Assignment #4 Report

Russell Sherman

April 4, 2017

CSCI 50700

Requirements for the Online Marketplace

After meeting with the client, it was determined that they would like an online marketplace built that would allow them to sell good and possibly services to users all over the world. They would like the ability to separate what the customer can see, and what an administrator of the marketplace would see. Customers should have to register for an account upon entering the marketplace. After registering, the customer would then be required to login to enter the marketplace. The customer should be able to browse the marketplace and see a number of items available for purchase. The items should be assigned a ‘type’, ‘description’, and ‘price’. Customers should be able to add these items to a shopping cart that is specific to the user. The items will have a supply amount, restricting customers from purchasing more than is available. The customers shopping cart should keep track of items added or deleted from the cart and be persistent throughout the customers interactions in the marketplace. Administrators will have the ability to update descriptions, prices, and quantities of the items. They will also be able to remove items from the marketplace. Administrators should have the ability to add other administrators and also add or remove customers from the marketplace. Administrators should not have the ability to purchase items as an administrator; they would have to do so as a customer. The marketplace system should be reliable and be able to handle multiple requests during execution. The system should be able to handle any scenario gracefully.

Requirements for Assignment #4

Assignment 4 asks us to take a look at the idea of threads and concurrency. Specifically, how Java RMI handles concurrency and threading. In this report, I will examine the consequences of concurrency on my Marketplace Application. In order to achieve this, I will take advantage of six machines to demonstrate how the Application works across a network. This will show how my server handles requests from different clients. For this assignment, we will use just one server and five additional clients.

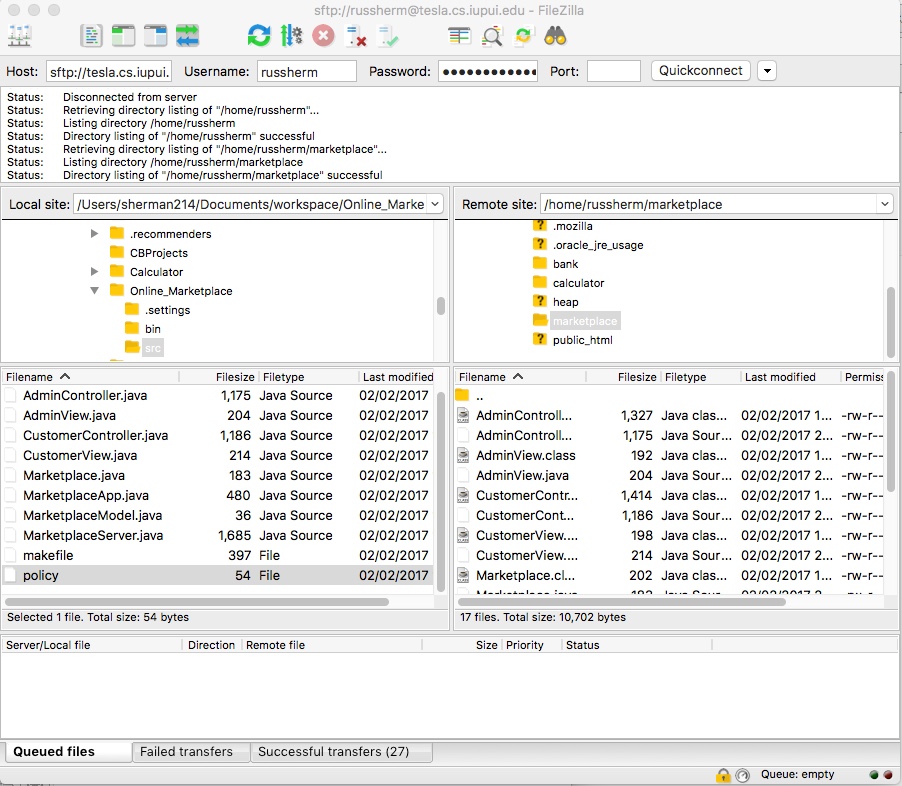
This assignment also requires the purchase item, add item and browse item functions to be implemented for the customer and administrator. For my Marketplace Application, this means customers should be able to browse and purchase items, while the administrator should have the ability to add items to the Marketplace.

