**ONLINE SHOPPING STORE**



Session: 2021 – 2024

**Submitted by:**

Muhammad Usman Asghar 2021-CS-46

**Supervised by:**

MISS MAIDA SHAHID

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Short Description about Project:**

This is a business application which mainly consists of three users. The users have different functionalities. Like user can form his account and then log in to his account to perform different functionalities which include the addition of the data of the user and generate total bill while the customer can see the no of items available.

**Users of Application:**

This project mainly has three users (Admin, Customer and Stock Manager). However, the Admin can add the further users too.

**Admin Functionalities:**

1. As an admin, he can add a user.
2. As an admin, he can Search a User.
3. As an admin, he can see a list of users.
4. As an admin, he can add an Item.
5. As an admin, he can search an Item.
6. As an admin, he can update an Item.
7. As an admin, he can remove an Item.
8. As an admin, he can give a discount.
9. As an admin, he can see the list of all Items.
10. As an admin, he can Sign Out.

**Customer Functionalities:**

1. As a customer, he can view all Items alphabetically.
   1. As a customer, he can view Item with respect to Price.
   2. As a customer, he can view Item with Discount.
   3. As a customer, he can search an Item.
   4. As a customer, he can buy an Items.
   5. As a customer, he can generate Bill.
   6. As a customer, he can sign out the menu.

**Stock Manager Functionalities:**

* 1. Items with dead stock
  2. As an stock manager, he can order the stock.
  3. As an stock manager, he can add the stock.
  4. As an stock manager, he can sign out.

**Data Structures:**

const int size = 500;

// All Data Structure of User

string userNames[size];

string passWords[size];

string roles[size];

string userName, passWord, role;

// Data Structure of all Items

string allItems[size];

float itemPrices[size];

int itemDiscounts[size];

int quantity[size];

bool valid = true;

bool signOut = false;

int option = 1;

// Arrays for Sorting

string allItemsSorted[size];

float priceSorted[size];

int discountSorted[size];

int quantitySorted[size];

// Loading Data

loadDataofUser(userNames, passWords, roles);

loadDataofItems(allItems, itemPrices, itemDiscounts, quantity);

/

**Functions Prototypes:**

// FUNCTION PROTOTYPES

bool loadDataofUser(string userNames[], string passWords[], string roles[]);

bool loadDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[]);

string DataSepraterUsingComma(string Record, int Field);

bool storeDataofUser(string userNames[], string passWords[], string roles[]);

bool storeDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[]);

string roleAssign();

bool isUserValid(string userNames[], string passWords[], string roles[], string userName, string passWord, string role);

bool AddaUser(string userNames[], string passWords[], string roles[]);

void view\_all\_user(string name[], string password[], string role[], int count);

bool wrong();

void Header();

void menu(string title);

int mainMenu();

int adminMenu();

void xit();

void TemplateUser();

void PrintUser(int index, string userNames[], string passWords[], string roles[]);

int SearchaUser(string userNames[], string passWords[], string roles[]);

bool AddanItem(string allItems[], float itemPrices[], int itemDiscounts[], int quantities[]);

int SearchanItem(string allItems[]);

void TemplateItem();

void PrintItem(int index, string allItems[], float allPrices[], int discount[], int quantities[]);

void StringSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[]);

void floatSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[]);

void SwapData(int indexi, int indexj, string arr1[], float arr2[], int arr3[], int quantities[]);

int customerMenu();

float priceAfterDiscount(float price, int discount);

int stockManagerMenu();

// Counts

int userCount = 0,

itemCount = 0;

**Complete Code:**

// ONLINE SHOPPING STORE

#include <iostream>

#include <fstream>

using namespace std;

// FUNCTION PROTOTYPES

bool loadDataofUser(string userNames[], string passWords[], string roles[]);

bool loadDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[]);

string DataSepraterUsingComma(string Record, int Field);

bool storeDataofUser(string userNames[], string passWords[], string roles[]);

bool storeDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[]);

string roleAssign();

bool isUserValid(string userNames[], string passWords[], string roles[], string userName, string passWord, string role);

bool AddaUser(string userNames[], string passWords[], string roles[]);

void view\_all\_user(string name[], string password[], string role[], int count);

bool wrong();

void Header();

void menu(string title);

int mainMenu();

int adminMenu();

void xit();

void TemplateUser();

void PrintUser(int index, string userNames[], string passWords[], string roles[]);

int SearchaUser(string userNames[], string passWords[], string roles[]);

bool AddanItem(string allItems[], float itemPrices[], int itemDiscounts[], int quantities[]);

int SearchanItem(string allItems[]);

void TemplateItem();

void PrintItem(int index, string allItems[], float allPrices[], int discount[], int quantities[]);

void StringSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[]);

void floatSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[]);

void SwapData(int indexi, int indexj, string arr1[], float arr2[], int arr3[], int quantities[]);

int customerMenu();

float priceAfterDiscount(float price, int discount);

int stockManagerMenu();

// Counts

int userCount = 0,

itemCount = 0;

// MAIN FUNCTION

int main()

{

const int size = 500;

