Mini Project - Online Store

Nilay Kamat - IMT2021096 (Nilay.Kamat@iiitb.ac.in)
Github : https://github.com/Nilsiloid/OS-MiniProject

1 Overview

This project simulates the working of an Online Retail Store via the terminal. I have used a client-server architecture for the implementation and the requests to the server from the client are sent using sockets. The server then processes each request and returns an appropriate result for the same. There are 2 different types of clients - based on privileges and access permissions - User or Admin. A regular user(i.e. User) is able to view the available products and add required products to their cart. The Admin on the other hand is able to add or edit any products that are currently available in the store.

The program uses Socket programming for communication between the server and the client. The client side and server side interact using system calls, and the server side interacts with the data files using file locking mechanisms.

2 Application Architecture

The application has the following files:

- User.c - this file contains the code to be run at the client side. Implements the Users(Admin and User).
- Server.c - this file contains the code to be run at the server side.
- header.h - this file contains the structures and macros to be used in the program.
- Cart - this file contains the carts of the customers registered at the store. A customer is registered at the store only if they have items available in their cart.
- Products - this file contains the inventory of the store, i.e. the products currently in the store with their product ID, quantity and price.
- Receipt - this file contains the receipt for the User after payment.

3 How to Run the Code?

1. Open 2 terminal windows.
2. In the first terminal window, run the following commands:

   ```
   $ gcc Server.c -o server
   $ ./server
   ```

   This starts up the server side of the project. It will cause the server to be ready for connections from clients.
3. In the second terminal window, run the commands:

   ```
   $ gcc User.c -o User
   $ ./User
   ```

   This connects the client to the server.
4. We can then run the program as is directed by the client program.

4 header.h

The following structures and macros are used throughout the program to pass data among the server and client.

- struct Product - the struct to store the product info (productID, productName, qty and price).
- struct Cart - the struct to store the cart info for a customer (customer id, the list of products in the cart). The max number of products that can be added to cart at a time has been set to 50.
- struct Receipt – the structure used to store the items that the user has billed and will be paying for. It stores information about the product(productID, productName, qty and price) and the paymentStatus along with msg for any comments regarding the item listed.

5 Functionalities of the user

Normal User (Customer) The program intends to provide the following functionalities to customers (normal users):

- View Available Products : To view the items available at the store with their respective quantities in stock and the prices, so that the customer can know what the store offers.
- Add item to Cart : To add a product to their cart, along with the quantity of the product to be ordered.
- Update Cart items : To edit the quantity of any existing product in the customer's cart. Also let's customer delete the item from their cart.
- View Cart : To allow a customer to view the items present in their cart currently along with their quantity and prices.
- Payment : To proceed for payment for all the products in their cart. Following successful payment, a receipt is generated for the user to view his order.
- Log out : To log out from being a normal user(customer).

Admin User The Admin user serves as the administrator of the OnlineStore. They can manage the products in the online store, and has the following functionalities available to them:

- Add Product : To add a new product to the store.
- Delete Product : To explicitly remove products from the store.
- Update Product : To update the quantity available and/or the price of an existing product in the store.
- Display Products : To see the current inventory, i.e. the items currently present in the store.
- Log out : To log out from being an Admin user.

6 Functionalities of the server

Server The server serves the requests of the client (whether a customer or the Admin) and has functionality to support the requests as mentioned above. It essentially handles all the "Back-End" work to display the required data to the users. It creates and stores the database for the Products, Cart items and Receipts and uses them to serve the user requests.

7 Implementation of User.c

7.1 Helper functions

The code uses the following helper functions for modularity:

1. loginMenu()
   Upon running the code, any user is greeted by the loginMenu() which gives 3 options - to log in as Admin, to log in as Customer or to exit the program. This function is called inside a while loop to display above options and let the user log in as per choice.

2. adminMenu() and userMenu()
   These functions are also called inside while loops to display the options available to the Admin/-Customer respectively. Looking at the options from these menus, the user enters his choices. (Customer has options 1-6 whereas Admin has 1-5).

3. displayProducts(struct Product pdt), displayReceipt(struct Receipt rcpt)
   Both of these functions are used to display information regarding the Products. displayProducts() displays the products that are available in the inventory of the store - the information provided is the productID, productName, quantity and price. On the other hand, displayReceipt() displays the products that a certain customer has added to their Cart - the information provided here is the same as displayProducts() but with 2 additional data items - paymentStatus of the products and a message(msg) to inform about any changes in quantity due to availability in the store inventory.

