Assignment #5 Report

Russell Sherman

April 28, 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 #5

In Assignment 5, the goal is to take a look at synchronization and examine its impact on our application. In the previous assignment, we found that without synchronization, our application will encounter a number of difficulties in regards to concurrent access. In order to guard against these types of problems, we need to take advantage of a number of patterns designed specifically for synchronization and concurrency. To ensure our application, we need to implement the Monitor object pattern, future pattern, guarded suspension pattern, scoped locking and thread-safe interface. Using these five patterns will help to alleviate the thread-safety issues we ran into on assignment 4.

Also in Assignment 5, we are tasked with creating a MySQL database to store our users and items. This replaces the items class and hard coded version of customer and administrator we used for past assignments. This should allow more flexibility when dealing with user login and roles, as well as eventually creating new users. It will also assist in keeping track of items and the information associated with them.

Finally, we need to complete the functionality of our application. As we conclude, administrators should be able to add, remove, update or browse items. Customers should be able to purchase items. We also want to make sure our application is polished and working efficiently.

Changes Made

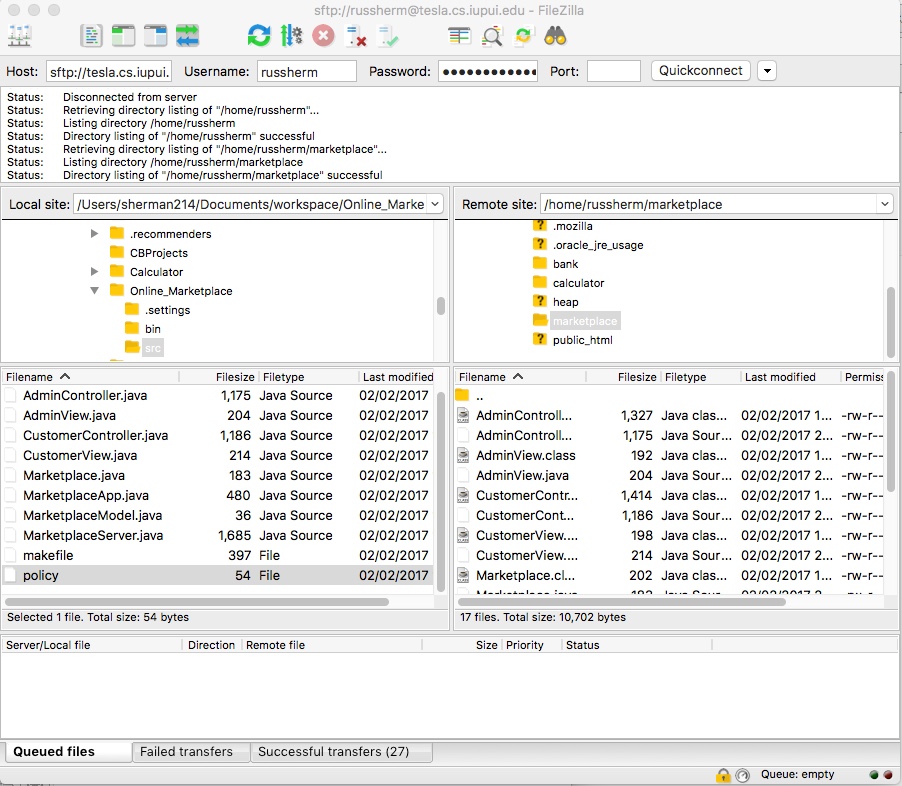
The first task for assignment 5 was to create our MySQL database on the first machine (in-csci-rrpc01.cs.iupui.edu). For my database, I have created two tables, one for items and one for users. My items table has five columns: ID, Item, Quantity, Price, Type. Using the item ID will allow easier access and manipulations to objects, as opposed to users having to select the item name. Having the quantity will help to keep track of which items are still available. One test that could be run for this assignment would be to have a Quantity of 1 for an item and have two machines concurrently attempt to purchase the item. This will help test our synchronization efforts. For this assignment I have given the item ‘Router’ a quantity of 1 in our marketplace. If we continued to develop our marketplace, we would be able to take advantage of the price and type columns in our database. Adding multiple items to a cart with allow us to total the items’ price to show the user what they owe. Users could search for items based on the type of item, showing them only items they are interested.