Changes Made

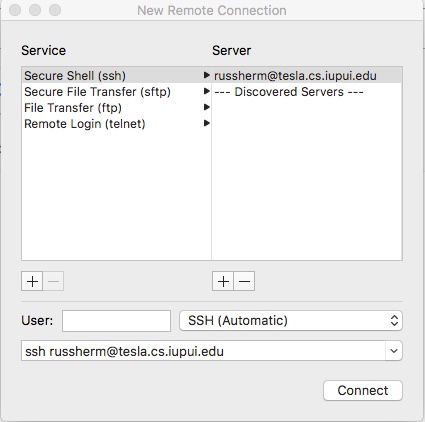
In order to make sure the Application had the required functionality, a number of changes had to be made. To begin, I needed to get rid of classes that were no longer needed. These included Marketplace, MarketplaceApp, and MarketplaceModel. These were classes that held other functions in the past, but have since become unneeded. Removing them helps the Application to evolve and stay organized. The next task was continuing to build upon the command and Java Annotation classes that were instrumental in Assignment 3. The server class stayed mostly the same, but our Client class required some changes to make sure it was looking the correct interface. In assignment 3, the Client class was looking for an incorrect interface, which led to errors in the output. Another issue from Assignment 3 was authorizing the user. In Assignment 3, I ran into problems with directing