**Mobile Shop Management System**



Session: 2022 – 2026

**Submitted by:**

Gul-e-Zahra 2022-CS-75

**Supervised by:**

Ma’am Maida Sahid

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

* **Short Description of project**

My project deals with the management of a mobile shop, an application which provide an elaborative interface for all the people associated with this application in any term like clients, suppliers, owner etc. In this application, the owner can see all details of his\her business, clients can go through all the desired models of mobile phone and furthermore.

* **Users of Application** 
  + Admin: The primary users of the system are admins. They will use the system to manage their sales, customer information, and other shop-related tasks.
  + Customers: These are the core users of the application. It provides them with an easy interface to purchase devices as per their desire.
  + Employees: These users are updated by admins and admins maintain their record. They can see their profile using the application.
  + Admins:
* **Functional Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| ***User Story ID*** | ***As a*** | ***I want to perform*** | ***So that I can*** |
| 1 | Admin | View Employees | View the employees that are currently working. |
| Add Employees | Add new employees if it is required. |
| Delete Employees | Delete the employee |
| View Stock | View current stock of the shop. |
| Update stock | Update stock if it gets finish. |
| Add a new item | Add a new item to stock. |
| Delete an item | Delete an item if it is outdated. |
| View Customers | View the customers who uses the application. |
| View Salaries | View the salaries of all the employees. |
| Change ID | Admin can change the ID of any user. |
| View Items | View all the available items. |
| Change name | Admin can change the name of any user. |
| View menu | Can view all the available devices. |
| 2 | Customer | Availability | He can check the availability of device. |
| Purchase | Purchase any device |
| Change pin code | Customer can change his pin code. |
| 2 | Customer | Change name | Customer can also change his name. |
| Wishlist | He can add anything to his wish list if he is not able to purchase it now. |
| See Wishlist | Customer can see his Wishlist. |
| Give feedback | Can also give feedback to the service. |
| 3 | Employee | Can view his profile | Can see his name, password, and salary. |
| Change ID | Change my ID if needed. |

* **Wireframes**

**Table

Description automatically generated**

Figure 1 Login page

**Table

Description automatically generated with medium confidence**

Figure 2 Sign in

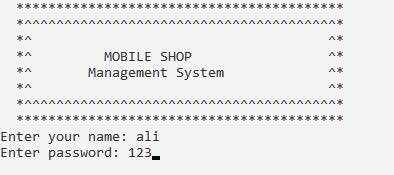
****

Figure 3 Sign up

*Text, letter

Description automatically generated*

Figure 4 Admin menu

**Text, letter

Description automatically generated**

Figure 5 Customer Menu

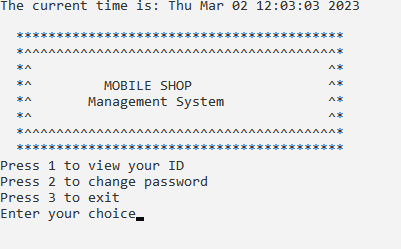
****

Figure 6 Employee menu

* **Data Structures (Parallel Arrays)**
  1. string user\_name[100];
  2. string passcode[100];
  3. string roles[100];
  4. int user\_salary[100];
  5. string item\_collections[100];
  6. int item\_price[100];
  7. int item\_count = 0;
  8. int item\_quantity[100];
  9. string wishList[100];
  10. string feed\_back[100];
* **Function Prototypes**

**// interface functions**

* 1. void interface\_of\_application();
  2. void header();
  3. void press();
  4. void print\_time();

**// file handling functions**

* 1. void store\_to\_file();
  2. void loadDataFromFile();
  3. void delete\_from\_file(string name, string password, string role);
  4. void store\_data\_of\_items\_to\_file();
  5. void load\_data\_of\_items();
  6. string getField(string record, int field);
  7. void readData();
  8. string get\_field\_for\_items(string line, int field);

**// some other functions and variables that support concern functionalities**

* 1. void view\_user();
  2. int login\_page();

**// validation functions**

* 1. int validation(int number);
  2. bool name\_check(string name);
  3. bool role\_check(string role);
  4. bool password\_validation(string password);functions for sign in
  5. void login\_in();
  6. bool sign\_in(string userName, string password, string role);
  7. string return\_role(string name, string password);

**//functions for sign up**

* 1. bool is\_valid\_user\_name(string userName, string password, string role);
  2. void add\_user(string userName, string password, string role, int salary);
  3. void sign\_up();

**// functions for admin**

* 1. int admin\_menu();
  2. void admin\_functionality();
  3. void returned\_function\_of\_admin();