7.2 Implementation of Client functions

Once the connection to the server has been made, the program then takes the input for the type of user (1 for Admin, 2 for customer).

Admin Functions:- If the user choose 1, i.e. the Admin role, the adminMenu() function is called and displays the functionalities offered to the Admin. This choice(i.e. 2) is sent to the server to inform/notify the server about what kind of user is currently accessing the program.
The functionalities that can be used by an Admin are as described above, and the implementations are listed below (all inputs are taken from the user in the main() function):

1. To add a product to the inventory, we take the product id, product name, quantity in stock and the price as inputs from the user, and communicate the same to the server, which adds the product to the inventory, or reports the error in case of invalid data.

2. To delete a product from the inventory, we simply ask for the product id. If the product id is correct, the product id is deleted, else an error message is displayed.

3. To update a product in the inventory, we take the product ID and the updated quantity and/or updated price. If the product was updated, we display the appropriate message, else an appropriate error message is generated and displayed. To edit a product's quantity / price, we take the product id of the product, and the new quantity / price. If no errors, the product is edited and the appropriate message is displayed, else appropriate error message is returned. Prices and quantities updated by the admin are not immediately seen in the cart, and old values are visible if the user sees his cart. However once the customer goes to his payment page, he sees the updated values.

4. To display available store Products, we use the displayProducts() function to get a response from the server, and display each Product's information.

5. When the Admin wants to log out, we simply exit the loop using break statement.

Customer Functions:- If the user chooses 2 - the userMenu() function is called and displays the functionalities offered to the Customer. This choice(i.e. 2) is sent to the server to inform/notify the server about what kind of user is currently accessing the program.
The functionalities that can be used by a Customer are as described above, and the implementations are listed below (all inputs are taken from the user in the main() function):

1. To display available store Products, we use the displayProducts() function to get a response from the server, and display each Product's information.

2. To add a product to a customer's cart, we take the customer's ID, the ID of the product they want to add and the quantity as inputs. In case of errors(such as invalid ID's), we communicate the error to the user, else the cart gets updated at the server side.

3. To update a product existing in the cart, the procedure is similar. Here, we query whether the customer wants to update the quantity of an item or delete it altogether from the cart(options 1 and 2 respectively). If the customer wants to update, we take the customer's ID, the ID of the product they want to update and the updated quantity as inputs. If the customer wants to delete, we take the customer's ID and the ID of the product they want to delete as inputs.
   Again, checks for invalid data are done here as well.

4. To view the cart of a customer, we take the customer ID as input, and return the cart obtained upon querying via the server, or return an empty cart if the customer id is wrong. Again, checks for invalid data are done here as well.

5. To pay for the items cart, the cart is first displayed to the customer, following which the total amount payable is calculated and displayed to the user. The customer is then asked to enter a confirmation(yes) to pay, else decline. If ijesïs entered, we generate the receipt, else we allow the customer to go back to the userMenu().

6. When the customer wants to log out, we simply exit the loop using break statement.

7.3 Socket Setup in Client program

The main() function first consists of socket setup then followed by the implementation for the project. The code for the socket setup is as follows:

```
int main(){
    printf("Connecting to the server\n");
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd == -1){
        perror("Error: ");
        return -1;
    }

    struct sockaddr_in server;

    serv.sin_family = AF_INET;
    // serv.sin_addr.s_addr = INADDR_ANY;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv.sin_port = htons(5000);

    if (connect(sockfd, (struct sockaddr *)&server, sizeof(server)) == -1){
        perror("Error: ");
        return -1;
    }
}
```

8 Implementation of Server.c

8.1 Helper functions

The code uses the following helper functions for modularisation:

1. createProductFile()
   This function is used to create a file "Products"that stores information about the products that are currently in the inventory of the store(using Product structure). The Products file is generated in such a way that the last Product structure to exist in the file will have all it's values set to -1. This will hence indicate to the program when reading from the Products file that we have reached the end of the products.

2. addProduct(productID, productName, quantity, price, clientFD)
   This function is used by the Admin user to add a new product to the inventory of the store. The product information inputted by user is passed to the server and to this function via the main() function as arguments. Using these parameters, this function checks if the product ID is a duplicate and thus, returns an appropriate message to the client. Here, we are overwriting the last product's information and thus add an extra -1 Product"to the end of the file in order to continue with the logic used while creating "Products"file.

