**AUTOMOTIVE MANAGEMNET SYSTEM**



Session: 2022 – 2025

**Submitted by:**

Saba Shahdin 2022-CS-112

**Supervised by:**

Sir Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

Table of Contents

[1.DESCRIPTION](#_Toc22187)

* OBJECTIVE
* USE IN COMPUTER SCIENCE FIELD
* OUTPUT

[2.USERS OF APPLICATION](#_Toc22188)

[3.FUNCTIONAL REQUIREMENTS](#_Toc22189)

[4.WIREFRAMES:](#_Toc22190)

[5.DATA STRUCTURE:](#_Toc22191)

[6.FUNCTION PROTOTYPES](#_Toc22192)

[7.COMPLETE CODE OF THE APPICATION](#_Toc22193)

[8.WEAKNESS IN THE APPLICATION](#_Toc22194)

[9.FUTURE DIRECTION](#_Toc22195)

[10.FUNCTIONS WORKING FLOW](#_Toc22196)

[RUBRICS](#_Toc22199)

**Description of Project:**

* **Objective:**

1. To manage the garage efficiently.
2. To make the system more manageable and organize it programmatically.
3. To help employees handle their own task properly.
4. As an owner handle all the affairs digitally in just one go.

* **Contribution in computer science field:**

1. Helps in digital management of the small business
2. Keep the previous record of firm.

* **Output:**

1. Number of products in the firm.
2. Number of products can be repaired.
3. Products prices
4. On manager and owner end, total money earned in one day.

**Users of Application:**

There are three users of my **“AUTOMOTIVE MANAGEMNT SYSTEM”**

* Owner of Garage
* Manager
* Employee
* **Functional Requirements:**

The functional requirement of the automotive management system is to keep the check on working environment by controlling different modules.

1. **OWNER:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***User Story ID*** | ***As a*** | ***I want to perform*** | ***So that I can*** |
| ***1*** | Owner | View sell products and their prices | **No of products sell in one day**  **No of products repair in one day**  **Money earned in one day by selling**  **Money earned in one day by repairing** |
| ***2*** | Owner | View repair products and their prices |
| ***3*** | Owner | Add selling products |
| ***4*** | Owner | Update selling products |
| ***5*** | Owner | Delete selling products |
| ***6*** | Owner | Add repairing products |
| ***7*** | Owner | Update repairing products |
| ***8*** | Owner | Delete repairing products |
| ***9*** | Owner | View employees record |
| ***10*** | Owner | View manager |
| ***11*** | Owner | Add manager |
| ***12*** | Owner | Remove manager |
| ***13*** | Owner | Update manager |
| ***14*** | Owner | Sort products with price |

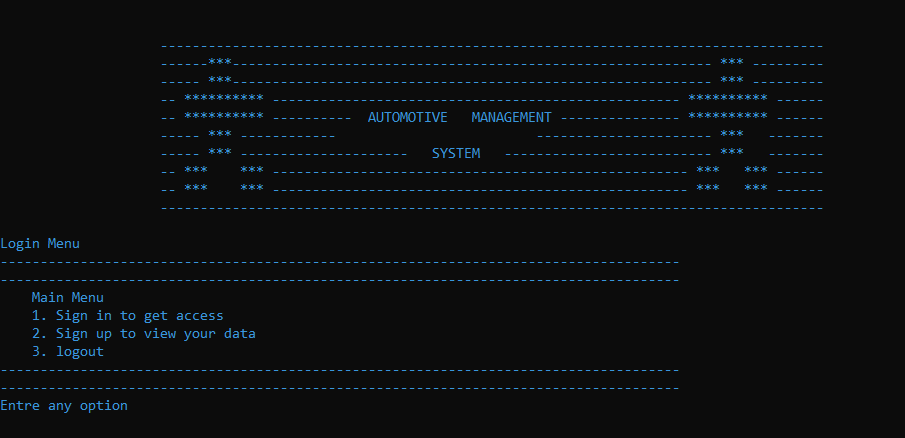
1. **EMPLOYEE:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***User Story ID*** | ***As a*** | ***I want to perform*** | ***So that I can*** |
| ***1*** | Employee | View products and their prices | **Add money earned in one day by selling**  **Add money earned in one day by repairing** |
| ***2*** | Employee | View repair products and their prices |
| ***3*** | Employee | Add number of products sell in one day |
| ***4*** | Employee | Add number of repair products in one day |

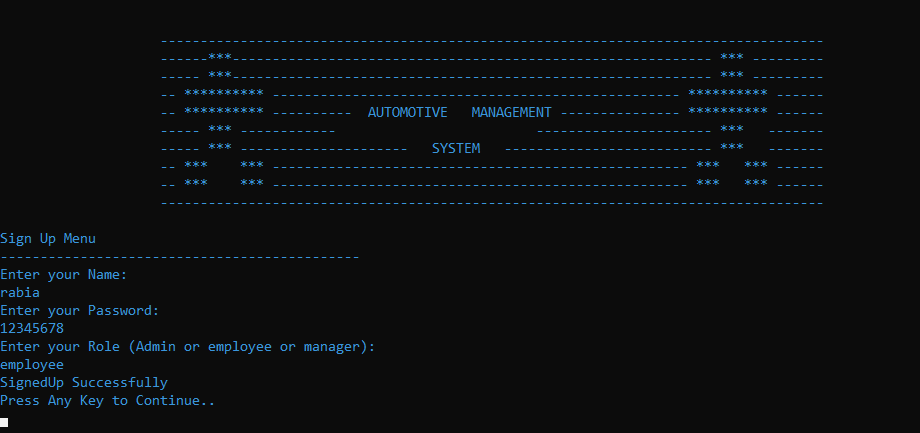
1. **MANAGER:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***User Story ID*** | ***As a*** | ***I want to perform*** | ***So that I can*** |
| ***1*** | Manager | View record of employee | **View money**  **earned in one day**  **by selling**  **view money earned in one day by repairing**  **No of products repair in one day**  **No of products sell in one day** |
| ***2*** | Manager | View selling products and their prices |
| ***3*** | Manager | View repair products and their prices |
| ***4*** | Manager | Add employees |
| ***5*** | Manager | Remove employees |
| ***6*** | Manager | Update employees date |

* **Wireframes**

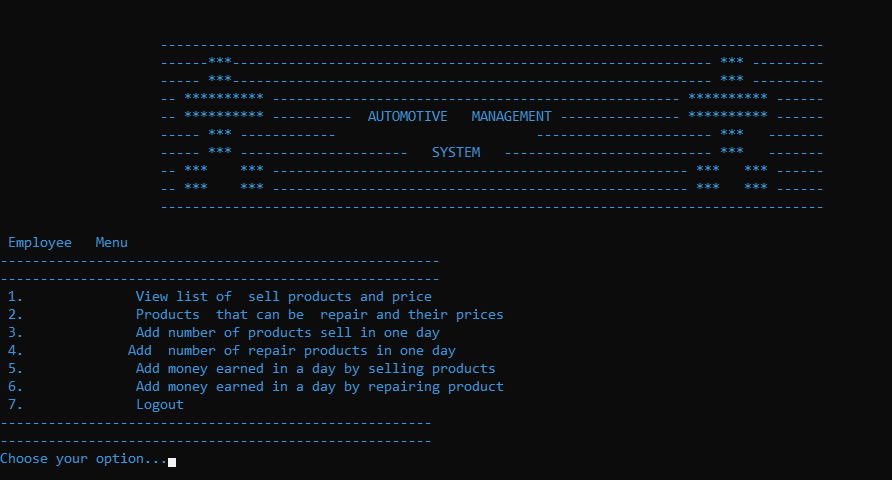


**Figure 1: LogIn Screen**

****

**Figure 2: Signup Screen**

**Figure 3: Admin Main Menu Screen**

****

**Figure 4: Employee Screen**

****

**Figure 4: Manager Screen**

* **Data Structures (Parallel Arrays)**

**Arrays for Users**

* const int arraySize = 20;
* string users[arraySize];
* string passwords[arraySize];
* string roles[arraySize];
* string userName;
* string password;
* string role;

