For the past few weeks, our software development team has been hard at work designing, testing, and programming a streamlined product which allows people trying to sell products to be able to connect with customers online. We designed our software to be user friendly, simple, and accomplish the goal of giving both sellers and buyers from around the world the marketplace of their dreams, all within a Java application. Our application utilizes both an easy to use client which can run on any computer, as well as a backend server which runs around the clock to process orders, sellers inventory, and even tracks the history of what customers have purchased. Alongside these classes, We have written several useful methods within our customer, product, seller, and store classes, which are used by both the server and the client. For example, the product class contains details on a product's number, description, as well as how many are in stock. Our seller and customer classes contain methods which are called within the server to place orders, add items to the store, add items to a shopping cart, and even allows sellers to view data on buyers past transactions so they can be more informed on what products to sell. Finally, our store class is used to store all of the products on the marketplace. Writing code within multiple classes allows our server and client classes to appear more concise, and allows us to have access to methods within both of them without having to redefine them. Our server and client classes work in tandem with each other to receive, process, and act upon user input.

Our client class’s primary function is to handle said user input, as well as show JoptionPanes which prompt further action. We use a variety of different JoptionPanes, such as dropdown menus, keyboard entry, and error messages to provide a clear and concise user experience. Once a user interacts with our clients cutting edge JoptionPane menu system, such as a user adding an item to their shopping cart, creating a new account, or a seller adding a product to the store, their request is then sent to the server for processing. This allows our client to have an impressive arsenal of features. For example, customers have access to every store on the marketplace, and can view products, their information, and be able to buy anything in just a few clicks. Our customers also have access to a shopping cart, so that they can browse the store at their own pace, and be able to checkout all of the products they wish to purchase in one simple checkout. Our customers can view the store directly, or view it after sorting it by price or quantity available. Our sellers also have access to a wide array of features, as well as access to data on how their store is doing. Sellers can directly add products to a store, as well as manage multiple stores at the same time. They can set the price, quantity available, as well as upload a description of their product. Sellers can also view data on what products have been purchased, as well as view the shopping carts of individual customers. Finally, sellers can use CSV files to add products to their store in bulk. On both the customer and seller side of our software, our client does an amazing job of implementing our products functionality, and conveying user inputs to our backend server. Within our server we not only process user inputs and send back directions for the client, we also store user data such as accounts made on the application and past transactions. Our server also handles errors, such as invalid logins, invalid inputs when creating accounts, or trying to buy more of an item than there is in stock, and directs the client to show error messages corresponding to the situation. Within our server class, there are comments documenting exactly what is happening in each step is processing, as well as the purpose of each individual method which is being called. This not only makes communication within our team more effective, it also serves to make future developers jobs easier if this software is ever updated, changed, or further improved upon. Our team was very deliberate in design choices like this, and has made similar choices to further improve our software. For example, we chose to write our client using JoptionPanes rather than Jframes, as it better fits with the use case of our product, an easy to use and simple program which is centered on user input. JoptionPanes provide a myriad of different ways to prompt user input, from simple keyboard entry, to buttons, and even dropdown menus. We had taken efforts to make sure the user has the simplest and most streamlined user experience, hence making the input system easy to use and straightforward. Additionally, we chose to use a simple number based way of processing user inputs, saving inputs as integers within the server, each corresponding to different actions taken by the client. This makes our server code easy to read, as one can simply look at the comment at the top of the class, and know where to look to find where in the server. All of our processing, data storage, and decision making happens on the server side of our program, allowing our client to run smoothly as it prompts the user with further input. Our backend processing is designed to be able to take user input, convert it into integers, and use said integers to prompt which actions the server should take next. Our processing is smooth, and our server is designed to make sure all actions are deliberate and that invalid inputs throw errors. We were very meticulous with our error handling, making sure that any invalid inputs, or invalid selections result in an error message to the user. It is important that our program works only as intended, and that all invalid values are dealt with accordingly. In most cases, this means showing the user an error window, telling them what was wrong with their input, and prompting them to retry the past input window. This can occur when the user inputs an invalid email at login, tries creating an account with a username not in email format, tries to view their purchase history when it is empty, or tries purchasing an invalid amount of a product, along with other cases. We also made an effort to make sure that after a user fully completes an action, such as adding a product to the store as a seller or buying one as a customer, that they will be redirected back to the main menu. This ensures a seamless user experience, and that users are able to accomplish multiple tasks without having to relaunch the program. These design choices not only make our product function smoother, but also ensure that our product is the most user friendly, complete, and error proof version of itself that it can be.