**// admin choices**

* 1. void admin\_choice1();
  2. void admin\_choice\_2();
  3. void admin\_choice\_3();
  4. void admin\_choice\_4();
  5. void admin\_choice\_5();
  6. void admin\_choice\_6();
  7. void admin\_choice\_7();
  8. void admin\_choice\_8();
  9. void admin\_choice\_9();
  10. void admin\_choice\_10();
  11. void admin\_choice\_11();

**// supporting fuctions**

* 1. void delete\_user\_from\_array(string name, string password, string role);
  2. void delete\_device\_from\_array(string device\_name);
  3. void add\_salary(string name, string password, string role, int salary);
  4. void update\_stock(string item\_name, int updated\_stock);
  5. void update\_customer\_id(string name, string previousPassword, string newPassword);

// functions related to items

* 1. void store\_data\_of\_items\_to\_array(string device\_name, int device\_price, int quantity);
  2. bool check\_item\_already\_exist(string device\_name);

**// customer functionalities**

* 1. void customer\_functionality();
  2. int customer\_menu();
  3. void go\_back\_to\_customer\_menu();

**// customer choice**

* 1. void customer\_choice\_1();
  2. void customer\_choice\_2();
  3. void customer\_choice\_3();
  4. void customer\_choice\_4();
  5. void customer\_choice\_5();
  6. void add\_to\_wishlist();
  7. void see\_wish\_list();
  8. void give\_feed\_back();

**// supporting function for customer**

* 1. bool search\_device(string device\_name);
  2. void purchase\_a\_device(string name);
  3. int view\_price(string name);
  4. bool check\_stock(string name);
  5. void clear\_item\_from\_stock(string name);
  6. void update\_password(string userName, string role, string newPassword);
  7. void update\_name(string userName, string role, string newName);
  8. void initialization\_of\_wishList();
  9. void initialization\_of\_feed\_back();

**// employee function**

* 1. void employee\_functionality();
  2. int employee\_menu();

**// employee choices**

* 1. void employee\_choice\_1();
  2. void employee\_choice\_2();

**// supporting function for employees**

* 1. int search\_salary(string name, string password, string role);
  2. bool is\_password\_already\_taken(string password);
  3. void go\_back\_to\_employee\_menu();
* **Functions Working Flow**
* **Complete Code of the Business Application**

#include <iostream>

#include <fstream>

#include <conio.h>

#include <windows.h>

#include <ctime>

using namespace std;

// global data for user

string user\_name[100];

string passcode[100];

string roles[100];

int user\_salary[100];

int user\_count = 0;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ function definations\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// interface functions

void interface\_of\_application();

void header();

void press();

void print\_time();

// file handling functions

void store\_to\_file();

void loadDataFromFile();

void delete\_from\_file(string name, string password, string role);

void store\_data\_of\_items\_to\_file();

void load\_data\_of\_items();

string getField(string record, int field);

void readData();

string get\_field\_for\_items(string line, int field);

// some other functions and variables that support concern functionalities

void view\_user();

int login\_page();

// count that stores the index of the user who signs up

int store = 0;

// validation functions

int validation(int number);

bool name\_check(string name);

bool role\_check(string role);

bool password\_validation(string password);

// functions for sign in

void login\_in();

bool sign\_in(string userName, string password, string role);

string return\_role(string name, string password);

// functions for sign up

bool is\_valid\_user\_name(string userName, string password, string role);

void add\_user(string userName, string password, string role, int salary);

void sign\_up();

// functions for admin

int admin\_menu();

void admin\_functionality();

void returned\_function\_of\_admin();

// admin choices

void admin\_choice1();

void admin\_choice\_2();

void admin\_choice\_3();

void admin\_choice\_4();

void admin\_choice\_5();

void admin\_choice\_6();

void admin\_choice\_7();

void admin\_choice\_8();

void admin\_choice\_9();

void admin\_choice\_10();

void admin\_choice\_11();

// supporting fuctions

void delete\_user\_from\_array(string name, string password, string role);

void delete\_device\_from\_array(string device\_name);

void add\_salary(string name, string password, string role, int salary);

void update\_stock(string item\_name, int updated\_stock);

void update\_customer\_id(string name, string previousPassword, string newPassword);

// global data related to items

string item\_collections[100];

int item\_price[100];

int item\_count = 0;

int item\_quantity[100];

// functions related to items

void store\_data\_of\_items\_to\_array(string device\_name, int device\_price, int quantity);

bool check\_item\_already\_exist(string device\_name);

// global data for customer

string wishList[100];

string feed\_back[100];

// customer functionalities

void customer\_functionality();

int customer\_menu();

void go\_back\_to\_customer\_menu();