// All Data Structure of User

string userNames[size];

string passWords[size];

string roles[size];

string userName, passWord, role;

// Data Structure of all Items

string allItems[size];

float itemPrices[size];

int itemDiscounts[size];

int quantity[size];

bool valid = true;

bool signOut = false;

int option = 1;

// Arrays for Sorting

string allItemsSorted[size];

float priceSorted[size];

int discountSorted[size];

int quantitySorted[size];

// Loading Data

loadDataofUser(userNames, passWords, roles);

loadDataofItems(allItems, itemPrices, itemDiscounts, quantity);

// Sign In

while (signOut != true)

{

while (option > 0 && option < 3)

{

option = 1;

valid = true;

Header();

menu("Main Menu");

option = mainMenu();

if (option == 1)

{

while (valid != false)

{

Header();

menu("Sign In Menu");

cout << "Enter Username.........";

cin >> userName;

cout << "Enter Password.........";

cin >> passWord;

role = roleAssign();

valid = isUserValid(userNames, passWords, roles, userName, passWord, role);

if (valid == false)

{

if (wrong() == false)

{

system("cls");

valid = false;

}

}

else

{

if (role == "Admin")

{

while (option > 0 && option < 11)

{

Header();

menu("Menu");

option = adminMenu();

Header();

if (option == 1)

{

menu("Add a User");

bool flag = AddaUser(userNames, passWords, roles);

if (flag)

{

userCount++;

}

xit();

}

else if (option == 2)

{

menu("Search a User");

int index = SearchaUser(userNames, passWords, roles);

PrintUser(index, userNames, passWords, roles);

xit();

}

else if (option == 3)

{

menu("List of Users");

view\_all\_user(userNames, passWords, roles, userCount);

xit();

}

else if (option == 4)

{

menu("Add an Item");

bool flag = AddanItem(allItems, itemPrices, itemDiscounts, quantity);

if (flag == true)

{

itemCount++;

}

xit();

}

else if (option == 5)

{

menu("Search an Item");

int index = SearchanItem(allItems);

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

xit();

}

else if (option == 6)

{

menu("Update Price");

int index = SearchanItem(allItems);

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

if (index > -1)

{

int newPrice;

cout << "Enter the new Price...";

cin >> newPrice;

itemPrices[index] = newPrice;

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

}

xit();

}

else if (option == 7)

{

menu("Delete Item");

int index = SearchanItem(allItems);

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

if (index > -1)

{

allItems[index] = "\0";

itemPrices[index] = 0;

itemDiscounts[index] = 0;

cout << "Item Deleted Successfully...";

}

xit();

}

else if (option == 8)

{

menu("Add OR Change Discount");

int index = SearchanItem(allItems);

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

if (index > -1)

{

cout << "ENter the Dicount on that Item...";

cin >> itemDiscounts[index];

cout << "Dicount Added Successfully...";

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

}

xit();

}

else if (option == 9)

{

menu("List of Items");

cout << "1- All Items in Alphabatical Order" << endl;

cout << "2- All Items in Ascending Order of Price" << endl;

cout << "3- All Items with Discount" << endl;

int choice;

cout << "Select Option.....";

cin >> choice;

Header();

if (choice == 1)

{

menu("All Items in Alphabatical Order");

StringSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

}

else if (choice == 2)

{

menu("All Items in Ascending Order of Price");

floatSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

}

else if (choice == 3)

{

menu("All Items with Discount");

floatSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

if (discountSorted[i] != 0)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

}

}

xit();

}

else if (option == 10)

{

xit();

valid = false;

option = 1;

break;

}

// Storing Data

for (int i = 0; i < itemCount; i++)

{

allItemsSorted[i] = allItems[i];

priceSorted[i] = itemPrices[i];

discountSorted[i] = itemDiscounts[i];

quantitySorted[i] = quantity[i];

}

storeDataofUser(userNames, passWords, roles);

storeDataofItems(allItems, itemPrices, itemDiscounts, quantity);

}

}

else if (role == "Customer")

{

option = 1;

int buyCount = 0;

int indexofBuyedItems[size];

int quantityofBuyedItems[size];

while (option > 0 && option < 7)

{

Header();

menu("Menu");

option = customerMenu();

Header();

if (option == 1)

{

menu("All Items in Alphabatical Order");

StringSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

xit();

}

else if (option == 2)

{

menu("All Items in Ascending Order of Price");

floatSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

xit();

}

else if (option == 3)

{

menu("All Items with Discount");

floatSorting(itemCount, allItemsSorted, priceSorted, discountSorted, quantitySorted);

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

if (discountSorted[i] != 0)

{

PrintItem(i, allItemsSorted, priceSorted, discountSorted, quantitySorted);

}

}

xit();

}

else if (option == 4)

{

menu("Search an Item");

int index = SearchanItem(allItems);

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

xit();

}

else if (option == 5)

{

menu("Buy an Item");

int index = SearchanItem(allItems);

bool isAlreadyBrought = false;

int indexinarr, iee;

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

for (int i = 0; i < buyCount; i++)

{

if (index == indexofBuyedItems[i])

