Assignment #1 Report

Russell Sherman

February 2, 2017

CSCI 50700

Requirements for the Online Marketplace

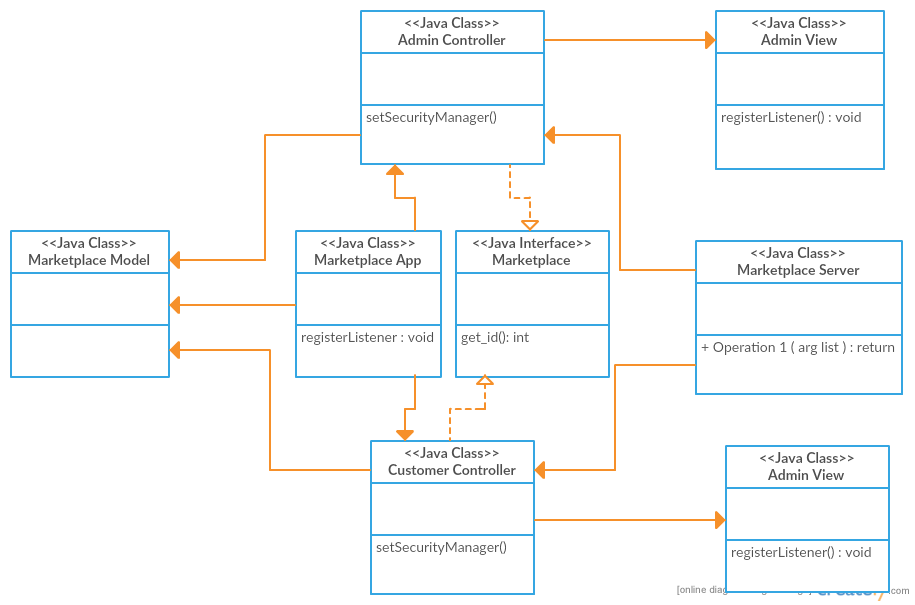
After meeting with the client, it was determined that they would like an online marketplace built that would allow them to sell good and possibly services to users all over the world. They would like the ability to separate what the customer can see, and what an administrator of the marketplace would 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 #1

With the bigger picture requirements in mind, the first step in creating the marketplace is getting a skeleton version of it up and running. Using what we learned in class on the domain model and the model-view-controller pattern, we needed to design our domain model for the first stage of our marketplace. The domain model should include any classes we need to get the marketplace connected to a server and running. To connect to the server, we need to use Java RMI to show a connection between the client and the server. The two fundamental things we are trying to accomplish are showing an understanding of Java RMI and how to implement the Model-View-Controller pattern.