**Arrays for Products**

* const int array = 100;
* string productName[array];
* int productPrice[array];
* string productName1[array];
* int productPrice1[array];

**Varriable for Products Selling**

* string productsSell;
* string priceSell;
* string productsSell1;
* string priceSell1;

**Arrays for Employees**

* const int size = 10;
* string employeeName[size];
* int employeeAge[size];
* string employeeCity[size];
* string employeeNames;
* int employeeAges;
* string employeeCitys;

**Arrays for Manager**

* const int size1 = 3;
* string managerName[size1];
* int managerAge[size1];
* string managerCity[size1];
* string managerNames;
* int managerAges;
* string managerCitys;

**Arrays for Selling Number**

* const int size2 = 100;
* string day[size2];
* int numberOfProducts[size2];
* string prod;
* string names\_prod[size2];
* int prices\_prod[size2];

**Arrays for Repairing Number**

* const int size3 = 100;
* string day1[size3];
* string productNames1[size3];
* int numberOfProducts1[size3];
* string prod1;
* string names\_prod1[size3];
* int prices\_prod1[size3];

**Arrays for Storing Days Data**

* const int productArray = 100;
* int oneDayRecord[productArray];
* string productNames[array];
* int productPrices[array];
* const int productArray1 = 100;
* int oneDayRecord1[productArray1];

**Counter Variables**

* int count = 0;
* int sellingCount = 0;
* int repairingCount = 0;
* int employeeCount = 0;
* int managerCount = 0;
* int productsCount = 0;
* int productsRepairCount = 0;
* **Function Prototypes:**

**Main Screen**

* void printMainScreen();

**Menus**

* void menuAdmin();
* void menuEmployee();
* void menuManager();

**Login screen**

* bool signUp(string userName, string password, string role);
* string signIn(string userName, string password);
* string firstScreen();

**Supportive functions**

* void clearScreen();
* void openRootMenu(string submenu);
* void subMenu(string submenu);
* void gotoxy(int x, int y);

**Main Panels of Users**

* void adminPanel();
* void ManagerPanel();
* void EmployeePanel();

**Product for selling**

* void viewProductForSelling();
* void addProductForSelling();
* void DeleteSellingProduct();
* void UpdateSellingValues();
* void product();

**Product for Repairing**

* void viewProductForRepairing();
* void addProductForRepairing();
* void DeleteRepairingProduct();
* void UpdateRepairingValues();
* void product1();

**Employees Function**

* void employees();
* void addEmployees();
* void viewEmployees();
* void removeEmployees();
* void updateEmployee();

**Manager Function**

* void addManager();
* void managers();
* void viewManager();
* void removeManager();
* void updateManager();

**Product for Selling**

* void AddSelledProducts ();
* void viewSelledProducts ();
* void moneyEarnedBySelling ();

**Product for Repairing**

* void viewRepairedProducts();
* void AddRepairedProducts();
* void moneyEarnedByRepairing();

**Sorted Data**

* void sortedDataSelled();
* void sortedDataRepaired();
* void viewUser();

**Validity**

* bool isValid(string name);
* bool validPassword(string password);
* bool isValidName1(string productsSell1);
* bool isValidName(string productsSell);
* void checkProduct(string productsSell);
* bool isValidPrice1(string priceSell1);
* bool isValidPrice(string priceSell);
* void checkProduct1(string productsSell1);
* void isValidEmployeeName(string employeeNames);
* void isValidManagerName(string managerNames);
* bool searchPrice();
* int search();
* bool searchPrice1();
* int search1();

**Function to Store File Data**

**Users Data Storing**

* void loadData();
* void storeUser(string userName, string password, string role);
* string parseRecord(string record, int field);

**Selling Product Storing**

* void loadProductData();
* void storeSellingProduct ();
* string parseProductRecord(string record, int field);

**Repairing Product Storing**

* void loadProductsData();
* string parseProductRepairingRecord(string record, int field);
* void storeRepairingProduct();

**Employee Data Storing**

* string parseEmployee(string record, int field);
* void loadEmployeeData();
* void storeEmployees();

**Manager Data Storing**

* void loadmanagerData();
* string parsemanager(string record, int field);
* void storeManager();

**Selling Number Storing**

* string parsemoney(string record, int field);
* void loadmoneyData();
* void storeMoney();

**Repairing Number Storing**

* void storeMoney1();
* void loadmoneyData1();
* string parsemoney1(string record, int field);
* **Functions Working Flow:**

main()

admin

employee

manager

View Selling

View Repairing

View Money earned by Selling Products in one day

View Money earned by Repairing Products in one day

View Money earned by Selling Products in one day

View Money earned by Repairing Products in one day

View Number of Repairing Products

Sort products

View Employees

View Number of Selling Products

Add Manager

View Manager

Remove Manager

Update Manager

View Number of Selling Products

Add Repairing

View Repairing

Remove Repairing

Update Repairing

View Number of Repairing Products

add Selling

View Selling

Delete Selling

Update Selling

View Selling

View Repairing

View Money earned by Selling Products in one day

View Money earned by Repairing Products in one day

View Number of Selling Products

View Number of Repairing Products

add Employee

View Employee

Delete Employee

Update Employee

**Complete Code of the Business Application**

// header files

#include <iostream>

#include <conio.h>

#include <windows.h>

#include <sstream>

#include <fstream>

using namespace std;

// Function Prototypes

void printMainScreen();

// menus

void menuAdmin();

void menuEmployee();

void menuManager();

// login screen

bool signUp(string userName, string password, string role);

string signIn(string userName, string password);

string firstScreen();

// supportive functions

void clearScreen();

void openRootMenu(string submenu);

void subMenu(string submenu);

void gotoxy(int x, int y);

// main panels of users

void adminPanel();

void ManagerPanel();

void EmployeePanel();

// product for selling

void viewProductForSelling();

void addProductForSelling();

void DeleteSellingProduct();

void UpdateSellingValues();

void product();

// product for repairing

void viewProductForRepairing();

void addProductForRepairing();

void DeleteRepairingProduct();

void UpdateRepairingValues();

void product1();

// employeesfunction

void employees();

void addEmployees();

void viewEmployees();

void removeEmployees();

void updateEmployee();

// manager function

void addManager();

void managers();

void viewManager();

void removeManager();

void updateManager();