{

isAlreadyBrought = true;

indexinarr = indexofBuyedItems[i];

iee = i;

}

}

if (index > -1)

{

cout << "How many \* " + allItems[index] + " \* do you want to buy(not zer0)";

cin >> quantityofBuyedItems[buyCount];

if (quantityofBuyedItems[buyCount] > 0 && quantityofBuyedItems[buyCount] <= quantity[index])

{

quantity[index] = quantity[index] - quantityofBuyedItems[buyCount];

if (isAlreadyBrought == false)

{

indexofBuyedItems[buyCount] = index;

buyCount++;

}

else

{

quantityofBuyedItems[iee] += quantityofBuyedItems[buyCount];

quantityofBuyedItems[buyCount] == 0;

}

}

else if (quantityofBuyedItems[buyCount] > quantity[index])

{

cout << "Wrong Quantity...........You can buy less than or equal to...." << quantity[index] << "." << endl;

}

else if (quantityofBuyedItems[buyCount] < 1)

{

cout << "Wrong Quantity...........You have to enter quantity more tha zero" << endl;

}

}

xit();

}

else if (option == 6)

{

menu("Bill");

float total = 0;

cout << "Item Name\t\t~~Item Price~~\t\tQuantity\t\tAfter Discounts\t\t Final Price\n\n";

for (int i = 0; i < buyCount; i++)

{

float discountedprice = priceAfterDiscount(itemPrices[indexofBuyedItems[i]], itemDiscounts[indexofBuyedItems[i]]);

float totalofanItem = discountedprice \* quantityofBuyedItems[i];

cout << allItems[indexofBuyedItems[i]] << "\t" << itemPrices[indexofBuyedItems[i]] << "\t" << quantityofBuyedItems[i] << "\t" << discountedprice << "\t" << totalofanItem << endl;

total = total + totalofanItem;

}

cout << "\nThe Amount is.........................................." << total << " Rs.";

cout << "\nThank You for Visiting Usman Shopping Store..............................." << endl;

xit();

}

else if (option == 7)

{

system("cls");

valid = false;

option = 1;

break;

}

for (int i = 0; i < itemCount; i++)

{

allItemsSorted[i] = allItems[i];

priceSorted[i] = itemPrices[i];

discountSorted[i] = itemDiscounts[i];

quantitySorted[i] = quantity[i];

}

storeDataofItems(allItems, itemPrices, itemDiscounts, quantity);

}

}

else if (role == "Stock Manager")

{

option = 1;

int orderCount = 0;

int indexofOrderedItems[size];

int QuantityofOrderedItems[size];

while (option > 0 && option < 5)

{

Header();

menu("Menu");

option = stockManagerMenu();

Header();

if (option == 1)

{

menu("Items With DeadStock");

TemplateItem();

for (int i = 0; i < itemCount; i++)

{

if (quantity[i] == 0)

{

PrintItem(i, allItems, itemPrices, itemDiscounts, quantity);

}

}

xit();

}

else if (option == 2)

{

menu("Order Stock");

int index = SearchanItem(allItems);

bool isAlreadyOrdered = false;

int indexinarr, iee;

TemplateItem();

PrintItem(index, allItems, itemPrices, itemDiscounts, quantity);

for (int i = 0; i < orderCount; i++)

{

if (index == indexofOrderedItems[i])

{

isAlreadyOrdered = true;

indexinarr = indexofOrderedItems[i];

iee = i;

}

}

if (index > -1)

{

cout << "How many \* " + allItems[index] + " \* do you want to buy(not zer0)";

cin >> QuantityofOrderedItems[orderCount];

if (QuantityofOrderedItems[orderCount] > 0)

{

if (isAlreadyOrdered == false)

{

indexofOrderedItems[orderCount] = index;

orderCount++;

}

else

{

QuantityofOrderedItems[iee] += QuantityofOrderedItems[orderCount];

QuantityofOrderedItems[orderCount] == 0;

}

}

else if (QuantityofOrderedItems[orderCount] > quantity[index])

{

cout << "Wrong Quantity...........You can buy less than or equal to...." << quantity[index] << "." << endl;

}

else if (QuantityofOrderedItems[orderCount] < 1)

{

cout << "Wrong Quantity...........You have to enter quantity more tha zero" << endl;

}

}

xit();

}

else if (option == 3)

{

menu("Add Stock");

cout << "Item Name"

<< "\t"

<< "Item Stock~`Order`"<<endl;

for (int i = 0; i < orderCount; i++)

{

cout <<i+1<<"- "<< allItems[indexofOrderedItems[i]] << "\t" << QuantityofOrderedItems[i] << endl;

}

if (orderCount != 0)

{

int choice = -1;

while (choice < 0 || choice > orderCount)

{

cout << "Which one do you want to enter....";

cin >> choice;

}

quantity[indexofOrderedItems[choice-1]]+=QuantityofOrderedItems[choice-1];

indexofOrderedItems[choice-1]=0;

QuantityofOrderedItems[choice-1]=0;

}

xit();

}

else if (option == 4)