the user to the correct view. In Assignment 4, the auth() function now handles the authentication correctly. Other major changes came in the Items, AdminView and CustomerView classes. The Items class needed an ArrayList of all the items in the Marketplace. The AdminView and CustomerView added the ability interact with the user and ask what function they would like to execute. Once the user selects which function they want, our JavaAnnotations checks to make sure the user has the correct role.

Sample Run

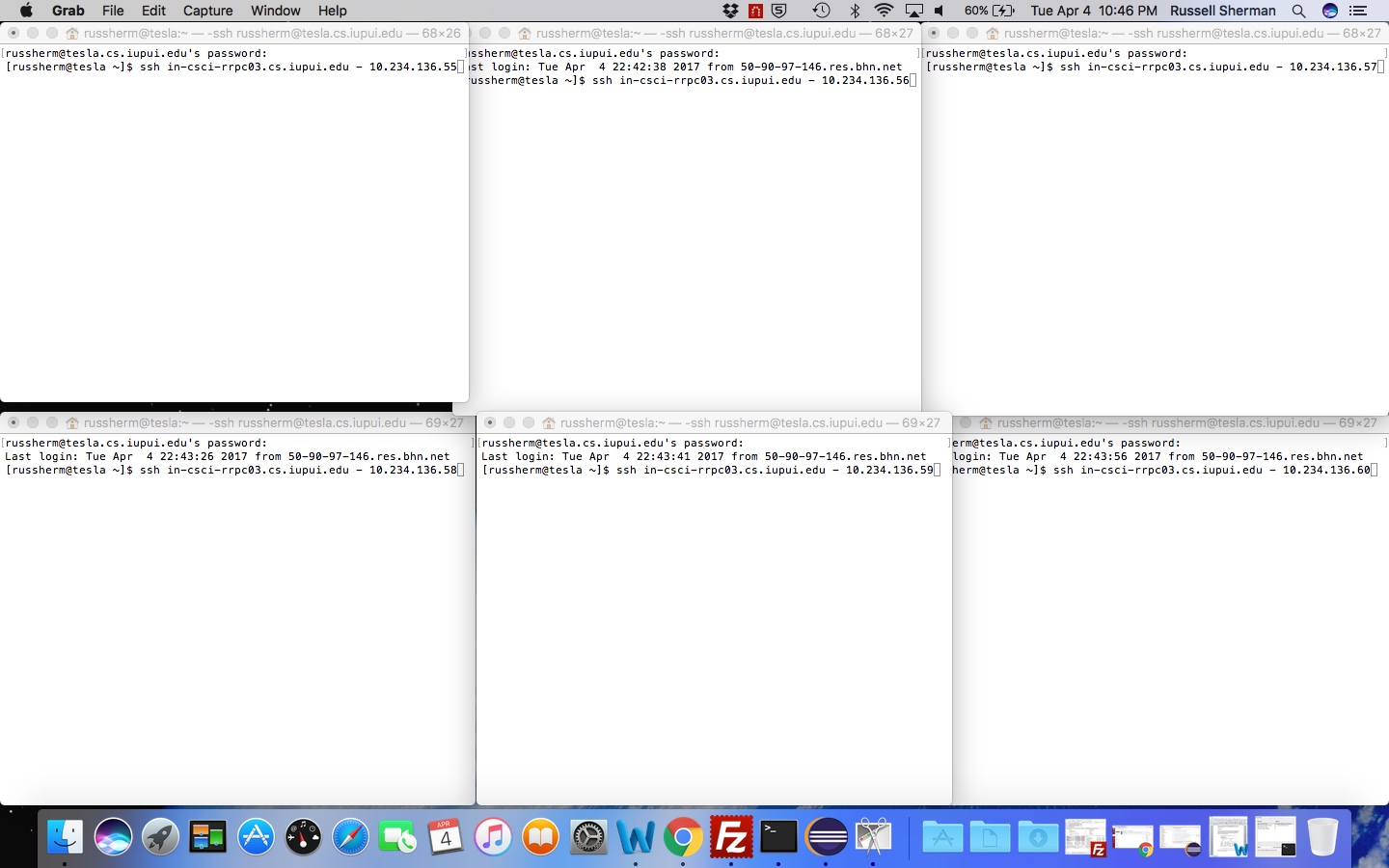
After saving my files in Eclipse, I open a program to transfer my files from my local machine to the Tesla server. Since I am unable to use PuTTY, I use FileZilla and sign on to the Tesla server using “tesla.cs.iupui.edu” followed by my username and password. Upon signing in the tesla, I am able to transfer my files to the correct directory that I have labeled marketplace.