// product for selling

void AddSelledProducts();

void viewSelledProducts();

void moneyEarnedBySelling();

// product for repairing

void viewRepairedProducts();

void AddRepairedProducts();

void moneyEarnedByRepairing();

// sorted data

void sortedDataSelled();

void sortedDataRepaired();

void viewUser();

// validity

bool isValid(string name);

bool validPassword(string password);

bool isValidName1(string productsSell1);

bool isValidName(string productsSell);

void checkProduct(string productsSell);

bool isValidPrice1(string priceSell1);

bool isValidPrice(string priceSell);

void checkProduct1(string productsSell1);

void isValidEmployeeName(string employeeNames);

void isValidManagerName(string managerNames);

bool searchPrice();

int search();

bool searchPrice1();

int search1();

// function to store file data

// users data storing

void loadData();

void storeUser(string userName, string password, string role);

string parseRecord(string record, int field);

// selling product storing

void loadProductData();

void storeSellingProduct();

string parseProductRecord(string record, int field);

// repairing product storing

void loadProductsData();

string parseProductRepairingRecord(string record, int field);

void storeRepairingProduct();

// employee data storing

string parseEmployee(string record, int field);

void loadEmployeeData();

void storeEmployees();

// manager data storing

void loadmanagerData();

string parsemanager(string record, int field);

void storeManager();

// selling number storing

string parsemoney(string record, int field);

void loadmoneyData();

void storeMoney();

// repairing number storing

void storeMoney1();

void loadmoneyData1();

string parsemoney1(string record, int field);

// arrays for users

const int arraySize = 20;

string users[arraySize];

string passwords[arraySize];

string roles[arraySize];

string userName;

string password;

string role;

// arrays for products

const int array = 100;

string productName[array];

int productPrice[array];

string productName1[array];

int productPrice1[array];

// varriable for proucts selling

string productsSell;

string priceSell;

string productsSell1;

string priceSell1;

// arrays for employees

const int size = 10;

string employeeName[size];

int employeeAge[size];

string employeeCity[size];

string employeeNames;

int employeeAges;

string employeeCitys;

// arrays for manager

const int size1 = 3;

string managerName[size1];

int managerAge[size1];

string managerCity[size1];

string managerNames;

int managerAges;

string managerCitys;

// arrays for selling number

const int size2 = 100;

string day[size2];

int numberOfProducts[size2];

string prod;

string names\_prod[size2];

int prices\_prod[size2];

// arrays for repairing number

const int size3 = 100;

string day1[size3];

string productNames1[size3];

int numberOfProducts1[size3];

string prod1;

string names\_prod1[size3];

int prices\_prod1[size3];

// arrays for storing days data

const int productArray = 100;

int oneDayRecord[productArray];

string productNames[array];

int productPrices[array];

const int productArray1 = 100;

int oneDayRecord1[productArray1];

// counter varriables

int count = 0;

int sellingCount = 0;

int repairingCount = 0;

int employeeCount = 0;

int managerCount = 0;

int productsCount = 0;

int productsRepairCount = 0;

main()

{

    // load data

    loadData();

    loadProductData();

    loadProductsData();

    loadEmployeeData();

    loadmanagerData();

    loadmoneyData();

    loadmoneyData1();

  int k = 3;

    HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    SetConsoleTextAttribute(hConsole, k);

    system(" cls ");

    string option;

    string role;

    while (option != "3")

    {

        printMainScreen();

        openRootMenu("Login");

        option = firstScreen();

        if (option == "1")

        {

            system("cls");

            string Username;

            string password;

            string role;

            char login;

            printMainScreen();

            openRootMenu("Sign In");

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

            cout << "Enter your Name: " << endl;

            cin >> Username;

            cout << "Enter your Password: " << endl;

            cin >> password;

            role = signIn(Username, password);

            if (role == "admin")

            {

                clearScreen();

                printMainScreen();

                openRootMenu("Admin");

                adminPanel();

            }

            else if (role == "employee")

            {

                clearScreen();

                printMainScreen();

                openRootMenu(" Employee  ");

                EmployeePanel();

            }

            else if (role == "manager")

            {

                clearScreen();

                printMainScreen();

                openRootMenu(" Manager ");

                ManagerPanel();

            }

            else if (role == "undefined")

            {

                cout << "You Entered wrong Credentials" << endl;

            }

        }

        else if (option == "2")

        {

            system("cls");

            string name;

            string password;

            string role;

            printMainScreen();

            openRootMenu("Sign Up");

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

            cout << "Enter your Name: " << endl;

            cin >> name;

            cout << "Enter your Password: " << endl;

            cin >> password;

            cout << "Enter your Role (Admin or employee or manager): " << endl;

            cin >> role;

            bool flag2 = isValid(name);

            bool flag3 = validPassword(password);

            if (flag2 == true && flag3 == true)

            {

                bool flag = signUp(name, password, role);

                storeUser(name, password, role);

                if (flag == true)

                {

                    cout << "SignedUp Successfully" << endl;

                }

                else if (flag == false)

                {

                    cout << "Users in the System have exceeded the Capacity" << endl;

                }

            }

            else if (flag2 == false)

            {

                cout << "userName  already exits " << endl;

            }

            else if (flag3 == false)

            {

                cout << "Password must be 8 character long" << endl;

            }

        }

        clearScreen();

    }

}

void printMainScreen()

{

    int x;

    int y;

    gotoxy(20, 2);

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

    gotoxy(20, 3);

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

    gotoxy(20, 4);

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

    gotoxy(20, 5);

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

    gotoxy(20, 6);

    cout << "-- \*\*\*\*\*\*\*\*\*\* ----------  AUTOMOTIVE   MANAGEMENT ---------------\*\*\*\*\*\*\*\*\*\* ------      ";

    gotoxy(20, 7);

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

    gotoxy(20, 8);

    cout << "----- \*\*\* ---------------------   SYSTEM   -------------------------- \*\*\*   -------      ";

    gotoxy(20, 9);

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

    gotoxy(20, 10);

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

    gotoxy(20, 11);

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

    cout << " " << endl;

}

void menuAdmin()

{

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

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

    cout << " 1.\t\t View list of products and price" << endl;

    cout << " 2.\t\t Products  that can be  repair and their prices" << endl;

    cout << " 3.\t\t Add product  and their prices" << endl;

    cout << " 4.\t\t Add repair products and their prices" << endl;

    cout << " 5.\t\t Delete  selling product and their prices" << endl;

    cout << " 6.\t\t Delete  repairing products and their prices" << endl;

    cout << " 7.\t\t Update selling products and their prcices" << endl;

    cout << " 8.\t\t Update repairing products and their prcices" << endl;

    cout << " 9.\t\t Money earned in one day by selling" << endl;

    cout << "10.\t\t Money earned in one day  by repairing" << endl;

    cout << "11.\t\t View number of products sell in one day" << endl;

    cout << "12.\t\t View number of products repair in one day" << endl;

    cout << "13.\t\t View Employees record" << endl;

    cout << "14.\t\t Add manager" << endl;

    cout << "15.\t\t View manager " << endl;

    cout << "16.\t\t Remove manager " << endl;

    cout << "17.\t\t Update manager " << endl;

    cout << "18.\t\t Sort products with price" << endl;

    cout << "19.\t\t Logout" << endl;