// customer choice

void customer\_choice\_1();

void customer\_choice\_2();

void customer\_choice\_3();

void customer\_choice\_4();

void customer\_choice\_5();

void add\_to\_wishlist();

void see\_wish\_list();

void give\_feed\_back();

// supporting function for customer

bool search\_device(string device\_name);

void purchase\_a\_device(string name);

int view\_price(string name);

bool check\_stock(string name);

void clear\_item\_from\_stock(string name);

void update\_password(string userName, string role, string newPassword);

void update\_name(string userName, string role, string newName);

void initialization\_of\_wishList();

void initialization\_of\_feed\_back();

// employee function

void employee\_functionality();

int employee\_menu();

// employee choices

void employee\_choice\_1();

void employee\_choice\_2();

// supporting function for employees

int search\_salary(string name, string password, string role);

bool is\_password\_already\_taken(string password);

void go\_back\_to\_employee\_menu();

main()

{

loadDataFromFile();

load\_data\_of\_items();

initialization\_of\_wishList();

initialization\_of\_feed\_back();

interface\_of\_application();

int choice = 3;

while (choice != 0)

{

choice = login\_page();

if (choice == 1)

{

sign\_up();

}

if (choice == 2)

{

login\_in();

}

if (choice == 3)

{

view\_user();

}

}

}

void interface\_of\_application()

{

system("cls");

print\_time();

header();

}

// function that prints time

void print\_time()

{

time\_t currentTime = time(0);

// Convert the current time to a string format

char \*timeString = ctime(&currentTime);

// Print the current time to the console

cout << "The current time is: " << timeString << endl;

}

// header function

void header()

{

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

cout << " \*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\* " << endl;

cout << " \*^ ^\* " << endl;

cout << " \*^ MOBILE SHOP ^\* " << endl;

cout << " \*^ Management System ^\* " << endl;

cout << " \*^ ^\* " << endl;

cout << " \*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\* " << endl;

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

}

// getch function

void press()

{

cout << "Press any key to continue ";

getch();

}

// function that prints first menu(sign in / sign up)

int login\_page()

{

int login\_choice;

cout << "Press 1 for sign up" << endl;

cout << "Press 2 for sign in" << endl;

cout << "Press 3 to view all users " << endl;

cout << "Press 0 to exit" << endl;

cout << "Enter your choice: ";

cin >> login\_choice;

login\_choice = validation(login\_choice);

return login\_choice;

}

// validation function for integer

int validation(int number)

{

while (cin.fail())

{

cin.clear();

cin.ignore(100, '\n');

cout

<< "Invalid input !!" << endl

<< "Enter an integer: ";

cin >> number;

}

return number;

}

// password validation

bool password\_validation(string password)

{

bool flag = false;

cout << password.length();

if (password.length() == 8)

{

flag = true;

}

return flag;

}

// sign up function

void sign\_up()

{

string name;

string password;

string role;

int salary = 0;

bool check;

bool check1 = false;

bool check2 = false;

bool check3 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

while (check3 == false)

{

cout << "Enter your password(must contains 8 letterrs): ";

cin >> password;

check3 = password\_validation(password);

}

while (check2 == false)

{

cout << "Enter your role(customer/admin/employee): ";

cin >> role;

check2 = role\_check(role);

}

check = is\_valid\_user\_name(name, password, role);

if (check == false)

{

cout << "User alredy exist" << endl;

}

else

{

add\_user(name, password, role, salary);

store\_to\_file();

cout << "Successfully sign up" << endl;

}

}

// validation function for name

bool name\_check(string name)

{

bool flag = false;

int i = 0;

while (i < name.length())

{

if (name.length() <= 2)

{

cout << "Your Name Should Be Atleast 3 Characters Long" << endl;

getch();

break;

}

if ((name[i] > 63 && name[i] < 91) || (name[i] > 96 && name[i] < 123))

{

i++;

flag = true;

}

else

{

flag = false;

break;

}

}

return flag;

}

// validation function for role

bool role\_check(string role)

{

bool value = false;

if (role == "customer" || role == "admin" || role == "employee")

{

value = true;

}

return value;

}

// function that check that either name or password is already taken or not

bool is\_valid\_user\_name(string userName, string password, string role)

{

bool flag = true;

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == userName && (passcode[index] == password && roles[index] == role))

{

flag = false;

break;

}

}

return flag;

}

// function that store user details in arrays

void add\_user(string userName, string password, string role, int salary)

{

user\_name[user\_count] = userName;

passcode[user\_count] = password;

roles[user\_count] = role;

user\_salary[user\_count] = salary;

user\_count++;