{

system("cls");

valid = false;

option = 1;

break;

}

storeDataofItems(allItems, itemPrices, itemDiscounts, quantity);

}

}

}

}

}

else if (option == 2)

{

system("cls");

signOut = true;

break;

}

}

}

return 0;

}

// END OF MAIN FUNCTION

// FUNCTIONS IMPLEMENTATION

// LOAD DATA OF USER

bool loadDataofUser(string userNames[], string passWords[], string roles[])

{

fstream file;

string lineFromFile;

file.open("UserData.txt", ios::in);

if (!file)

{

return false;

}

else

{

for (userCount = 0; !file.eof(); userCount++)

{

getline(file, lineFromFile);

if (lineFromFile == "\0")

{

return false;

}

else

{

roles[userCount] = DataSepraterUsingComma(lineFromFile, 1);

userNames[userCount] = DataSepraterUsingComma(lineFromFile, 2);

passWords[userCount] = DataSepraterUsingComma(lineFromFile, 3);

}

}

file.close();

return true;

}

}

// LOAD DATA OF ITEM

bool loadDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[])

{

fstream file;

string lineFromFile;

file.open("ItemData.txt", ios::in);

if (!file)

{

return false;

}

else

{

for (itemCount = 0; !file.eof(); itemCount++)

{

getline(file, lineFromFile);

if (lineFromFile == "\0")

{

return false;

}

else

{

allItems[itemCount] = DataSepraterUsingComma(lineFromFile, 1);

itemPrices[itemCount] = stof(DataSepraterUsingComma(lineFromFile, 2));

quantity[itemCount] = stoi(DataSepraterUsingComma(lineFromFile, 3));

itemDiscounts[itemCount] = stoi(DataSepraterUsingComma(lineFromFile, 4));

}

}

file.close();

return true;

}

}

// SEPARATE COMMA

string DataSepraterUsingComma(string Record, int Field)

{

int Comma = 1;

string RequiredRecord;

for (int i = 0; Record[i] != '\0'; i++)

{

if (Record[i] == ',')

{

Comma = Comma + 1;

}

else if (Comma == Field)

{

RequiredRecord = RequiredRecord + Record[i];

}

}

return RequiredRecord;

}

// STORE DATA OF USER

bool storeDataofUser(string userNames[], string passWords[], string roles[])

{

fstream file;

string lineFromFile;

file.open("UserData.txt", ios::out);

if (!file || userCount == 0)

{

return false;

}

else

{

for (int i = 0; i < userCount - 1; i++)

{

file << roles[i] << "," << userNames[i] << "," << passWords[i] << endl;

}

file << roles[userCount - 1] << "," << userNames[userCount - 1] << "," << passWords[userCount - 1];

file.close();

return true;

}

}

// STORE DATA OF ITEM

bool storeDataofItems(string allItems[], float itemPrices[], int itemDiscounts[], int quantity[])

{

fstream file;

string lineFromFile;

file.open("ItemData.txt", ios::out);

if (!file || itemCount == 0)

{

return false;

}

else

{

for (int i = 0; i < itemCount - 1; i++)

{

file << allItems[i] << "," << itemPrices[i] << "," << quantity[i] << "," << itemDiscounts[i] << endl;

}

file << allItems[itemCount - 1] << "," << itemPrices[itemCount - 1] << "," << quantity[itemCount - 1] << "," << itemDiscounts[itemCount - 1];

file.close();

return true;

}

}

// VALID USER

bool isUserValid(string userNames[], string passWords[], string roles[], string userName, string passWord, string role)

{

for (int i = 0; i < userCount; i++)

{

if (userName == userNames[i] && passWord == passWords[i] && role == roles[i])

{

return true;

}

}

return false;

}

// WRONG

bool wrong()

{

bool valid;

cout << "The credientals you entered are wrong...";

cout << "Press 1 to Continue Else 0 to Exit...";

cin >> valid;

return valid;

}

// HEADER

void Header()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\t(Online Shopping Store)\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

// MENU

void menu(string title)

{

cout << "\n-------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "\t\t\t\t\t\t\t<...>\t" + title + "\t<...>" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

}

// MAIN MENU

int mainMenu()

{

int mainMenu;

cout << "1- login" << endl;

cout << "2- Exit" << endl;

cout << "Select your choice...\n";

cin >> mainMenu;

return mainMenu;

}

// ADMIN MENU

int adminMenu()

{

int menu;

cout << "User DataBase\n\n";

cout << " 1- Add a User......" << endl;

cout << " 2- Search a User..." << endl;

cout << " 3- List of Users...\n " << endl;

cout << "Item DataBase\n\n";

cout << " 4- Add an Item......" << endl;

cout << " 5- Search an Item..." << endl;

cout << " 6- Update an Item..." << endl;

cout << " 7- Remove an Item..." << endl;

cout << " 8- Add Discount..." << endl;

cout << " 9- List of all Items..." << endl;

cout << " 10- Sign Out...\n " << endl;

cout << "Select your choice...\n";

cin >> menu;

return menu;

}

// ADD USER

bool AddaUser(string userNames[], string passWords[], string roles[])

