Overview

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 execution.

Aim: To wrap up the development of our Online Marketplace application and to ensure that all feedback given during the semester has been appropriately addressed in our code and that all required functionality is present and working in the submission. The focus here will be submitting a polished product (source code + report)

Requirements: The final assignment of the semester: online marketplace should handle the following basic events:

- Users /Administrators Login & Registration
- Browsing Items
- Updating Items
- Removing Items
- Adding Items
- Purchasing Items

Version Overviews

Version #0

To demonstrate a working application using Java RMI through an implementation of the MVC software design pattern

For this assignment, the working application uses Java RMI through an implementation of the MVC software design pattern. The Java RMI makes use of the Client-Server Architecture along with the three components of the MVC pattern: Model, View and Controller.

- *Model:* The model manages the functionality (logic) of the program.
- *View:* The view manages the user interface.
- *Controller:* The controller is the connection between view and model.

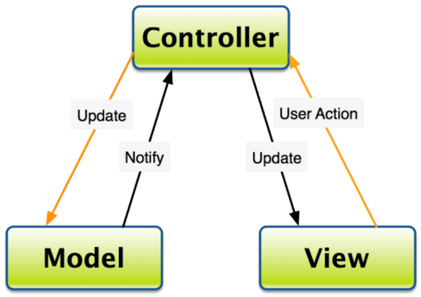


Image source

In this assignment, I made several mistakes and the main improvement made by me was *Separation of concerns i.e. each section will address a separate concern.* In my system, Model is responsible for handling the functionalities, View is responsible for the user interface and Controller is responsible for the connection between model and view.

Version #1

To implement the Front Controller pattern as part of our existing Marketplace application along with partial implementation of the logic for the Login requirement. The function should be included for both Administrators as well as Customers with distinct views along with the use of Command and Abstract Factory patterns

Here, I have made use of the following patterns:

- Front Controller Pattern
- Command Pattern
- Abstract Factory Pattern

Front Controller Pattern: It consists of 3 entities: Front Controller, Dispatcher and View. In my implementation, the MarketplaceFrontController is the entry point of the application. It handle's all the client request coming into the application. Request in our case is the login, user input's his login credentials into the application and the front controller uses a dispatcher object by creating an instance of the MarketplaceDispatcher class which then dispatches the request to the model for authentication.

Command Pattern: In command pattern, invoker plays an important role. The invoker handles the command and manages execution of the command. The main question asked in command pattern is what is to be done? In my implementation, I have tried to specify the same question what? In this assignment the question is what view should be displayed. For this purpose, I have created an Invoker class that has commands to display the views for customer and administrator. The Invoker class commands to call for the view based on the user type.

Abstract Factory Pattern: According to the Gang of Four book the abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. In my implementation, the AbstractFactory defines an interface UserPage that all the concrete factories need to implement. The concrete factories in my case are: CustomerFactory and AdminFactory, both the class have implemented the UserPage interface, creating two separate families. The AbstractFactory has two abstract methods which are implemented in the MarketplaceDispatcher. The MarketplaceDispatcher acts as the factory producer and thus, on implementing this we achieve to get two distinct views for each of the users.

Version #2

To expand the application to make use of the Authorization pattern through a role-based access control (RBAC) approach that will make use of Java Annotations. As part of this process we will also implement and explore further the Proxy pattern and the Reflection pattern.

Here, I have made use of the following patterns:

- Reflection Pattern
- Authorization Pattern
- Proxy Pattern

Reflection Pattern: Reflection pattern is an architectural pattern. It provides a mechanism for changing the structure and behavior of the system dynamically. In this assignment, I have tried to implement the reflection by creating a customized java annotation for role-based access control. The RequriesRole.java is the java file responsible for the customized java annotation.

Authorization Pattern: Authorization pattern is a structural pattern. In this assignment, I have tried to implement the authorization pattern for providing role-based access control. AuthorizationInvocationHandler.java file is responsible for implementing the role-based access control. AuthorizationException.java file is used to create a customized exception for invalid authorization. I have made use of Session.java to create the session and UserType.java to maintain the role (admin or customer) and implement session in the application, thus avoiding individual rights and setting associate rights based on the roles within the system.

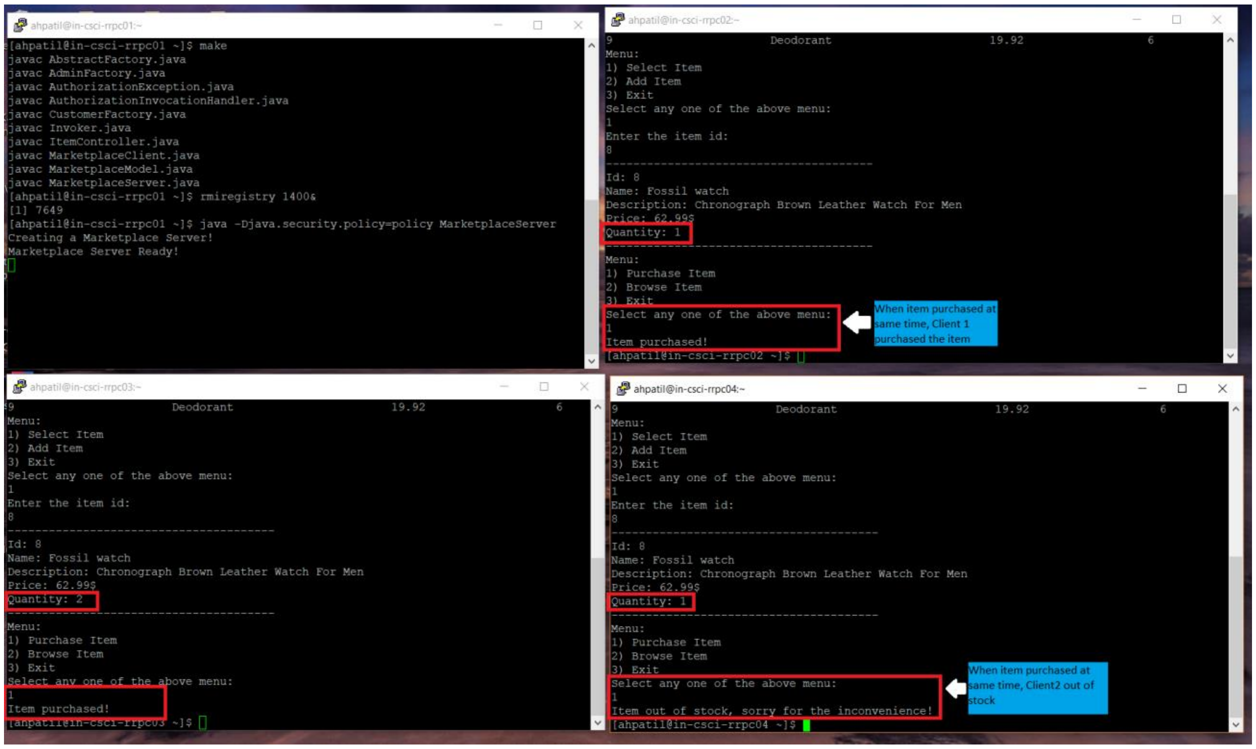
Proxy Pattern: I have tried to implement this pattern by using the Java Dynamic Proxy along with the customized annotation. I have created an instance in MarketplaceServer.java which acts as a pass through to the real object based on the runtime implementations of Marketplace interface.

Version #3
To implement previously unimplemented functionality in our system as well as to explore the consequences of concurrency in our Marketplace Application.

Here, the main objective was to observe the concurrency in the application. For this purpose, I have made use of the following machines:

- in-csci-rrpc01.cs.iupui.edu - 10.234.136.55 (Server)
- in-csci-rrpc02.cs.iupui.edu - 10.234.136.56
- in-csci-rrpc03.cs.iupui.edu - 10.234.136.57
- 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

A thread is an execution context which can run together with other threads in the same environment. Threads in java play an important role for the internal working of Java RMI. In previous assignment I noticed that the system still works well even when multiple clients request the server to execute their queries at same time. This is because every time when a RMI call comes in, the server- thread will generate a new thread which handles the request. Thus, this java threading mechanism in Java RMI makes it possible for one remote object to be concurrently invoked multiple times by different threads.


ahpatil@in-csci-rrpc01:~
ahpatil@in-csci-rrpc02:~
ahpatil@in-csci-rrpc03:~
ahpatil@in-csci-rrpc04:~
When item purchased at same time, Client 1 purchased the item
When item purchased at same time, Client2 out of stock

```
[ahpatil@in-csci-rrpc01 ~]$ make
javac AbstractFactory.java
javac AdminFactory.java
javac AuthorizationException.java
javac AuthorizationInvocationHandler.java
javac CustomerFactory.java
javac Invoker.java
javac ItemController.java
javac MarketplaceClient.java
javac MarketplaceModel.java
javac MarketplaceServer.java
[ahpatil@in-csci-rrpc01 ~]$ rmiregistry 1400&
[1] 7649
[ahpatil@in-csci-rrpc01 ~]$ java -Djava.security.policy=policy MarketplaceServer
Creating a Marketplace Server!
Marketplace Server Ready!
```

```
9                      Deodorant                      19.92                 6
Menu:
1) Select Item
2) Add Item
3) Exit
Select any one of the above menu:
1
Enter the item id:
8
---------------------------------------
Id: 8
Name: Fossil watch
Description: Chronograph Brown Leather Watch For Men
Price: 62.99$
Quantity: 1
---------------------------------------
Menu:
1) Purchase Item
2) Browse Item
3) Exit
Select any one of the above menu:
1
Item purchased!
[ahpatil@in-csci-rrpc02 ~]$
```

```
9                      Deodorant                      19.92                 6
Menu:
1) Select Item
2) Add Item
3) Exit
Select any one of the above menu:
1
Enter the item id:
8
---------------------------------------
Id: 8
Name: Fossil watch
Description: Chronograph Brown Leather Watch For Men
Price: 62.99$
Quantity: 2
---------------------------------------
Menu:
1) Purchase Item
2) Browse Item
3) Exit
Select any one of the above menu:
1
Item purchased!
[ahpatil@in-csci-rrpc03 ~]$
```

```
9                      Deodorant                      19.92                 6
Menu:
1) Select Item
2) Add Item
3) Exit
Select any one of the above menu:
1
Enter the item id:
8
---------------------------------------
Id: 8
Name: Fossil watch
Description: Chronograph Brown Leather Watch For Men
Price: 62.99$
Quantity: 1
---------------------------------------
Menu:
1) Purchase Item
2) Browse Item
3) Exit
Select any one of the above menu:
1
Item out of stock, sorry for the inconvenience!
[ahpatil@in-csci-rrpc04 ~]$
```

In figure above, we notice this concurrent behavior of Java RMI wherein, *customer1* and *customer* both try to purchase the item whose quantity is 1 concurrently. Since, *customer1* purchases it slightly prior to *customer* the item is purchased by *customer1* and for *customer* it displays item out of stock. Thus, through this test I have observed that RMI is multi-threaded, allowing the server to exploit Java threads for better concurrent processing of client requests
The only concern here is that the state of the remote object will soon be invalid if not provided with the guard access to the remote method. In scenario, when I ran the on two physically different desktops and tried to purchase item whose quantity is 1 at same time, both customers were able to purchase the item as it couldn't synchronize the request. Thus, doesn't ensure thread safety. This can be achieved by the *synchronized* keyword in Java. We will discuss about this in more detail in this assignment.

