**ASSIGNMENT-5 PROJECT REPORT**

**PROJECT REQUIREMENTS**

**ANALYSIS:**

**GOAL:** To design an Online market place that should handle some of the basic operations such as:

* Registration and Login
* Browsing Items
* Updating Items
* Removing Items
* Adding Items
* Purchasing Items

The above-mentioned operations will be implemented by using various software design patterns and frameworks for distributed computing. This must ensure the reliability and software quality. The entire coding is done in JAVA (RMI) and ECLIPSE being the compiler.

**ASSIGNMENT 5**

In Assignment 5 will mainly be examining the concept of Synchronization. As seen in previous assignment there are many significant challenges with respect to synchronization and construction of multi-threaded applications. The focus of this assignment is to overcome these problems arising and find the solutions to it.

This assignment applies the information we have learned regarding the use of synchronization in the Java programming language to ensure that access to our shared resources are indeed thread-safe and mainly examining how Java implements the following patterns: Monitor Object, Future, Guarded Suspension, Scoped Locking, and Thread-Safe Interface.

**REQUIREMENT ANALYSIS:**

* Main aim of the client is to sell the goods to the customers around the world.
* The System should be constructed in a portable language like JAVA and makes use of their existing network.
* Separate Interface for both Client and Server
* Customer should be able to browse the products and view the details of the desired product along with its availability and add them to the cart.
* If the product is not available for the customer It should prompt to the customer of its unavailability.
* Administrators can add the Items to inventory and can also modify the details of the

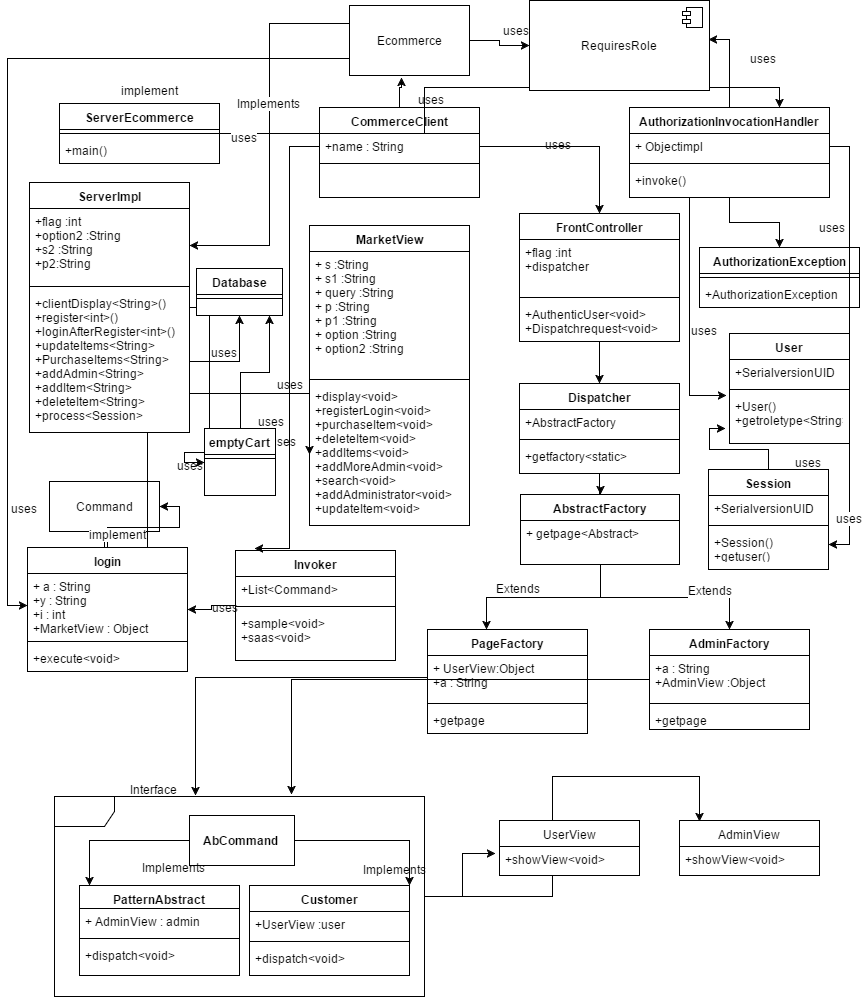
Item. They should also have the access to add another administrator’s as well as add or remove customer accounts.

* Default administrator account is already created.
* The customers should be able to register their accounts.
* The roles should be distinct and independent.
* The system should be reliable and should allow and process multiple customer requests during execution.

**DOMAIN MODEL:**

C:\Users\saipa\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Domainmodel.png

**CLASS DIAGRAM**



**CODE:**

For implementing RMI we must first construct a remote interface with methods in it.

There are 3 methods that are declared in Interface

* Register ()
* clientDisplay()
* loginAfterRegister()
* addAdmin()
* updateItems()
* purchaseItems()
* process()
* addItem()
* deleteItem()
* dispatchRequest()

We also have a SERVER class and a CLIENT class.

In **SERVER** class RMI functionality is implemented by using JAVA in general:

**SERVERIMPL:** This has the entire functionality of the project. All the methods are defined in this **SERVERIMPL CLASS.**

The **SERVERIMPL** class acts as the controller It contains all methods that are to be implemented.

**Register method** is used for registering the customers. It is an integer method with 2 strings (USERNAME AND PASSWORD) as the parameters. It shows whether the customer is successfully registered or not. **Login method** is also implemented in this and is mainly used for logging in the portal. It shows whether the customer or the admin is successfully logged in or not. The default admin credentials is taken as **Pavan (USERNAME) and java@2345(PASSWORD).**

**CLIENT** In this a remote object is created and connected to server all the implementations of the server methods are done in client.

**VIEW mainly** contains the code which is generally used during the runtime. All the details the user has to enter are present in this and the client access this by using a remote object created.It has two methods in it which are register() and registerLogin() which are void methods.

**AbCommand** is an Interface which has void dispatch method.

**AbstractFactory :** Abstract class

* getpage [Abstract Method]
* Attributes are Parameterized which is a String

**AdminFactory:** It Extends Abstract Factory which is an abstract class

* It implements getpage method.

**PatternAbstract :** It implements AbCommand interface.

* Parameterized Constructor having object of AdminView.
* Implementation of dispatch method.

**PageFactory** : It extendsthe abstract class

* Implements getpage method.

**Customer:** It implements AbCommand interface.

* Parameterized Constructor having object of UserView

**AdminView and UserView:**

* These are the generic views created.
* They both have showView() as a method. It is a void method.

**Command:** It is an Interface.

* It has a method execute () and it is a void method.

**Front Controller:** Parameterized Constructor.

* Integer Attribute.
* AuthenticUser : It is a void method with String as parameter.

**Dispatcher:** Dispatches the control to either AdminFactory or PageFactory.

* getFactory method is present which is static type of AbstractFactory.

**Invoker:** It has a List and 2 methods

* Sample : void method.
* Saas : void method.

**Login:** It is used for implementing the command pattern.

* It implements Command interface.
* It overrides Execute method.

**User:** It determines the role-type and it returns to the Session which can get which role type user is using the Application.

* Parameterized constructor.
* Method getroletype().

**Session:** It is implemented from the serializable interface.

* It returns the user-type by using the getUser() function.

**RequiresRole:** It is a custom defined annotation.

* Retained till runtime and applied on all types and methods as defined in @Target and @Retention
* It has a value attribute of type String.

**AuthorizationInvocationHandler:** It is implemented from the serializable interface and Invocation Handler.

* It has a method invoke which has parameters in it.
* It has a final long serialVersionUID.

**AuthorizationException:** It extends Runtime Exception.

* It has a final long serialVersionUID.
* It denys the access for a method if the user is not authorized**.**

**emptyCart:** It mainly contains the implementation of future pattern.

**Database**: It has all the Database related functions connections and queries:

**OVERVIEW:**

The focus of this application is to create an application where the customer can purchase the desired product and the admin can do the manipulations to the Inventory.

From the Domain model, we can say that there are 2 main users **Administrator** and **Customer.** Only the customer can register and we assume there is one administrator by default.

The Datatypes and the functions that are going to be used or used are specified in the block.

The User block just validates the login of administrators and customers. The administrator block can add other administrators and add items, delete items, modify item details from the inventory.

The Customer on the other hand can register and login to the system. He can only search the items from the Inventory rather than modifying them. Additional functionality possessed by the customer is he can also purchase the product i.e. Adding items to the cart.

After adding items to the cart the customer can finalize the purchase if he does so the appropriate amount of stock gets cut from items and gets added to the customers cart .

**PATTERNS USED:**

**3.1Front Controller Pattern**

Front controller receives the request from the view I,e about the Login and dispatch this request to the Server Controller to verify the credentials provided. Once the controller returns the result to the Front Controller the Front Controller can then pass this to the Dispatcher which will render the appropriately according to the input provided whether it is a customer or admin if its an admin it moves to admin page else customers page.

3.2 Command Pattern

Command Pattern tells us what to do instead of how to do. We will be having a command interface and an execute method executing all the methods in the server controller.

3.3 Abstract Factory Pattern

Abstract Factory Pattern we abstract classes to stop the client know about how the method is called and keep abstracted from the user. We create Abstract factories from a factory. Here we have created 2 factories for admin and user.

3.4 Role Based Access Control using Annotations

It provides control on the functions specific to the roles I,e only customer can access the customer function and admin can access admin function.

3.5 Proxy Pattern

Creating a proxy for main class instead of calling again and again. We create a proxy class and use it again and again instead of a main class. If the main class fails proxy class will be available and it works as a main class.

3.6 Reflection Pattern

We get the entire copy of a class without even calling object of the class. Functions and variables everything can be accessed by making a copy of the class.

5.1 Monitor Object

In monitor object you define one critical section of a code to make it synchronize with other threads to achieve this we use synchronize keyword.So, when synchronize keyword is used monitor object acquires a lock via monitor enter so no other thread can access it until its execution after execution it releases lock via monitor exit. By applying monitor object we explicitly implement GUARDED SUSPENSION AND SCOPE LOCKING in which whenever control comes into the critical section guarded suspension checks whether the condition is satisfied or not if not satisfied it keeps the thread in wait state once condition satisfies it notifies all. While in scope locking whenever control gets into critical section it acquires lock but once it leaves it makes sure it releases the lock.

5.2 Future Pattern

We use this pattern when user requires access to the variable which result is not yet achieved at that time we provide virtual object to the client until the result is not available. So, virtual object keeps on checking onto the condition whether the variable is received or not. once variable result is there virtual object notifies client thread to execute its execution.

5.3 Thread Safe Interface

Putting a lock on interface rather than implementation of interface to avoid a deadlock condition in application

**ANALYSIS:**

Distributed Application: The term Distributed Application implies able to run the application in several machines at the same time provided all the machines are connected in the same network. JAVA supports this behavior with the help of RMI which is an API that helps in creating the distributed Application. The RMI produces this with the help of remote communication between the applications using stub and skeleton.

Stub: Gateway for the client side.

Skeleton: Gateway for the Server side.

**Application Point of View:**

In our Market Place application, we used RMI to support the distribuends for the application as the application can be run on several machines that are controlled by the server. The Server will be running on one of the machine and the clients will be running on the other machines where all the machines are in the same network. The server process the requests of various clients that are requesting it and provides them with appropriate results.

In our application we used:

in-csci-rrpc03.cs.iupui.edu - 10.234.136.57 SERVER

in-csci-rrpc02.cs.iupui.edu - 10.234.136.56

in-csci-rrpc01.cs.iupui.edu - 10.234.136.55 CLIENTS

in-csci-rrpc04.cs.iupui.edu - 10.234.136.58

in-csci-rrpc05.cs.iupui.edu - 10.234.136.59

in-csci-rrpc06.cs.iupui.edu - 10.234.136.60

**RMI CONCURRENCY:**

It is the ability to run several programs or parts of the program in parallel mainly to improve the throughput of the program.

JAVA RMI supports concurrency as the process or functions are present on the host(Server) and the communication happens through internet.

Java RMI does support concurrency but there is no guarantee that if offers the best as the number of clients increases and there is a single server. There may be chances of bottleneck and also deadlock as there may be a circular dependency between the remote objects. RMI cannot control concurrent access to remote objects present. In JAVA RMI the remote objects make use of sockets and serialization for transfer of variables etc (transfer of control).

To invoke the remote object multiple times JAVA RMI uses threading. Hence, we can say that JAVA RMI is multithreaded. The JAVA keyword Synchronized is used to make sure that only one thread can enter remote object at any given point of time thus ensuring no two threads runs on same object at same time. The other threads will be present in a waiting stage when one is executing. In the application, I have performed some tests to check the concurrency and its impact. Synchronization between the threads can be achieved by placing the synchronized keyword in front of the method.

Firstly, I have run server on one of the machine and clients on 3 different machines present in the same network. All the clients are running as customers and are connected to the same server on this I have implemented all the client methods such as registration, login, search, purchase and I observe all the requests are handled by the server and the results are given to the appropriate clients based on their request. However, there was a slight or considerable delay in the results and the delay was negligible. The Server has managed all the requests effectively though all the requests were concurrent.

Secondly, I have run server and administrator role on all the 3 machines. I have implemented all the methods of the administrator like adding admins, search items, update items, delete items and add items and all requests are handled seamlessly and the results are given to the respective admins.

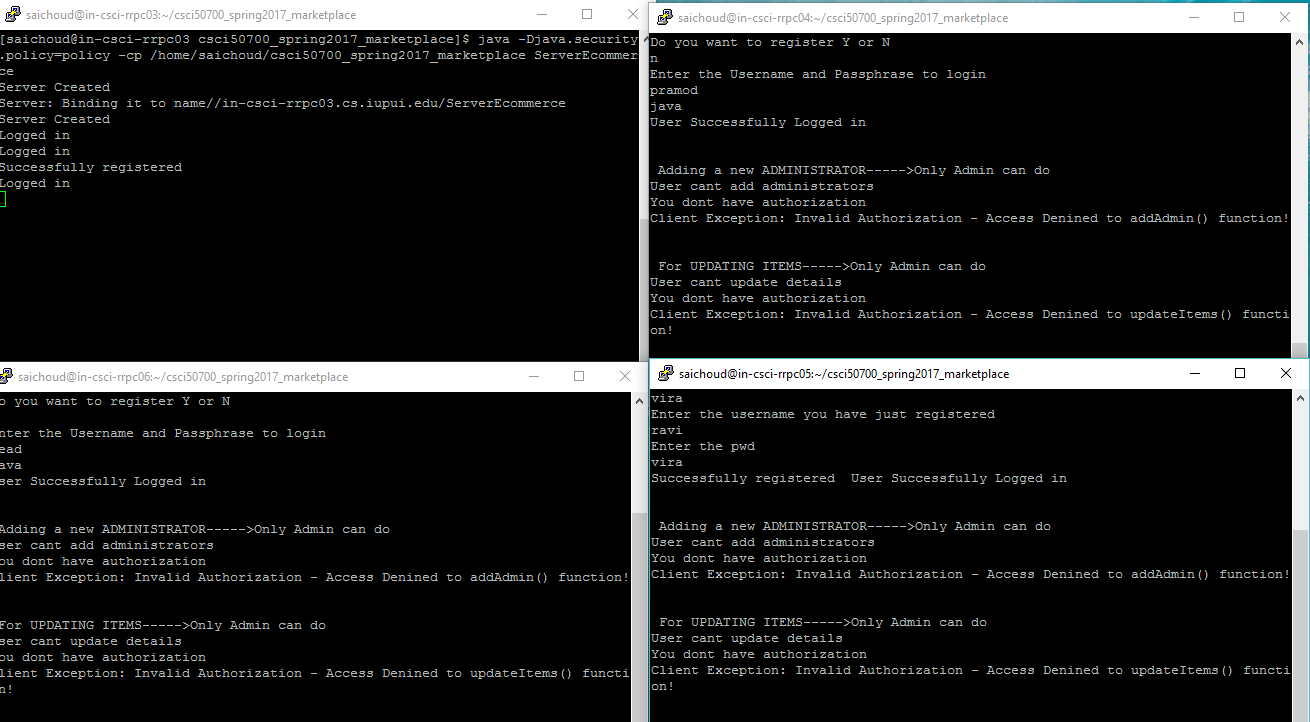
Finally, I have run server on one machine and a client and admin on 2 different machines all the operations corresponding to both the customer and admin are run and the server has responded and processed the requests appropriately and returned the results correctly all the role based operations are performed and returned exactly. This shows JAVA RMI is multi-threaded.

**SYNCHRONIZATION:**

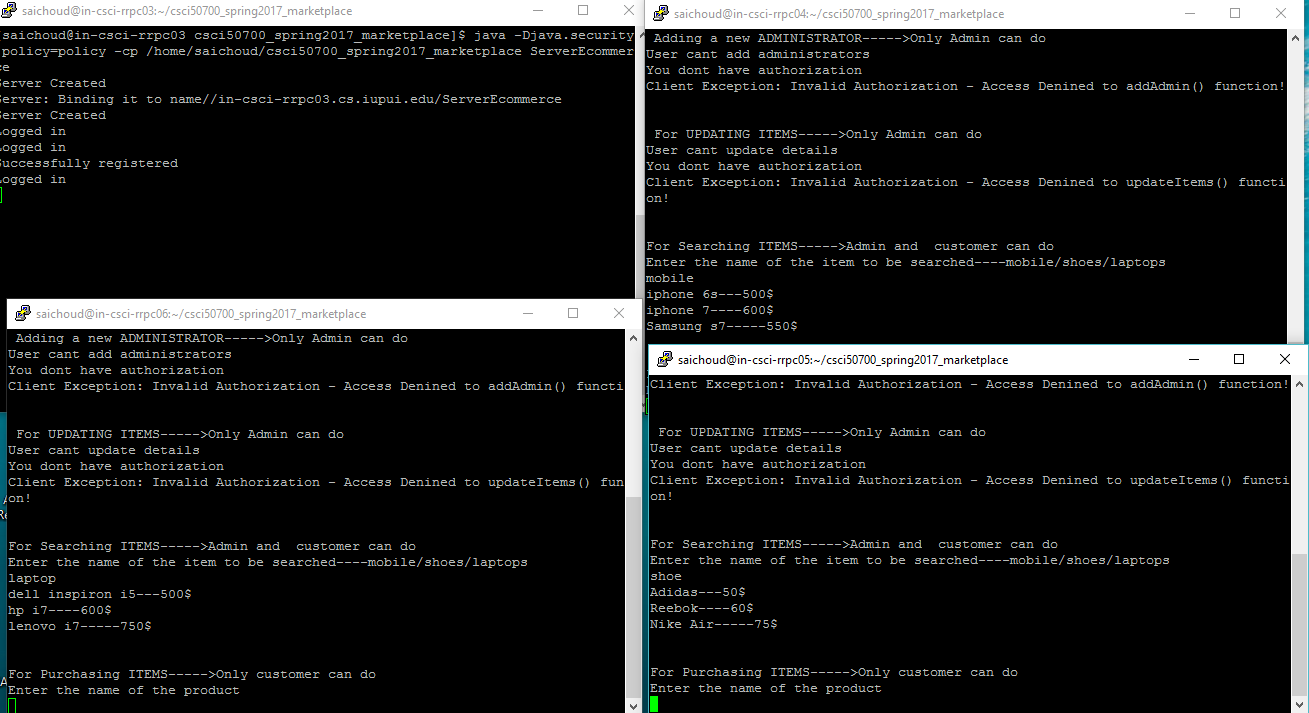
Providing access to the client in a concurrent way without getting into a deadlock and livelock condition. We achieved synchronization in our application by implementing many patterns like Monitor object, Future Pattern ,Thread safe ,Guarded suspension and scope lock pattern.

**SAMPLE RUNS:**

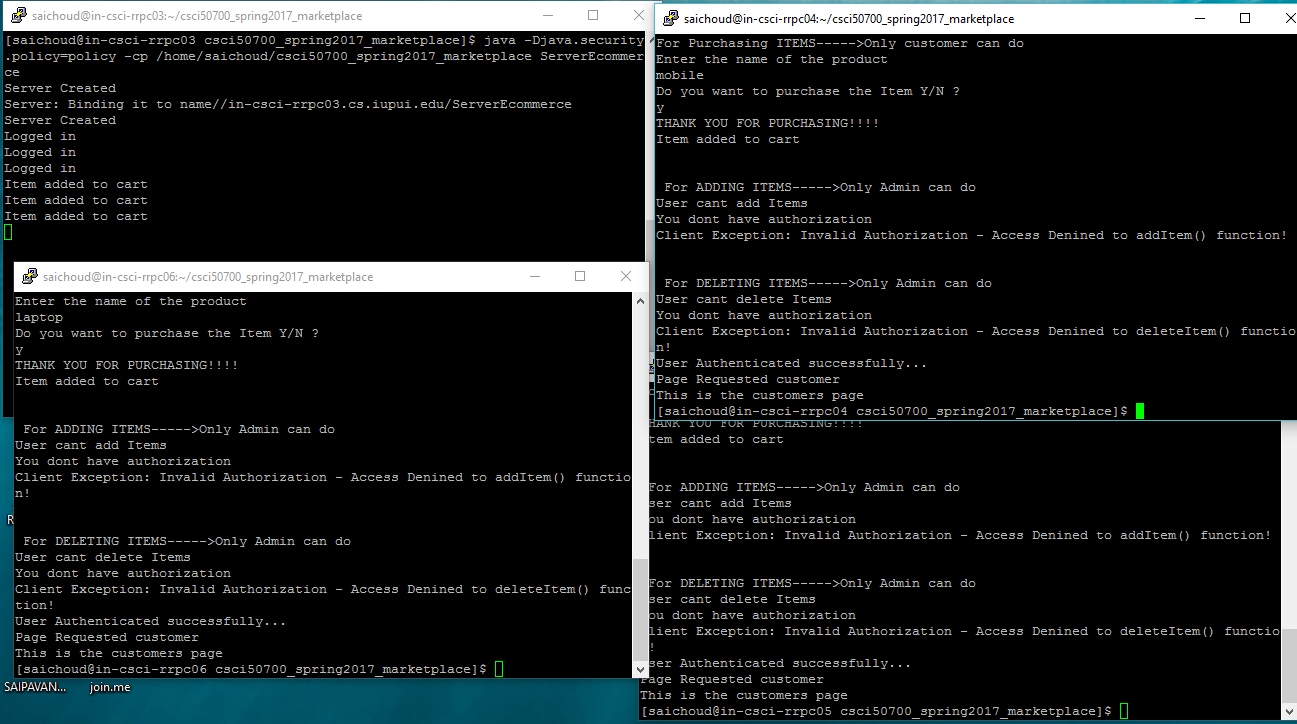
* The below sample run shows 3 clients and one server .
* The Customer role is running on all the clients .
* One customer registers and the other 2 just perform the login functionality**.**



**Searching operation on all the clients where the role is customer:**

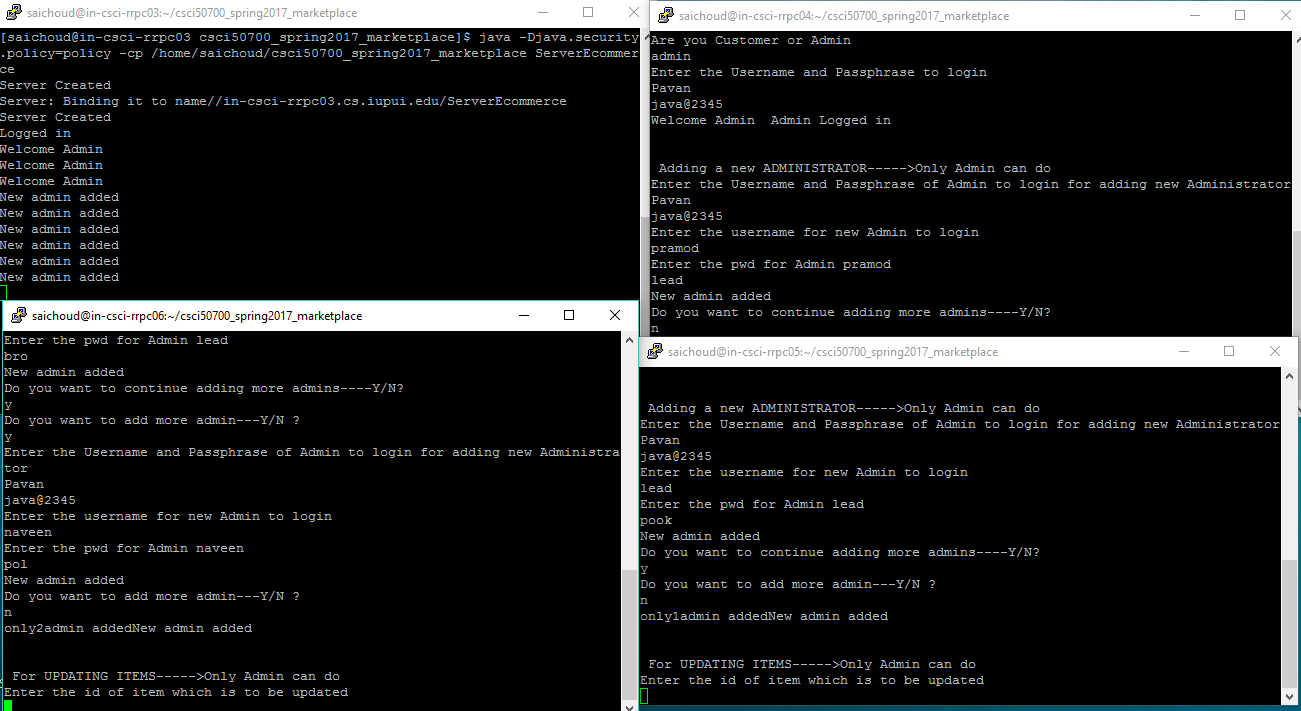


**Purchasing operation where the role is customer.**

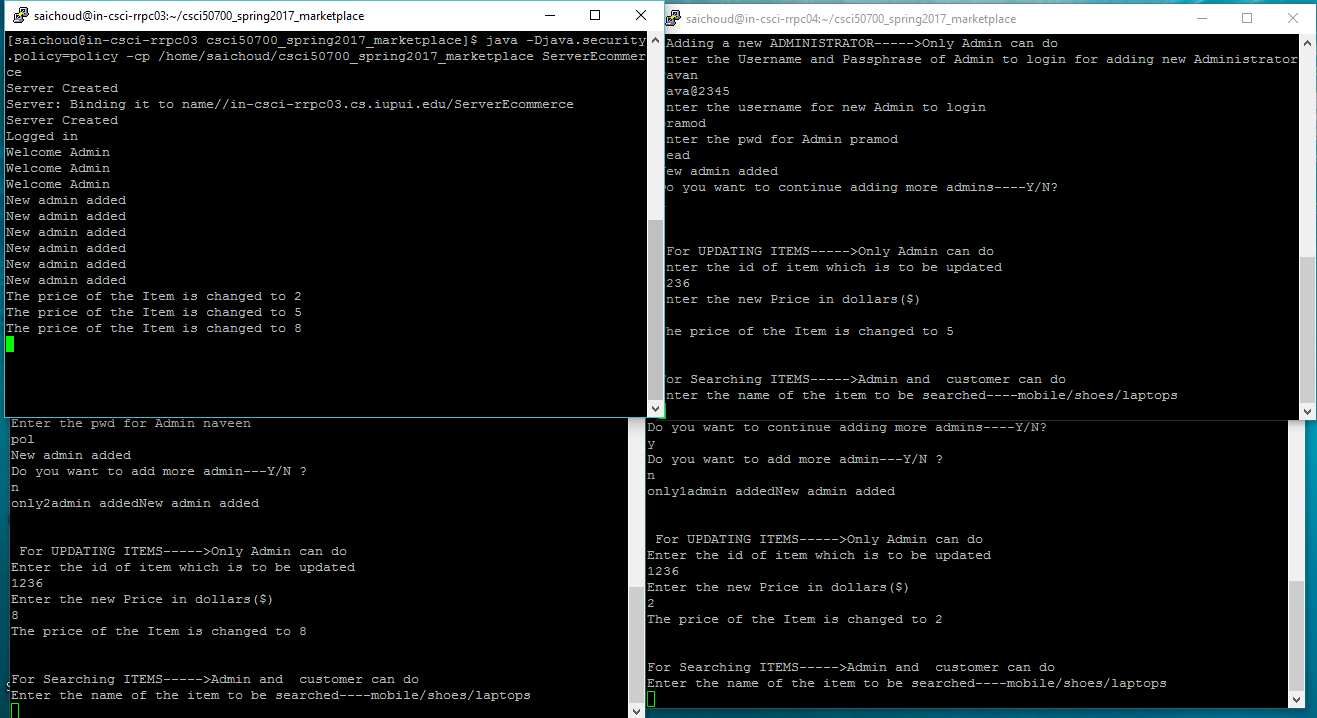


**ADMIN ROLE :**

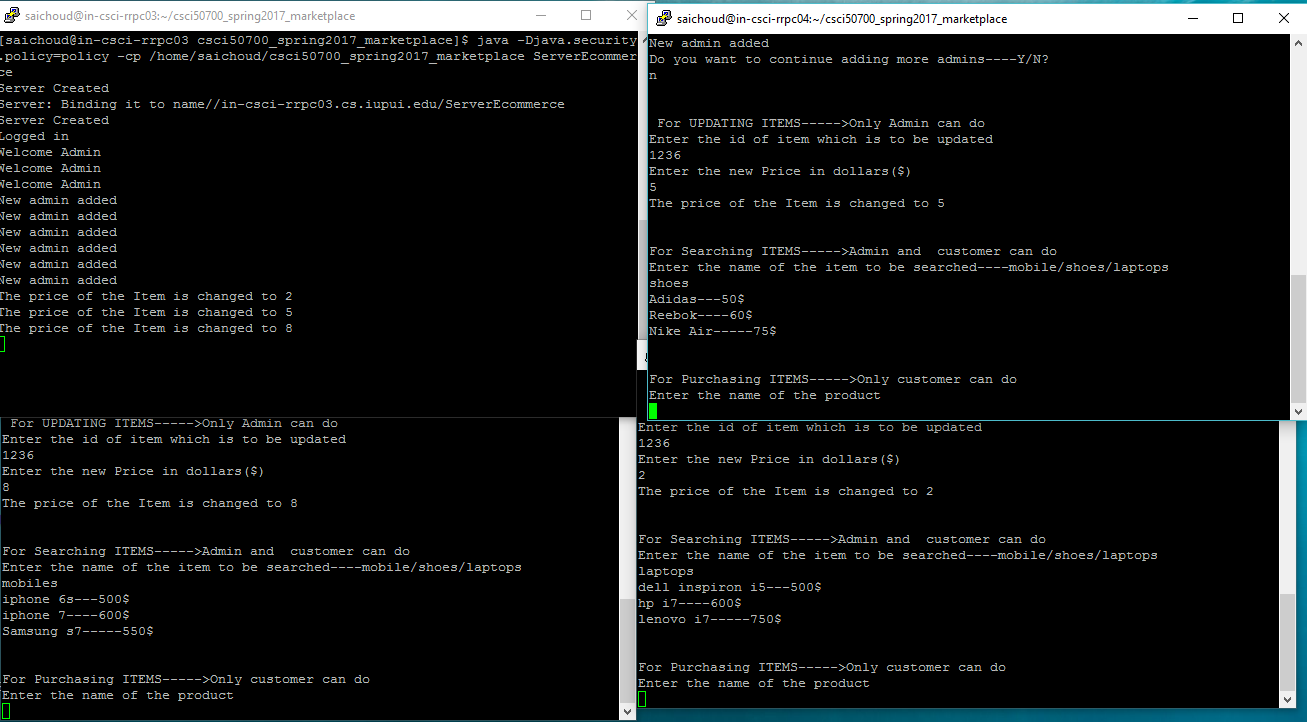
**The below image shows admin add functionality where one administrator tries to add more admins.**



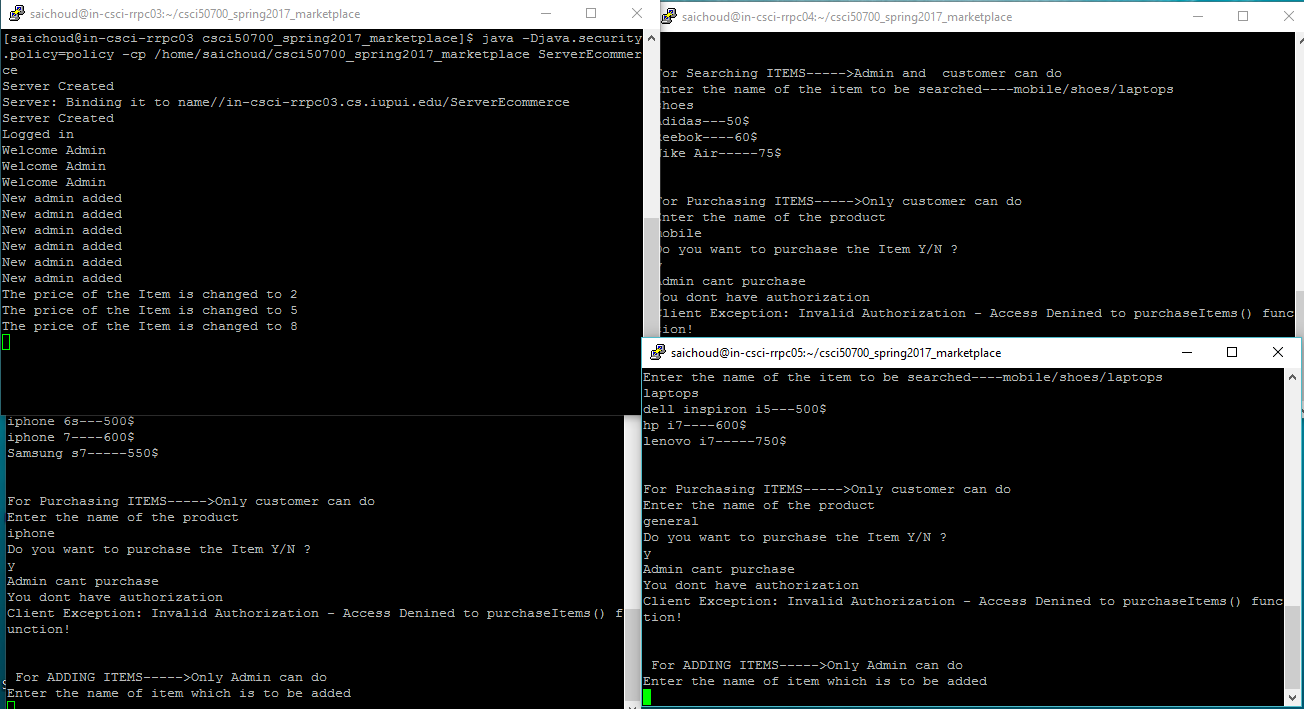
**Updating functionality performed by the admins**



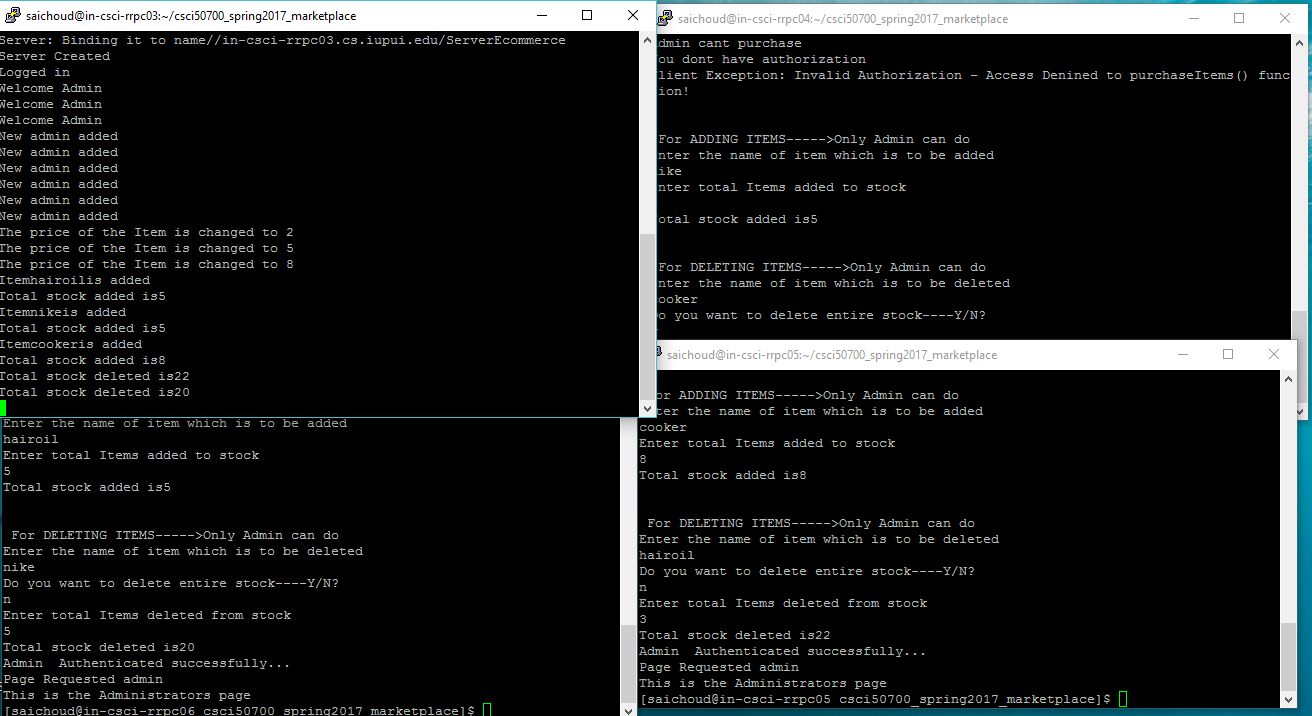
**Search Item functionality performed by admins**



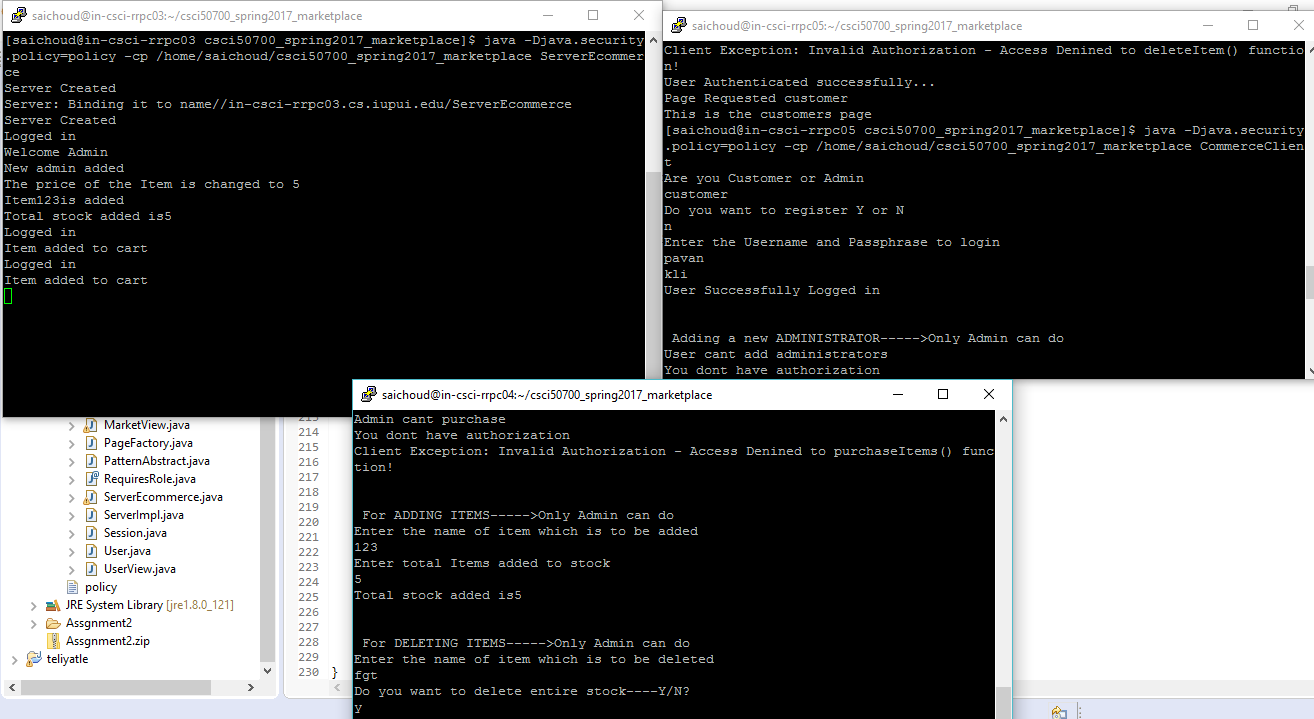
**ADDItem functionality implementation by admins it also shows RBAC where admin cant purchase item.**



**DeleteItem functionality performed by Admins**



**Admin and Customer both running simultaneously on the server.**



**CONCLUSION**

The assignment mainly includes the additional functionalities and the impact concurrency has on our application.