Now that we have implemented our database, the need for our Items class becomes redundant. Our items will be tracked in the database and not in our actual application via an array list. I have removed the Items class from our application. I have also removed CustomerController and AdminController, as they were no longer need. One final change was to the Admin and Customer views. Previously the user could only do one task before the application would exit. To improve the user experience I have added the ability for users to do multiple functions using a while loop. The users also have the ability to ‘Quit’, which will exit them from the application.

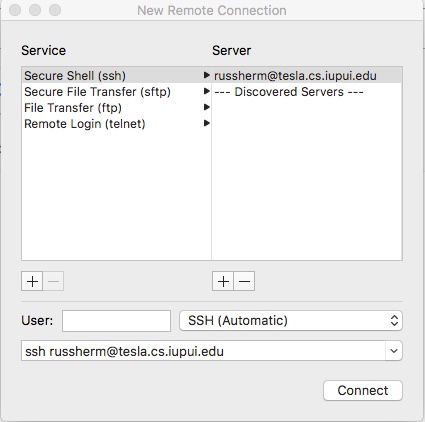
The major changes made to our application came when implementing our thread-safe patterns. The first patterns I added were the Monitor Object and guarded suspension. These two worked together to block access to objects until other processes on that object had completed. Using notify, notifyall and wait on our methods were we able to add some synchronization to our functions. Implementing the Future pattern allowed us to ensure that in the future our task/result is ready. The goal when using a thread-safe interface is to have publicly accessibly interface methods and private implementation methods. This will allow for easier acquisition and releasing of locks. Finally, with the scoped locking pattern, we again use the Monitor Object pattern to ensure access to the critical section. We can use monitorenter and monitorexit for Scoped Locking.

Sample Run

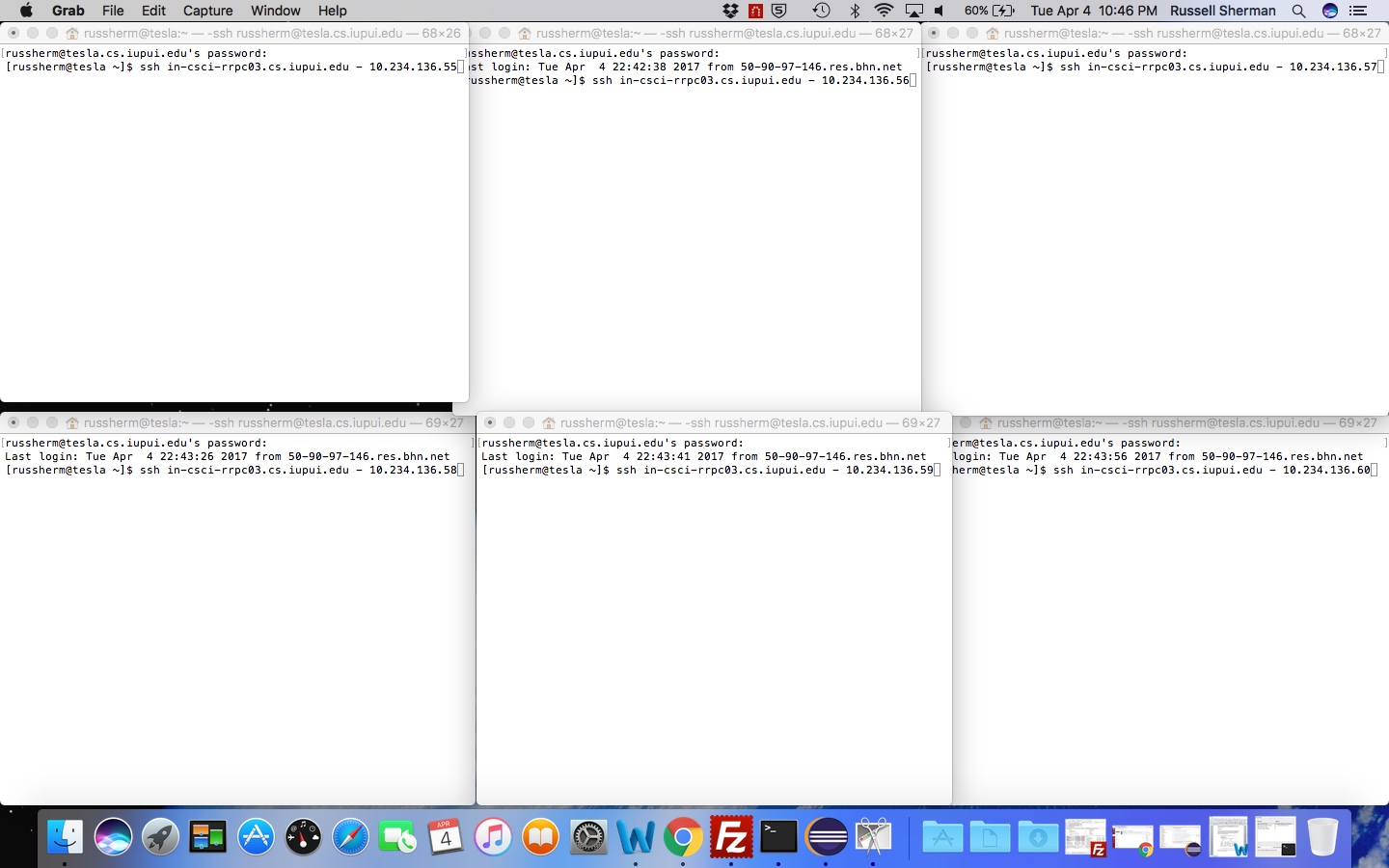
After saving my files in Eclipse, I open a program to transfer my files from my local machine to the Tesla server. Since I am unable to use PuTTY, I use FileZilla and sign on to the Tesla server using “tesla.cs.iupui.edu” followed by my username and password. Upon signing in the tesla, I am able to transfer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