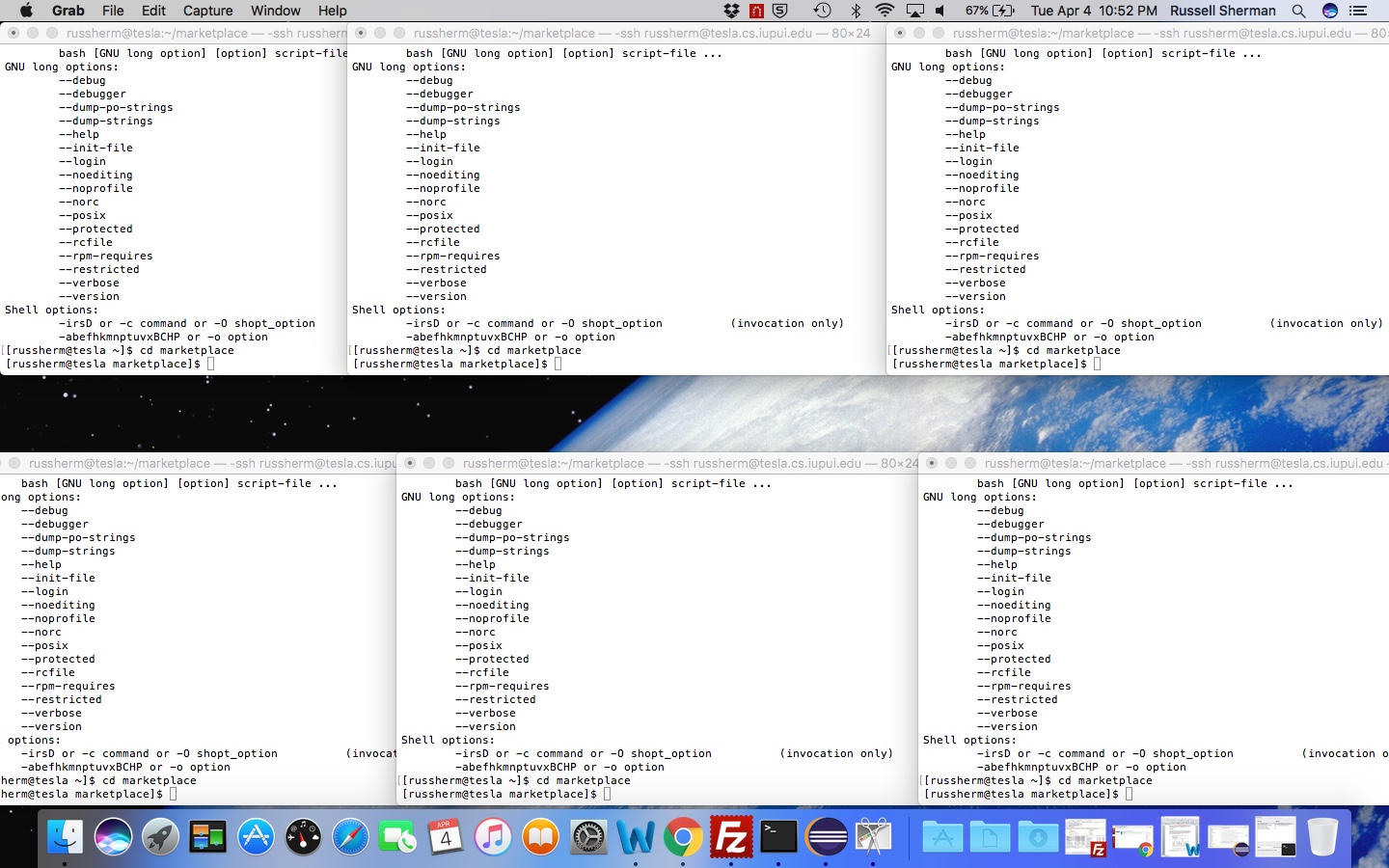
Now that my files are on the tesla server, I can open a terminal and connect to the tesla server via a remote connection.