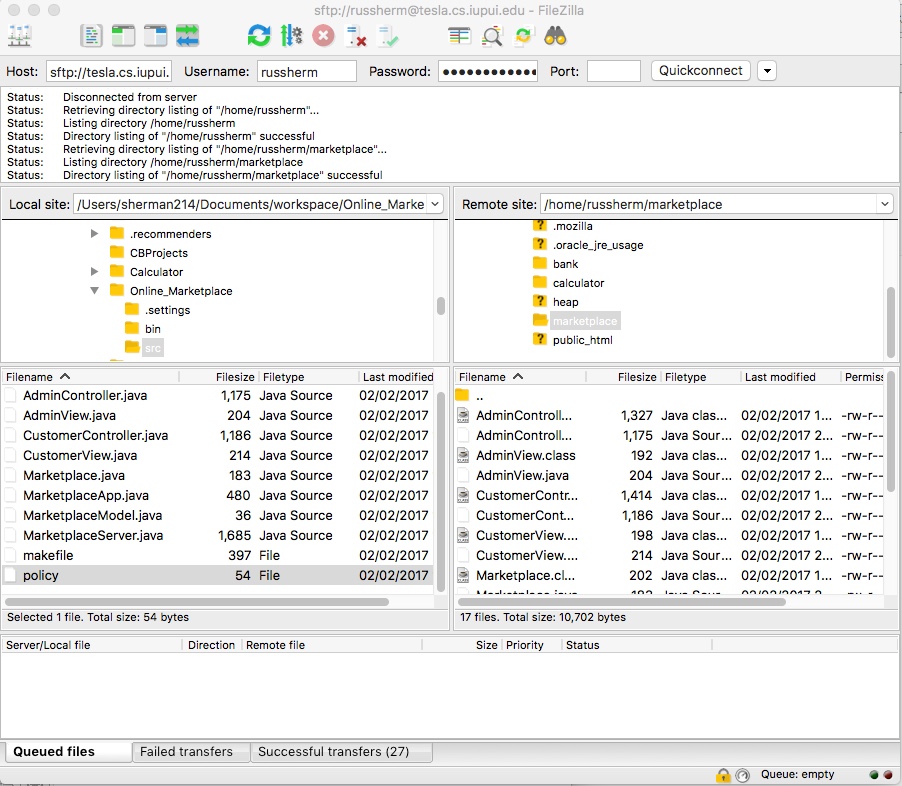
Currently, I have classes for views for the administrators and customers, as well as controllers for the administrators and customers. The controller is taking care of the client side of connecting to the server. The view classes are not being utilized quite yet, as I have not implemented anything for either the administrator or the customer to see. The other class that is doing a large portion of the work is the Marketplace Server class. This class takes care of the server side of the relationship. It establishes the Java RMI connection. I have hard coded the port number, so that I am not relying on the default port of 1099. As it stands, the Marketplace Model class is waiting on being developed as we begin to introduce more requirements. I may end up moving the server connection into the model, but for this stage I thought it would be best to keep a separate server class. The last class I have is the Marketplace App class that attempts to create the Model-View-Controller structure. At this point, this class is not complete and needs to be developed as the marketplace and requirements expand. Finally, I have a Marketplace Interface for the get\_id method used remotely by the client. All of these classes may not be present for future assignments but for getting our skeleton up and running, I thought it would be beneficial that they are included.

Domain Model

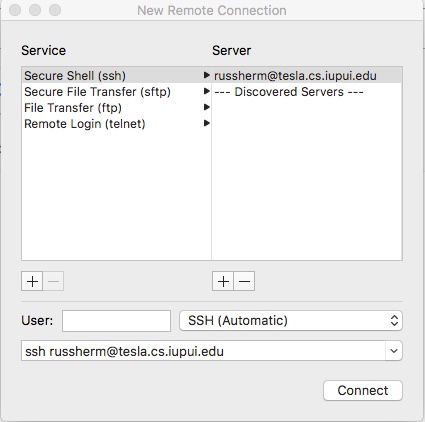


Sample Run

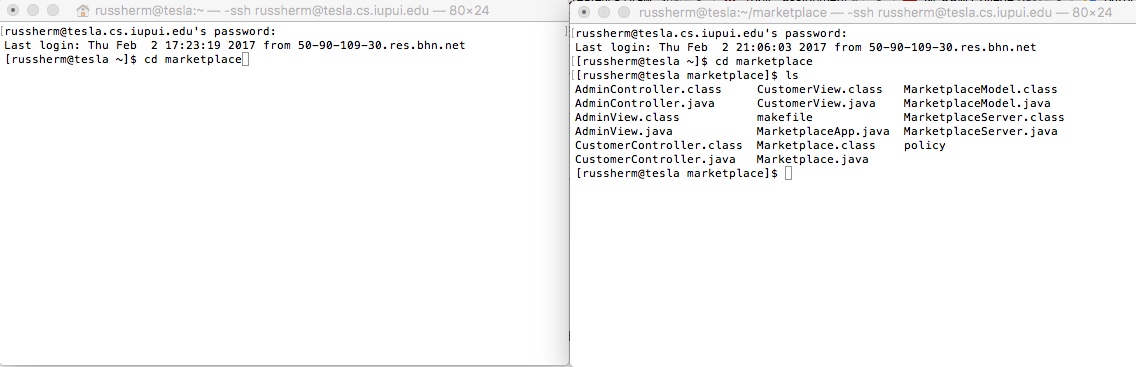
After saving my files in Eclipse, I open a program to transfer my files from my local machine to the Tesla server. Since I am unable to use PuTTY, I use FileZilla and sign on to the Tesla server using “tesla.cs.iupui.edu” followed by my username and password. Upon signing in the tesla, I am able to transfer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