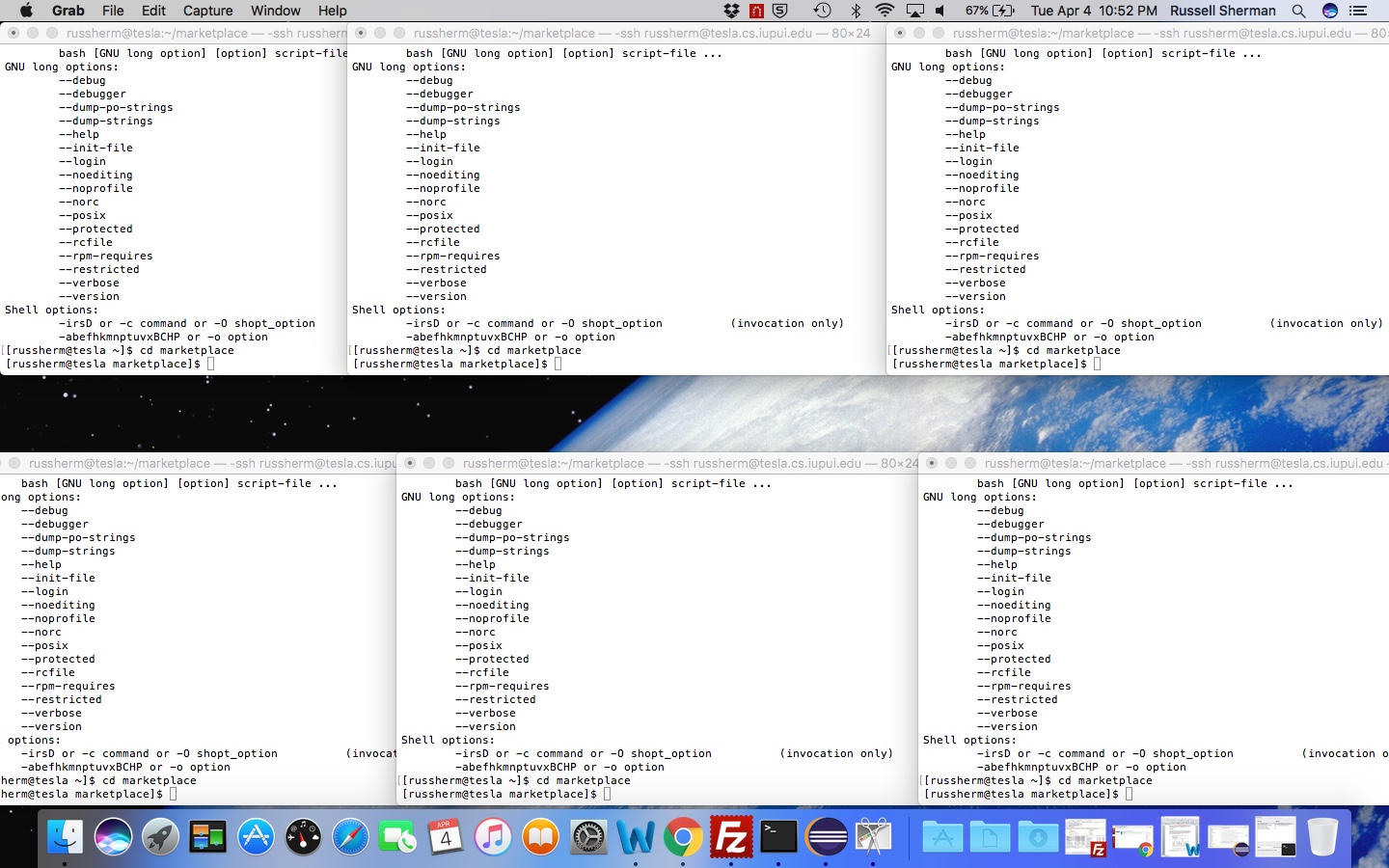
Now that my files are on the tesla server, I can open a terminal and connect to the tesla server via a remote connection.