{

int role = 0;

bool flag = false;

bool roll;

while (flag != true)

{

cin.ignore();

cout << "Enter the User Name(To be used at Login)...\n";

getline(cin, userNames[userCount]);

for (int i = 0; i < userCount; i++)

{

if (userNames[userCount] == userNames[i])

{

cout << "Username already exsist...\n";

TemplateUser();

PrintUser(i, userNames, passWords, roles);

return false;

}

}

flag = true;

}

cout << "Enter the Password\n";

getline(cin, passWords[userCount]);

roles[userCount] = roleAssign();

cout << roles[userCount] + "\tADDED\tSUCESSFULLY....\n";

TemplateUser();

PrintUser(userCount, userNames, passWords, roles);

return true;

}

// ROLE ASSIGNMEENT

string roleAssign()

{

int value;

string role = "\0";

cout << "Enter the Role\nPress\n 1- for Admin \n 2- for Customer\n 3- for Stock Manager\nYour choice is....";

cin >> value;

while ((value < 1 && value > 3))

{

cout << "Enter the Role\nPress\n 1- for Admin \n 2- for Customer\n 3- for Stock Manager\nYour choice is....";

cin >> value;

}

if (value == 1)

{

role = "Admin";

}

else if (value == 2)

{

role = "Customer";

}

else if (value == 3)

{

role = "Stock Manager";

}

return role;

}

// EXIT

void xit()

{

int null;

cout << "ENTER ANY KEY TO EXIT...";

cin >> null;

}

// TEMPLATE USER

void TemplateUser()

{

cout << "Role\t\t~~UserName~~\n\n";

}

// PRINT USER

void PrintUser(int index, string userNames[], string passWords[], string roles[])

{

if (index < 0)

{

cout << "No\tRcord\tFound" << endl;

}

if (roles[index] != "\0" && userNames[index] != "\0")

{

cout << roles[index] << "\t\t" << userNames[index] << endl;

}

}

// SEARCH USER

int SearchaUser(string userNames[], string passWords[], string roles[])

{

int choice = 0;

string searchstring;

cin.ignore();

cout << "Enter UserName..." << endl;

getline(cin, searchstring);

TemplateUser();

for (int i = 0; i < userCount; i++)

{

if (searchstring == userNames[i])

{

return i;

}

}

return -1;

}

// AND AN ITEM

bool AddanItem(string allItems[], float itemPrices[], int itemDiscounts[], int quantities[])

{

bool flag = false;

while (flag != true)

{

cin.ignore();

cout << "Enter the Item Name...\n";

getline(cin, allItems[itemCount]);

for (int i = 0; i < itemCount; i++)

{

if (allItems[itemCount] == allItems[i])

{

cout << "Item already exist...\n";

TemplateItem();

PrintItem(i, allItems, itemPrices, itemDiscounts, quantities);

return false;

}

}

flag = true;

}

cout << "Enter the Item Price...\n";

cin >> itemPrices[itemCount];

cout << "Enter the item quantity (1 or more)...\n";

cin >> quantities[itemCount];

while (quantities[itemCount] < 1)

{

cout << "Enter the item quantity (1 or more)...\n";

cin >> quantities[itemCount];

}

itemDiscounts[itemCount] = 0;

cout << allItems[itemCount] + "\tADDED\tSUCESSFULLY....\n";

TemplateItem();

PrintItem(itemCount, allItems, itemPrices, itemDiscounts, quantities);

return true;

}

// TEMPLATE ITEM

void TemplateItem()

{

cout << "Item Name\t\t~~Item Price~~\t\tQuantity\t\tDiscounts\n\n";

}

// PRINT ITEM

void PrintItem(int index, string allItems[], float allPrices[], int discount[], int quantities[])

{

if (index < 0)

{

cout << "No\tRcord\tFound" << endl;

}

else if (allItems[index] != "\0" && allPrices[index] != 0)

{

cout << allItems[index] << "\t\t" << allPrices[index] << "\t\t" << quantities[index] << "\t\t" << discount[index] << "%" << endl;

}

}

// SEARCH AN ITEM

int SearchanItem(string allItems[])

{

string searchstring;

cin.ignore();

cout << "Enter Item..." << endl;

getline(cin, searchstring);

for (int i = 0; i < itemCount; i++)

{

if (searchstring == allItems[i])

{

return i;

}

}

return -1;

}

// STRING STORING

void StringSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[])

{

for (int i = 0; i < count; i++)

{

for (int j = i + 1; j < count; j++)

{

string stringi, stringj;

stringi = arr[i];

stringj = arr[j];

for (int k = 0; stringj[k] != '\0' || stringi[k] != '\0'; k++)

{

int ASCII\_I;

int ASCII\_J;

ASCII\_I = stringi[k];

ASCII\_J = stringj[k];

if (ASCII\_I > 64 && ASCII\_I < 91)

{

ASCII\_I = ASCII\_I + 32;

}

if (ASCII\_J > 64 && ASCII\_J < 91)

{

ASCII\_J = ASCII\_J + 32;

}

if (ASCII\_I == ASCII\_J)