3. deleteProduct(productID, clientFD)
   This function is used by the Admin user to delete a currently existing product from the store inventory. The productID inputted by user is passed to server and then to this function as argument via the main(). Here, the logic used in implementation is to copy all the products whose productID doesn't match the argument from the file "Productsïnto a temporary file "temp"which we rename "Productsüpon the transfer of all required products and deletion of the original "Products"file. We have used a variable deleteFlag to know if that product was deleted or not. An appropriate message is then sent to the client by the server.

4. updateProduct(productID, quantity, price, clientFD)
   This function is used by the Admin user to update a currently existing product's quantity and/or price. We search through the "Products"file and match the productID passed as an argument. If we are able to match it, we update the values in that specific Product structure and rewrite it into the file(also, set the variable updateFlag to 1). If we are unable to match the ID, updateFlag remains 0 and thus an appropriate message is sent to the client in both the above scenarios.

5. displayProduct()
   This function displays all the products available in the store inventory. In fluidity with the logic used in the creation of "Products"file, we read from the file and output the product information to the client until we encounter the -1 Product", i.e the product with all values assigned -1. We actually write this product to the client program as well but upon reading it, the client decides to not call it's own displayProducts() function to output the information.

6. createCart()
   This function is called when a new customer adds an item to their cart. This function opens the "Cart"file which stores instances of the Customer structure - the customerID and an array of 50 Product structure instances. This implies that each customer's cart can have at max 50 different products.
   Whenever a new customer adds an item to their cart, a new instance of struct Customer is created with just the customerID set to the correct value and all the productID's in the array set to -1.

7. EditCart(customerID, productID, quantity, update, delete, clientFD)
   This function is used by the Customer to edit their cart items. It has a 3 way usage in the sense that it is used for addition/updation and deletion of items to and from an already existing customer cart. The update and delete variables indicate to the function as to what function it must implement among the 3.

   - update = 0 and delete = 0: The function will add an item to an already existing cart.
   - update = 1 and delete = 0: The function will update an item in an already existing cart with the help of all the mentioned parameters. It will update the existing quantity to the one mentioned in the parameter(if possible).
   - update = 0 and delete = 1: The function will delete an item from an already existing cart with the help of the customerID and productID.

   An appropriate message is sent to the client regarding the transaction that took place - be it adding an item, updating an item, deleting an item or any errors in doing so.

8. emptyCart(customerID)
   This function is called by the Payment() function only after the Customer has succesfully paid for their items. This clears the cart of the customer by resetting all productID's in the array to -1.

9. displayCart(customerID, clientFD)
   This function is used by the Customer to view items that they have added to their cart. The server opens the "Cart"file and matches the customerID with the ID passed as a parameter. Once the ID

matches, we iterate through all the products in the array and write(to the client) the ones who's productID is not -1 or essentially the ones who's ID's have been set to some value.

10. totalCalc(customerID, clientFD)
    This function is used to calculate the total amount payable by the user for the items that exist in their cart. Here, we have to handle the situation in which the items were added to cart before any changes were made to the "Products"file - such as a change in quantity or price. The quantity of each product in the cart is checked with the currently available quantity for that product in the store inventory and the receipt(before payment) is generated with all the product information and 2 new attributes - the payment status and a message regarding the quantity. This is written into a new file called "Receipt". The total amount payable is calculated by summing the multiplicative product of the price of the Product in the "Products"file and the quantity that is being bought.

    The total value is then returned to the main() and is written to the client to be outputted along with the receipt.

11. Payment(customerID, clientFD, paymentFlag)
    This function is used to verify and process the payment made by a customer for their product. If the paymentFlag is 1, the program understands that the customer has confirmed to buying the items present in their cart. The program calls the function emptyCart() defined above to reset the customer's cart and sets all the paymentStatus' to "Payment Confirmedïn the customer's receipt. This new receipt is then written to the client program to output it for the user.

8.2 Implementation of Server functions

The program reads the input sent by the clients about the type of user(1-Admin, 2-Customer) and the choice of the user, and calls the appropriate helper functions for the case.