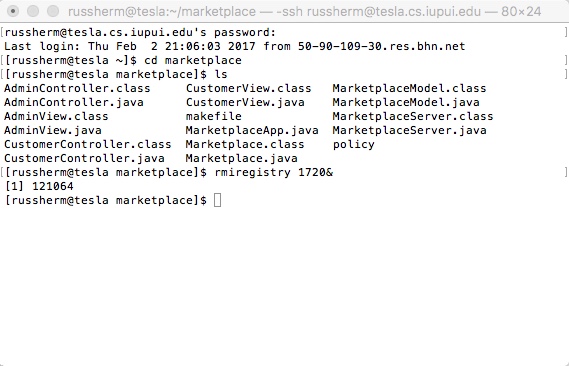
Now that my files are on the tesla server, I can open a terminal and connect to the tesla server via a remote connection.



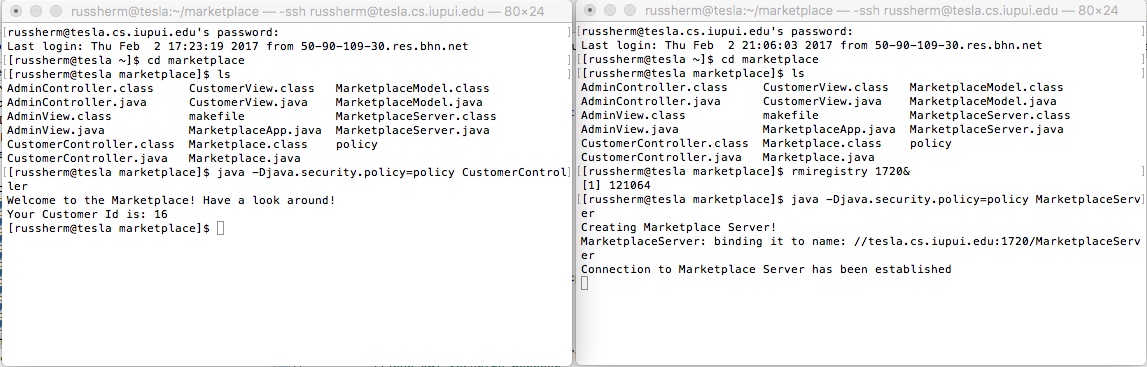
After logging in with my password, I am connected to the tesla server. I need to repeat these steps so I have two terminal windows open that are logged into tesla. I need to find the correct directory, and can do this by changing the directory to marketplace. Once I have done this, I can type ‘ls’ to see all the files in this directory. This will include all the files I just moved from my local machine to the tesla server.



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 know I am connected based on the “[1] 121064”. 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 Marketplace Server class, and also one of my controller classes. For simplicity, I will just run the Customer Controller, as the Admin Controller is similar. To run the Marketplace Server file, I will enter the following “java -Djava.security.policy=policy MarketplaceServer”. I will do the same in the other terminal for the Customer Controller file, entering “java -Djava.security.policy=policy CustomerController”.



As we can see, the Marketplace Server file was able to create a server and establish a connection. The client side using the Customer Controller file was able to welcome the customer and give them a customer id. The final step is closing the Java RMI registry connection. We can do this by bringing it to the foreground using ‘fg’ and then using the ‘Ctrl-C’ command.