{

continue;

}

else if (ASCII\_I > ASCII\_J)

{

SwapData(i, j, arr, floatarr, intarr, quantities);

break;

}

else

{

break;

}

}

}

}

}

// FLOAT STRING

void floatSorting(int count, string arr[], float floatarr[], int intarr[], int quantities[])

{

for (int i = 0; i < count; i++)

{

for (int j = i + 1; j < count; j++)

{

int floati, floatj;

floati = floatarr[i];

floatj = floatarr[j];

if (floati > floatj)

{

SwapData(i, j, arr, floatarr, intarr, quantities);

}

}

}

}

// SWAP DATA

void SwapData(int indexi, int indexj, string arr1[], float arr2[], int arr3[], int quantities[])

{

string tempstring;

float tempfloat;

float tempint;

tempstring = arr1[indexj];

arr1[indexj] = arr1[indexi];

arr1[indexi] = tempstring;

tempfloat = arr2[indexj];

arr2[indexj] = arr2[indexi];

arr2[indexi] = tempfloat;

tempint = arr2[indexj];

arr3[indexj] = arr3[indexi];

arr3[indexi] = tempint;

tempint = quantities[indexj];

quantities[indexj] = quantities[indexi];

quantities[indexi] = tempint;

}

// CUSTOMER MENU

int customerMenu()

{

int menu;

cout << " 1- View all Items Alphabatically...." << endl;

cout << " 2- View Item with Rescpect to Price..." << endl;

cout << " 3- View Item with Dicount..." << endl;

cout << " 4- Search an Item....." << endl;

cout << " 5- Buy an Items......" << endl;

cout << " 6- Generate Bill....." << endl;

cout << " 7- Sign Out...\n " << endl;

cout << "Select your choice...\n";

cin >> menu;

return menu;

}

// PRICE DISCOUNT

float priceAfterDiscount(float price, int discount)

{

float newPrice = price - (price \* (discount / 100.0));

return newPrice;

}

// VIEW ALL USER

void view\_all\_user(string name[], string password[], string role[], int count)

{

cout << "Name "

<< "\t\t"

<< "Password "

<< "\t\t"

<< "Role " << endl;

for (int i = 0; i < count; i++)

{

cout << name[i] << "\t\t " << password[i] << "\t\t " << role[i] << endl;

}

}

// STOCK MANAGER MENU

int stockManagerMenu()

{

int menu;

cout << " 1- Items with DeadStock...." << endl;

cout << " 2- Order Stock..." << endl;

cout << " 3- Add Stock....." << endl;

cout << " 4- Sign Out...\n " << endl;

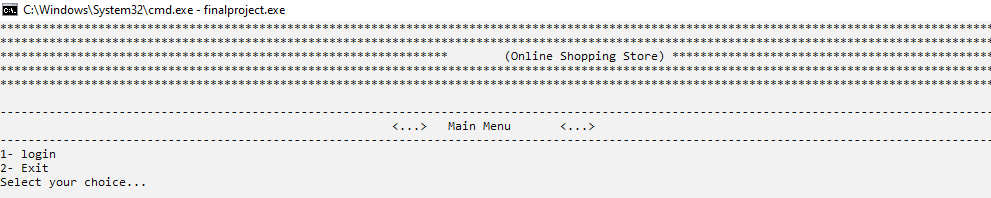
cout << "Select your choice...\n";

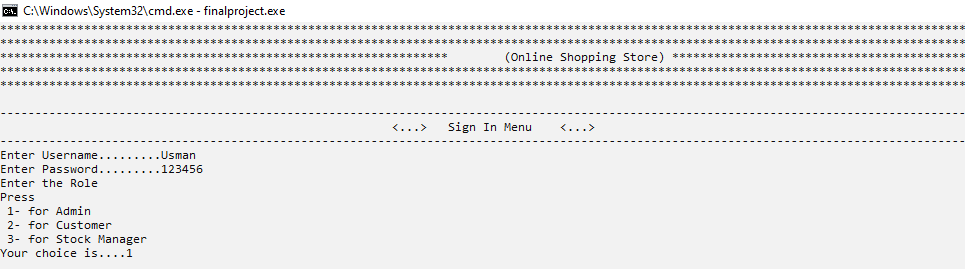
cin >> menu;

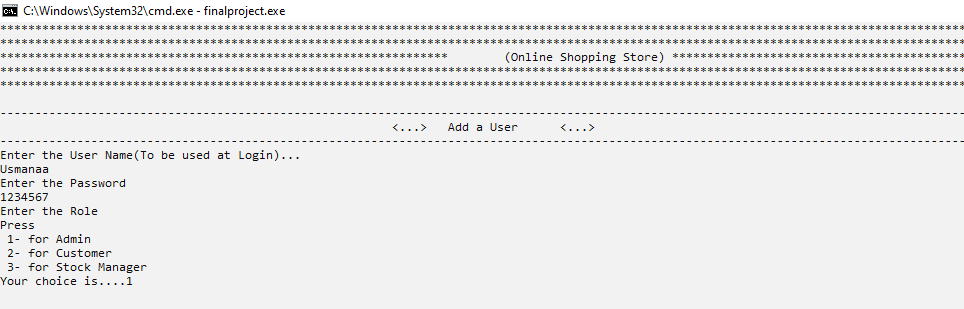
return menu;