After logging in with my password, I am connected to the tesla server. I need to repeat these steps so I have six terminal windows open that are logged into tesla. The reason for this is so we can ssh into all six machines needed for this assignment. Once I have all six terminals open, I will ssh into all six machines.

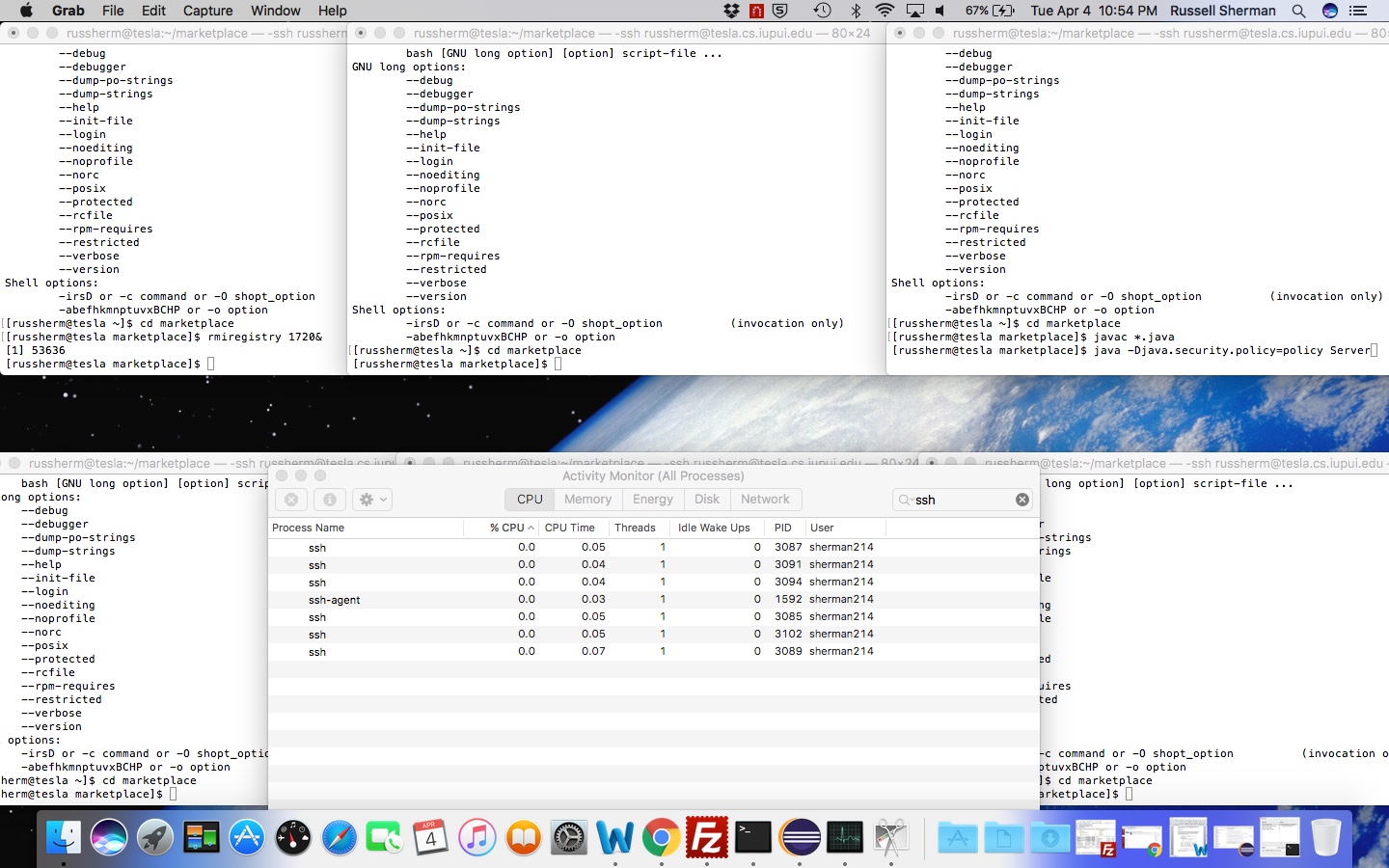


As you can see, I have typed in the corresponding address for each machine. Once I log in to the machines I need to find the correct directory, and can do this by changing the directory to marketplace.