Admin functions

1. If the Admin chooses to add a new product to the store inventory, the Server program will read the various product attributes sent by the Client program and call the addProduct() function passing these parameters. The addProduct() will then carry out its task and add the product to the store inventory if the data is valid.

2. If the Admin chooses to delete an existing product from the store inventory, the Server program will read the productID sent by the Client program and call delProduct() function to delete the product corresponding to the received productID.

3. If the Admin chooses to update an existing product in the store inventory, the Server program will read the updated attributes sent by the ClientProgram and call the updateProduct() function passing above parameters to update the product corresponding to received productID.

4. If the Admin chooses to view the products available in the store inventory, we will simply call the displayProduct() function which will carry out the required task.

5. If the Admin chooses to log out, we will exit the loop using break statement.

Customer functions

1. If the Customer chooses to view the products available in the store inventory, we will simply call the displayProduct() function which will carry out the required task.

2. If the Customer chooses to add a product to their cart, the Server program will read the customerID, productID and quantity sent by the Client program and call the EditCart() function passing these attributes along with update and delete values, both set at 0.(this will indicate an addition in cart)

3. If the Customer chooses to update a product in their cart, the Server program will read the customerID and choice sent by the Client program and call the EditCart() function. Now based on if the choice was update(1) or delete(1) the extra attributes will be read and passed along with the corresponding update/delete value set to 1.

4. If the Customer chooses to view the items present in their cart, the Server program will call the displayCart() function and pass the customerID read from the Client program.
5. If the Customer chooses to pay for the items in their cart, the Server program will first calculate the total payable amount using the totalCalc() function and then upon receiving confirmation to execute payment, will call the Payment() function.
6. If the Customer chooses to log out, we exit the loop using break statement.

8.3 Socket Setup in Server Program

The main function first consists of socket setup, code for which is as follows:

```
int main(){
    int new_fd, sockfd;
    printf("Setting up the server!\n");

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        perror("Error: ");
        return -1;
    }

    struct sockaddr_in server, client;
    server.sin_family = AF_INET;
    // server.sin_addr.s_addr = INADDR_ANY;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(5000);

    int opt = 1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)))
    {
        perror("Error: ");
        return -1;
    }
    if (bind(sockfd, (struct sockaddr *)&server, sizeof(server)) == -1)
    {
        perror("Error: ");
        return -1;
    }
    if (listen(sockfd, 5) == -1)
    {
        perror("Error: ");
        return -1;
    }
    printf("Server set up successfully!\n");
}
```

Following this, any client requests are accepted, and a fork() is called to implement a concurrent server mechanism.

9 OS Concepts used in Implementation

9.1 File Locking

The program uses mandatory locking in the program whenever we are trying to read/write from a file. The 3 files used are - "Products", "Cartänd "Receipt".

1. When we are writing to the "Products"file to add/update/delete a product or change it quantity upon payment, a write lock has been used to ensure incorrect data is not read by another user.

2. When we are reading from the "Products"file to display the available products, we implement a read lock on the file.

3. Similarly with the "Cart"file. When we are editing the cart items, we implement a write lock and when displaying a customer's cart to them, we implement a read lock.

4. With the "Receipt"file, we implement a write lock both while calculating total payable amount and during payment confirmation.

9.2 Socket programming

The program uses sockets to communicate between the server and the client.

1. The program uses the server side socket system calls (socket, bind, listen, accept) to setup the socket at the server side, and client side socket system calls (socket, connect) at the client side.

2. The fork() system call is used to setup a concurrent server.

3. All reads and writes (from the socket) use read and write system calls. These read and write system calls are used for communication between the server and client - in the project, to send a customer's ID or the quantity of a product or an appropriate message.

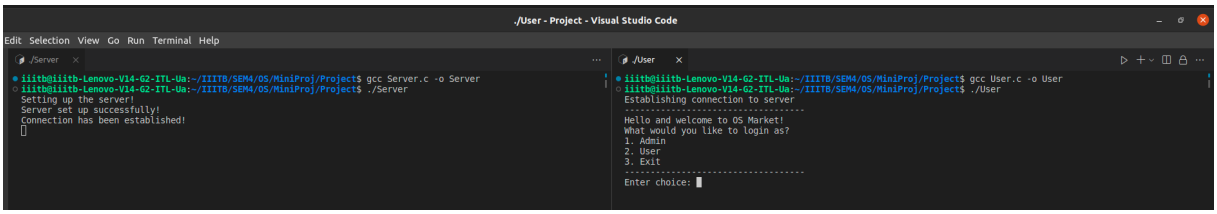
9.3 File handling

1. We use basic functions such as open, read, write to perform the basic read and write operations on the files.

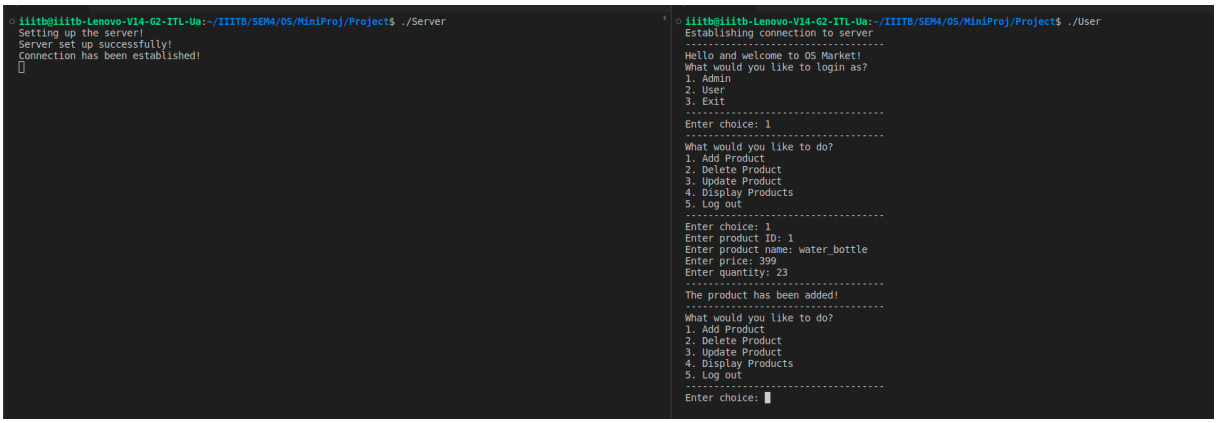
10 Screenshots of working

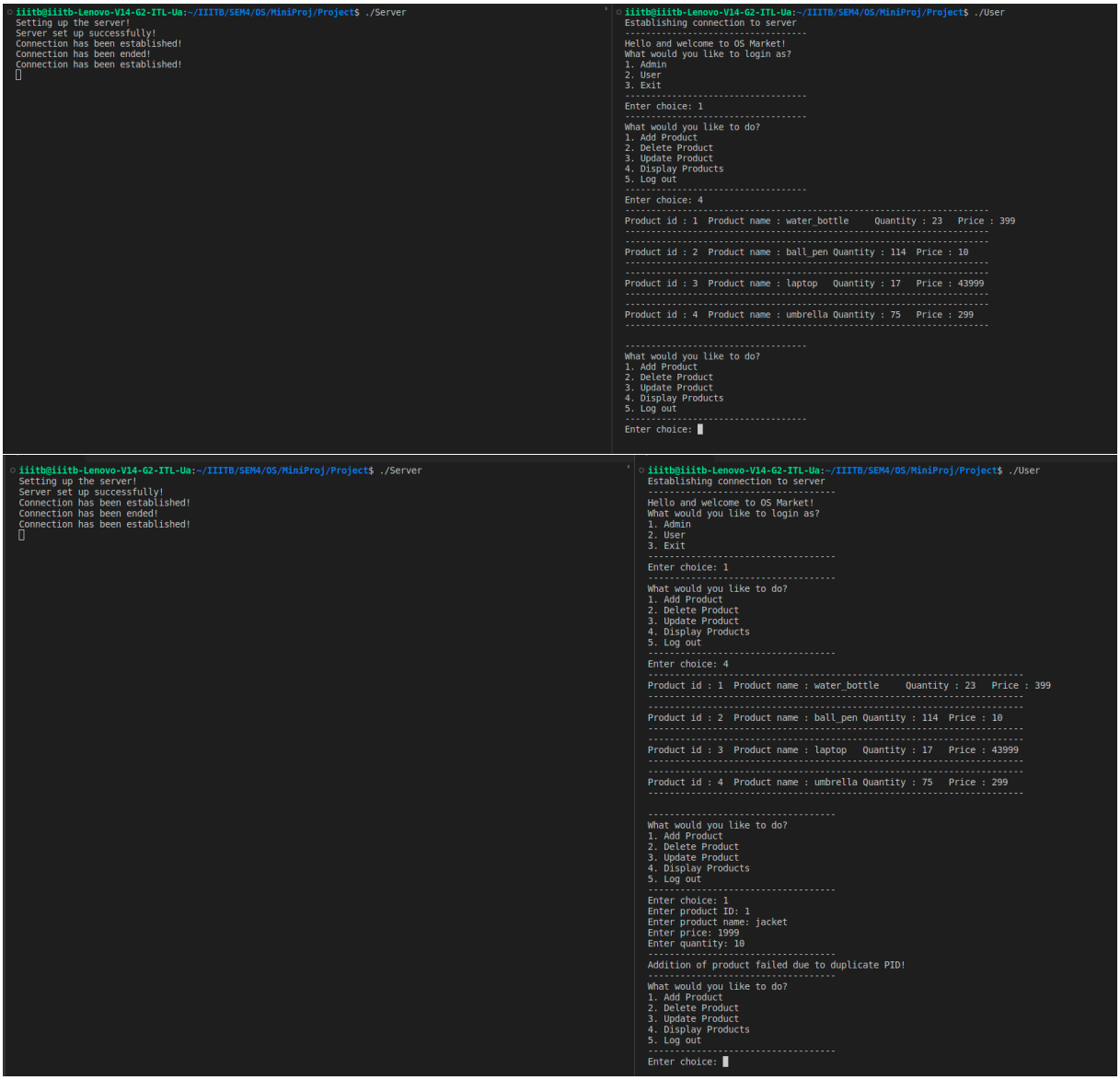
Attached below are a few screenshots of the working of my project.

