Implementation and Testing

For the implementation of the system. We used a client-server architecture similar to the kind used in most web applications but restricted to the secure network of the store. The cash register application, customer display application, and payment authorization system application all function as clients connecting to a websocket server when they are run. The checkout system application functions as the websocket server and it is the central controller for the entire system. The checkout system coordinates everything event in the system by sending and forwarding messages to their appropriate destinations. These messages trigger actions in the appropriate client applications and the clients send the response back to the server. By using this architecture the subsystems (cashier_register, customer_display, and payment_authorization_system) can all operate independently of one another. They are only aware of the server which simplifies things. However, this system does have cons associated with it. Primarily, it creates a central point of failure. Should the server ever go down for any reason, the entire system goes offline. To prevent the catastrophic loss of data we replicated the data across multiple redundant databases (see workspace folders).

In the demo I forgot to discuss restock/reorder of stock items. The cashier_register application checks for stock availability in real time with each item scanned and the server reorders the item from the supplier when supplies are low (see Supplier class in checkout_system). The cashier_register application has a Printer class that prints the receipt for the customer (see Printer class in cashier_register). It also has a Register class for the register drawer in case they pay with cash or check (see Register class in cashier_register).

While testing the application we used a deep copy database containing the current state of the final database. By doing this we avoid corrupting the data in the project database. The most challenging part was deciding how the “scale item” function worked. We decided to give bulk items a per item weight in the record and the price was a per unit of weight price. So, if an items record has a price of 10, and weight of 2, the cost for a single item would be $20 (2 * 10). Generating the test data for the other application functions was relatively straightforward.