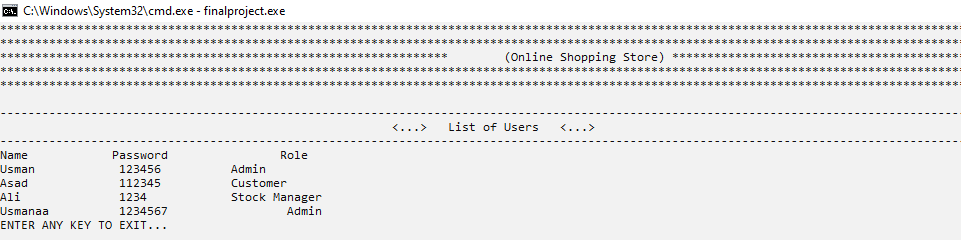
}

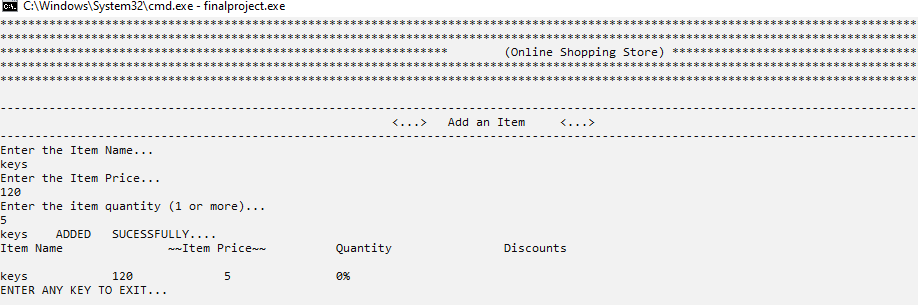
**Wireframes and Test Cases:**

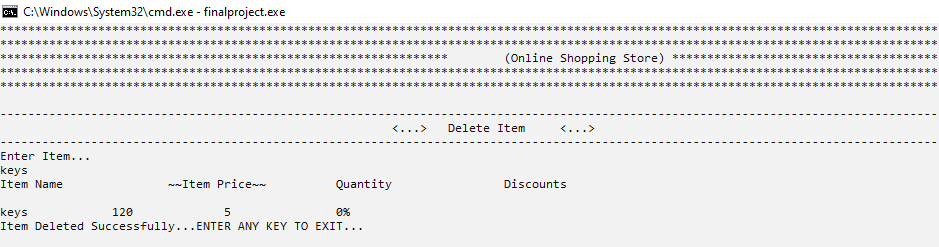


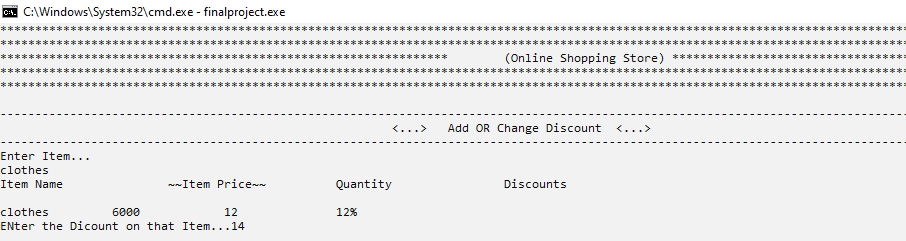
****

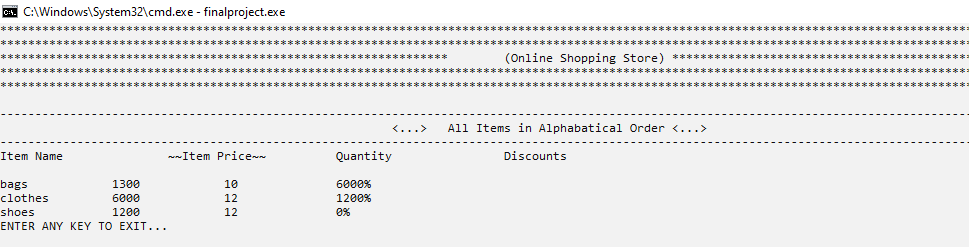
****

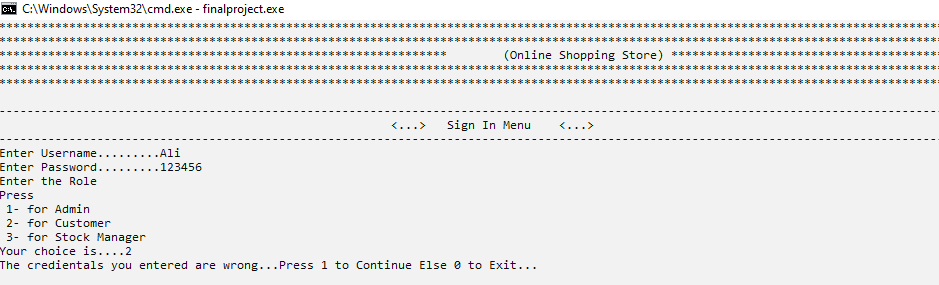
****

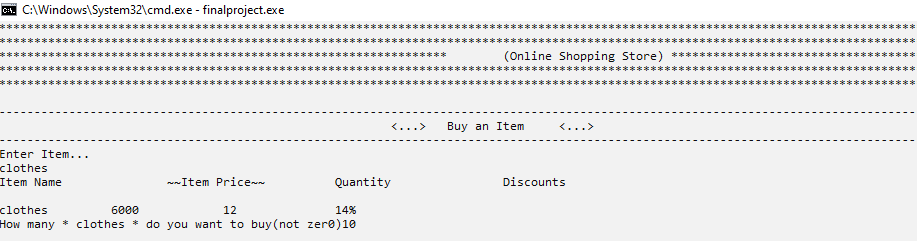
****

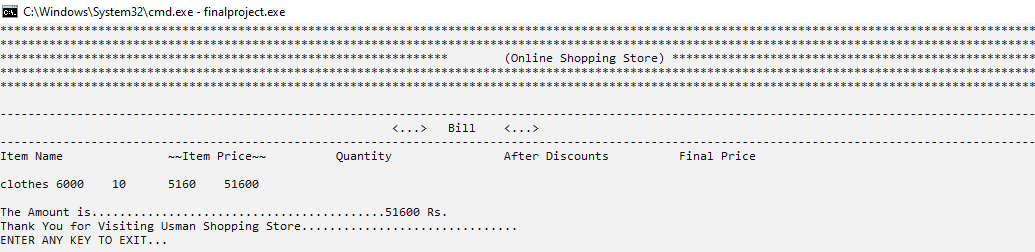
****

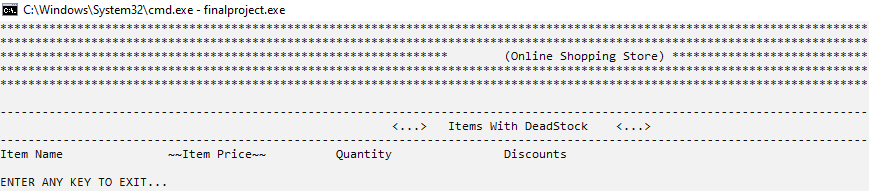


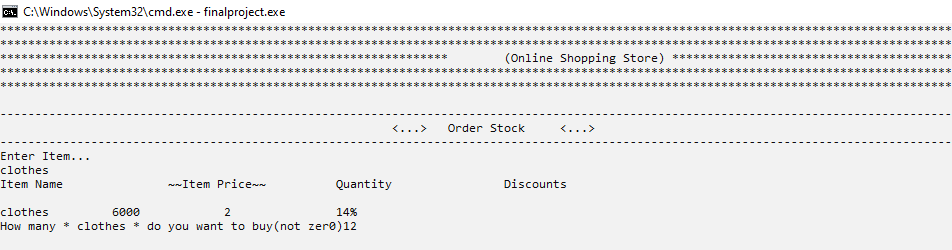


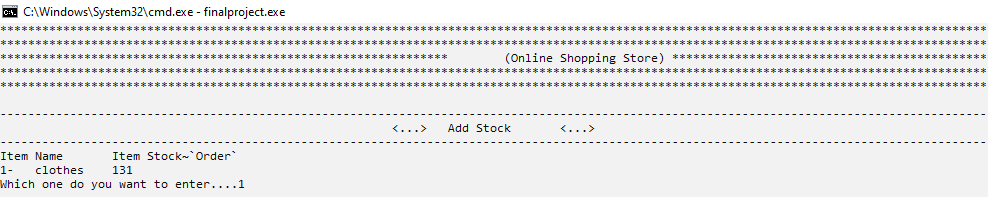


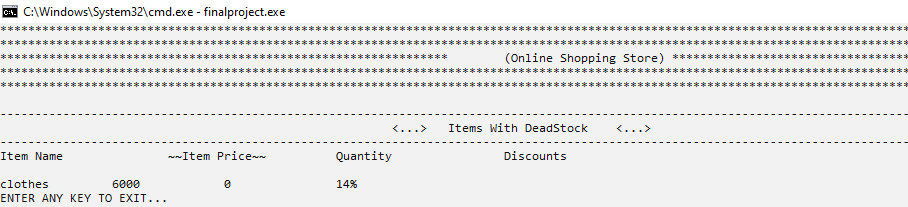












**Student Reg. No. :**   **Student Name.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chec** | **A-Extensive Evidence**  **ked by:** Click | or tap here to enter text.**B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation  Formatting **Grade:** | All the documentation meets all the criteria. |  |  |  |
| Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow**  Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project  Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50%  criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style **Grade:** | All Code style criteria is followed | All code style criteria followed but some  improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added. | | | | |
| Code  Documentation Mapping **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure  (Arrays) **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria  but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendatio n Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and  Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not  know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |