**PROJECT REPORT**

**Assignment #5**

**Overview:** Design and implementation of BestSale (online market place). The online marketplace will handle the following basic events:

• Registration & Login

• Browsing Items

• Updating Items

• Removing Items

• Adding Items

• Purchasing Items

**Goal:** The goal of this document is to give a detailed understanding of the project. This will be helpful to implement the BestSale in JAVA RMI using MVC architecture.

**Assignment #5:** For Assignment #5 my job is to modify the code submitted for assignment #4 and implement the database connectivity to my code and check for its **synchronization** and implement the design patterns that will help us to solve the challenges that can occur.

My main goal for this assignment is to apply the information I learned regarding the use of Synchronization to ensure that access to the shared resources are thread-safe and should find if there is a need to implement the patterns Monitor Object, Future, Guarded Suspension, Scoped Locking and Thread-Safe Interface to achieve the synchronization.

**Project Requirements**

The client desires an online marketplace where they can sell goods (and possibly services) to customers geographically dispersed around the world. Think Amazon but on a smaller scale and budget. Their desire is to have a system that is constructed in a portable language (Java) and makes use of their existing network. The system itself should present a view for the customer to interact with as well as a view for the employees or administrators of the company to interface with. Therefore, you will need to provide a separate interface for both Clients and Administrators. For the customer/client, there is a need for them to be able to browse available products – this should present the customer with the type, description and price of the item with the options to add to their shopping cart. If the customer attempts to add a quantity of the item more than the current supply the system should prevent the customer from adding these and prompt them with a message on the availability of the item. The customer should be able to also purchase their items from the shopping cart. This shopping cart should maintain state and be persistent through interactions with the application. The administrators should be able to update an item’s description within the system, update its price, and update its quantity. The administrator should also be able to remove items from the system if so desired. Administrators should be able to add other administrators as well as add/remove customer accounts. We will assume a default administrator account is already created – as administrators cannot register for accounts. On the other hand, a customer should be able to initially register for their account by themselves. We also assume that the roles are distinct and independent, meaning an Administrator cannot purchase items in that role but must have a Customer account to do so. The system should handle any faults or unexpected scenarios gracefully. It should be reliable and should allow for multiple customer requests during the course of execution. Any other questions or clarifications should be directed at the client representative (Me).

Specific actions/events that should be handled as part of the system:

• Registration (Customer)

• Login (Administrator/Customer)

• Browsing Items (Administrator/Customer)

• Updating Items (Administrator)

• Removing Items (Administrator)

• Adding Items (Administrator)

• Purchasing Items (Customer)

**Code:**

**Interface:** BestSale – BestSale interface extends Remote Class.

* BestSaleServer() – String method.
* Login() – int method.
* ReturnValidate ()– String method.
* addItems () – String method.
* updateItems() – String method.
* browseItems() – String method.
* removeItems() – String method.
* sessionLogin() – Session type.
* addToCart() – String type.

AbstractFactory –

* dispatchRequest() – void method – Attributes are parameterized attributes.

CommandController –

* execute() – void method

RequiresRole –

* RequiresRole – String type.

**Class:** BestSaleServer**:** extends BestSale interface.

* BestSaleServer() – Constructor
* Main() – Creates instance for BestSaleServer and binds it to another string name where Server name binding occurs. Throws

BestSaleClient:

* Main() – throws RemoteException. Creates an instance for the interface and binds it to the a string.
* Creates instances for Invoker class, BestSale, Registration, FrontController

BestSaleView:

* RegistrationDisplay() – void method: Attributes used are username, pwd, email. This class acts a view part for the project.
* LoginAuthentication() – void method: Attributes used are option, username, pwd.
* showView() – void method.
* addItems() – void method: Attributes used are name, price.
* browseItems() – void method: Attributes used are browse, name.
* addToCart() – void method: Attributes used are option2, name.
* updateItems() – void method: Attributes used are option2, updateName, updatePrice.
* removeItems() – void method. Attributes used are option2, deletedItem.

ClientDispatcher: implements AbstractFactory interface.

* ClientDispatcher () – Constructor
* dispatchRequest() – void method

FrontController:

* FrontController() – Constructor
* DispatchRequest() - void

AbstractClass:

* AbstractClass: returnPage() – method – Attributes are parameterized attributes.

Registration: implements CommandController

* Registration() - Constructor
* execute() – void method.

AdminDispatcher: Implements AbstractFactory interface.

* AdminDispatcher () – Parameterized Constructor.
* dispatchRequest() – void method – Attributes are parameterized attributes.

PageReturn: Extends AbstractClass class.

* returnPage() – Attributes are parameterized.

AdminView:

* showView – void method.

Invoker:

* addToList() – void method
* inTheList – void method

ServerImpl: extends UnicastRemoteObject

* ServerImpl() – Constructor
* Register() – String type.
* Login() – int method
* ReturnValidate() – String method
* SessionLogin() – Session type.
* addItems() – String type.

Session: implements Serializable

* Session() – Constructor
* User() – User type.

User: implements Serializable

* User() – Constructor
* getRoleType() – String type.

AuthorizationException: extends RuntimeException

* AuthorizationException() – Constructor

AuthorizationInvocationHandler: implements InvocationHandler, Serializable

* AuthorizationInvocationHandler() – Constructor
* Invoke() – Object type.

**Abstract Class:** AbstractClass:

* returnPage() – Attributes are parameterized.

**Project Overview:**

The online market place handles the following events**:**

• Login (Client/Admin) & Registration(Client)

• Browsing Items (Client/Admin)

• Updating Items (Admin)

• Removing Items (Admin)

• Adding Items (Admin)

• Purchasing Items (Client)

**Patterns Applied:**

**Front-Controller Pattern:**

It receives the request from login page and dispatches it to MVC to verify credentials. After getting the results it forwards it to dispatcher and dispatcher will return the appropriate page.

**Command Pattern:**

It declares an interface for all commands and provides a simple execute() method which carries out the operation to the Receiver. The Receiver knows what to do to carry out request. Invoker can get command to execute by calling execute method. Creating the command and setting a receiver to command will be done by client.

**Abstract Factory Pattern:**

It provides best way to create an object. It creates interface that creates a factory for both the objects without specifying their classes explicitly.

**Role Based Access Control Using Annotations:**

The privileges of particular role will be mapped to user.

**Reflection Pattern:**

It can inspect other code in the same system. An architecture that is open for modify and extension should be specified.

**Synchronization:**

An array list of active users is provided on server and it will be updated if users go online or offline. When a new connection is made a thread will be created and the array list will be updated to indicate the new user and same will happen for disconnection.

Multi threaded applications development is difficult since incorrect use of locks can lead to deadlock.

**Monitor Object:**

It allows threads to have mutual exclusion and block for a condition to be true and also a mechanism is provided to indicate other threads if the condition is satisfied

1. mutual exclusion
2. signaling

**Future:**

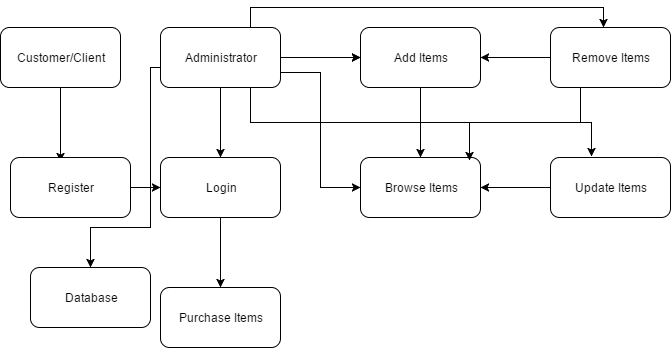
It allows a client to check the progress of process that is being executed on other thread. Future Object is used to check the status and retrieve the result.

**Other patterns like Guarded Suspension, Scoped Locking, Thread-Safe Interface can be implemented by using synchronized keyword.**

**Domain Model of Entire System:**



**Domain Model of Entire System:**



**Operational Flow: UserLogin Flow:**

****

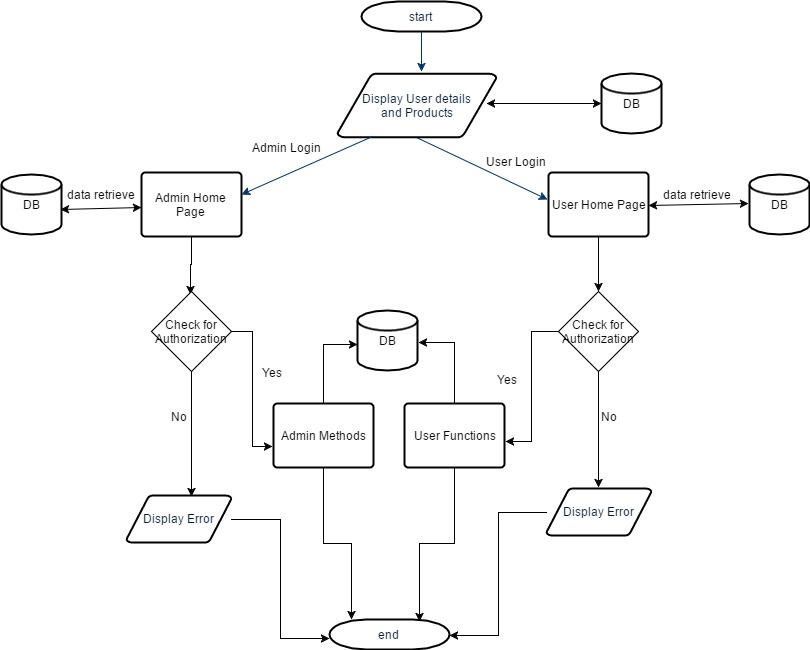
**AdminLogin Flow:**

****

**Registration Flow:**

****

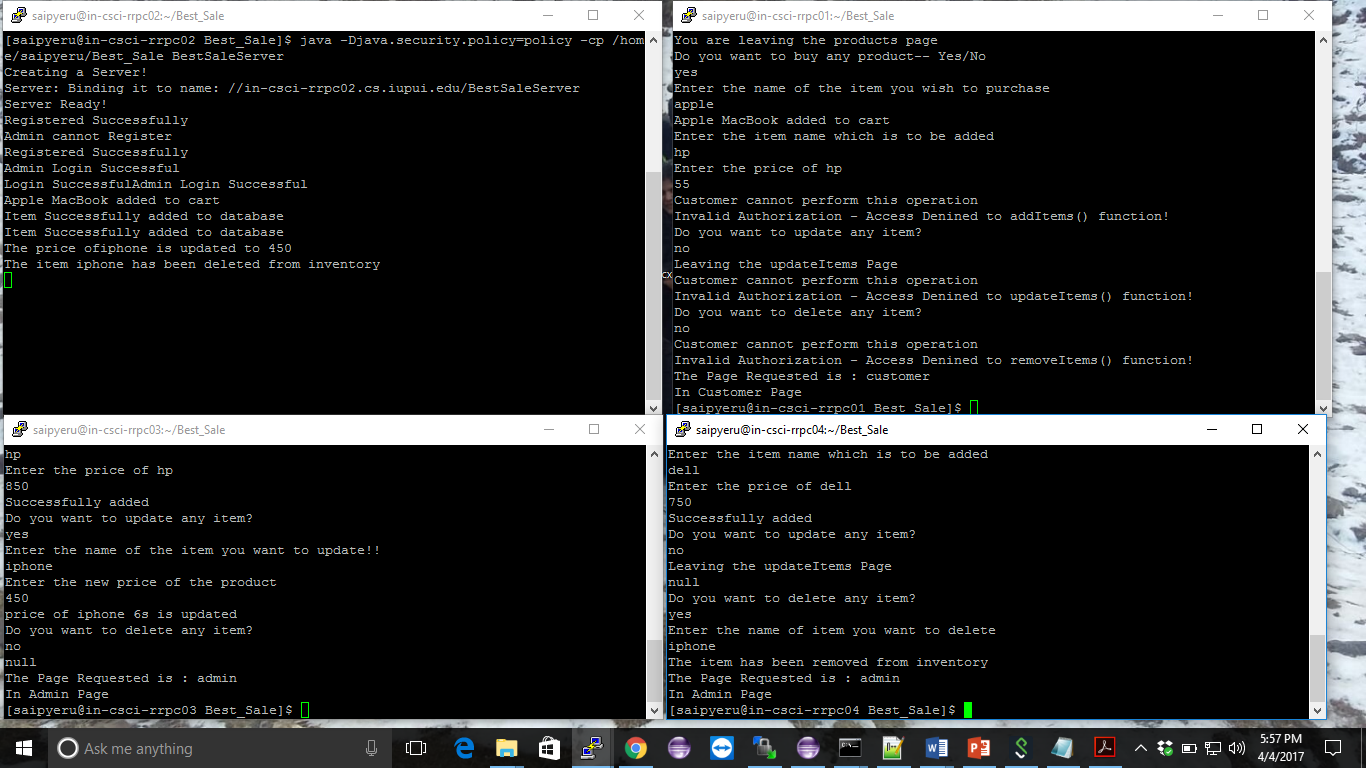
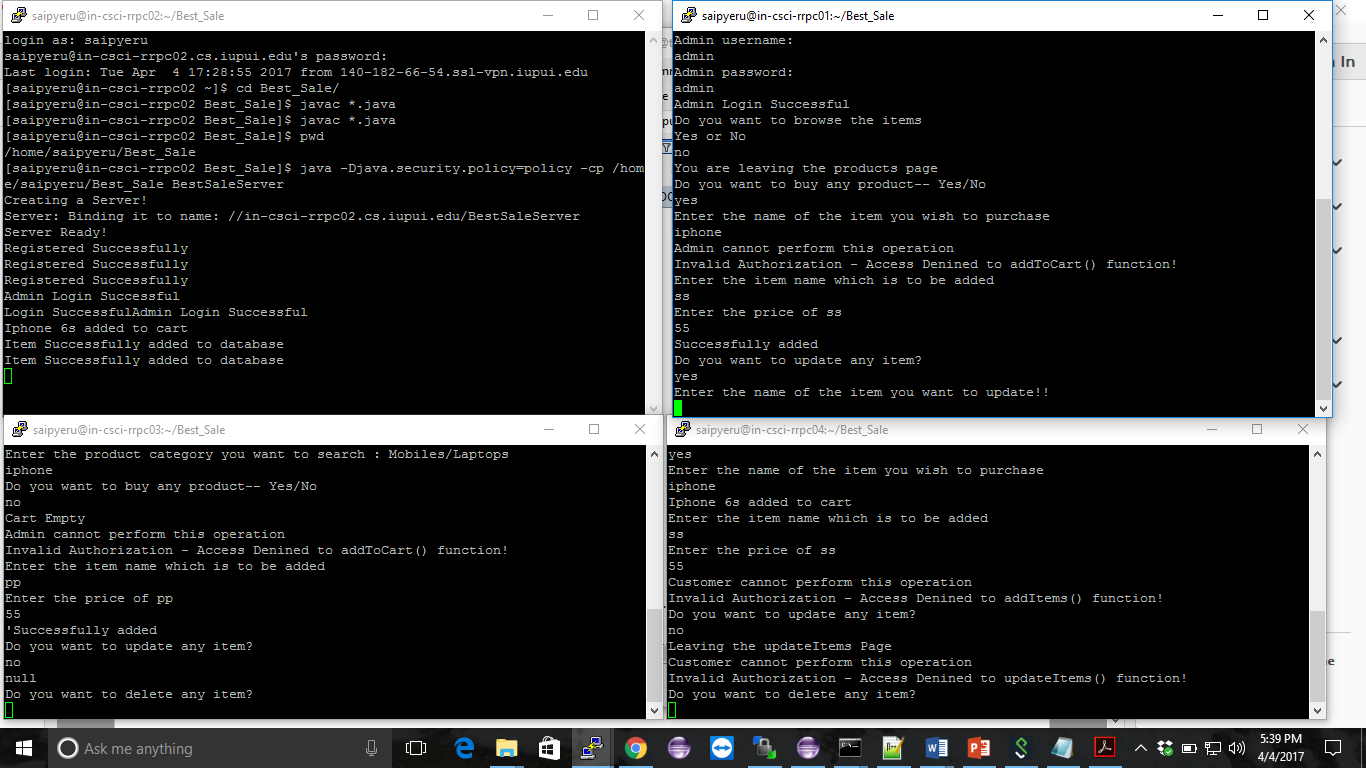
**Post Login Flow:**

****

**Java RMI Concurrency:**

1. Concurrency is the ability to run several programs or several parts of a program in parallel.
2. Concurrency and parallelism can be best described as – “Concurrency is about dealing with a lot of things at once and Parallelism is about doing lot of things at once”.
3. Concurrency in Java can occur by using ‘synchronized’ keyword. In Java synchronization can be achieved by using Thread Synchronization concept.
4. Java RMI provides the facility to implement the applications concurrently, but Java RMI makes no guarantees with respect to mapping remote object invocations to threads.
5. Java RMI cannot control the access of the remote objects.
6. Java RMI provides the communication between the server and the clients without repeatedly creating the new sockets.
7. Java RMI is multi-threaded and concurrency in Java RMI can be carried out by commuting between remote objects.
8. RMI applications are prone to dangers that might occur in concurrent applications, like the increase in number of customers can result in deadlock as the dependency between the remote objects increases.
9. Thread Concept is used to invoke a remote object multiple times on a concurrent basis by the different threads.
10. Synchronize keywords can be used for threads as well as processes to prevent the execution of more than one thread/process on the same object.
11. By using synchronized keyword, only one thread can access the remote object at a time and all the other threads will be in waiting state, the threads in waiting state will proceed to execute stage after the object is available or it will lead to deadlock.
12. RMI concurrency is not visible to the outside world but it has a strong impact on the working of our system.

**Sample Runs(of Assignment 5):**

****