Version #4

To examine the impact that the concept that Synchronization has on our Marketplace Application. In Assignment 5, the main objective was to observe the synchronization in the application. For this purpose, I have made use of the following machines:

- in-csci-rrpc01.cs.iupui.edu - 10.234.136.55 (*Server and MySQL Database*)
- in-csci-rrpc02.cs.iupui.edu - 10.234.136.56
- in-csci-rrpc03.cs.iupui.edu - 10.234.136.57
- 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

The server runs on the 10.234.136.55 machine.
In this assignment, I'll be discussing the following patterns

- Monitor Object Pattern
- Future Pattern
- Guarded Suspension Pattern
- Scoped Locking Pattern
- Thread-Safe Interface Pattern

In our application, the main problem is when multiple customers try to access the same object or same resource from the server and in return the server is not able distinguish between the multiple service requests which results in error result. So, it is very important to ensure synchronization in the application so that all requests are serviced by the server accurately additionally, providing concurrency in the system. Thus, it is very important to ensure that only one thread can access the resource at a given instance of time.
This can be achieved using "*synchronized*" keyword in Java. For this purpose, I have made use of synchronized blocks. Synchronized blocks provide a way by which only one thread can attempt to enter the synchronized block and until it exits the block no other thread is permitted to enter the synchronized block. Using this we can achieve both concurrency and thread safety in our system.
Following is the code snippet from our marketplace application source code:

```
//To purchase the items
@Override
public synchronized String purchase(Session session, int id) throws RemoteException{
   ResultSet resultSet;
```

```
    int quantity=0;
    // Synchronized block
    synchronized (this){
            try {
                resultSet = databaseConnection.getItem(id);
            while(resultSet.next()){
                quantity = Integer.parseInt(resultSet.getString("itemQuantity"));
                }
            resultSet.beforeFirst();
            if(quantity>0) {
                quantity--;
                databaseConnection.updateQuantity(quantity, id);
                return "Item Purchased!";
                }
            } catch (SQLException e) {
                System.out.println("Purchase:: Result set error");
            }
        }
        return "Item out of stock!";
}
```

Thus, using this synchronized block only one thread can access the share variable and other shared resource at a given time

Monitor Object Pattern: This pattern helps in the execution of multi-threaded applications. It ensures that only one thread can access the shared resource at a given time, all the other threads attempting to access the same resource are blocked until the time thread is accessing the resource. It consists of three components: Monitor Object, Monitor Condition and Monitor Lock. The monitor uses the lock to ensure that only a single thread is active in the monitor at a given time, it does this using the Monitor Lock *acquire()* and releases the lock when thread completes it's execution using the *release()*. While, the monitor object is locked it prevents other threads from accessing the lock by using the Monitor Conditions: *wait()*, *notify()* and *notifyAll()*.
In this assignment, I have achieved this by using the *synchronized* keyword by implementing the *synchronized* blocks and methods at places where I think the lock acquire and release mechanism is needed. The *synchronized* keyword by Java does this for us.

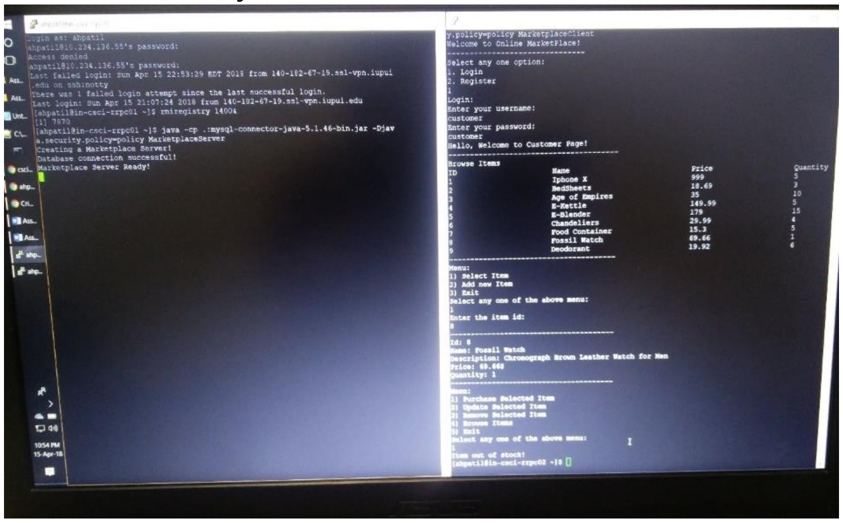
Future Pattern: In a single threaded application when we call a method it returns only after the computation is successfully done. At times, this call takes arbitrary long time to compute and thus increases the latency. If a we get a way to do other independent processing while the called method is being computed it might reduce the overall latency. This can be achieved using Future Pattern. Using this pattern, you don't need to wait for the results of the called method rather you get a virtual object which promises to be available in future and can be kept as reference until we are able retrieve it. In simple terms, Future is a proxy for an object that is not yet there.
In this assignment, I have achieved this by using the *synchronized* keyword by implementing the at places where I found the need of future object. This keyword informs java whether it needs to synchronize the method or block and provides a virtual object for referencing and notifies the user when on availability of result. The *synchronized* keyword by Java does all of this for us

Guarded Suspension Pattern: This pattern is implemented to allow only one thread to access the shared resource and blocks the other threads from accessing the same resource until it is being used the thread. This pattern requires both a lock to be acquired and a pre-condition to be satisfied

unlike monitor object pattern. It is used when a method call will be suspended for a finite period. To assist the guarded suspension we have *wait()* and *notify()* methods.
In this assignment, I have achieved this by using the *synchronized* keyword by implementing the *synchronized* blocks and methods at places where I think the lock and pre-conditions are required to be satisfied by the thread. If the conditions are not satisfied the thread moves to the wait state and once the conditions are met the threads waiting are notified. The *synchronized* keyword by Java does this for us.

Scoped Locking Pattern: Imagine, a scenario if the programmer forgets to release the lock from a method or block. In this case the even after the thread finishes its execution the object will be locked, and no other thread will be able to access the resource. Thus, we require a mechanism using which we ensure that a lock should always be acquired and released properly when control enters and leaves critical sections. We can do this using the scope locking pattern, we define a guard class that acquires and releases a particular lock automatically within a method or block scope.
In this assignment, I have made use of *synchronized* block in Java at some places, using this the compiler generates the corresponding byte code that includes the instructions *monitorenter* and *monitorexit*. An exception handler is also created to handle those cases when unexpected termination occurs.

Thread-Safe Interface Pattern: This pattern minimizes locking overhead and ensures that intra-component method calls do not incur self- deadlock by trying to reacquire a lock that is held by the component already. In this assignment, I have made use of *synchronized* keyword in Java at top of all methods which ensures that there is no deadlock condition. The *synchronized* keyword by Java does provides thread safety for us.

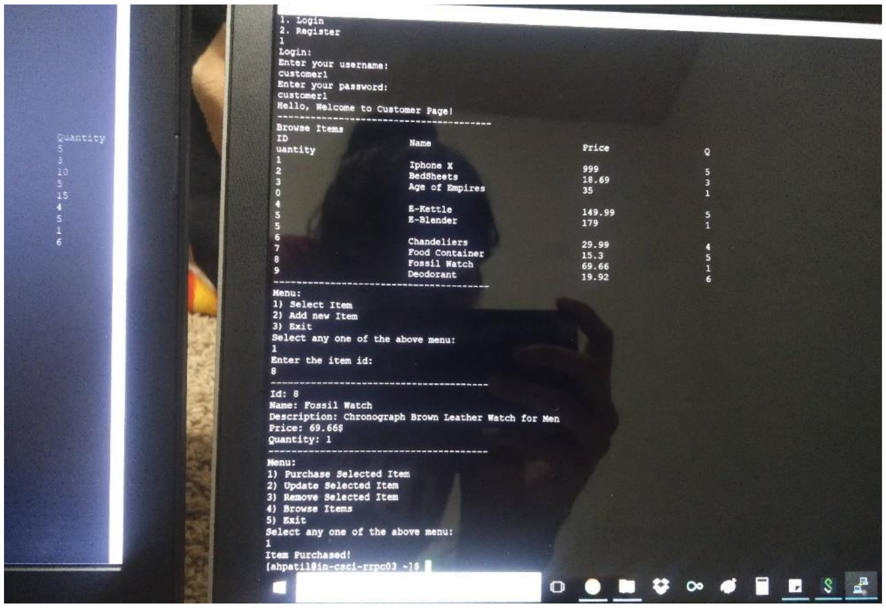


```
login as: ahpatil
ahpatil@10.234.136.55's password:
Access denied
ahpatil@10.234.136.55's password:
Last failed login: Sun Apr 15 22:53:29 EDT 2018 from 140-182-67-19.ssl-vpn.iupui
.edu on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Sun Apr 15 21:07:24 2018 from 140-182-67-19.ssl-vpn.iupui.edu
[ahpatil@in-csci-rrpc01 ~]$ rmiregistry 14004
[1] 7870
[ahpatil@in-csci-rrpc01 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceServer
Creating a Marketplace Server!
Database connection successful!
Marketplace Server Ready!
```

```
y.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
-----------------------------
Select any one option:
1. Login
2. Register
1
Login:
Enter your username:
customer
Enter your password:
customer
Hello, Welcome to Customer Page!
-----------------------------------
Browse Items
                      Name                          Price                 Quantity
ID
1                     Iphone X                      999                   5
2                     BedSheets                     18.69                 3
3                     Age of Empires                35                    10
4                     E-Kettle                      149.99                5
5                     E-Blender                     179                   15
6                     Chandeliers                   29.99                 4
7                     Food Container                15.3                  5
8                     Fossil Watch                  69.66                 1
9                     Deodorant                     19.92                 6
-----------------------------------
Menu:
1) Select Item
2) Add new Item
3) Exit
Select any one of the above menu:
1
Enter the item id:
8
-----------------------------------
Id: 8
Name: Fossil Watch
Description: Chronograph Brown Leather Watch for Men
Price: 69.668
Quantity: 1
-----------------------------------
Menu:
1) Purchase Selected Item
2) Update Selected Item
3) Remove Selected Item
4) Browse Items
5) Exit
Select any one of the above menu:
1
Item out of stock!
[ahpatil@in-csci-rrpc02 ~]$
```

I have applied this *synchronized* keyword on various places, which ensures that the system runs accurately providing thread synchronization and concurrency. Thus, this demonstrates the implementation of ***Monitor Object Pattern, Future Pattern, Guard Suspension Pattern, Scope Locking Pattern*** and ***Thread-Safe Interface Pattern*** using *synchronized* keyword.

The above image shows the server running on machine 10.234.136.55 and client running on other machines. I have opened two instances of putty where *customer* and *customer1* have logged into the marketplace application. Now, *customer* and *customer1* select item fossil whose quantity is 1 and both try to purchase the item at same time. Since, we are making use of the *synchronized* keyword, only a single thread can access the shared resource at a given time and thus, *customer1* is able to purchase the item whereas for *customer* it displays out of stock.