}

// function that store user details in file

void store\_to\_file()

{

fstream file;

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

for (int x = 0; x < user\_count; x++)

{

file << user\_name[x] << ",";

file << passcode[x] << ",";

file << roles[x] << ",";

file << user\_salary[x] << ",";

file << wishList[x] << ",";

file << feed\_back[x] << endl;

}

file.close();

}

// function that load data from file to array

void loadDataFromFile()

{

string record;

fstream file;

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

while (getline(file, record))

{

user\_name[user\_count] = getField(record, 1);

passcode[user\_count] = getField(record, 2);

roles[user\_count] = getField(record, 3);

// user\_salary[user\_count] = stoi(getField(record, 4));

wishList[user\_count] = (getField(record, 5));

feed\_back[user\_count] = (getField(record, 6));

user\_count++;

}

file.close();

}

// function to separate data

string getField(string record, int field)

{

int commmaCount = 1;

string item;

for (int x = 0; x < record.length(); x++)

{

if (record[x] == ',')

{

commmaCount++;

}

else if (commmaCount == field)

{

item = item + record[x];

}

}

return item;

}

// login function

void login\_in()

{

string name;

string password;

string role;

int salary = 0;

bool check;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

while (check2 == false)

{

cout << "Enter your password: ";

cin >> password;

check2 = password\_validation(password);

}

role = return\_role(name, password);

check = sign\_in(name, password, role);

if (check == true)

{

cout << "Welcome" << endl

<< endl;

if (role == "admin")

{

admin\_functionality();

}

else if (role == "customer")

{

customer\_functionality();

}

else if (role == "employee")

{

employee\_functionality();

}

}

else

{

cout << "Go and first sign up" << endl

<< endl;

}

}

// function that check either user has signed up or not

bool sign\_in(string userName, string password, string role)

{

bool flag = false;

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == userName && (passcode[index] == password && roles[index] == role))

{

store = index;

flag = true;

break;

}

}

return flag;

}

// funtion that search role of the user who is signing in

string return\_role(string name, string password)

{

string value\_to\_be\_returned;

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == name && passcode[index] == password)

{

value\_to\_be\_returned = roles[index];

break;

}

}

return value\_to\_be\_returned;

}

// function that load data of items from file to array

void load\_data\_of\_items()

{

string line;

fstream file;

string record;

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

while (getline(file, record))

{

item\_collections[item\_count] = get\_field\_for\_items(record, 1);

item\_price[item\_count] = stoi(get\_field\_for\_items(record, 2));

item\_quantity[item\_count] = stoi(get\_field\_for\_items(record, 3));

item\_count++;

}

file.close();

}

// funtion that store data of items to file

void store\_data\_of\_items\_to\_file()

{

fstream file;

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

for (int x = 0; x < item\_count; x++)

{

file << item\_collections[x] << ",";

file << item\_price[x] << ",";

file << item\_quantity[x] << endl;

}

file.close();

}

// function that separate data of items according to comma

string get\_field\_for\_items(string line, int field)

{

int commmaCount = 1;

string item;

for (int i = 0; i < line.length(); i++)

{

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

{

commmaCount++;

}

else if (commmaCount == field)

{

item = item + line[i];

}

}

return item;

}

// funtion that store items to array during execution

void store\_data\_of\_items\_to\_array(string device\_name, int device\_price, int quantity)

{

item\_collections[item\_count] = device\_name;

item\_price[item\_count] = device\_price;

item\_quantity[item\_count] = (quantity);

item\_count++;

}

// function to view all users

void view\_user()

{

cout << "Names "

<< "\t\t"

<< "Password "

<< "\t\t"

<< "Role" << endl;

for (int index = 0; index < user\_count; index++)

{

cout << user\_name[index] << "\t\t" << passcode[index] << "\t\t" << roles[index] << "\t\t" << user\_salary[index] << endl;

}

cout << endl

<< endl;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_admin functions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// functions that display admin menu

int admin\_menu()

{

int admin\_choice;

interface\_of\_application();

cout << "Press 1 to add an employee" << endl;

cout << "Press 2 to view yours employee" << endl;

cout << "Press 3 to delete an employee" << endl;

cout << "Press 4 to add a new product " << endl;

cout << "Press 5 to view your products" << endl;

cout << "Press 6 to delete product " << endl;

cout << "Press 7 to update salary of any employee " << endl;

cout << "Press 8 to view customers" << endl;

cout << "Press 9 to update stock" << endl;

cout << "Press 10 to update credentials of any employee " << endl;

cout << "Press 11 to view feed back of customers " << endl;

cout << "Press 0 to exit " << endl;

cout << "Enter your choice: ";

cin >> admin\_choice;

admin\_choice = validation(admin\_choice);

return admin\_choice;