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

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

}

void menuEmployee()

{

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

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

    cout << " 1.\t\t View list of  sell products and price" << endl;

    cout << " 2.\t\t Products  that can be  repair and their prices" << endl;

    cout << " 3.\t\t Add number of products sell in one day" << endl;

    cout << " 4.\t\tAdd  number of repair products in one day" << endl;

    cout << " 5.\t\t Add money earned in a day by selling products" << endl;

    cout << " 6.\t\t Add money earned in a day by repairing product" << endl;

    cout << " 7.\t\t Logout" << endl;

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

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

}

void menuManager()

{

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

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

    cout << " 1.\t\t View employee record" << endl;

    cout << " 2.\t\t View list of  sell products and price" << endl;

    cout << " 3.\t\t Products  that can be  repair and their prices" << endl;

    cout << " 4.\t\t Add employees " << endl;

    cout << " 5.\t\t Remove employees " << endl;

    cout << " 6.\t\t Update employees data" << endl;

    cout << " 7.\t\t View number of products sell in one day" << endl;

    cout << " 8.\t\t View number of products repair in one day" << endl;

    cout << " 9.\t\t Money earned in one day by selling" << endl;

    cout << "10.\t\t Money earned in one day  by repairing" << endl;

    cout << "11.\t\t Logout" << endl;

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

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

}

string signIn(string userName, string password)

{

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

    {

        if (users[idx] == userName && passwords[idx] == password)

        {

            return roles[idx];

        }

    }

    return "undefined";

}

bool signUp(string userName, string password, string role)

{

    bool flag = true;

    if (count < arraySize)

    {

        users[count] = userName;

        passwords[count] = password;

        roles[count] = role;

        count++;

        return true;

    }

    else

    {

        return false;

    }

}

string firstScreen()

{

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

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

    cout <MainMenu                                                                        " << endl;

    cout << "    1. Sign in to get access                                                         " << endl;

    cout << "    2. Sign up to view yourdata                                                     " << endl;

    cout << "    3. logout                                                                        " << endl;

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

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

    string option;

    cout << "Entre any option ";

    cin >> option;

    return option;

}

void clearScreen()

{

    cout << "Press Any Key to Continue.." << endl;

    getch();

    system("cls");

}

void openRootMenu(string submenu)

{

    string message = submenu + " Menu";

    cout << message << endl;

}

void subMenu(string submenu)

{

    string message = "Main Menu > " + submenu;

    cout << message << endl;

    cout << endl;

}

void gotoxy(int x, int y)

{

    COORD coordinates;

    coordinates.X = x;

    coordinates.Y = y;

    SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coordinates);

}

void storeUser(string userName, string password, string role)

{

    fstream file;

    file.open("signIn.txt", ios ::app);

    file << userName << ",";

    file << password << ",";

    file << role << endl;

    file.close();

}

void loadData()

{

    fstream file;

    string word;

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

    while (getline(file, word))

    {

        users[count] = parseRecord(word, 1);

        passwords[count] = parseRecord(word, 2);

        roles[count] = parseRecord(word, 3);

        count++;

    }

    file.close();

}

string parseRecord(string record, int field)

{

    int commaCount = 1;

    string item;

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

    {

        if (record[x] == ',')

        {

            commaCount = commaCount + 1;

        }

        else if (commaCount == field)

        {

            item = item + record[x];

        }

    }

    return item;

}

bool isValid(string name)

{

    bool flag = true;

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

    {

        if (users[idx] == name)

        {

            return false;

            break;

        }

    }

    return true;

}

bool validPassword(string password)

{

    bool flag3 = true;

    int idx = 0;

    while (password[idx] != '\0')

    {

        if (password.length() != 8)

        {

            return false;

            break;

        }

        else

        {

            return true;

        }

    }

}

bool isValidName(string productsSell)

{

    bool flag7 = true;

    int idx = 0;

    while (productsSell[idx] != '\0')

    {

        if (productsSell[idx] >= '0' && productsSell[idx] <= '9')

        {

            flag7 = false;

            break;

        }

        idx = idx + 1;

    }

    return flag7;

}

bool isValidPrice(string priceSell)

{

    bool flag7 = true;

    int idx = 0;

    while (priceSell[idx] != '\0')

    {

        if (!(priceSell[idx] >= '0' && priceSell[idx] <= '9'))

        {

            flag7 = false;

            break;

        }

        idx = idx + 1;

    }

    return flag7;

}

void product()

{

    cin.ignore(389, '\n');

    cout << "Product name:   ";

    getline(cin, productsSell);

    cout << "Product price: ";

    cin >> priceSell;

}

void addProductForSelling()

{

    bool flag7;

    bool flag8;

    bool takeInput = true;

    while (takeInput == true)

    {

 if (sellingCount > array)

        {

            cout << "No more space." << endl;

        }

        else if (sellingCount < array)

        {

            product();

            flag7 = isValidName(productsSell);

            flag8 = isValidPrice(priceSell);

            checkProduct(productsSell);

            if (flag8 == true && flag7 == true)

            {

                productName[sellingCount] = productsSell;

                stringstream temp;

                temp >> priceSell;

                productPrice[sellingCount] = (stoi(priceSell));

            }

            else

            {

                cout << "Wrong format of input" << endl;

                sellingCount--;

            }

            cout << "If you want to enter another record press 1 otherwise 0: ";

            cin >> takeInput;

            sellingCount = sellingCount + 1;

        }

    }

}

void checkProduct(string productsSell)

{

    for (int idx = 0; idx < sellingCount; idx++)

    {

        if (productsSell == productName[idx])

        {

            cout << " Product already present. Use update function to change price" << endl;

            sellingCount = sellingCount - 1;

            break;

        }

    }

}

void storeSellingProduct()

{

    fstream file;

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

    for (int idx = 0; idx < sellingCount; idx++)

    {

        file << productName[idx] << ",";

        file << productPrice[idx] << endl;

    }

    file.close();

}

void loadProductData()

{

    fstream file;

    string word;

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

    while (getline(file, word))

    {

        stringstream temp(parseProductRecord(word, 2));

        temp >> productPrice[sellingCount];

        productName[sellingCount] = parseProductRecord(word, 1);

        sellingCount++;

    }

    file.close();

}

string parseProductRecord(string record, int field)

{

    int commaCounter = 1;

    string products;

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

    {

        if (record[x] == ',')

        {

            commaCounter = commaCounter + 1;

        }

        else if (commaCounter == field)

        {

            products = products + record[x];

        }

    }

    return products;

}

bool isValidName1(string productsSell1)

{

    bool flag7 = true;

    int idx = 0;

    while (productsSell1[idx] != '\0')

    {

        if (productsSell1[idx] >= '0' && productsSell1[idx] <= '9')

        {

            flag7 = false;

            break;

        }

        idx = idx + 1;

    }

    return flag7;

}

bool isValidPrice1(string priceSell1)

{

    bool flag7 = true;

    int idx = 0;

    while (priceSell1[idx] != '\0')

    {

        if (!(priceSell1[idx] >= '0' && priceSell1[idx] <= '9'))

        {

            flag7 = false;

            break;

