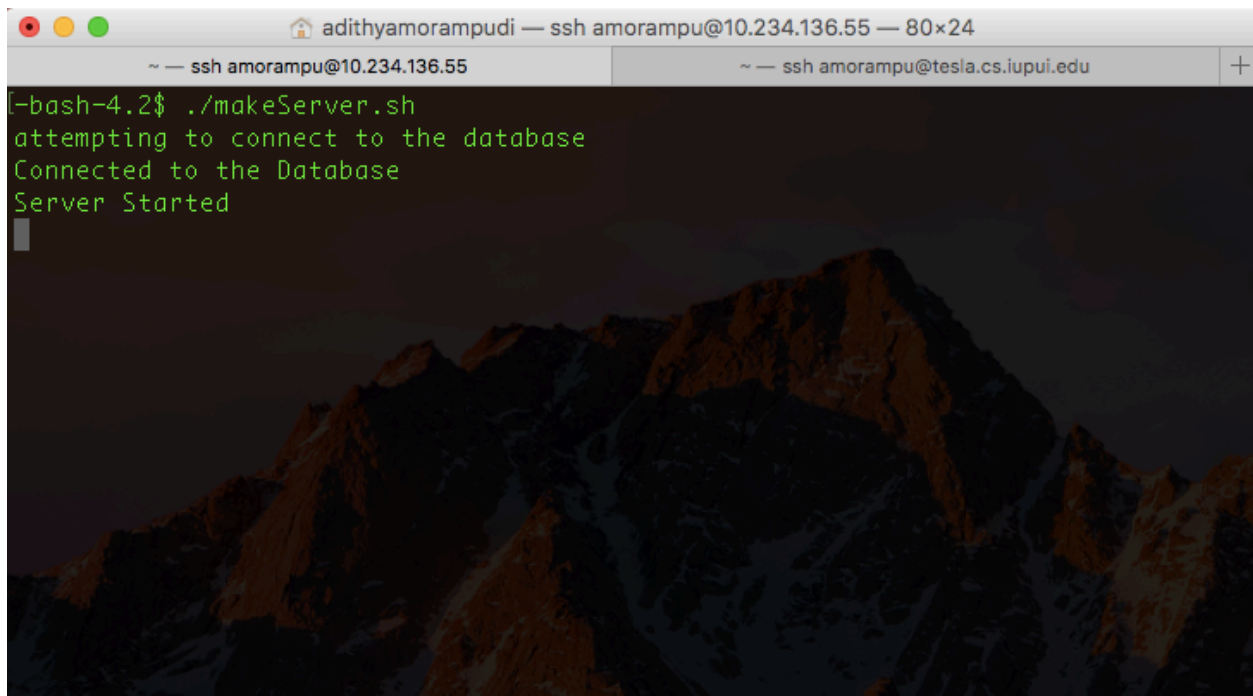
By having this database access layer, all calls to the database are centralized through this class and makes it easier for the application to make the contact through this layer. So just in case if I want to change the database from MySQL to a different database, I can simply change this database access layer and then application works totally fine.

All mappings between objects and tables are encapsulated within this database layer, Application logic should NOT concern itself with any database-related constructs

My database access layer class is DbConnection.java, which has all the functionality of DML. I have used **Singleton pattern** to make the database connection and return only this single instance interact with the database all the time, without having to create a new instance for every new user. I have made the Connection object **Volatile**, so as to not block the connection object unnecessarily.

## Sample Runs of completed Application

Screen Shot to show the running Instance of the Server, please note, the server should always be run on 10.234.136.55 only. As soon as the server is started it first connects to the database. As you can see from the below figure. Now we can run different clients on different machines and our server can handle concurrent requests, without any trouble.

A terminal window titled 'adithyamorampudi — ssh amorampu@10.234.136.55 — 80x24'. The window has two tabs: '~ — ssh amorampu@10.234.136.55' and '~ — ssh amorampu@tesla.cs.iupui.edu'. The active tab shows the following output: 

```
[~bash-4.2$ ./makeServer.sh  
attempting to connect to the database  
Connected to the Database  
Server Started  
█
```

The background of the terminal is a dark, high-contrast image of a mountain range.

A running Instance of the client in the figure below, which is being executing and sending requests to the server, we can see that the client can get the items from the **database** and display it to the user. Now, the user can select the other available options based on the output shown to him. The client side of the application displays the following, the entry screen will display a view which asks the user to select if it is a customer login or Admin login, based on the user input, the respective view is rendered and displayed to the user, once the view is displayed, the user is prompted to enter credentials and if they are valid then a number of options are available to him/her, based on the choices available the user can select the required one and can start accessing the application, all the options are displayed in the below screenshots.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80x24
~ — ssh amorampu@10.234.136.55
^C-bash-4.2$ ./makeClient.sh
welcome to RMI Username: amorampu Password: Port: Quickconnect
Enter 1 for Admin Login
Enter 2 for Customer Login
Enter your choice: 2
Retrieving directory listing of /home/amorampu/OOAD/src/.git/logs/refs/heads/master
File transfer successful, transferred 41 bytes in 1 second
File transfer successful, transferred 1,066 bytes in 1 second
Enter your userID:
[amorampu]adithyamorampudi/Documents/Courses/OOAD/Assignments/A1/
Enter your password:
1234
Welcome Customer
csci50700_spring2017_marketplace-Assignment-3
csci50700_spring2017_marketplace-Assignment-4
csci50700_spring2017_marketplace-master
1.browse Items
2.Purchase Items
Enter Choice
1
----- Server -----
Software Quality Assurance
Item ID Item Name Stock Filetype Price Last modified Description
1 sample 2 Directory $20 04/01/2018 17:3... sample product
2 Fedex 1 Directory $30 04/01/2018 17:3... Shipping price
3 sanitizer 0 Directory $20 03/06/2018 16:0... hand sanitizer
```

Screenshot to show the **purchase** functionality, when a user tries to purchase an item he needs to display enter the id related to the product and if the stock is available then user can purchase it, otherwise an error message is displayed saying the product is out of stock.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80x24
~ — ssh amorampu@10.234.136.55
1.browse Items
2.Purchase Items
Enter Choice
^C-bash-4.2$ ./makeClient.sh
[-bash-4.2$ ./makeClient.sh
welcome to RMI Username: amorampu Password: Port: Quickconnect
Enter 1 for Admin Login
Enter 2 for Customer Login
Enter your choice: 2
Enter your userID:
[amorampu]adithyamorampudi/Documents/Courses/OOAD/Assignments/A1/
Enter your password:
1234
Welcome Customer
csci50700_spring2017_marketplace-Assignment-3
csci50700_spring2017_marketplace-Assignment-4
csci50700_spring2017_marketplace-master
1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemID:
1
Item purchase success!
Directory 04/01/2018 17:3...
Directory 04/01/2018 17:3...
Directory 03/06/2018 16:0...
Directory 02/12/2018 22:3...
```

Screenshot to show the add items functionality, this function is restricted to the administrator of the application, once the admin is logged in, he or she can then access the add item function and add the items into the database, please note that the id of the item is auto generated and auto populated, so the admin only needs to enter the remaining fields.

```

adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80x24
~ — ssh amorampu@10.234.136.55
Enter your userID:
admin@tesla.cs.iupui. Username: amorampu Password: Port: Quickconnect
Enter your password:
12345
Starting upload of /home/amorampu/Documents/Courses/OOAD/Assignments/A1/src/.git/logs/refs/heads/master
Listing directory listing of "/home/amorampu/OOAD/src/.git/logs/refs/heads"...
File transfer successful, transferred 41 bytes in 1 second
File transfer successful, transferred 1,066 bytes in 1 second
Welcome Admin
1.browseItems
2.Update items
3.removeItems
4.add Items
5.Purchase Items, This Method wont work in ADMIN View(RBAC Demo)
Enter Choice
csci50700_spring2017_marketplace-Assignment-3
csci50700_spring2017_marketplace-Assignment-4
csci50700_spring2017_marketplace-master
Design_Patterns
Lecture_slides
javaapplication16
Senior
Software Quality Assurance
Enter the details of the items to be added!
Enter the price of the product
5
Enter the Stock of the product
5
Enter the name of the product
earphones
enter the description of the product
apple earphones
Item added into the database successfully.

```

Below screenshot shows New User registration.

```

adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
~/OOAD/src/.git/logs/refs/heads/...
.git/logs/refs/heads
-bash-4.2$ sh makeClient.sh
welcome to MarketPlace Application
s/Enters/1 for Admin Login
Enter 2 for Customer Login
Enter 3 for Customer Registration
Enter your choice:
3
Enter the First Name:
Mike
Enter the Last Name:
Tyson
Enter the User Name:
mtyson
Enter the password:
1234
success
Welcome new User