}

// funtion that bring user to admin menu

void returned\_function\_of\_admin()

{

press();

admin\_functionality();

}

// function that go to the concern function according to admin choice

void admin\_functionality()

{

int returned\_choice = -1;

returned\_choice = admin\_menu();

while (returned\_choice != 0)

{

system("cls");

header();

if (returned\_choice == 1)

{

admin\_choice1();

}

else if (returned\_choice == 2)

{

admin\_choice\_2();

}

else if (returned\_choice == 3)

{

admin\_choice\_3();

}

else if (returned\_choice == 4)

{

admin\_choice\_4();

}

else if (returned\_choice == 5)

{

admin\_choice\_5();

}

else if (returned\_choice == 6)

{

admin\_choice\_6();

}

else if (returned\_choice == 7)

{

admin\_choice\_7();

}

else if (returned\_choice == 8)

{

admin\_choice\_8();

}

else if (returned\_choice == 9)

{

admin\_choice\_9();

}

else if (returned\_choice == 10)

{

admin\_choice\_10();

}

else if (returned\_choice == 11)

{

admin\_choice\_11();

}

}

}

// admin choice number 1

void admin\_choice1()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

bool check;

string name;

string password;

int salary;

string role;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter name of employee: ";

cin >> name;

check1 = name\_check(name);

}

while (check2 == false)

{

cout << "Enter password of employee (must contains 8 letterrs): ";

cin >> password;

check2 = password\_validation(password);

}

cout << "Enter the salary of employee: ";

cin >> salary;

salary = validation(salary);

role = "employee";

check = is\_valid\_user\_name(name, password, role);

if (check == true)

{

add\_user(name, password, role, salary);

store\_to\_file();

cout << "Your desired entry has successfully entered." << endl;

}

else

{

cout << "Can't enter your record because it is repititive!" << endl;

}

returned\_function\_of\_admin();

}

// admin choice number 2

void admin\_choice\_2()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Name"

<< "\t\t"

<< "Password"

<< "\t\t"

<< "Salary " << endl;

for (int index = 0; index < user\_count; index++)

{

if (roles[index] == "employee")

{

cout << user\_name[index] << "\t\t" << passcode[index] << "\t\t" << user\_salary[index] << endl;

}

}

returned\_function\_of\_admin();

}

// admin choice number 3

void admin\_choice\_3()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

string name;

string del\_password;

string del\_role;

bool check;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

cout << "enter password: ";

cin >> del\_password;

while (check2 == false)

{

cout << "Enter your role: ";

cin >> del\_role;

check2 = role\_check(del\_role);

}

check = is\_valid\_user\_name(name, del\_password, del\_role);

if (check == false)

{

delete\_user\_from\_array(name, del\_password, del\_role);

store\_to\_file();

cout << "Succesfully deleted" << endl;

}

else

{

cout << "User does no exist" << endl;

}

returned\_function\_of\_admin();

}

// funtion that delete user from array

void delete\_user\_from\_array(string name, string password, string role)

{

for (int index = 0; index < user\_count; index++)

{

if ((user\_name[index] == name && passcode[index] == password) && roles[index] == role)

{

passcode[index] = passcode[index + 1];

passcode[index + 1] = " ";

user\_name[index] = user\_name[index + 1];

user\_name[index + 1] = " ";

roles[index] = roles[index + 1];

roles[index + 1] = " ";

user\_salary[index] = user\_salary[index + 1];

user\_salary[index] = 0;

wishList[index] = wishList[index + 1];

wishList[index] = " ";

user\_count--;

}

}

}

// admin choice number 4

void admin\_choice\_4()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

bool check;

string device\_name;

int price;

int quantity;

cout << "Enter name of device: ";

cin.ignore();

getline(cin, device\_name);

cout << "Enter price of your device: ";

cin >> price;

price = validation(price);

cout << "Enter how many pieces you want to enter: ";

cin >> quantity;

quantity = validation(quantity);

check = check\_item\_already\_exist(device\_name);

if (check == true)

{

cout << "Your entered item is already present. " << endl;

}

else

{

store\_data\_of\_items\_to\_array(device\_name, price, quantity);

store\_data\_of\_items\_to\_file();

cout << "Successfully added " << endl;

}

returned\_function\_of\_admin();

}

// function that check either item already exist or not to avoid repitition of items

bool check\_item\_already\_exist(string device\_name)

{

bool find = false;

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == device\_name)

{

find = true;

}

}

return find;

}

// admin choice number 5

void admin\_choice\_5()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Product Name "