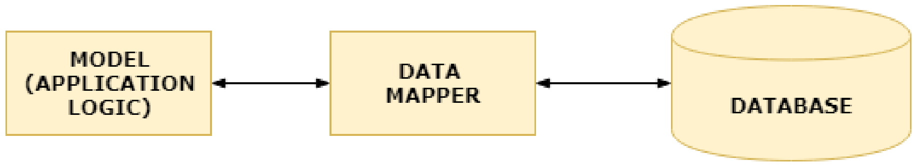
```
1. Login
2. Register
1
Login:
Enter your username:
customer1
Enter your password:
customer1
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID
uantity                  Name                        Price              Q
1
2                        Iphone X                    999                5
3                        BedSheets                   18.69              3
0                        Age of Empires              35                 1
4
5                        E-Kettle                    149.99             5
5                        E-Blender                   179                1
6
7                        Chandeliers                 29.99              4
8                        Food Container              15.3               5
9                        Fossil Watch                69.66              1
                         Deodorant                   19.92              6
---------------------------------------
Menu:
1) Select Item
2) Add new Item
3) Exit
Select any one of the above menu:
1
Enter the item id:
8
---------------------------------------
Id: 8
Name: Fossil Watch
Description: Chronograph Brown Leather Watch for Men
Price: 69.66$
Quantity: 1
---------------------------------------
Menu:
1) Purchase Selected Item
2) Update Selected Item
3) Remove Selected Item
4) Browse Items
5) Exit
Select any one of the above menu:
1
Item Purchased!
[ahpatil@in-csci-rrpc03 ~]$
```

I have implemented the *register* functionality which allows new users to register into the marketplace application. Additionally, I have included the *update* and *remove* functionalities. Update allows the admin to update the item from inventory and *remove* allows the admin to remove the selected item from the inventory. I have added the *cart*, which allows customer to add items into the cart and eventually purchase all items from the cart. The customer can even select an item from the browse list and purchase that selected item without adding it into the cart. Moreover, I have switched from using text file as storage to MySQL database for storage. The *DatabaseConnection.java* file is responsible for establishing the connection to the database.

Discussion:

In this assignment, our major focus is over implementing the entire functionality of the online marketplace application and to implement the Database Access Layer.

Database Access Layer: This layer helps to provide access to data stored in persistent storage. In our case, we have stored the data on MySQL database which is installed on machine in-csci-*rrpc01.cs.iupui.edu - 10.234.136.55*. This layer helps to efficiently access database in a distributed object-oriented application.
In this assignment, I have implemented this layer using the *DatabaseMapper.java* class. The *MarketplaceModel.java* consist of methods that are used to perform the functionalities like login, register, update item, purchase item, remove item, browse, etc. All these methods require database access to complete their functionalities. I have another class file: *DatabaseConnection.java* class file which is responsible for establishing database connection. I have put the *DatabaseMapper.java* class i.e. the data mapper class between the model and database connection.



This data mapper class receives the necessary parameters to perform the functionality from model and gets the data object from database connection. It then does all the validation and application functionality and return the respective results/ action on both side i.e. incase of purchase it receives the item id and quantity from model and then receives the result set from database. It checks whether the quantity for that item id is enough to make a purchase. If the item has enough quantity, it informs the database connection to execute update for that item and return item purchased successfully to the model. This is how I have implemented the Database Access Layer in my marketplace application, it decouples the application logic from the database functionality and all the mapping between objects and tables are encapsulated within the *DatabaseMapper.java* class.

<u>Overview of the entire marketplace application:</u>

- Functionalities:
    - Login & Registration
    - Purchase item
    - Update item
    - Add item
    - Remove item
    - Browse
    - Cart
- Role-based authorization: This allows only role specific actions i.e.
    - Admin: Login and Registration, Add item, Update item, Remove item, Browse

    - Customer: Login and Registration, Purchase item, Cart, Browse
- Database: I have used MySQL database to provide database storage and access which is installed on machine in-csci-*rrpc01.cs.iupui.edu - 10.234.136.55*
- Distributed application: This application is distributed in nature and provides concurrency and synchronization. It allows multiple users to connect concurrency and helps to maintain synchronization in the system.
- Design Patterns: To design the application following patterns are used:
    - MVC Pattern
    - Front Controller Pattern
    - Command Pattern
    - Abstract Factory Pattern
    - Reflection Pattern
    - Authorization Pattern
    - Proxy Pattern
    - Monitor Object Pattern
    - Future Pattern
    - Guarded Suspension Pattern
    - Scoped Locking Pattern
    - Thread-Safe Interface Pattern
    - Database Access Layer

Domain Model

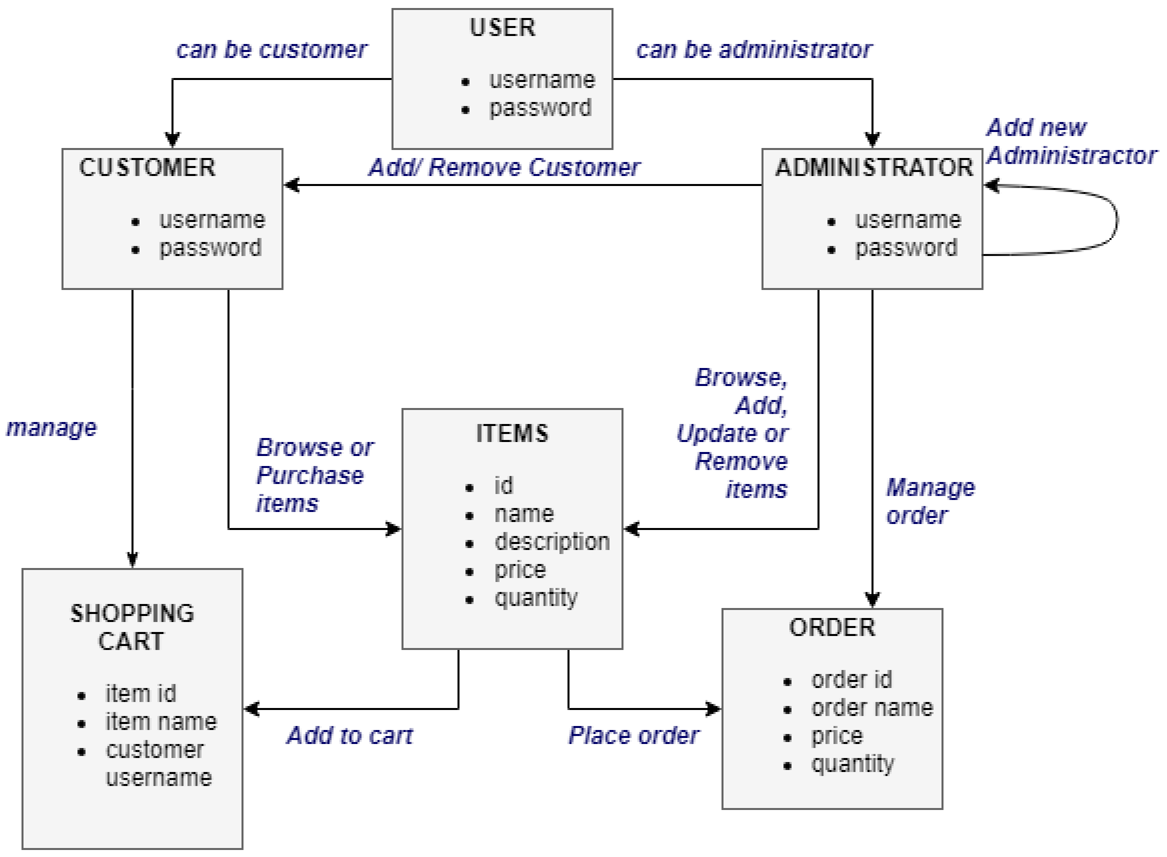


Fig (1) Domain Model for Online Marketplace

Each entity in the above domain model has a role to play in this Online Marketplace. User could be a customer or administrator. Each of the user-type has its own unique username for identification purposes. The user is authenticated on basis of the username password combination, correct username and password results into successful login. If the customer is new to the marketplace s/he can register and manage all the customer activities. A customer can browse or purchase items whereas, an administrator can browse, add, update or remove items. Each of the user-type has its own role in this marketplace.

UML

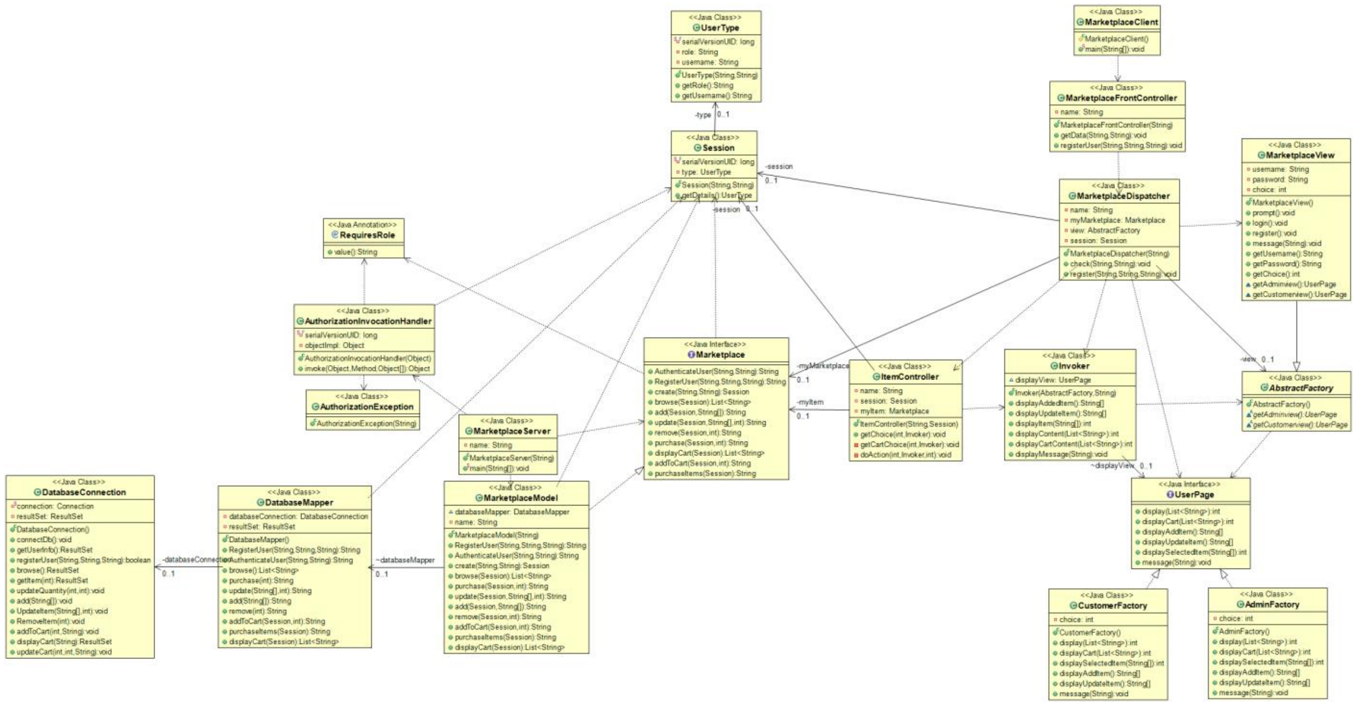
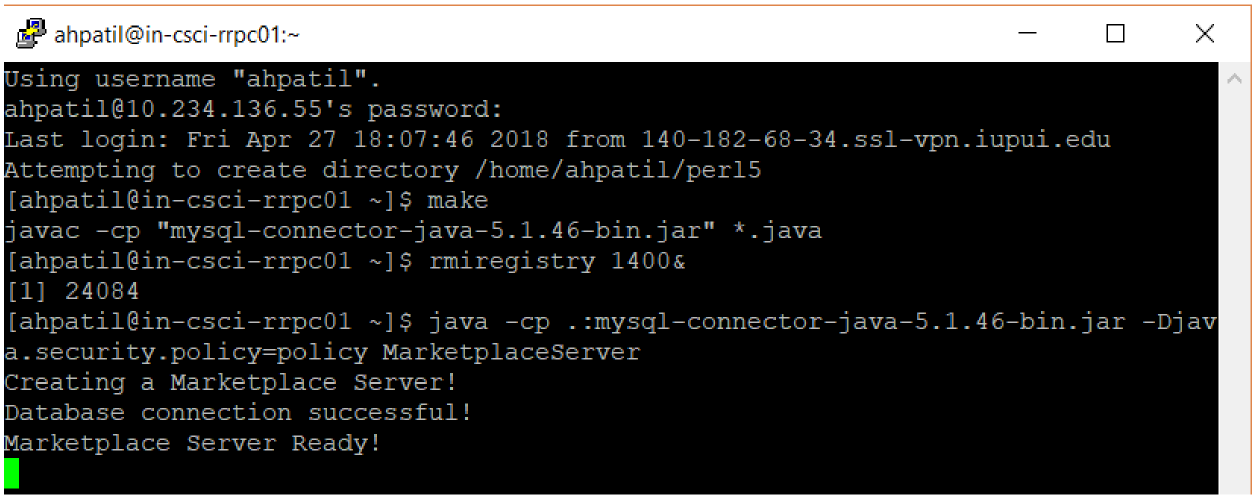