```

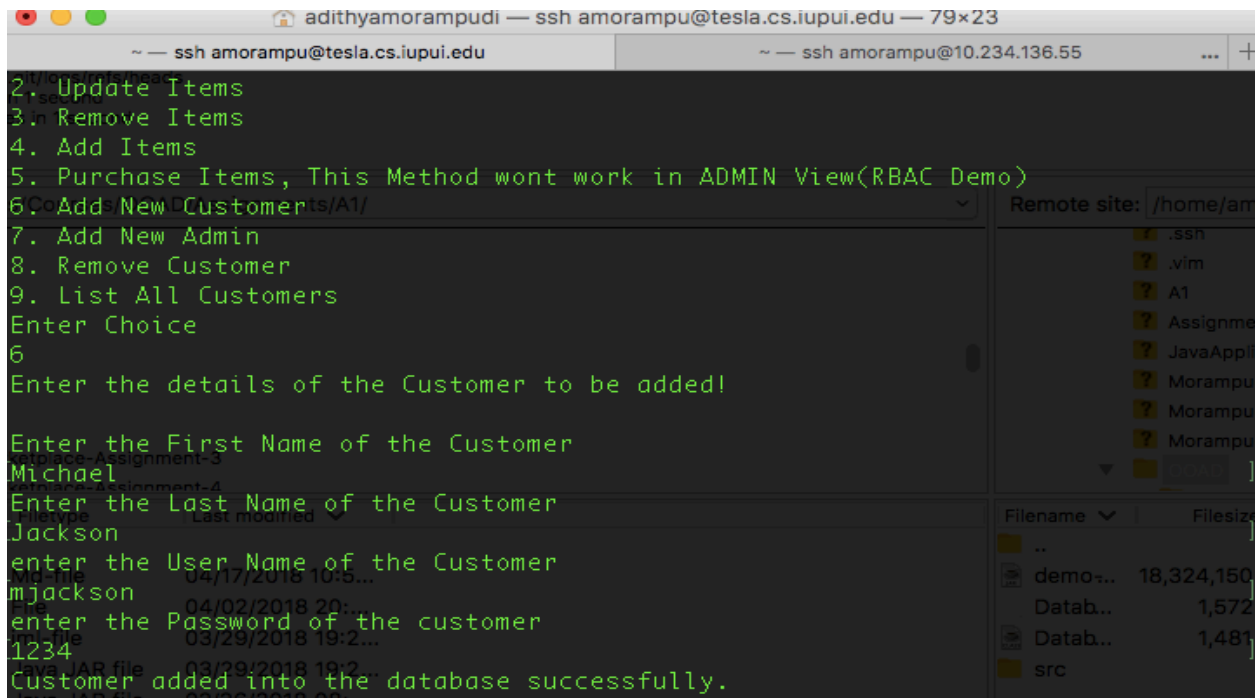
Adding items to the cart.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
~/COAD/src/git/logs/refs/heads...
git/logs/refs/heads
7 in 1 second phone 2 $10 iphone 7 plus
=====
8 Courses/Id Assignments/A2 $10 crimson cards Remote site: /home/amorampu
=====
9 earphones 2 $5 apple earphones
=====
10 Monitors 1 $20 sfadd
=====
12 Diamond 16 $2030 fasdf
=====
Filetype Last modified
Enter the Id of the product to be added to the cart from above list
12
Enter the quantity of the product.
3
Item added to the cart successfully.
File JAR file 02/26/2018 08:...
```

Purchasing the user cart, the user can remove the items from the cart as well.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
~/COAD/src/git/logs/refs/heads...
git/logs/refs/heads
7 in 1 second
Items present in the cart are:
ItemId quantity
12 10
Welcome MTYSON Enjoy your customized portal
1. browse Items
2. Purchase Items
3. List Items in the cart
4. Add Items to cart
5. Delete From cart
Enter Choice
2
12 Purchased
Welcome MTYSON Enjoy your customized portal
1. browse Items
2. Purchase Items
3. List Items in the cart
4. Add Items to cart
5. Delete From cart
Enter Choice
```

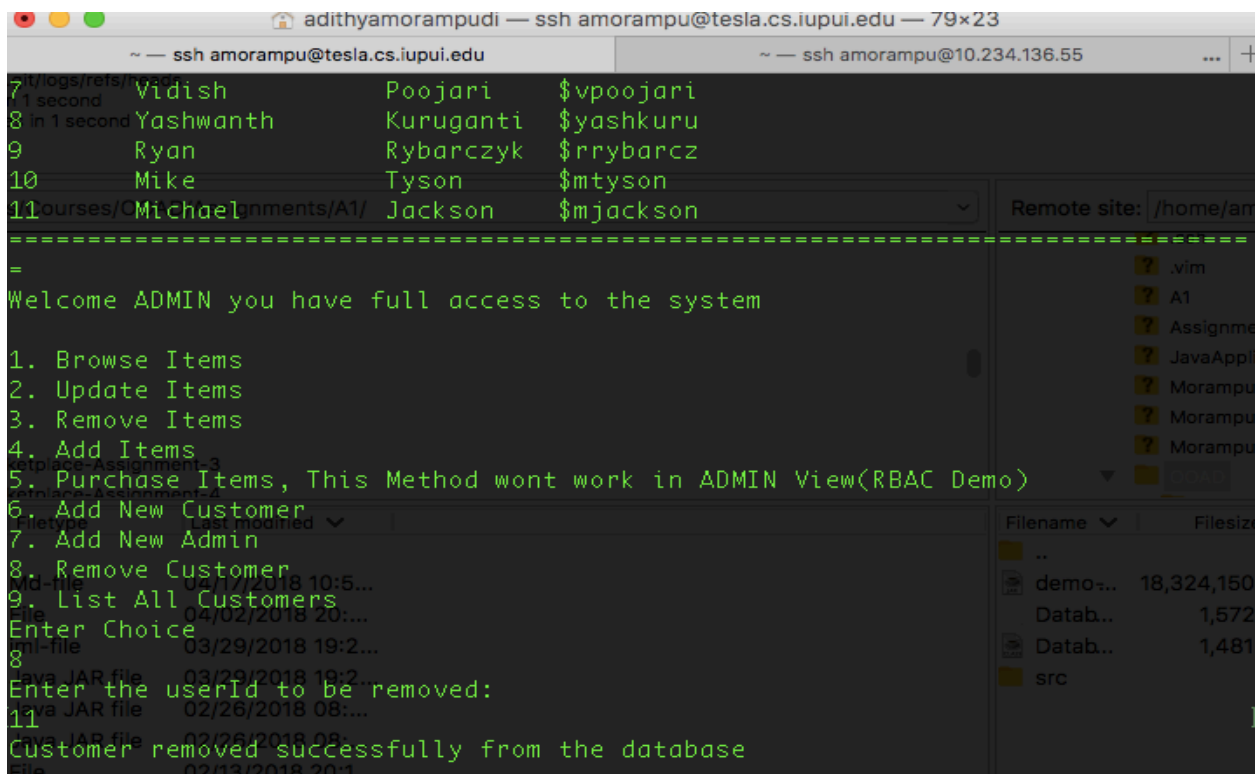
Admin adding a new customer.



```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
2. Update Items
3. Remove Items
4. Add Items
5. Purchase Items, This Method wont work in ADMIN View(RBAC Demo)
6. Add New Customer
7. Add New Admin
8. Remove Customer
9. List All Customers
Enter Choice
6
Enter the details of the Customer to be added!

Enter the First Name of the Customer
Michael
Enter the Last Name of the Customer
Jackson
enter the User Name of the Customer
mjackson
enter the Password of the customer
1234
Customer added into the database successfully.
```

Removing a customer from the database



```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
7. Add New Admin
8. Remove Customer
9. List All Customers
Enter Choice
8
Enter the user ID to be removed:
11
Customer removed successfully from the database
```

## Conclusion

Finally, looks like the project is completed. But I need to do a lot more to make it a fully functional application. Anyway, let me tell that this is the first project at IUPUI that I am proud of mentioning it in any of my interviews or anywhere else. This project gave me an opportunity to design a real-time application which is scalable, fault tolerant and secure.

I wasn't aware there are these many patterns at first, and even though I knew some of these exist, I wasn't sure where to use them and when to use them, this project gave me an opportunity to learn them and apply them in a real-world situation. Which I think wouldn't have been possible without taking this course. This project taught me how to design an application which can be scaled and I also learned what it requires to keep in mind when designing an application which must be extendible. I mainly liked learning about architecting an application from the scratch.

I think there are not many things that I didn't like about this assignment, but sometimes I felt like it would be better if we could've used some of the build tools like Maven.

- Testing the application, which is important when designing an application for production, I think this was missing in the project, in the sense that we have never emphasized anywhere in any assignment about writing simple unit test cases. I would try to test the application after the final exam.
- Sometimes it was also an overhead to add new feature into the application, to add a single feature I had to make changes in almost 6-8 files, which I didn't like.

If there was a chance for me to redesign the application, I would've made use of web services such as REST to develop this application, since we are dealing with distributed applications, we need to interoperate between different platforms at some point and I don't have to redesign the entire application if I wanted a mobile version of this application, I could simply use the API to access the service, given if I made use of web services.

I Wouldn't have used Java RMI as Rachel mentioned in the class that after spinning up ~1000 threads RMI would hang, this would become a serious scale issue if I were designing this application for production.

Maybe I would've also designed my UI from the beginning. Because this will heavily impact my design as well.

## References

- [1]. <https://stackoverflow.com/questions/3507253/java-rmi-and-synchronized-methods>
- [2]. <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>
- [3]. <http://www.cs.wustl.edu/~schmidt/PDF/ScopedLocking.pdf>
- [4]. <https://docs.oracle.com/javase/tutorial/rmi/overview.html>
- [5]. <https://stackoverflow.com/questions/27759566/concurrent-access-to-a-remote-object->
- [6]. <https://wiki.hsr.ch/PnProg/files/ThreadSafeInterface.pdf>