<< "\t"

<< "Product price"

<< "\t"

<< "No. of pieces " << endl;

for (int i = 0; i < item\_count; i++)

{

cout << item\_collections[i] << "\t\t"

<< item\_price[i] << "\t\t"

<< item\_quantity[i] << endl;

}

returned\_function\_of\_admin();

}

// admin choice number 6

void admin\_choice\_6()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Menu 6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

string name;

bool check;

cout << "Enter the name of device you want to delete: ";

cin >> name;

check = check\_item\_already\_exist(name);

if (check == true)

{

delete\_device\_from\_array(name);

store\_data\_of\_items\_to\_file();

cout << "Successfully deleted" << endl;

}

else

{

cout << "Device does not exist" << endl;

}

returned\_function\_of\_admin();

}

// funtion that delete device from array

void delete\_device\_from\_array(string device\_name)

{

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == device\_name)

{

item\_collections[index] = item\_collections[index + 1];

item\_collections[index + 1] = "0";

item\_price[index] = item\_price[index + 1];

item\_price[index + 1] = 0;

item\_quantity[index] = 0;

item\_count--;

}

}

}

// admin choice number 7

void admin\_choice\_7()

{

int salary;

string name;

string password;

string role;

bool check;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

cout << "Enter password of employee: ";

cin >> password;

while (check2 == false)

{

cout << "Enter your role: ";

cin >> role;

check2 = role\_check(role);

}

cout << "Enter the updated salary: ";

cin >> salary;

salary = validation(salary);

check = is\_valid\_user\_name(name, password, role);

if (check == false)

{

add\_salary(name, password, role, salary);

cout << "Successfully updated " << endl;

}

else

{

cout << "Can't find your entered employee!" << endl;

}

returned\_function\_of\_admin();

}

// function that update salary

void add\_salary(string name, string password, string role, int salary)

{

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == name && passcode[index] == password && roles[index] == role)

{

user\_salary[index] = salary;

store\_to\_file();

}

}

}

// admin choice number 8

void admin\_choice\_8()

{

cout << "Customers are: " << endl;

for (int index = 0; index < user\_count; index++)

{

if (roles[index] == "customer")

{

cout << user\_name[index] << endl;

}

}

returned\_function\_of\_admin();

}

// admin choice number 9

void admin\_choice\_9()

{

string item\_name;

int updated\_stock;

cout << "Enter the name of the device of which you wanted to add stock: ";

cin >> item\_name;

cout << "Enter the updated stock: ";

cin >> updated\_stock;

update\_stock(item\_name, updated\_stock);

store\_data\_of\_items\_to\_file();

returned\_function\_of\_admin();

}

// function that update the stock

void update\_stock(string item\_name, int updated\_stock)

{

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == item\_name)

{

item\_quantity[index] = item\_quantity[index] + updated\_stock;

}

}

}

// admin choice number 10

void admin\_choice\_10()

{

string name;

string previousPassword;

string newPassword;

bool check1 = false;

while (check1 == false)

{

cout << "Enter name of the employee of which you wanted to change passowrd: ";

cin >> name;

check1 = name\_check(name);

}

cout << "Enter previous password of that employee: ";

cin >> previousPassword;

cout << "Enter new password of that employee: ";

cin >> newPassword;

update\_customer\_id(name, previousPassword, newPassword);

store\_to\_file();

returned\_function\_of\_admin();

}

// function that update customer credentials

void update\_customer\_id(string name, string previousPassword, string newPassword)

{

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == name && passcode[index] == previousPassword)

{

passcode[index] = newPassword;

}

}

}

// function to view\_feed\_back

void admin\_choice\_11()

{

for (int index = 0; index < user\_count; index++)

{

if (feed\_back[index] != " " || feed\_back[index] != "e")

{

cout << user\_name[index] << "\t" << feed\_back[index] << endl;

}

}

returned\_function\_of\_admin();

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_employee functions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// function that display employee menu

int employee\_menu()

{

int choice = 0;

cout

<< "Press 1 to view your ID" << endl;

cout << "Press 2 to change password" << endl;

cout << "Press 3 to exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

choice = validation(choice);

return choice;

}

// function that call function according to choice of employee

void employee\_functionality()

{

interface\_of\_application();

int choice = -1;

while (choice != 3)

{

choice = employee\_menu();

if (choice == 1)

{

employee\_choice\_1();

}

else if (choice == 2)

{

employee\_choice\_2();

}

}

}

// function that bring user back to menu of employee

void go\_back\_to\_employee\_menu()

{

interface\_of\_application();

employee\_functionality();

press();

}

// function of employee choice number 1

void employee\_choice\_1()

{

string name;

string password;

int salary;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

while (check2 == false)

{

cout << "Enter your password: ";

cin >> password;

check2 = password\_validation(password);

}

string role = "employee";

salary = search\_salary(name, password, role);

cout << "Your salary is: " << salary << endl;

go\_back\_to\_employee\_menu();

}

// function that search for salary of employee

int search\_salary(string name, string password, string role)

{

int salary;

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == name && passcode[index] == password && roles[index] == role)

{

user\_salary[index] = salary;

}

}

return salary;

}

// function of employee choice number 2

void employee\_choice\_2()

{

string name;

string previousPassword;

string newPassword;

string role = "employee";

bool is\_present\_check;

bool check;

bool check1 = false;

bool check2 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

cout << "Enter your previous password: ";

cin >> previousPassword;

is\_present\_check = is\_valid\_user\_name(name, previousPassword, role);

if (is\_present\_check == false)

{

while (check2 == false)

{

cout << "Enter your new password: ";

cin >> newPassword;

check2 = password\_validation(newPassword);

}

check = is\_password\_already\_taken(newPassword);

if (check == false)

{

update\_customer\_id(name, previousPassword, newPassword);

store\_to\_file();

cout << "Successfully change your password!" << endl;

}

else

{

cout << "This password has already taken! " << endl;

}

}

else

{

cout << "Invalid credentials! " << endl;

}

go\_back\_to\_employee\_menu();

}

// function that check either password is already taken or not

bool is\_password\_already\_taken(string password)

{

bool is\_taken = false;

for (int index = 0; index < user\_count; index++)

{

if (passcode[index] == password)

{

is\_taken = true;

}

}

return is\_taken;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_customer functions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// function that initialize entire string to empty to avoid error

void initialization\_of\_wishList()

{

for (int x = 0; x < 100; x++)

{

wishList[x] = " ";

}

}

// function that brings user back to customer menu

void go\_back\_to\_customer\_menu()

{

press();

customer\_functionality();

}

// function that prints customer menu

int customer\_menu()

{

int choice;

cout << "Welcome!" << endl;

cout << "Press 1 to view mneu " << endl

<< "Press 2 to check the availability of your desired device" << endl

<< "Press 3 to purchase phone" << endl

<< "Press 4 to change your pin code" << endl

<< "Press 5 to change your name" << endl

<< "Press 6 to add anything to your wish list" << endl

<< "Press 7 to see your wish list" << endl

<< "Press 8 to give feedback " << endl

<< "Press 0 to exit" << endl

<< "Enter your choice: ";

cin >> choice;

choice = validation(choice);

return choice;

}

// function that calls the concern functions according to choice

void customer\_functionality()

{

int customer\_choice = -1;

while (customer\_choice != 0)

{

system("cls");

header();

customer\_choice = customer\_menu();

if (customer\_choice == 1)

{

customer\_choice\_1();

}

else if (customer\_choice == 2)

{

customer\_choice\_2();

}

else if (customer\_choice == 3)

{

customer\_choice\_3();

}

else if (customer\_choice == 4)

{

customer\_choice\_4();

}

else if (customer\_choice == 5)

{

customer\_choice\_5();

}

else if (customer\_choice == 6)

{

add\_to\_wishlist();

}

else if (customer\_choice == 7)

{

see\_wish\_list();

}

else if (customer\_choice == 8)

{

give\_feed\_back();

}

}

}

// function for customer choice number 1

void customer\_choice\_1()

{

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] != "0")

{

cout << item\_collections[index] << endl;

}

}

go\_back\_to\_customer\_menu();

}

// function for customer choice number 2

void customer\_choice\_2()

{

string name;

bool is\_found;

cout << "Enter the name of the device you wanted to purchase: ";

cin >> name;

is\_found = search\_device(name);

if (is\_found == true)

{

cout << "Your desired device is available! " << endl;

}

else

{

cout << "Sorry for inconvinience your desired device is not available! " << endl;

}

go\_back\_to\_customer\_menu();

}

// function that check either device is present or not

bool search\_device(string device\_name)

{

bool flag = false;

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == device\_name)

{

flag = true;

}

}

return flag;

}

// function for customer choice number 3

void customer\_choice\_3()

{

char conform;

string name;

int price;

bool is\_found;

bool is\_available;

cout << "Enter the name of the device you wanted to purchase: ";

cin >> name;

is\_found = search\_device(name);

is\_available = check\_stock(name);

if (is\_found == true)

{

if (is\_available == true)