        }

        idx = idx + 1;

    }

    return flag7;

}

void checkProduct1(string productsSell1)

{

    for (int idx = 0; idx < repairingCount; idx++)

    {

        if (productsSell1 == productName1[idx])

        {

            cout << " Product already present. Use update function to change price" << endl;

            repairingCount = repairingCount - 1;

            break;

        }

    }

}

void product1()

{

    cin.ignore(389, '\n');

    cout << "Product name:   ";

    getline(cin, productsSell1);

    cout << "Product price: ";

    cin >> priceSell1;

}

void addProductForRepairing()

{

    bool flag8;

    bool flag7;

    bool takeInput = true;

    while (takeInput == true)

    {

        if (repairingCount > array)

        {

            cout << "No more space." << endl;

        }

        else if (repairingCount < array)

        {

            product1();

            flag7 = isValidName1(productsSell1);

            flag8 = isValidPrice1(priceSell1);

            checkProduct1(productsSell1);

            if (flag8 == true && flag7 == true)

            {

                productName1[repairingCount] = productsSell1;

                stringstream temp;

                temp >> priceSell1;

                productPrice1[repairingCount] = (stoi(priceSell1));

            }

            else

            {

                cout << "Wrong format of input" << endl;

                repairingCount--;

            }

            cout << "If you want to enter another record press 1 otherwise 0: ";

            cin >> takeInput;

            repairingCount = repairingCount + 1;

        }

    }

}

void storeRepairingProduct()

{

    fstream file;

    file.open("product2.txt", ios ::out);

    for (int idx = 0; idx < repairingCount; idx++)

    {

        file << productName1[idx] << ",";

        file << productPrice1[idx] << endl;

    }

    file.close();

}

void loadProductsData()

{

    fstream file;

    string word;

    file.open("product2.txt", ios ::in);

    while (getline(file, word))

    {

        stringstream temp(parseProductRepairingRecord(word, 2));

        temp >> productPrice1[repairingCount];

        productName1[repairingCount] = parseProductRepairingRecord(word, 1);

        repairingCount++;

    }

    file.close();

}

string parseProductRepairingRecord(string record, int field)

{

    int commaCounter1 = 1;

    string productsRepairing;

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

    {

        if (record[x] == ',')

        {

            commaCounter1 = commaCounter1 + 1;

        }

        else if (commaCounter1 == field)

        {

            productsRepairing = productsRepairing + record[x];

        }

    }

    return productsRepairing;

}

void viewProductForSelling()

{

    cout << "ProductName"<< "\t"<< "ProductPrice" << endl;

    for (int idx = 0; idx < sellingCount; idx = idx + 1)

    {

        cout << productName[idx] << "\t\t" << productPrice[idx] << endl;

    }

}

void viewProductForRepairing()

{

    cout << "ProductName"<< "\t"<< "ProductPrice" << endl;

    for (int idx = 0; idx < repairingCount; idx = idx + 1)

    {

        cout << productName1[idx] << "\t\t" << productPrice1[idx] << endl;

    }

}

void DeleteSellingProduct()

{

    int pos;

    cout << "Entre position you want to delete...";

    cin >> pos;

    if (pos > sellingCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Product at " << pos << " is... " << productName[pos - 1] << "\t" << productPrice[pos - 1] << "    is deleted successfully" << endl;

        for (int idx = pos - 1; idx < sellingCount - 1; idx++)

        {

            productName[pos - 1] = productName[idx + 1];

            productPrice[pos - 1] = productPrice[idx + 1];

        }

        sellingCount = sellingCount - 1;

    }

}

void DeleteRepairingProduct()

{

    int pos1;

    cout << "Entre position you want to delete...";

    cin >> pos1;

    if (pos1 > repairingCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Product at " << pos1 << " is...  " << productName1[pos1 - 1] << "\t" << productPrice1[pos1 - 1] << "    is deleted successfully" << endl;

        for (int idx = pos1 - 1; idx < repairingCount - 1; idx++)

        {

            productName1[pos1 - 1] = productName1[idx + 1];

            productPrice1[pos1 - 1] = productPrice1[idx + 1];

        }

        repairingCount--;

    }

}

void UpdateSellingValues()

{

    bool flag1 = true;

    char choice;

    int pos;

    cout << "Entre position you want to update...";

    cin >> pos;

    if (pos > sellingCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Product at " << pos << " is...... " << productName[pos - 1] << "\t" << productPrice[pos - 1] << endl;

        while (flag1 == true)

        {

            cout << "Entre 0  if you want to update product name and 1 to update product price......";

            cin >> choice;

            if (choice == '0')

            {

                cout << "Entre the value of product name you want to display after updateing....... ";

                cin >> productName[pos - 1];

            }

            else if (choice == '1')

            {

                cout << "Entre the value  of product price you want to display after updateing...... ";

                cin >> productPrice[pos - 1];

            }

            else

            {

                flag1 = false;

            }

        }

    }

}

void UpdateRepairingValues()

{

    bool flag1 = true;

    char choice;

    int pos;

    cout << "Entre position you want to update.... ";

    cin >> pos;

    if (pos > repairingCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Product at " << pos << " is..." << productName1[pos - 1] << "\t" << productPrice1[pos - 1] << endl;

        while (flag1 == true)

        {

            cout << "Entre 0  if you want to update product name and 1 to update product price.......";

            cin >> choice;

            if (choice == '0')

            {

                cout << "Entre the value of product name you want to display after updateing.......";

                cin >> productName1[pos - 1];

            }

            else if (choice == '1')

            {

                cout << "Entre the value  of product price you want to display after updateing...... ";

                cin >> productPrice1[pos - 1];

            }

            else

            {

                flag1 = false;

            }

        }

    }

}

void isValidEmployeeName(string employeeNames)

{

    bool flag7 = true;

    int idx = 0;

    while (employeeNames[idx] != '\0')

    {

        if (employeeNames[idx] >= '0' && employeeNames[idx] <= '9')

        {

            cout << "Wrong format of Name" << endl;

            employeeCount = employeeCount - 1;

        }

        idx = idx + 1;

    }

}

void employees()

{

    cin.ignore(389, '\n');

    cout << "Employee name: ";

    getline(cin, employeeNames);

    cout << "Employee age: ";

    cin >> employeeAges;

    cout << "Employee city:";

    cin >> employeeCitys;

}

bool isValidEmployee()

{

    bool flag = true;

    for (int idx = 0; idx < employeeCount; idx++)

    {

        if (employeeName[idx] == employeeNames && employeeAge[idx] == employeeAges)

        {

            return false;

            break;

        }

    }

    return true;

}

void addEmployees()

{

    bool takeInput = true;

    bool flag3 = true;

    string employee;

    while (takeInput == true)

    {

        if (employeeCount > size)

        {

            cout << "NO more space....";

        }

        if (employeeCount < size)

        {

            employees();

            bool flag = isValidEmployee();

            if (flag == true)

            {

                employeeName[employeeCount] = employeeNames;

                employeeAge[employeeCount] = employeeAges;

                employeeCity[employeeCount] = employeeCitys;

            }

            else

            {

                cout << "Already present" << endl;

                cout << " ";

                employeeCount--;

            }

            isValidEmployeeName(employeeNames);

            cout << "If you want to enter another record press 1 otherwise 0:  ";

            cin >> takeInput;

            employeeCount = employeeCount + 1;

        }

    }

}

void storeEmployees()

{

    fstream file;

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

    for (int idx = 0; idx < employeeCount; idx++)

    {

        file << employeeName[idx] << ",";

        file << employeeAge[idx] << ",";

        file << employeeCity[idx] << endl;

    }

    file.close();

}

void loadEmployeeData()

{

    fstream file;

    string line;

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

    while (getline(file, line))

    {

        stringstream temp(parseEmployee(line, 2));

        temp >> employeeAge[employeeCount];

        employeeName[employeeCount] = parseEmployee(line, 1);

        employeeCity[employeeCount] = parseEmployee(line, 3);

        employeeCount++;

    }

    file.close();

}

string parseEmployee(string record, int field)

{

    int commaCounter2 = 1;

    string employee;

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

    {

        if (record[x] == ',')

        {

            commaCounter2 = commaCounter2 + 1;

        }

        else if (commaCounter2 == field)

        {

            employee = employee + record[x];

        }

    }

    return employee;

}

void viewEmployees()

{

    cout << "Employee Name"<< "\t"<< "Employee Age"<< "\t"<< "Employee City" << endl;

    for (int idx = 0; idx < employeeCount; idx = idx + 1)

    {

        cout << employeeName[idx] << "\t\t" << employeeAge[idx] << " \t\t" << employeeCity[idx] << endl;

    }

}

void removeEmployees()

{

    int pos;

    cout << "Entre position you want to delete...";

    cin >> pos;

    if (pos > employeeCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Employee at " << pos << " is... " << employeeName[pos - 1] << "\t" << employeeAge[pos - 1] << "\t" << employeeCity[pos - 1] << "    is removed successfully" << endl;

        for (int idx = pos - 1; idx < employeeCount - 1; idx++)

        {

            employeeName[pos - 1] = employeeName[idx + 1];

            employeeAge[pos - 1] = employeeAge[idx + 1];

            employeeCity[pos - 1] = employeeCity[idx + 1];

        }

        employeeCount--;

    }

}

void updateEmployee()

{

    bool flag1 = true;

    char choice;

    int pos;

    cout << "Entre position you want to update.... ";

    cin >> pos;

    if (pos > employeeCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Employee at " << pos << " is..." << employeeName[pos - 1] << "\t" << employeeAge[pos - 1] << "/t" << employeeCity[pos - 1] << endl;

        while (flag1 == true)

        {

            cout << "Entre 0  if you want to update employee name and 1 to update employee age and  and 2 to update employee city ..........";

            cin >> choice;

            if (choice == '0')

            {

                cout << "Entre the value of employee  name you want to display after updateing.......";

                cin >> employeeName[pos - 1];

            }

            else if (choice == '1')

            {

                cout << "Entre the value  of  employee age you want to display after updateing...... ";

                cin >> employeeAge[pos - 1];

            }

            else if (choice == '2')

            {

                cout << "Entre the value  of  employee city you want to display after updateing...... ";

                cin >> employeeCity[pos - 1];

            }

            else

            {

                cout << " No  record found" << endl;

                flag1 = false;

            }

        }

    }

}

void isValidManagerName(string managerNames)

{

    bool flag7 = true;

    int idx = 0;

    while (managerNames[idx] != '\0')

    {

        if (managerNames[idx] >= '0' && managerNames[idx] <= '9')

        {

            cout << "Wrong format of Name" << endl;

            managerCount = managerCount - 1;

        }

        idx = idx + 1;

    }

}

void managers()

{

    cin.ignore(389, '\n');

    cout << "Manager name: ";

    getline(cin, managerNames);

    cout << "Manager Age : ";

    cin >> managerAges;

    cout << "Manager city:";

    cin >> managerCitys;

}

bool isValidManager()

{

    bool flag = true;

    for (int idx = 0; idx < managerCount; idx++)

    {

        if (managerName[idx] == managerNames && managerAge[idx] == managerAges)

        {

            return false;

            break;

        }

    }

    return true;

}

void addManager()

{

    bool takeInput = true;

    bool flag1 = true;

    while (takeInput == true)

    {

        if (managerCount > size1)

        {

            cout << "No more space ....";

        }

        else if (managerCount < size1)

        {

            managers();

            bool flag1 = isValidManager();

            if (flag1 == true)

            {

                managerName[managerCount] = managerNames;

                managerAge[managerCount] = managerAges;

                managerCity[managerCount] = managerCitys;

            }

            else

            {

                cout << "Already present" << endl;

                cout << " ";

                managerCount--;

            }

            isValidManagerName(managerNames);

            cout << "If you want to enter another record press 1 otherwise 0: ";

            cin >> takeInput;

            managerCount = managerCount + 1;

        }

    }

}

void storeManager()

{

    fstream file;

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

    for (int idx = 0; idx < managerCount; idx++)

    {

        file << managerName[idx] << ",";

        file << managerAge[idx] << ",";

        file << managerCity[idx] << endl;

    }

    file.close();

}

void loadmanagerData()

{

    fstream file;

    string line;

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

    while (getline(file, line))

    {

        stringstream temp(parsemanager(line, 2));

        temp >> managerAge[managerCount];

        managerName[managerCount] = parsemanager(line, 1);

        managerCity[managerCount] = parsemanager(line, 3);

        managerCount++;

    }

    file.close();

}

string parsemanager(string record, int field)

{

    int commaCounter3 = 1;

    string manager;

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

    {

        if (record[x] == ',')

        {

            commaCounter3 = commaCounter3 + 1;

        }

        else if (commaCounter3 == field)

        {

            manager = manager + record[x];

        }

    }

    return manager;

}

void viewManager()

{

    cout << "Manager Name"<< "\t"<< "Manager Age"<< "\t"<< "Manager City" << endl;

    for (int idx = 0; idx < managerCount; idx = idx + 1)

    {

        cout << managerName[idx] << "\t\t" << managerAge[idx] << " \t\t" << managerCity[idx] << endl;

    }

}

void removeManager()

{

    int pos;

    cout << "Entre position you want to delete...";

    cin >> pos;

    if (pos > managerCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Manager at " << pos << " is... " << managerName[pos - 1] << "\t" << managerAge[pos - 1] << "\t" << managerCity[pos - 1] << "    is removed successfully" << endl;

        for (int idx = pos - 1; idx < managerCount - 1; idx++)

        {

            managerName[pos - 1] = managerName[idx + 1];

            managerAge[pos - 1] = managerAge[idx + 1];

            managerCity[pos - 1] = managerCity[idx + 1];

        }

        managerCount--;

    }

}

void updateManager()

{

    bool flag1 = true;

    char choice;

    int pos;

    cout << "Entre position you want to update.... ";

    cin >> pos;

    if (pos > managerCount)

    {

        cout << "No data found" << endl;

    }

    else

    {

        cout << "Manager at " << pos << " is..." << managerName[pos - 1] << "\t" << managerAge[pos - 1] << "/t" << managerCity[pos - 1] << endl;

        while (flag1 == true)

        {

            cout << "Entre 0  if you want to update manager name and 1 to update manager age and  and 2 to update manager city ..........";

            cin >> choice;

            if (choice == '0')

            {

                cout << "Entre the value of manager  name you want to display after updateing.......";

                cin >> managerName[pos - 1];

            }

            else if (choice == '1')

            {

                cout << "Entre the value  of manager age you want to display after updateing...... ";

                cin >> managerAge[pos - 1];

            }

            else if (choice == '2')

            {

                cout << "Entre the value  of  manager city you want to display after updateing...... ";

                cin >> managerCity[pos - 1];

            }

            else

            {

                cout << " No  record found" << endl;

                flag1 = false;

            }

        }

    }

}

int search()

{

    int counter;

    for (int idx2 = 0; idx2 < sellingCount; idx2++)

    {

        if (productName[idx2] == prod)

        {

            counter = idx2;

        }

    }

    return counter;

}

bool searchPrice()

{

    bool flag = false;

    for (int idx2 = 0; idx2 < sellingCount; idx2++)

    {

        if (productName[idx2] == prod)

        {

            flag = true;

            break;

        }

    }

    return flag;

}

void AddSelledProducts()

{

    bool flag = true;

    bool flag2 = true;

    int price;

    while (flag == true)

    {

        cout << "Entre day...";

        cin >> day[productsCount];

        cin.ignore(389, '\n');

        if (productsCount < array)

        {

            cout << "Product name:   ";

            getline(cin, prod);

            flag2 = searchPrice();

            price = search();

            if (flag2 == true)

            {

                cout << "Price of products....." << productPrice[price] << endl;

                prices\_prod[productsCount] = productPrice[price];

                cout << "Number of Products sell:";

                cin >> numberOfProducts[productsCount];

                names\_prod[productsCount] = prod;

            }

            else if (flag2 == false)

            {

                cout << "Product not found" << endl;

                productsCount--;

            }

            oneDayRecord[productsCount] = numberOfProducts[productsCount] \* prices\_prod[productsCount];

            cout << "Entre 1 to get input for another day ....";

            cin >> flag;

            productsCount = productsCount + 1;

        }

        else if (productsCount > array)

        {

            cout << "No more space" << endl;

        }

    }

}

void viewSelledProducts()

{

    cout << "Day"<< "\t"<< "ProductName"<< "\t"<< "ProductPrice<< "\t"<< "Number of products Sell" << endl;

    for (int idx = 0; idx < productsCount; idx = idx + 1)

    {

        cout << day[idx] << "\t" << names\_prod[idx] << "\t\t" << prices\_prod[idx] << "\t\t" << numberOfProducts[idx] << endl;

    }

}

void storeMoney()

{

    fstream file;

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

    for (int idx = 0; idx < productsCount; idx++)

    {

        file << day[idx] << ",";

        file << names\_prod[idx] << ",";

        file << prices\_prod[idx] << ",";

        file << numberOfProducts[idx] << ",";

        file << oneDayRecord[idx] << endl;

    }

    file.close();

}

void loadmoneyData()

{

    fstream file;

    string line;

    file.open("money.txt", ios ::in);

    while (getline(file, line))

    {

        stringstream temp(parsemoney(line, 3));

        temp >> prices\_prod[productsCount];

        stringstream temp1(parsemoney(line, 4));

        temp1 >> numberOfProducts[productsCount];

        stringstream temp3(parsemoney(line, 5));

        temp3 >> oneDayRecord[productsCount];

        day[productsCount] = parsemoney(line, 1);

        names\_prod[productsCount] = parsemoney(line, 2);

        productsCount++;

    }

    file.close();

}

string parsemoney(string record, int field)

{

    int commaCounter4 = 1;

    string money;

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

    {

        if (record[x] == ',')

        {

            commaCounter4 = commaCounter4 + 1;

        }

        else if (commaCounter4 == field)

        {

            money = money + record[x];

        }

    }

    return money;

}

int search1()

{

    int counter;

    for (int idx2 = 0; idx2 < repairingCount; idx2++)

    {

        if (productName1[idx2] == prod1)

        {

            counter = idx2;

        }

    }

    return counter;

}

bool searchPrice1()

{

    bool flag = false;

    for (int idx2 = 0; idx2 < repairingCount; idx2++)

    {

        if (productName1[idx2] == prod1)

        {

            flag = true;

            break;

        }

    }

    return flag;

}

void AddRepairedProducts()

{

    bool flag3 = true;

    bool flag = true;

    int price1;

    while (flag == true)

    {

        if (productsRepairCount < array)

        {

            cin.ignore(389, '\n');

            cout << "Entre day...";

            getline(cin, day1[productsRepairCount]);

            cout << "Entre product : ";

            getline(cin, prod1);

            flag3 = searchPrice1();

            price1 = search1();

            if (flag3 == true)

            {

                cout << "Price of products....." << productPrice1[price1] << endl;

                prices\_prod1[productsRepairCount] = productPrice1[price1];

                cout << "Number of Products sell:";

                cin >> numberOfProducts1[productsRepairCount];

                names\_prod1[productsRepairCount] = prod1;

            }

            else

            {

                cout << "Product Not found." << endl;

                productsRepairCount--;

            }

            oneDayRecord1[productsRepairCount] = numberOfProducts1[productsRepairCount] \* prices\_prod1[productsRepairCount];

            cout << "Entre 1 to get input for another day ....";

            cin >> flag;

            productsRepairCount = productsRepairCount + 1;

        }

        else if (productsRepairCount > array)

        {

            cout << "No more space" << endl;

        }

    }

}

void viewRepairedProducts()

{

    cout << "Day"<< "\t"<< "Name"<< " \t"<< "ProductPrice"<< "\t"<< "Number of products Sell" << endl;

    for (int idx = 0; idx < productsRepairCount; idx = idx + 1)

    {

        cout << day1[idx] << "\t\t" << names\_prod1[idx] << "\t" << prices\_prod1[idx] << "\t\t" << numberOfProducts1[idx] << endl;

    }

}

void storeMoney1()

{

    fstream file;

    file.open("money1.txt", ios ::out);

    for (int idx = 0; idx < productsRepairCount; idx++)

    {

        file << day1[idx] << ",";

        file << names\_prod1[idx] << ",";

        file << prices\_prod1[idx] << ",";

        file << numberOfProducts1[idx] << ",";

        file << oneDayRecord1[idx] << endl;

    }

    file.close();

}

void loadmoneyData1()

{

    fstream file;

    string line;

    file.open("money1.txt", ios ::in);

    while (getline(file, line))

    {

        stringstream temp(parsemoney(line, 3));

        temp >> prices\_prod1[productsRepairCount];

        stringstream temp1(parsemoney(line, 4));

        temp1 >> numberOfProducts1[productsRepairCount];

        stringstream temp3(parsemoney(line, 5));

        temp3 >> oneDayRecord1[productsRepairCount];

        day1[productsRepairCount] = parsemoney(line, 1);

        names\_prod1[productsRepairCount] = parsemoney(line, 2);

        productsRepairCount = productsRepairCount + 1;

    }

    file.close();

}

void moneyEarnedBySelling()

{

    bool takeInput = true;

    while (takeInput == true)

    {

        string days;

        int sum = 0;

        cout << "Entre day to which you want to see money   ";

        cin >> days;

        int count;

        for (int idx = 0; idx < productsCount; idx++)

        {

            if (days == day[idx])

            {

                count = idx;

                sum = sum + oneDayRecord[idx];

            }

        }

        cout << endl;

        cout << "Money earned  is ...." << sum << endl;

        cout << endl;

        cout << "Entre 1 to get input for another day ....";

        cin >> takeInput;

        cout << endl;

    }

}

void moneyEarnedByRepairing()

{

    bool takeInput = true;

    while (takeInput == true)

    {

        string days1;

        cout << "Entre day to which you want to view money ";

        cin >> days1;

        int count;

        int sum1 = 0;

        for (int idx = 0; idx < productsRepairCount; idx++)

        {

            if (days1 == day1[idx])

            {

                count = idx;

                sum1 = sum1 + oneDayRecord1[idx];

            }

        }

        cout << "Money earend is..." << sum1 << endl;

        cout << "Entre 1 to get input for another day ....";

        cin >> takeInput;

    }

}

void sortedDataSelled()

{

    for (int idx = 0; idx < sellingCount; idx++)

    {

        for (int idx1 = idx + 1; idx1 < sellingCount; idx1++)

        {

            if (productName[idx] > productName[idx1])

            {

                string temp;

                temp = productName[idx];

                productName[idx] = productName[idx + 1];

                productName[idx + 1] = temp;

            }

            if (productPrice[idx] > productPrice[idx1])

            {

                int temp;

                temp = productPrice[idx];

                productPrice[idx] = productPrice[idx + 1];

                productPrice[idx + 1] = temp;

            }

        }

    }

    cout << "ProductName"<< "\t"<< "ProductPrice" << endl;

    for (int idx = 0; idx < sellingCount; idx++)

    {

        cout << productName[idx] << "\t";

        cout << productPrice[idx] << endl;

    }

}

void sortedDataRepaired()

{

    for (int idx = 0; idx < repairingCount; idx++)

    {

        for (int idx1 = idx + 1; idx1 < repairingCount; idx1++)

        {

            if (productName1[idx] > productName1[idx1])

            {

                string temp;

                temp = productName1[idx];

                productName1[idx] = productName1[idx + 1];

                productName1[idx + 1] = temp;

            }

            if (productPrice1[idx] > productPrice1[idx1])

            {

                int temp;

                temp = productPrice1[idx];

                productPrice1[idx] = productPrice1[idx + 1];

                productPrice1[idx + 1] = temp;

            }

        }

    }

    cout << "ProductName"<< "\t"<< "ProductPrice" << endl;

    for (int idx = 0; idx < repairingCount; idx++)

    {

        cout << productName1[idx] << "\t";

        cout << productPrice1[idx] << endl;

    }

}

void adminPanel()

{

    bool flag = true;

    menuAdmin();

    while (flag == true)

    {

        string option;

        cout << "Choose your option...";

        cin >> option;

        if (option == "1")

        {

            string productName;

            int productPrice;

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  View  Selling  Products" << endl;

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

            viewProductForSelling();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "2")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  View  Repairing Products" << endl;

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

            viewProductForRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "3")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "   Add  Selling  Products" << endl;

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

            addProductForSelling();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "4")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Add Repairing Products" << endl;

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

            addProductForRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "5")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Delete Selling Products" << endl;

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

            DeleteSellingProduct();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "6")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Delete Repairing Products" << endl;

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

            DeleteRepairingProduct();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "7")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Update Selling Products" << endl;

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

            UpdateSellingValues();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "8")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Update Repairing Products" << endl;

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

            UpdateRepairingValues();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "9")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Money earned in a day by selling" << endl;

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

            moneyEarnedBySelling();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "10")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "   Money earned in a day by reapiring" << endl;

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

            moneyEarnedByRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "11")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "      View number of products sell              " << endl;

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

            viewSelledProducts();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "12")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "      View number of products repair             " << endl;

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

            viewRepairedProducts();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        if (option == "13")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "     View Employees             " << endl;

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

            viewEmployees();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "14")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Add Manager" << endl;

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

            addManager();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "15")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  View Manager" << endl;

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

            viewManager();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "16")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Remove Manager" << endl;

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

            removeManager();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "17")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Update Manager" << endl;

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

            updateManager();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "18")

        {

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

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

            cout << "  Sorted Data" << endl;

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

            sortedDataSelled();

            sortedDataRepaired();

            clearScreen();

            printMainScreen();

            openRootMenu("Admin");

            menuAdmin();

        }

        else if (option == "19")

        {

            flag = false;

            clearScreen();

            printMainScreen();

            openRootMenu(" login ");

            storeSellingProduct();

            storeRepairingProduct();

            storeManager();

        }

    }

}

void ManagerPanel()

{

    bool flag = true;

    menuManager();

    while (flag == true)

    {

        string option;

        cout << "choose your option...";

        cin >> option;

        if (option == "1")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "     View Employees             " << endl;

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

            viewEmployees();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "2")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View Selling products             " << endl;

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

            viewProductForSelling();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "3")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View Repairing products             " << endl;

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

            viewProductForRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "4")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "  Add Employees                              " << endl;

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

            addEmployees();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "5")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      Remove Employees             " << endl;

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

            removeEmployees();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "6")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      Upadate Employees             " << endl;

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

            updateEmployee();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "7")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View number of products sell        " << endl;

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

            viewSelledProducts();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "8")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View number of products repair        " << endl;

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

            viewRepairedProducts();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "9")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View  money earned in one day by selling  " << endl;

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

            moneyEarnedBySelling();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "10")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

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

            cout << "      View  money earned in one day by repairing  " << endl;

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

            moneyEarnedByRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu(" Manager ");

            menuManager();

        }

        else if (option == "11")

        {

            flag = false;

            clearScreen();

            printMainScreen();

            openRootMenu("Login ");

            storeEmployees();

        }

    }

}

void EmployeePanel()

{

    bool flag = true;

    menuEmployee();

    while (flag == true)

    {

        string option;

        cout << "Choose your option...";

        cin >> option;

        if (option == "1")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      View Selling products             " << endl;

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

            viewProductForSelling();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "2")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      View Repairing products             " << endl;

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

            viewProductForRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "3")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      Add number of products sell              " << endl;

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

            AddSelledProducts();

            storeMoney();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "4")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      Add number of products repair             " << endl;

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

            AddRepairedProducts();

            storeMoney1();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "5")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      View money earend by selling           " << endl;

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

            moneyEarnedBySelling();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "6")

        {

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

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

            cout << "      View money earend by repairing           " << endl;

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

            moneyEarnedByRepairing();

            clearScreen();

            printMainScreen();

            openRootMenu(" Employee  ");

            menuEmployee();

        }

        else if (option == "7")

        {

            flag = false;

            clearScreen();

            printMainScreen();

            openRootMenu("Main ");

        }

    }

}

* **Weakness in the Business Application**

It is a drawback that total money earned in the month can not estimated.

* **Future Directions**

To make the application free of drawback like handling quarterly and monthly income

efficiently, make it user friendly, handle complex problems and accessed globally.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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:** |  |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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:** |  |