Fig (2) UML for Online Marketplace

Sample Runs

Server:

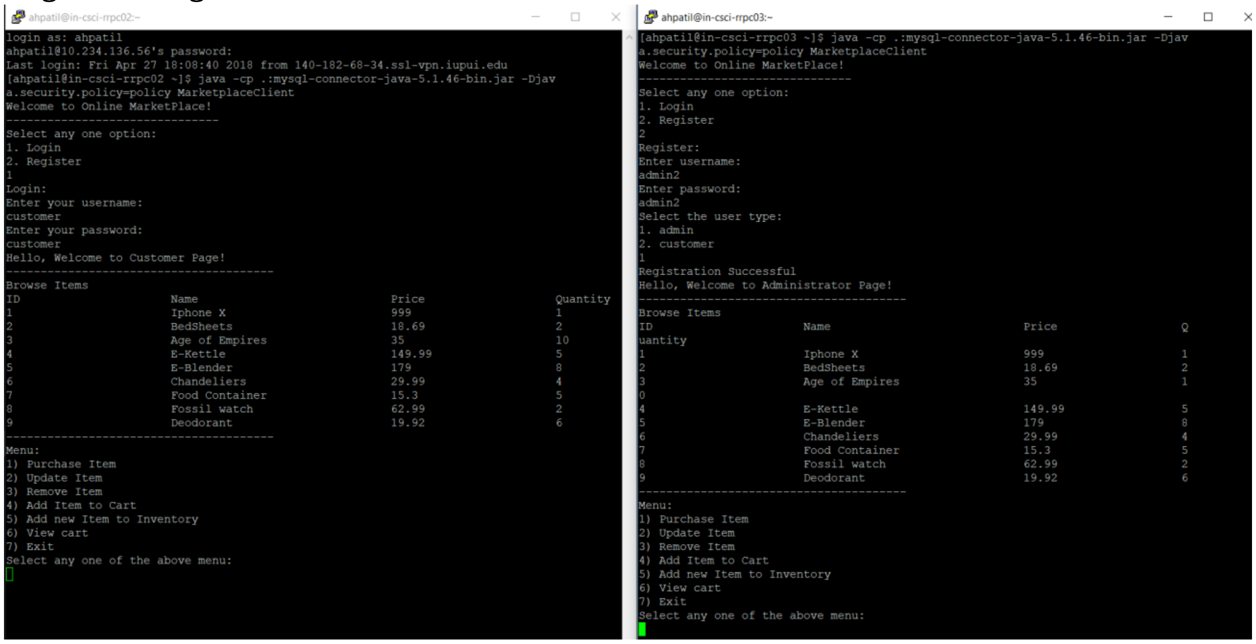


```
Using username "ahpatil".
ahpatil@10.234.136.55's password:
Last login: Fri Apr 27 18:07:46 2018 from 140-182-68-34.ssl-vpn.iupui.edu
Attempting to create directory /home/ahpatil/perl5
[ahpatil@in-csci-rrpc01 ~]$ make
javac -cp "mysql-connector-java-5.1.46-bin.jar" *.java
[ahpatil@in-csci-rrpc01 ~]$ rmiregistry 1400&
[1] 24084
[ahpatil@in-csci-rrpc01 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceServer
Creating a Marketplace Server!
Database connection successful!
Marketplace Server Ready!
```

Fig (3) Registration, user already exist

Server running on machine 10.234.136.55 and establishing database connection

Login and Registration:

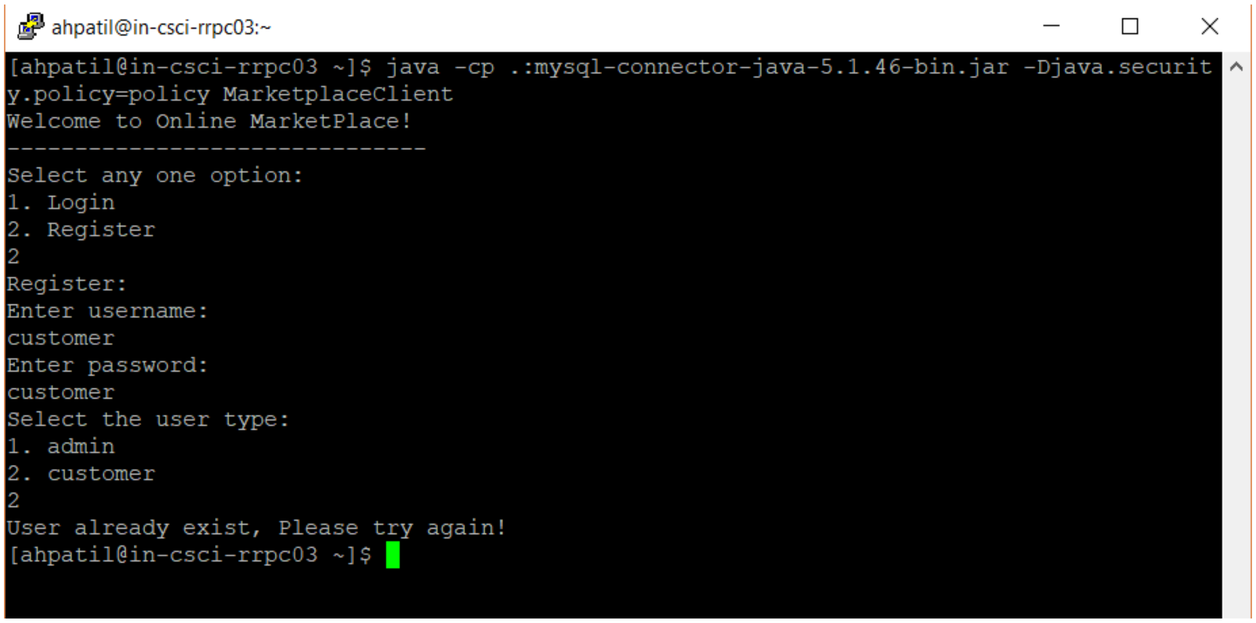


```
login as: ahpatil
ahpatil@10.234.136.56's password:
Last login: Fri Apr 27 18:08:40 2018 from 140-182-68-34.ssl-vpn.iupui.edu
[ahpatil@in-csci-rrpc02 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
-------------------------------
Select any one option:
1. Login
2. Register
1
Login:
Enter your username:
customer
Enter your password:
customer
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      10
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

```
[ahpatil@in-csci-rrpc03 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
-------------------------------
Select any one option:
1. Login
2. Register
2
Register:
Enter username:
admin2
Enter password:
admin2
Select the user type:
1. admin
2. customer
1
Registration Successful
Hello, Welcome to Administrator Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Q
uantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      1
0
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (4a) Login (left) and Registration (right)

Login into the system with already registered login credentials (left) and Registering as a new user (right) of type admin.

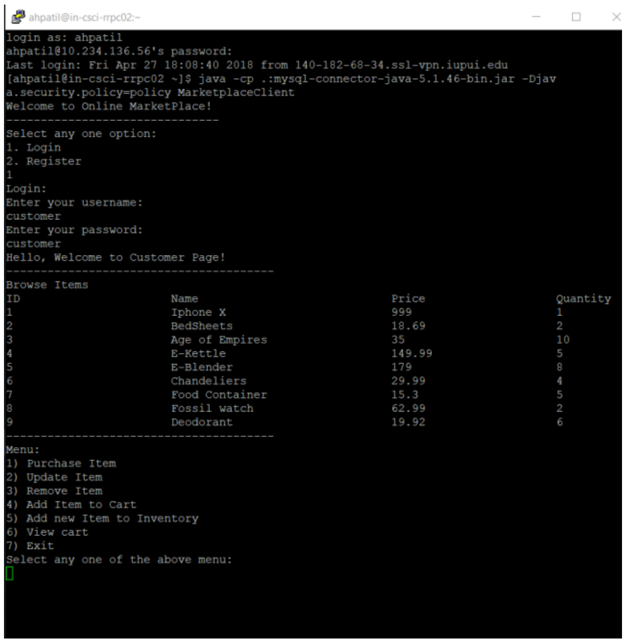


```
[ahpatil@in-csci-rrpc03 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djava.securit
y.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
-------------------------------
Select any one option:
1. Login
2. Register
2
Register:
Enter username:
customer
Enter password:
customer
Select the user type:
1. admin
2. customer
2
User already exist, Please try again!
[ahpatil@in-csci-rrpc03 ~]$
```

Fig (4b) Registration, user already exist

If user already exist it prompts "User already exist" and doesn't allow the user to register.

Browse Item:

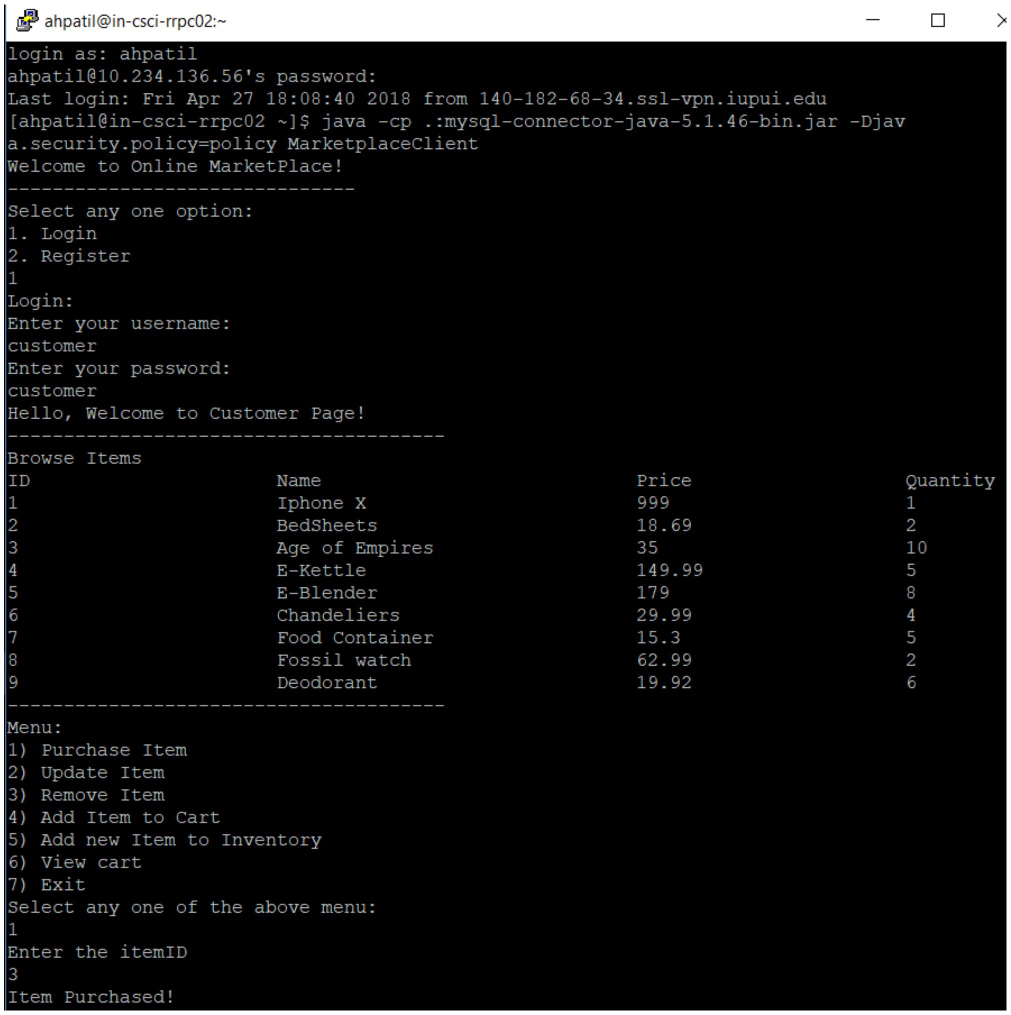


```
login as: ahpatil
ahpatil@10.234.136.56's password:
Last login: Fri Apr 27 18:08:40 2018 from 140-182-68-34.ssl-vpn.iupui.edu
[ahpatil@in-csci-rrpc02 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
-------------------------------
Select any one option:
1. Login
2. Register
1
Login:
Enter your username:
customer
Enter your password:
customer
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                       Name                            Price                    Quantity
1                        Iphone X                        999                      1
2                        BedSheets                       18.69                    2
3                        Age of Empires                  35                       10
4                        E-Kettle                        149.99                   5
5                        E-Blender                       179                      8
6                        Chandeliers                     29.99                    4
7                        Food Container                  15.3                     5
8                        Fossil watch                    62.99                    2
9                        Deodorant                       19.92                    6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (5) Browse items

An item list is displayed to the user for browsing and displays menu for the user.

Purchase Item:

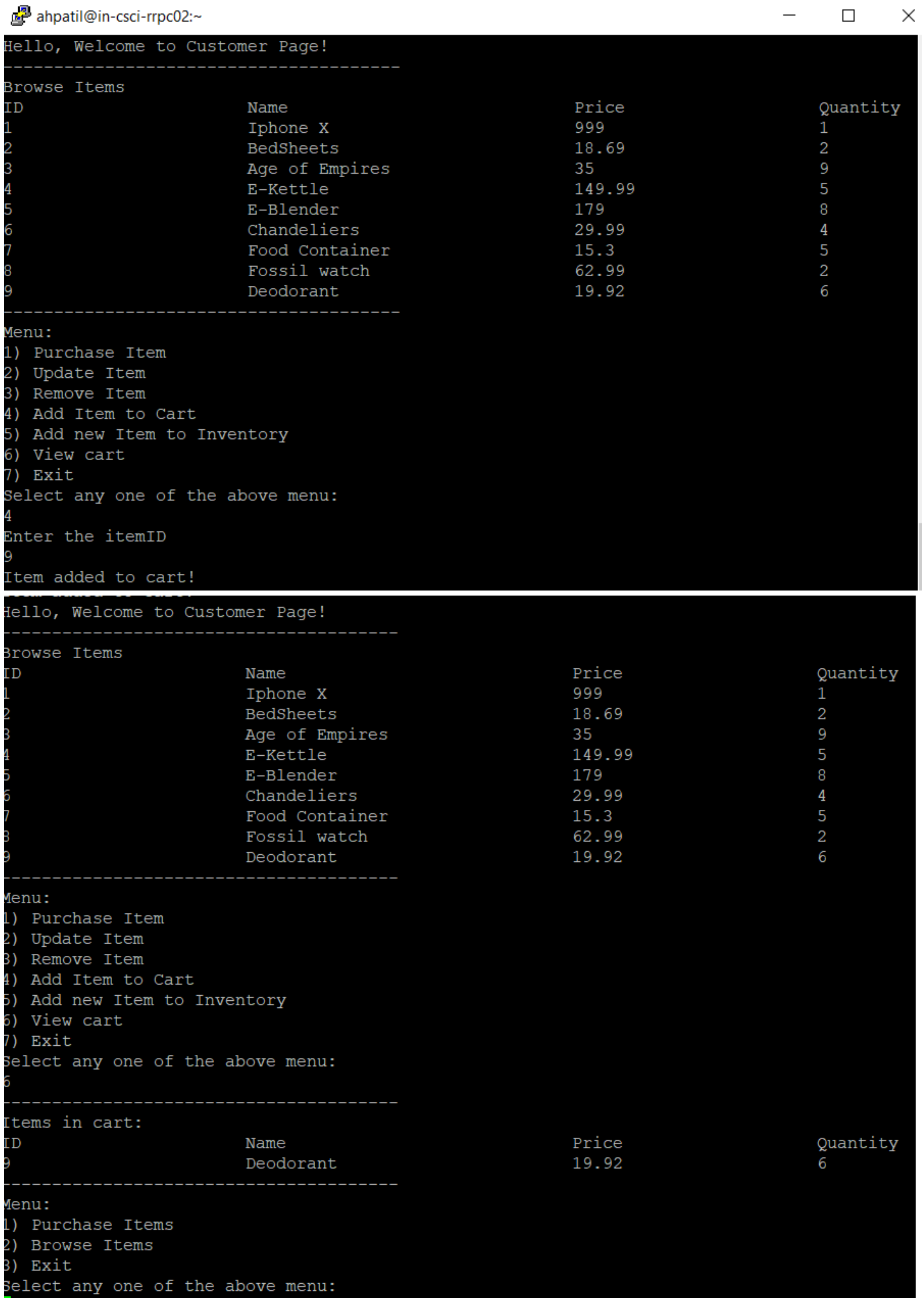

ahpatil@in-csci-rrpc02:~

```
login as: ahpatil
ahpatil@10.234.136.56's password:
Last login: Fri Apr 27 18:08:40 2018 from 140-182-68-34.ssl-vpn.iupui.edu
[ahpatil@in-csci-rrpc02 ~]$ java -cp .:mysql-connector-java-5.1.46-bin.jar -Djav
a.security.policy=policy MarketplaceClient
Welcome to Online MarketPlace!
------------------------------
Select any one option:
1. Login
2. Register
1
Login:
Enter your username:
customer
Enter your password:
customer
Hello, Welcome to Customer Page!
--------------------------------------
Browse Items
ID                       Name                            Price                   Quantity
1                        Iphone X                        999                     1
2                        BedSheets                       18.69                   2
3                        Age of Empires                  35                      10
4                        E-Kettle                        149.99                  5
5                        E-Blender                       179                     8
6                        Chandeliers                     29.99                   4
7                        Food Container                  15.3                    5
8                        Fossil watch                    62.99                   2
9                        Deodorant                       19.92                   6
--------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
1
Enter the itemID
3
Item Purchased!
```

Fig (6) Purchase Item

This functionality is authorized for the customer role type and here the customer selected item id 3 and purchased it. Since the item was in stock the customer was able to purchase it.

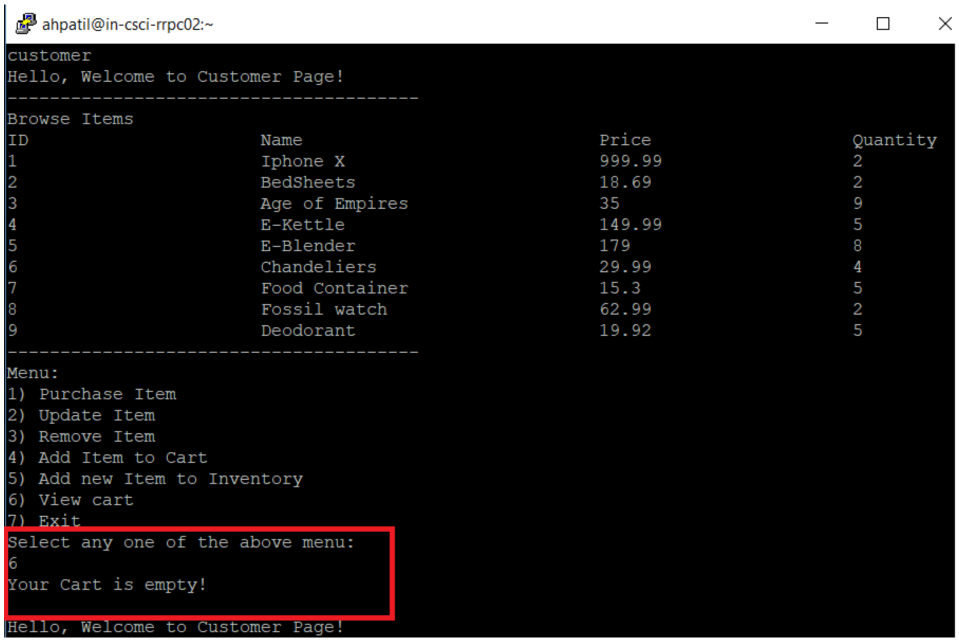
Cart Functionalities:



```
ahpatil@in-csci-rrpc02:~
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
4
Enter the itemID
9
Item added to cart!
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
6
---------------------------------------
Items in cart:
ID                      Name                            Price                   Quantity
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Items
2) Browse Items
3) Exit
Select any one of the above menu:
```

Fig (7a) Add item to the cart and view the cart

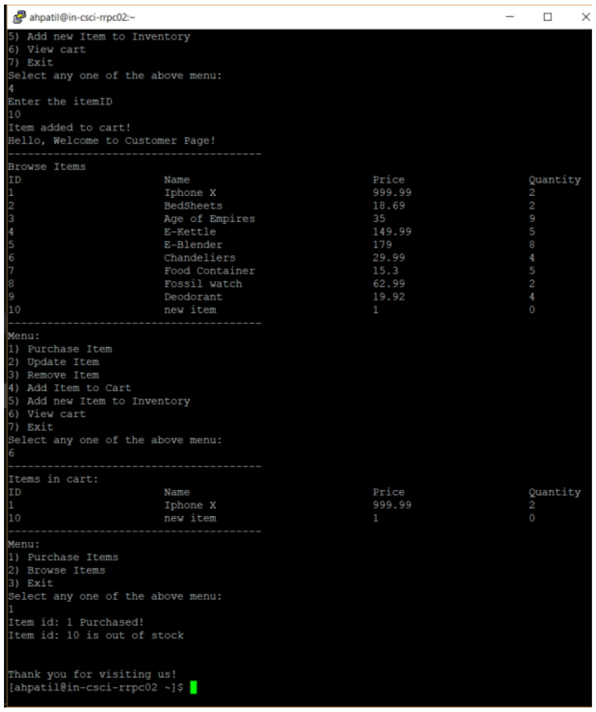
The customer added item id 9 to the cart and the added item was displayed in the cart



```
customer
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999.99                  2
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   5
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
6
Your Cart is empty!