{

// cout << "Your desired device is available! " << endl;

price = view\_price(name);

cout << "Price of your desired device is: " << price << endl;

cout << "Press y to confirm your choice and 'n' to cancel it: ";

cin >> conform;

if (conform == 'y')

{

purchase\_a\_device(name);

store\_data\_of\_items\_to\_file();

}

}

else

{

clear\_item\_from\_stock(name);

store\_data\_of\_items\_to\_file();

cout << "Out of stock" << endl;

}

go\_back\_to\_customer\_menu();

}

}

// funtion that display price of device

int view\_price(string name)

{

int price;

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == name)

{

price = item\_price[index];

}

}

return price;

}

// function that check either device is in the stock or not

bool check\_stock(string name)

{

bool is\_avialable = true;

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == name && item\_quantity[index] == 0)

is\_avialable = false;

}

return is\_avialable;

}

// function that decrease stock if device is purchased by user

void purchase\_a\_device(string name)

{

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == name && item\_quantity[index] != 0)

{

item\_quantity[index] = item\_quantity[index] - 1;

}

}

}

// function that clear item when it quantity becomes zero

void clear\_item\_from\_stock(string name)

{

for (int index = 0; index < item\_count; index++)

{

if (item\_collections[index] == name && item\_quantity[index] == 0)

{

item\_collections[index] = item\_collections[index + 1];

item\_collections[index + 1] = "0";

item\_price[index] = item\_price[index + 1];

item\_price[index + 1] = 0;

item\_quantity[index] = item\_quantity[index + 1];

item\_quantity[index + 1] = 0;

item\_count--;

store\_data\_of\_items\_to\_file();

}

}

}

// function for customer choice number 4

void customer\_choice\_4()

{

string name;

string password;

string newPassword;

bool found;

string role = "customer";

bool check1 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

cout << "Enter your previous password: ";

cin >> password;

found = is\_valid\_user\_name(name, password, role);

if (found == true)

{

cout << "You have entered wrong credentials: ";

}

else

{

cout << "Enter your new password: ";

cin >> newPassword;

cout << newPassword << endl;

update\_password(name, role, newPassword);

store\_to\_file();

}

go\_back\_to\_customer\_menu();

}

// function that update password

void update\_password(string userName, string role, string newPassword)

{

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == userName && roles[index] == role)

{

passcode[index] = newPassword;

}

}

}

// function for customer choice number 5

void customer\_choice\_5()

{

string name;

string password;

string newName;

bool found;

string role = "customer";

bool check1 = false;

while (check1 == false)

{

cout << "Enter your name: ";

cin >> name;

check1 = name\_check(name);

}

cout << "Enter your password: ";

cin >> password;

found = is\_valid\_user\_name(name, password, role);

if (found == true)

{

cout << "You have entered wrong credentials: ";

}

else

{

cout << "Enter your new name: ";

cin >> newName;

update\_name(name, role, newName);

store\_to\_file();

}

go\_back\_to\_customer\_menu();

}

// function that update name

void update\_name(string userName, string role, string newName)

{

for (int index = 0; index < user\_count; index++)

{

if (user\_name[index] == userName && roles[index] == role)

{

user\_name[index] = newName;

}

}

}

// function for customer choice number 6

void add\_to\_wishlist()

{

string device\_name;

cout << "Enter a device which you wanted to add in your wish list: ";

cin >> device\_name;

wishList[store] = device\_name;

store\_to\_file();

go\_back\_to\_customer\_menu();

}

// function for customer choice number 7

void see\_wish\_list()

{

cout << user\_name[store] << endl;

cout << wishList[store] << endl;

store\_to\_file();

go\_back\_to\_customer\_menu();

}

// initialization of wish list

void initialization\_of\_feed\_back()

{

for (int x = 0; x < 100; x++)

{

feed\_back[x] = "e";

}

}

// function for customer choice number 8

void give\_feed\_back()

{

string sentence;

cout << "Enter a device which you wanted to add in your wish list: ";

cin >> sentence;

feed\_back[store] = sentence;

store\_to\_file();

go\_back\_to\_customer\_menu();

}

* **Weakness in the Business Application**

I will improve the salary system and stock system and also want to add attendance system as well.

* **Future Directions**
* Stock management
* Salary management.
* Attendance system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **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 -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. |
| 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). | | | | |
| 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 |
| File Handling  **Grade:** | Separate files for separate data. Data in csv format | File handing require some improvements | File handing require a lot of improvements | Not implemented |
| Aesthetics of the User Interface  **Grade:** | UI is presentable. Proper coloring, Headers and clear screen is done | UI require some improvements | UI require a lot of improvements | 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. |

|  |  |
| --- | --- |
| **Checked by:** |  |
| **Comments:** |  |