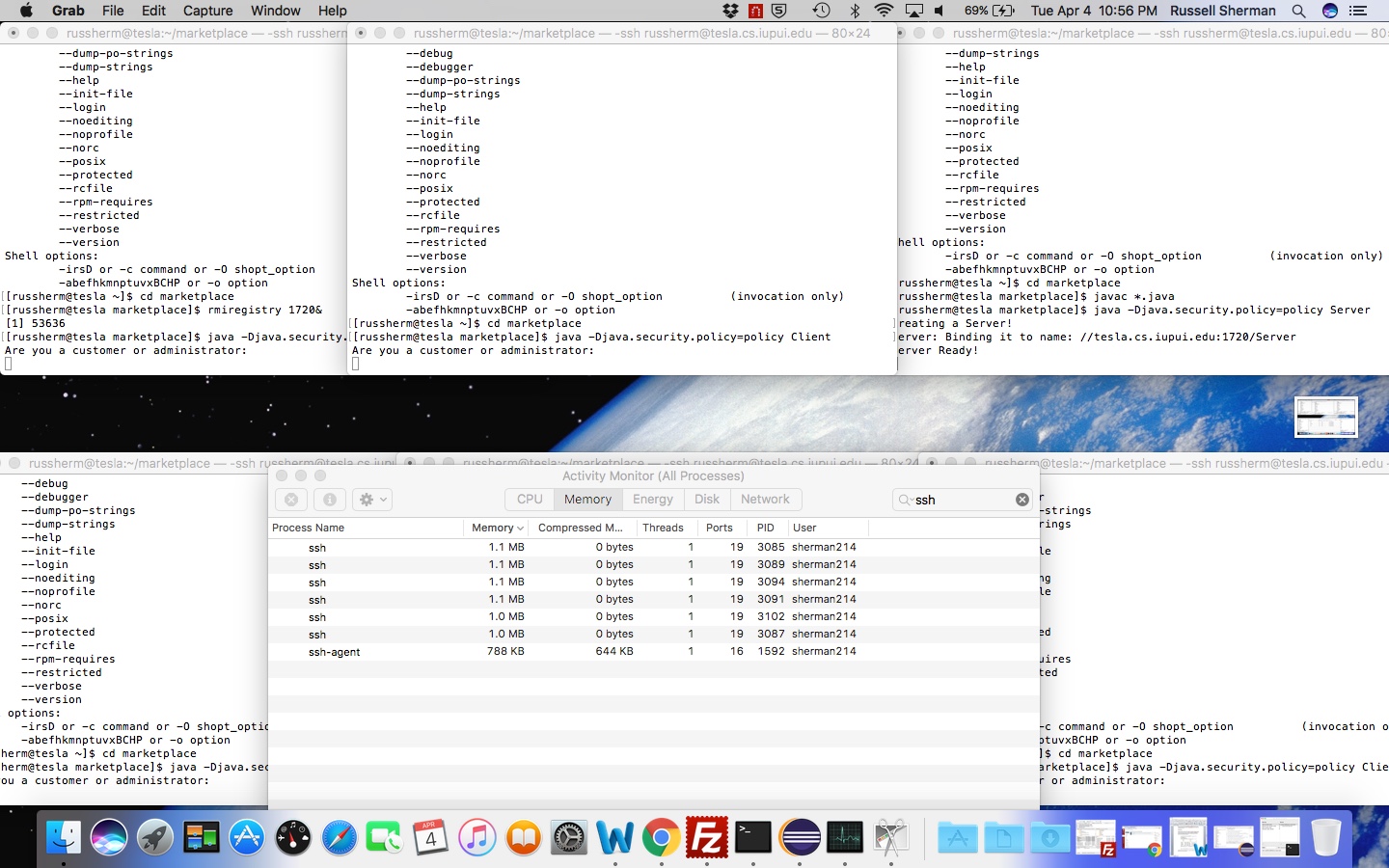


I need to make sure that the class files have all been compiled. I do this by typing “javac \*.java”. This will compile all of my files so they are ready to run. Next, I need to set up the Java RMI registry. I was able to hard code my registry port for port 1720. I can connect to this port by typing ‘rmiregistry 1720&’, with the & allowing the process to run in the background. I have selected machine 10.234.136.57 as my server for this assignment. That machine will run my server side, while the five other machines will run the client side.

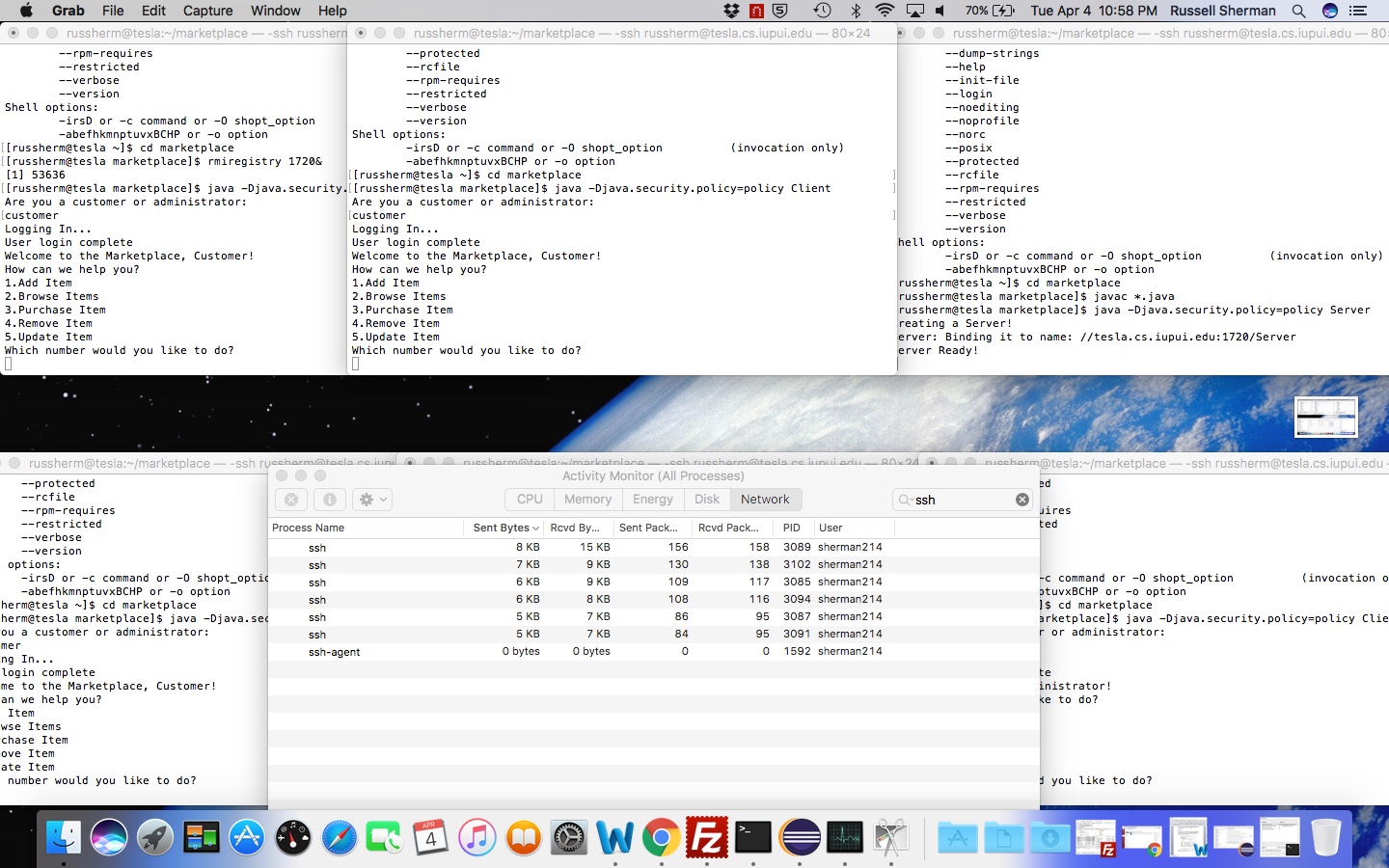
I have also opened the Activity Monitor to track the concurrency and threads of my Marketplace. I will track the CPU, Memory and Network to see how the JavaRMI is dealing with having five clients from different addresses.



I know I am connected based on the “[1] 53636”. If the port had been in use, an Exception would have been thrown. Now that I have my port, I can run my server and client sides of my Java RMI connection. I will do this by calling my Server class, and also my Client class. To run the Server file, I will enter the following “java -Djava.security.policy=policy Server” on the Server machine. I will do the same in the other five terminals, entering “java -Djava.security.policy=policy Client”.



As we can see, the Server establishes a connection and then the Client connects to the Java RMI. The Server will announce that it is creating a server, binding it to a name and then letting you know that the server is ready. On the Client side, if run correctly, the user should be asked for their role in the marketplace.



Upon choosing a role, they will be logged in as that specific role and have access to only role specific functions. The system will ask the user what actions they would like to take. Depending on their response and role, they may be blocked from accessing this action. The first client logs in as a customer and attempts to add an item and is blocked. The second client, also a customer, asks to browse items and is shown the items in the marketplace. The third client is a customer and tries to purchase an item and is asked what item they would like to purchase. The final two clients are administrators and try to remove and update items. They are allowed to proceed based on their role.

Java RMI Findings

As we can see from the Activity Monitor, each client concurrently executed with only 1 thread. Whether it was the CPU or Memory, all six of the ssh connections used only 1 thread. However, as stated in the definition, the RMI runtime makes no guarantees with respect to mapping remote object invocations to threads. If this is the case, it could be tricky for our Marketplace. If two clients desire the same item and close to the same time, it may be disastrous for the Marketplace. It could lead to the Marketplace “inventory” being unstable.