Hello, Welcome to Customer Page!
```

Fig (7b) View cart: Empty



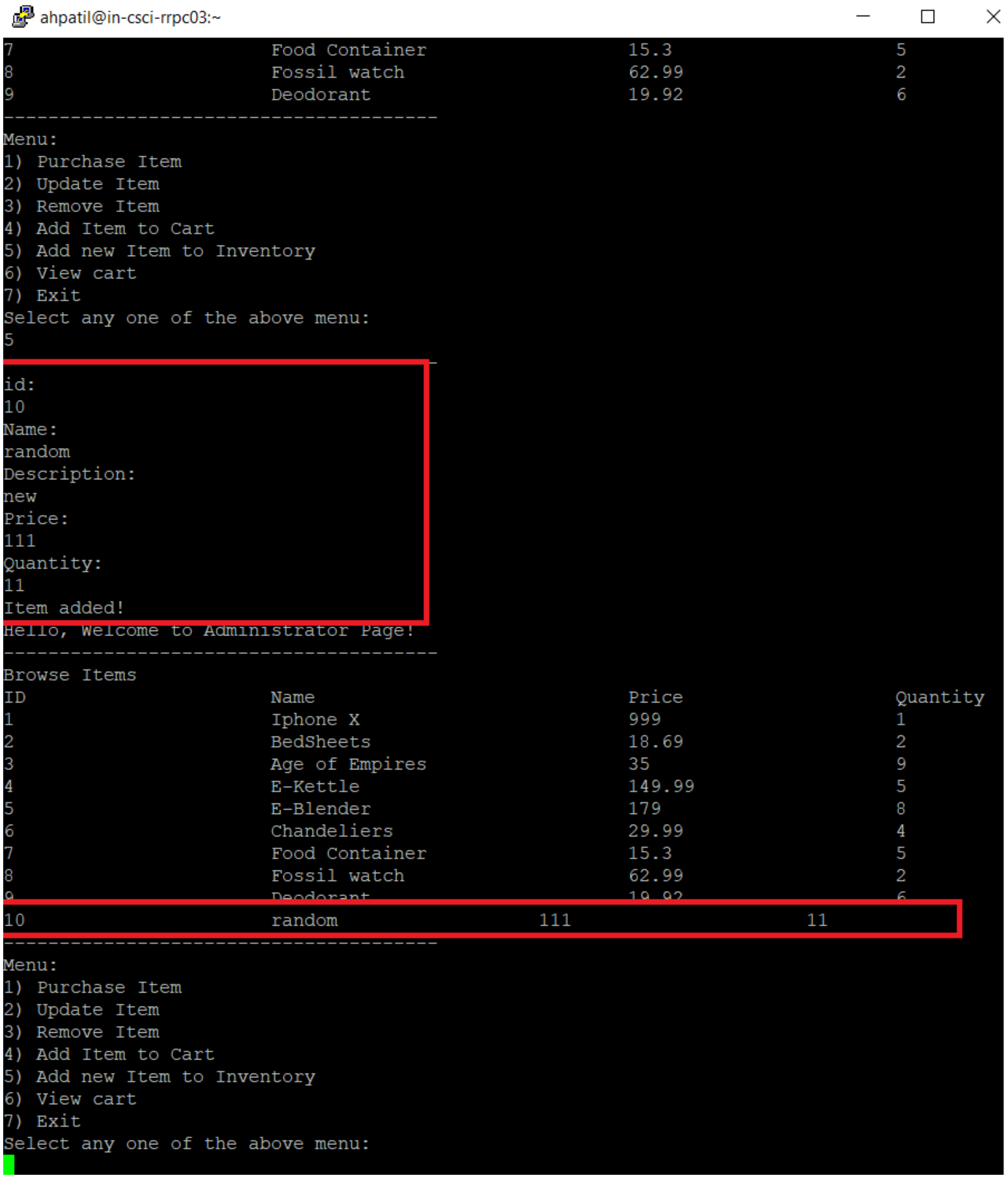
```
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
4
Enter the itemID
10
Item added to cart!
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999.99                  2
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   4
10                      new item                        1                       0
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
6
---------------------------------------
Items in cart:
ID                      Name                            Price                   Quantity
1                       Iphone X                        999.99                  2
10                      new item                        1                       0
---------------------------------------
Menu:
1) Purchase Items
2) Browse Items
3) Exit
Select any one of the above menu:
1
Item id: 1 Purchased!
Item id: 10 is out of stock


Thank you for visiting us!
[ahpatil@in-csci-rrpc02 ~]$
```

Fig (7c) Purchasing from the cart

The customer has item 1 and 10 in his cart. The item 10 has 0 quantities. When the customer proceeds for the purchase after adding items from the cart. It purchases only those items that are in the stock and for other items it displays item out of stock.

Add item:

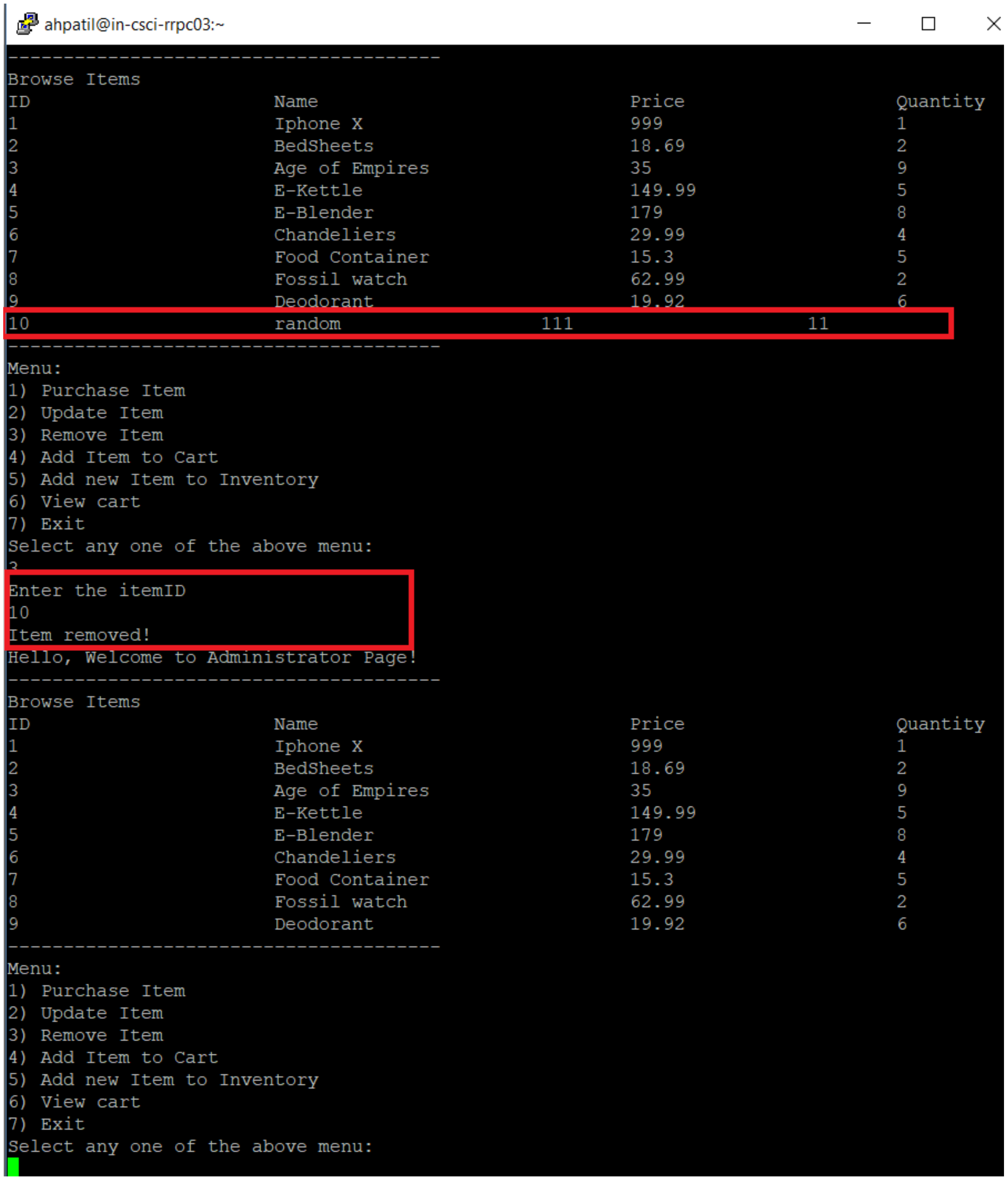


```
ahpatil@in-csci-rrpc03:~
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
5
---------------------------------------
id:
10
Name:
random
Description:
new
Price:
111
Quantity:
11
Item added!
Hello, Welcome to Administrator Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
10                      random                  111                     11
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (8) Add item to the inventory

This functionality is authorized for the admin role type and here the admin adds new item to the inventory and the added item reflects in the browse option.

Remove Item:

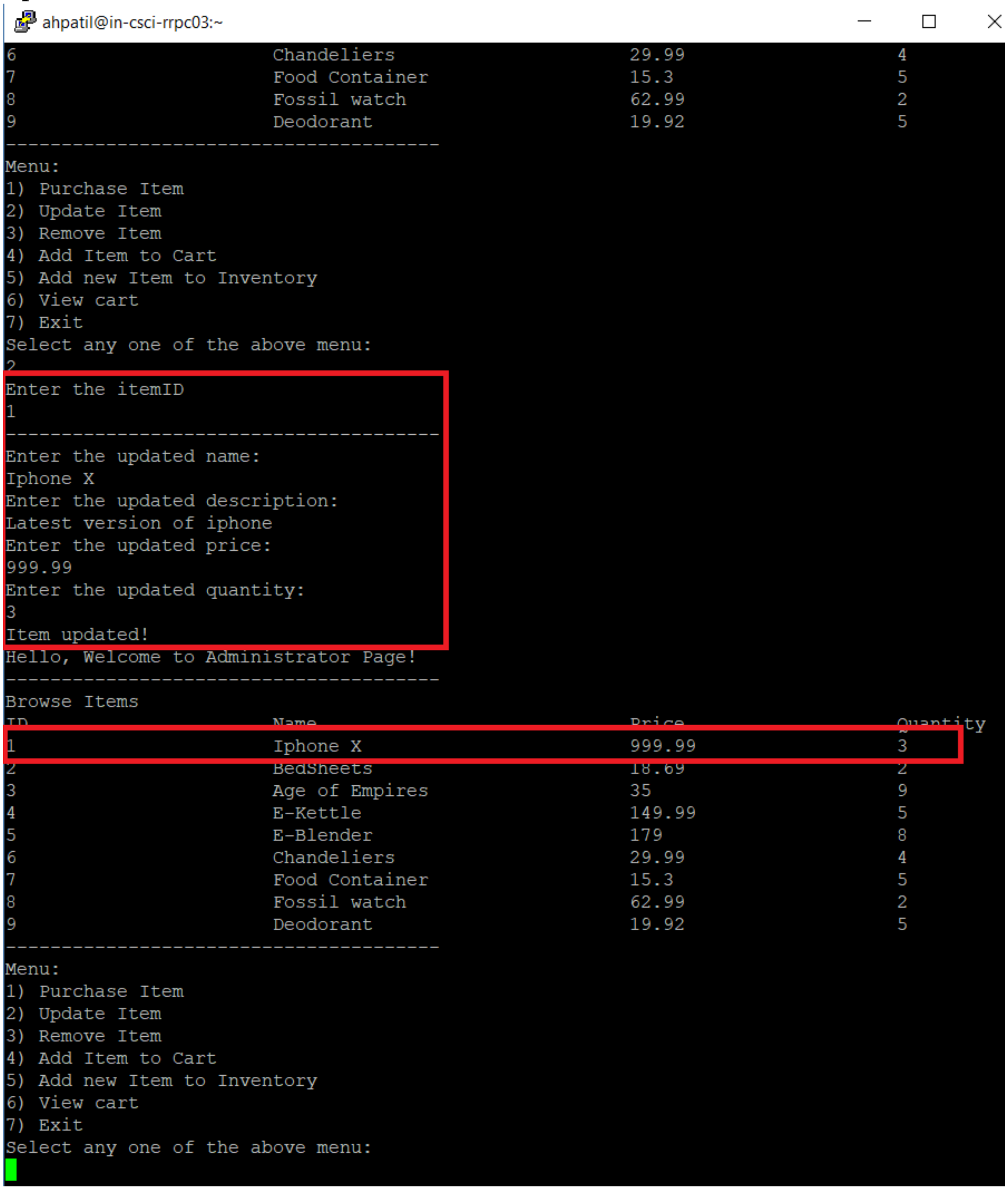


```
ahpatil@in-csci-rrpc03:~
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
10                      random                  111                     11
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
3
Enter the itemID
10
Item removed!
Hello, Welcome to Administrator Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999                     1
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   6
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (9) remove item from the inventory

This functionality is authorized for the admin role type and here the admin removes item with item id 10 from the inventory and the removed item changes reflects in the browse option.

Update Item:

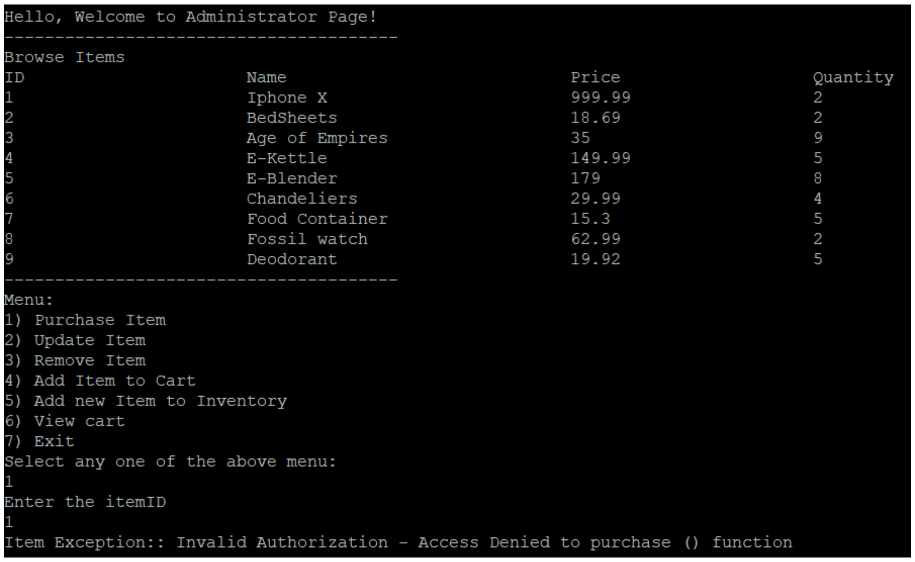


```
ahpatil@in-csci-rrpc03:~
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   5
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
2
Enter the itemID
1
---------------------------------------
Enter the updated name:
Iphone X
Enter the updated description:
Latest version of iphone
Enter the updated price:
999.99
Enter the updated quantity:
3
Item updated!
Hello, Welcome to Administrator Page!
---------------------------------------
Browse Items
ID                      Name                            Price                   Quantity
1                       Iphone X                        999.99                  3
2                       BedSheets                       18.69                   2
3                       Age of Empires                  35                      9
4                       E-Kettle                        149.99                  5
5                       E-Blender                       179                     8
6                       Chandeliers                     29.99                   4
7                       Food Container                  15.3                    5
8                       Fossil watch                    62.99                   2
9                       Deodorant                       19.92                   5
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (10) update item from the inventory

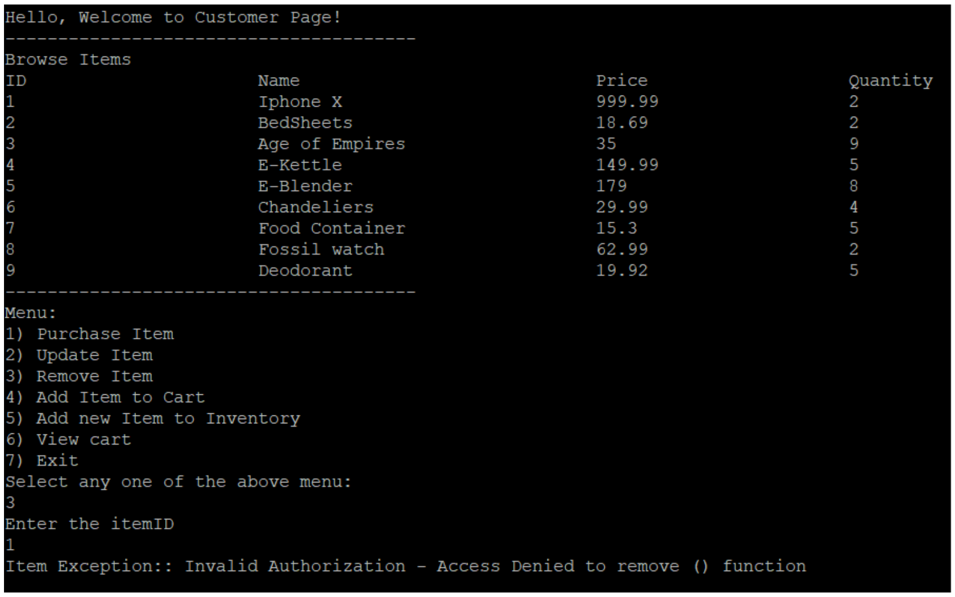
This functionality is authorized for the admin role type and here the admin updates the item from the inventory and the updated item changes reflects in the browse option

Role based Access Control:
We have provided role-based access for the functionalities. Following are the access authorizations based on the role-types:
Admin:
Login and Registration, Add item, Update item, Remove item, Browse
Customer:
Login and Registration, Purchase item, Cart, Browse



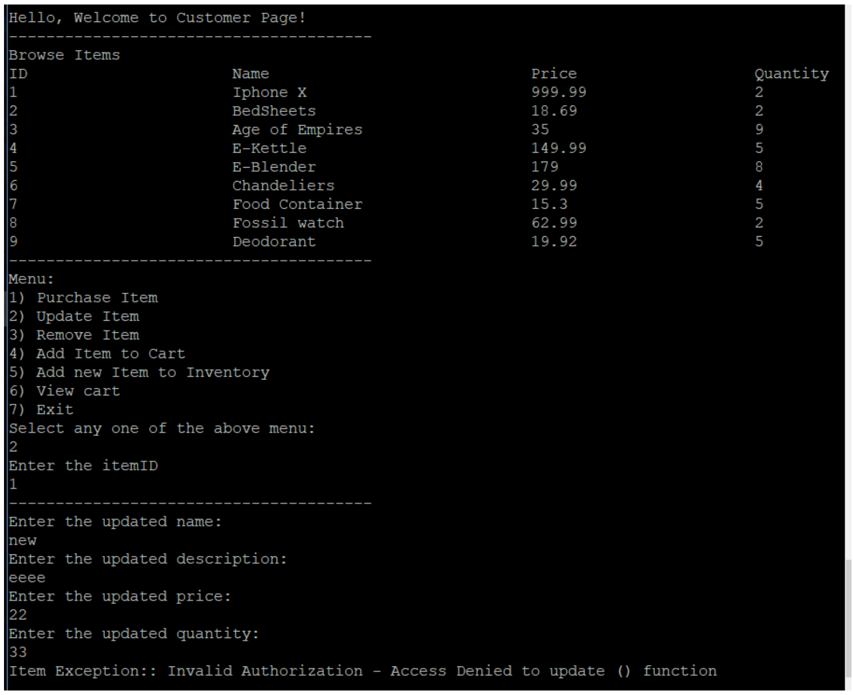
```
Hello, Welcome to Administrator Page!
---------------------------------------
Browse Items
ID                       Name                           Price                   Quantity
1                        Iphone X                       999.99                  2
2                        BedSheets                      18.69                   2
3                        Age of Empires                 35                      9
4                        E-Kettle                       149.99                  5
5                        E-Blender                      179                     8
6                        Chandeliers                    29.99                   4
7                        Food Container                 15.3                    5
8                        Fossil watch                   62.99                   2
9                        Deodorant                      19.92                   5
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
1
Enter the itemID
1
Item Exception:: Invalid Authorization - Access Denied to purchase () function
```

Fig (11a) Access denied for purchase, admin



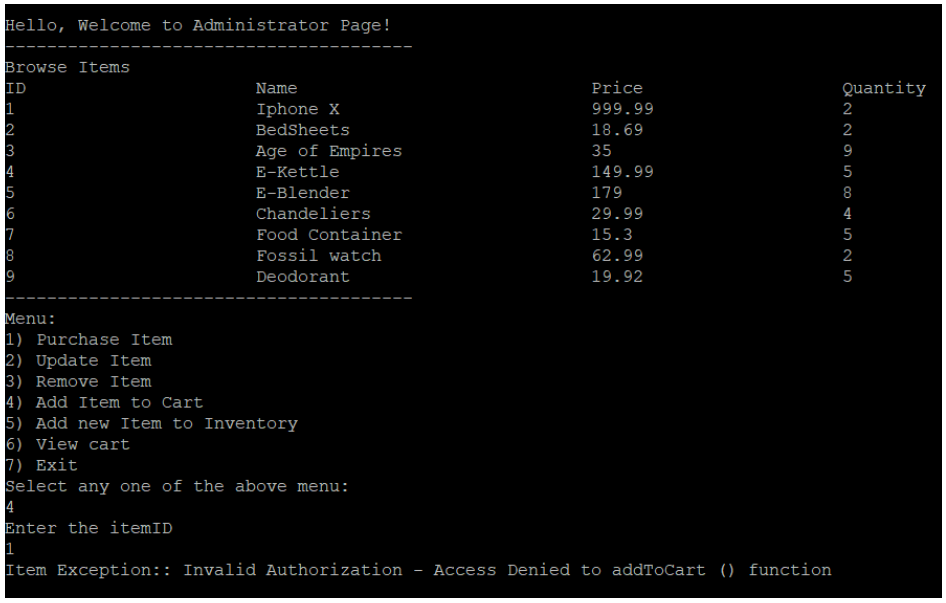
```
Hello, Welcome to Customer Page!
---------------------------------------
Browse Items
ID                       Name                           Price                   Quantity
1                        Iphone X                       999.99                  2
2                        BedSheets                      18.69                   2
3                        Age of Empires                 35                      9
4                        E-Kettle                       149.99                  5
5                        E-Blender                      179                     8
6                        Chandeliers                    29.99                   4
7                        Food Container                 15.3                    5
8                        Fossil watch                   62.99                   2
9                        Deodorant                      19.92                   5
---------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
3
Enter the itemID
1
Item Exception:: Invalid Authorization - Access Denied to remove () function
```

Fig (11b) Access denied for remove, customer



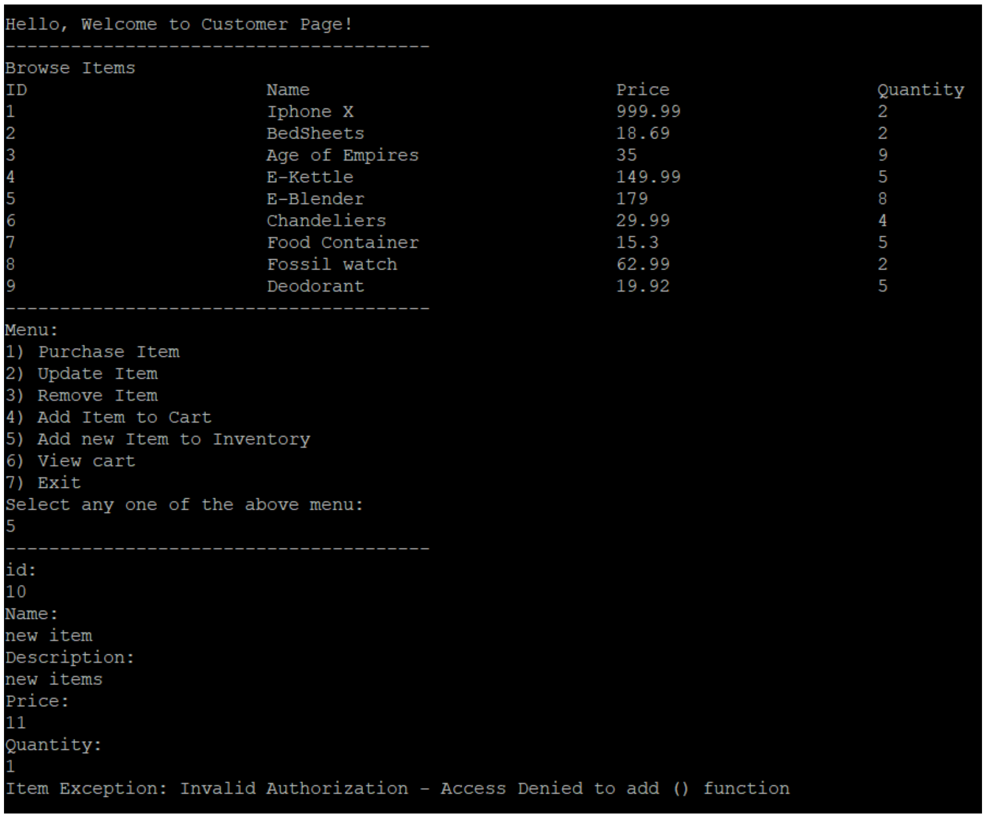
```
Hello, Welcome to Customer Page!
--------------------------------------
Browse Items
ID                       Name                               Price                    Quantity
1                        Iphone X                           999.99                   2
2                        BedSheets                          18.69                    2
3                        Age of Empires                     35                       9
4                        E-Kettle                           149.99                   5
5                        E-Blender                          179                      8
6                        Chandeliers                        29.99                    4
7                        Food Container                     15.3                     5
8                        Fossil watch                       62.99                    2
9                        Deodorant                          19.92                    5
--------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
2
Enter the itemID
1
--------------------------------------
Enter the updated name:
new
Enter the updated description:
eeee
Enter the updated price:
22
Enter the updated quantity:
33
Item Exception:: Invalid Authorization - Access Denied to update () function
```

Fig (11c) Access denied for update, customer



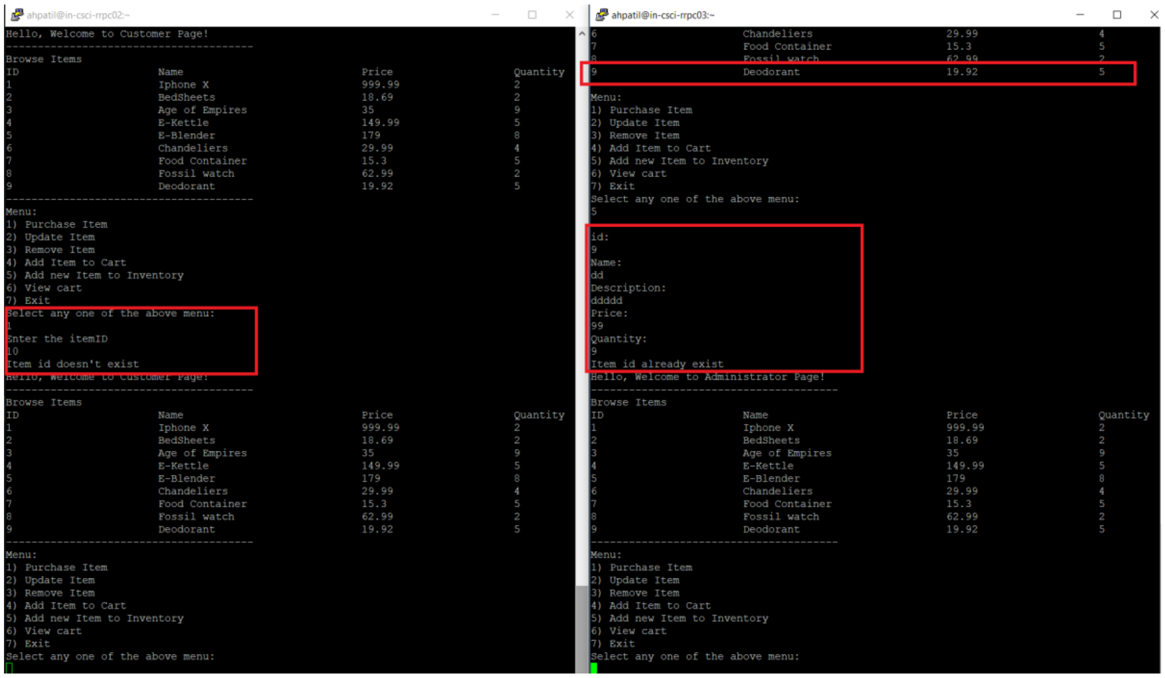
```
Hello, Welcome to Administrator Page!
--------------------------------------
Browse Items
ID                       Name                               Price                    Quantity
1                        Iphone X                           999.99                   2
2                        BedSheets                          18.69                    2
3                        Age of Empires                     35                       9
4                        E-Kettle                           149.99                   5
5                        E-Blender                          179                      8
6                        Chandeliers                        29.99                    4
7                        Food Container                     15.3                     5
8                        Fossil watch                       62.99                    2
9                        Deodorant                          19.92                    5
--------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
4
Enter the itemID
1
Item Exception:: Invalid Authorization - Access Denied to addToCart () function
```

Fig (11d) Access denied for add to cart, admin



```
Hello, Welcome to Customer Page!
----------------------------------------
Browse Items
ID                        Name                            Price                    Quantity
1                         Iphone X                        999.99                   2
2                         BedSheets                       18.69                    2
3                         Age of Empires                  35                       9
4                         E-Kettle                        149.99                   5
5                         E-Blender                       179                      8
6                         Chandeliers                     29.99                    4
7                         Food Container                  15.3                     5
8                         Fossil watch                    62.99                    2
9                         Deodorant                       19.92                    5
----------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
5
----------------------------------------
id:
10
Name:
new item
Description:
new items
Price:
11
Quantity:
1
Item Exception: Invalid Authorization - Access Denied to add () function
```

Fig (11d) Access denied for add to inventory, customer



```
ahpatil@in-csci-rrpc02:~
Hello, Welcome to Customer Page!
----------------------------------------
Browse Items
ID                        Name                            Price                    Quantity
1                         Iphone X                        999.99                   2
2                         BedSheets                       18.69                    2
3                         Age of Empires                  35                       9
4                         E-Kettle                        149.99                   5
5                         E-Blender                       179                      8
6                         Chandeliers                     29.99                    4
7                         Food Container                  15.3                     5
8                         Fossil watch                    62.99                    2
9                         Deodorant                       19.92                    5
----------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
1
Enter the itemID
10
Item id doesn't exist
Hello, Welcome to Customer Page!
----------------------------------------
Browse Items
ID                        Name                            Price                    Quantity
1                         Iphone X                        999.99                   2
2                         BedSheets                       18.69                    2
3                         Age of Empires                  35                       9
4                         E-Kettle                        149.99                   5
5                         E-Blender                       179                      8
6                         Chandeliers                     29.99                    4
7                         Food Container                  15.3                     5
8                         Fossil watch                    62.99                    2
9                         Deodorant                       19.92                    5
----------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

```
ahpatil@in-csci-rrpc03:~
6                         Chandeliers                     29.99                    4
7                         Food Container                  15.3                     5
8                         Fossil watch                    62.99                    2
9                         Deodorant                       19.92                    5

Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
5

id:
9
Name:
dd
Description:
ddddd
Price:
99
Quantity:
9
Item id already exist
Hello, Welcome to Administrator Page!
----------------------------------------
Browse Items
ID                        Name                            Price                    Quantity
1                         Iphone X                        999.99                   2
2                         BedSheets                       18.69                    2
3                         Age of Empires                  35                       9
4                         E-Kettle                        149.99                   5
5                         E-Blender                       179                      8
6                         Chandeliers                     29.99                    4
7                         Food Container                  15.3                     5
8                         Fossil watch                    62.99                    2
9                         Deodorant                       19.92                    5
----------------------------------------
Menu:
1) Purchase Item
2) Update Item
3) Remove Item
4) Add Item to Cart
5) Add new Item to Inventory
6) View cart
7) Exit
Select any one of the above menu:
```

Fig (12) If item id exists or doesn’t exist

Conclusion

What I liked about the project?
I learned about the version controls in Git. I liked the way the project was setup. We learned about the patterns in theory and had to implement it in practice in each version. This really helped me to understand the design patterns. If it was just theory, I wouldn't take much efforts to google about it and study in-depth. The coursework was designed brilliantly. Moreover, every time I got interviewed they asked me same question "Can you implement design patterns?" and I was always unsure what to answer. But, today if they ask me same question, I know which pattern works where and how. The problem, solution and consequences way of learning the design patterns helped me a lot.

What changes I would make if redesigning the same application?
While attending the presentation, I liked the approach used by one of my classmates i.e. server resource request and request handling way for designing the system. Furthermore, separation of login and registration from the functionalities i.e. having authorization end and resource end points would be another change in my system. I always wanted to split my model into two parts authorization and functionality using 2 RMI's but, couldn't work on it due to time constraints and priorities.
Talking about RMI, I would not use RMI. RMI has scalability issues, it limits the number of users using application at same time. It won't support more than 1000-2000 threads running concurrently. When looking ahead of time, using RMI isn't a practical choice. I would choose using JSON or HashMap’s for the data exchange.
Moreover, if I had enough time to work on this assignment. I would implement synchronization without using the Java synchronized keyword for more depth in learning the patterns. This change is from the learning prospective.

What I learned from these assignments?
I always wanted to learn the git version control and agile methodology. These series of assignments helped me work using agile methodologies, wherein I would update my work each day on my local repository and then on the git. Maintaining each assignment as versions and merging it with the master branch helped me understand the version control in git. I believe this is something that will help me while working in the industry.

Moreover, working on this assignment was really a nice experience and will reflect on my resume as one of the strong academic projects. I look forward to work on something similar using all my learnings from these assignments. Overall, a successful implementation of an Online Marketplace Application.