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.

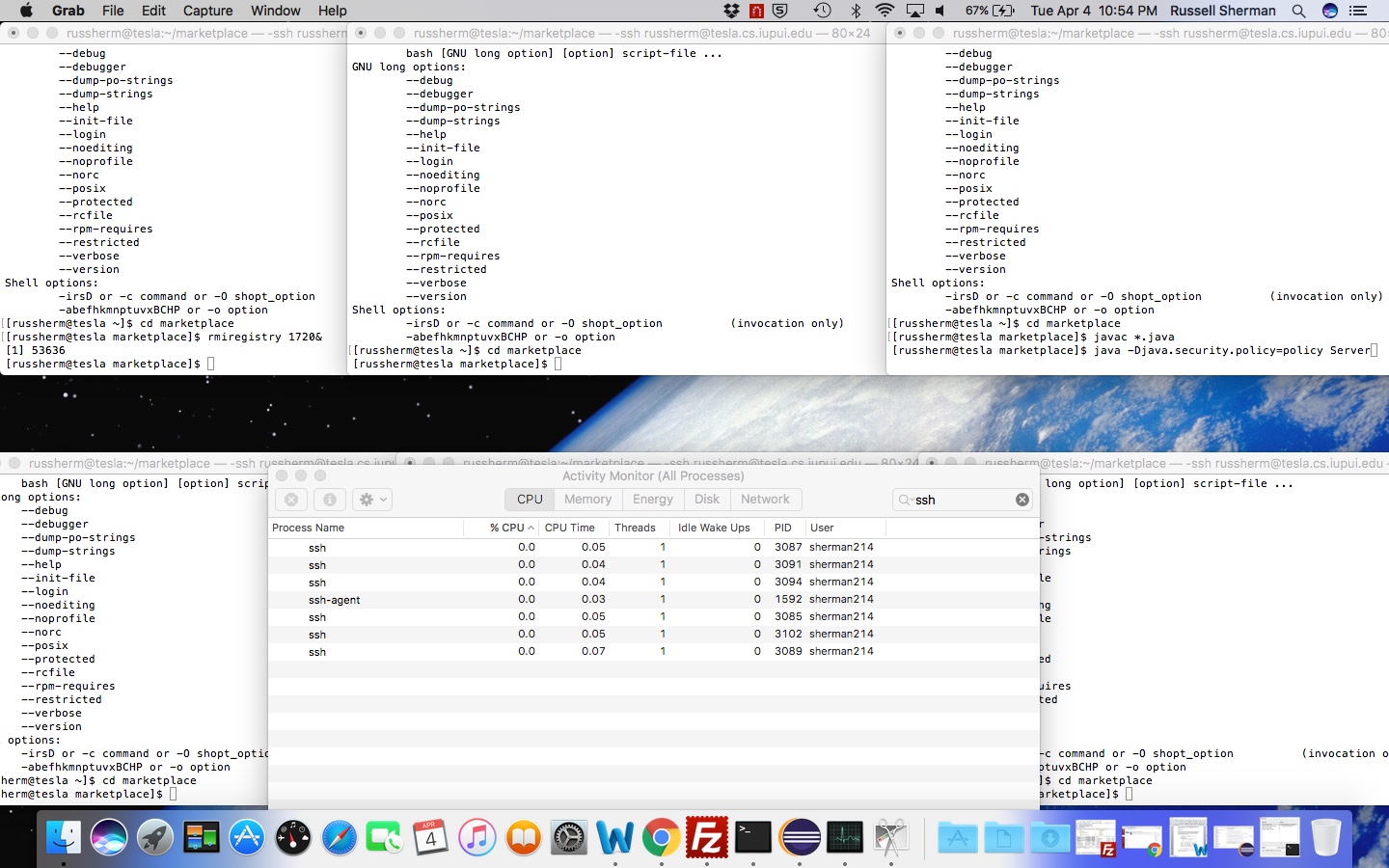


As you can see, I have typed in the corresponding address for each machine. Once I log in to the machines I need to find the correct directory, and can do this by changing the directory to marketplace.