- This screenshot shows the initial server set up and the connection made with a client program.

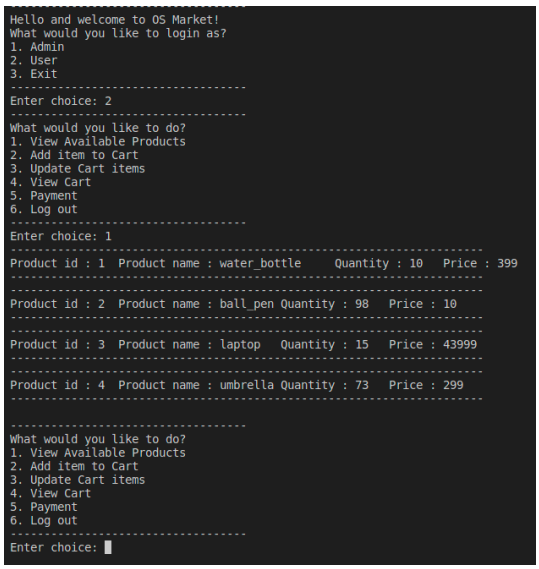


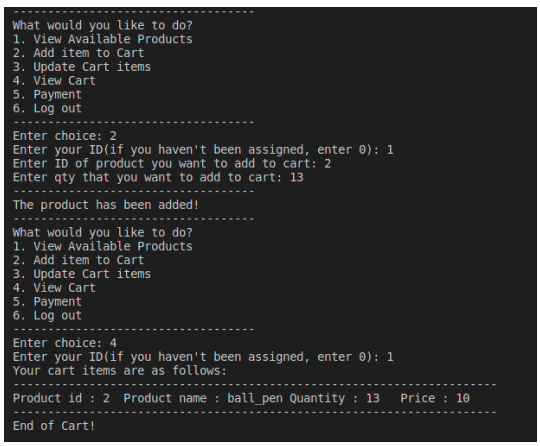
- The next 3 images shows the addition of a product to the store inventory by the Admin user, the list of available products upon addition of a few more products and a failed addition due to duplicated productID respectively.





- The next 3 images shows the list of available products when a customer requests to view the available products, the addition of an item to cart and the failure to add a product to cart due to invalid details respectively.





```
---------------------------------
What would you like to do?
1. View Available Products
2. Add item to Cart
3. Update Cart items
4. View Cart
5. Payment
6. Log out
---------------------------------
Enter choice: 2
Enter your ID(if you haven't been assigned, enter 0): 4
Enter ID of product you want to add to cart: 12
Enter qty that you want to add to cart: 999
---------------------------------
The product has not been added/updated/deleted due to invalid details!

---------------------------------
```