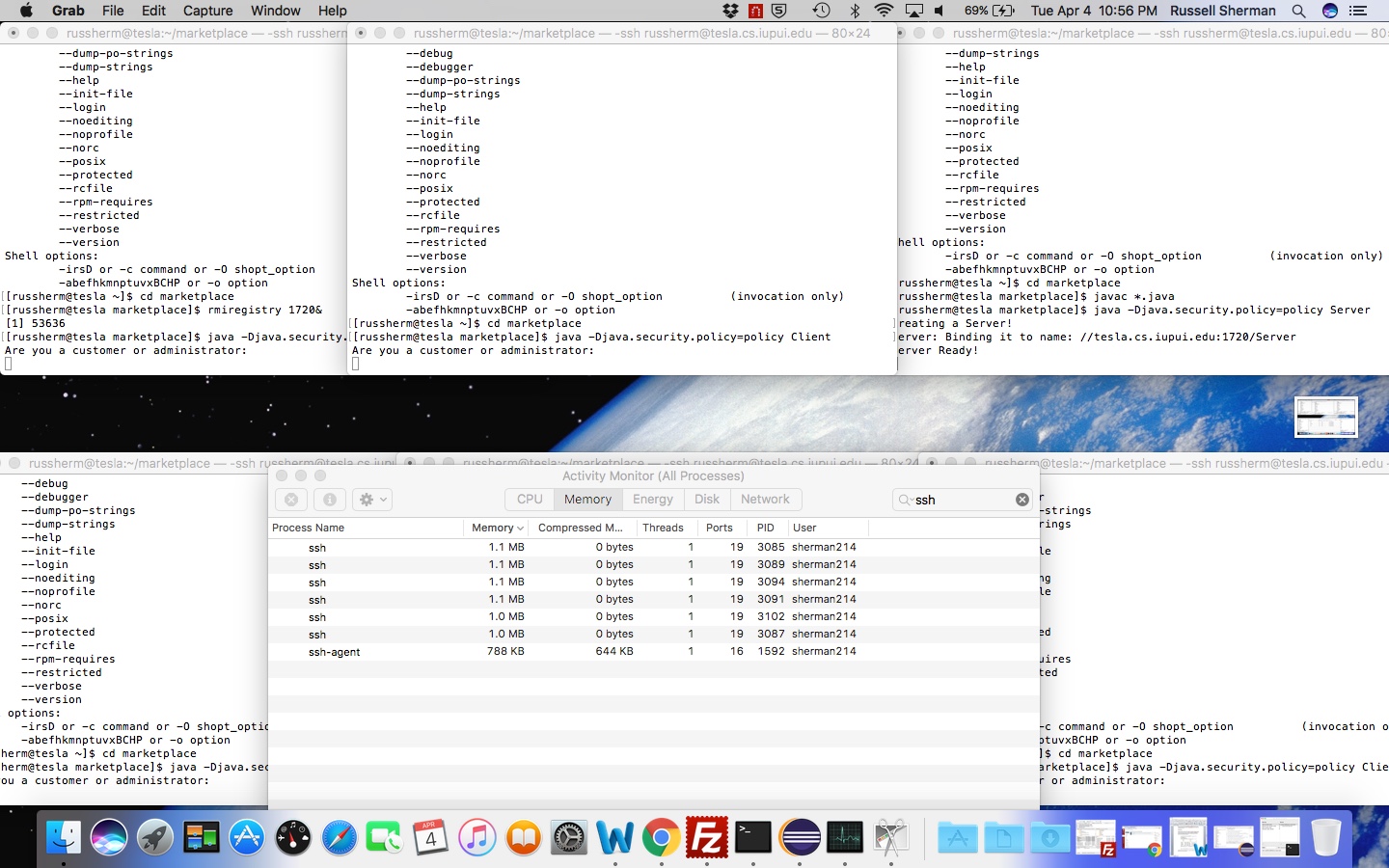


I need to make sure that the class files have all been compiled. I do this by typing “javac \*.java”. This will compile all of my files so they are ready to run. Next, I need to set up the Java RMI registry. I was able to hard code my registry port for port 1720. I can connect to this port by typing ‘rmiregistry 1720&’, with the & allowing the process to run in the background. I have selected machine 10.234.136.55 as my server for this assignment. That machine will run my server side, while the five other machines will run the client side.

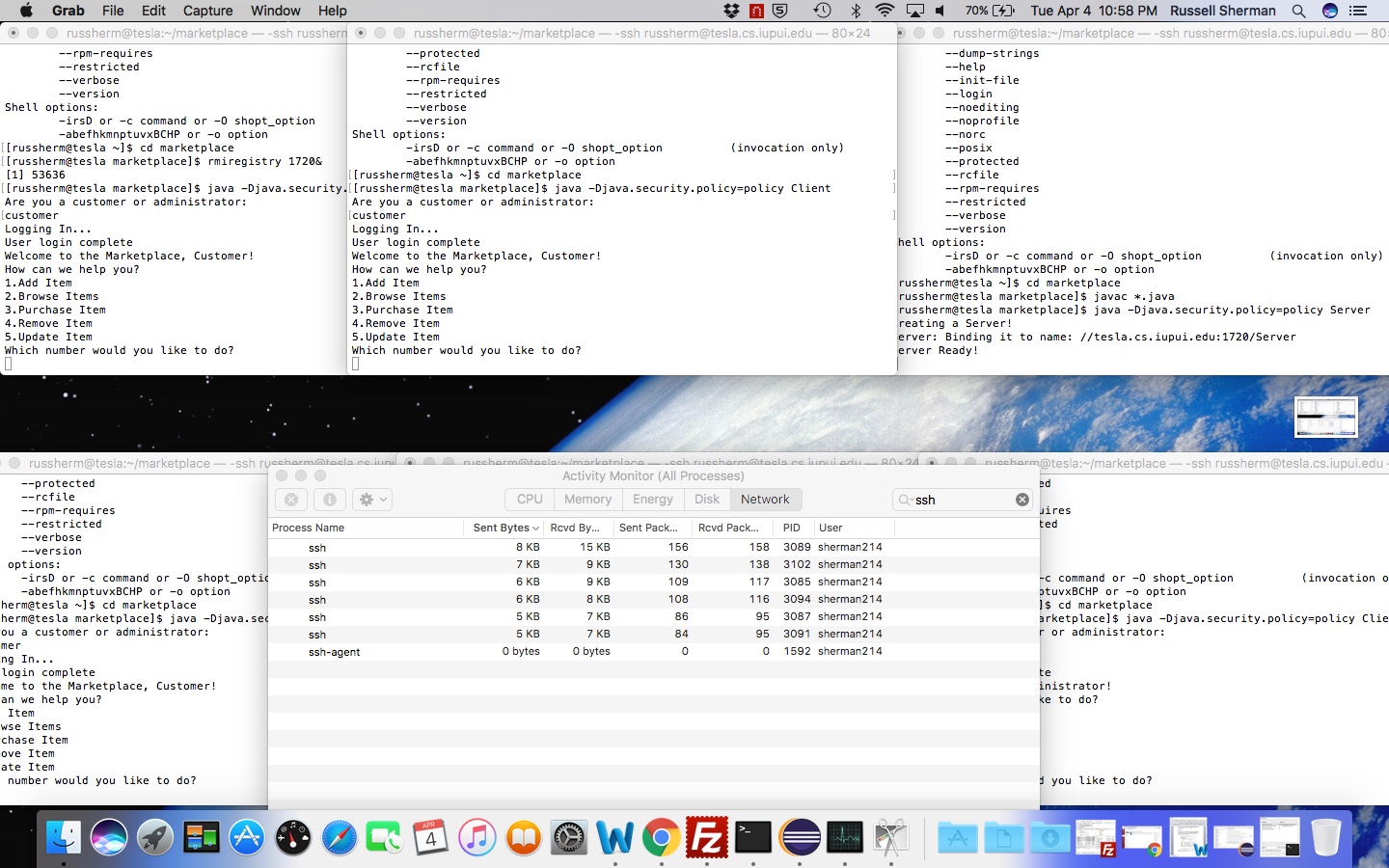
I have also opened the Activity Monitor to track the concurrency and threads of my Marketplace. I will track the CPU, Memory and Network to see how the JavaRMI is dealing with having five clients from different addresses.



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.

My databases looked like this:

