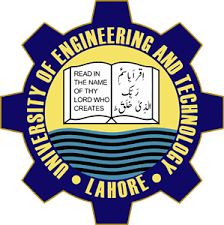
**Car Rental Management System**

****

Session:2021 – 2024

**Submitted by:**

Bisma Muhammad Ali 2021-CS-170

**Supervised by:**

Mam Maida Shahid

Department of Computer Science

**University of Engineering and Technology**

**Lahore, Pakistan**

**Description:**

This project will handle the management of car rent company. CRMS has most of the facilities that required for rent a car. First of all, Admin add customers in CRMS. Then add cars and their rent. After that, customers can login in the CRMS. They can see car status. They can select car under their budget. After selecting a car invoice will create.

**Users of System:**

Users of CRMS are

* Admin
* Customer

**Functional Requirements:**

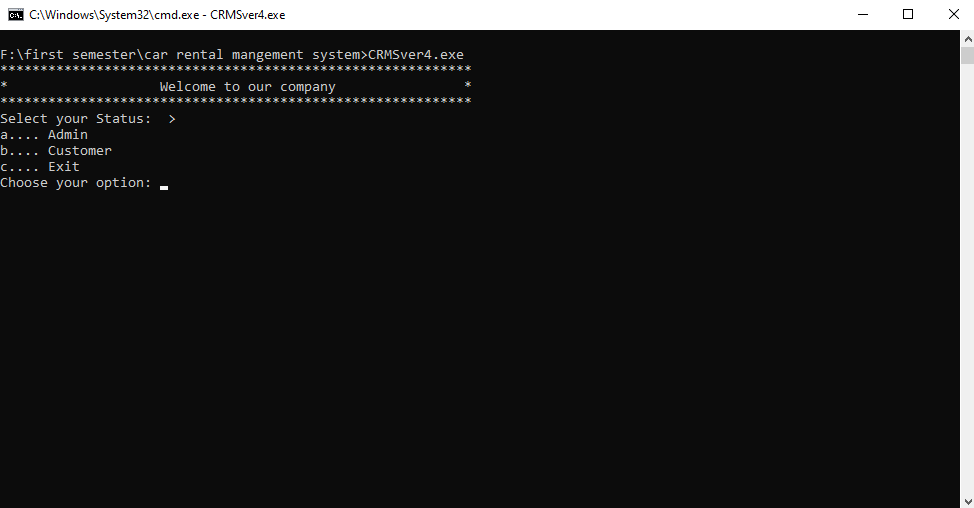
**Admin**

* As an admin I can access login page for CRMS
* As an admin I can select my status (Admin, Customer)
* As an admin I can display car’s model
* As an admin I can add new customers
* As an admin I can Remove customer
* As an admin I can change rent of car
* As an admin I can add new car
* As an admin I can remove car
* As an admin I can view all customer’s record

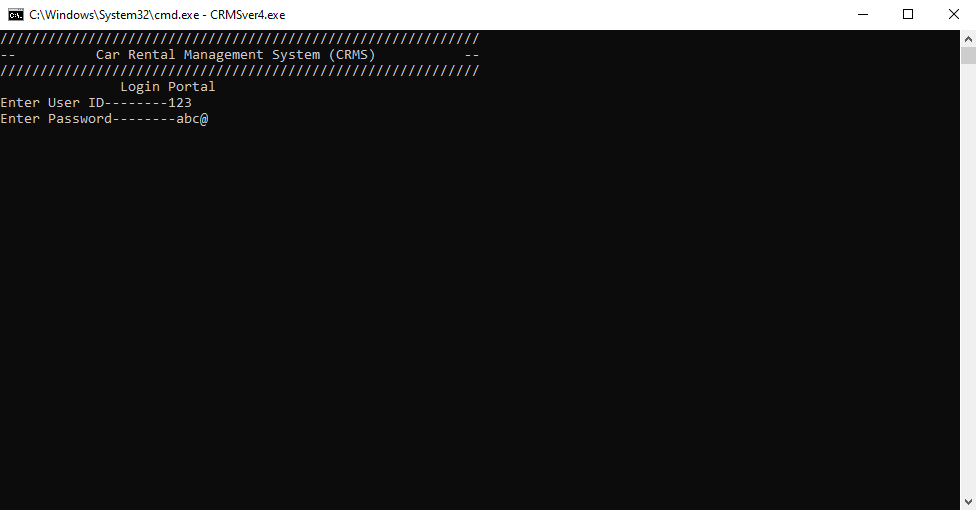
**Customer**

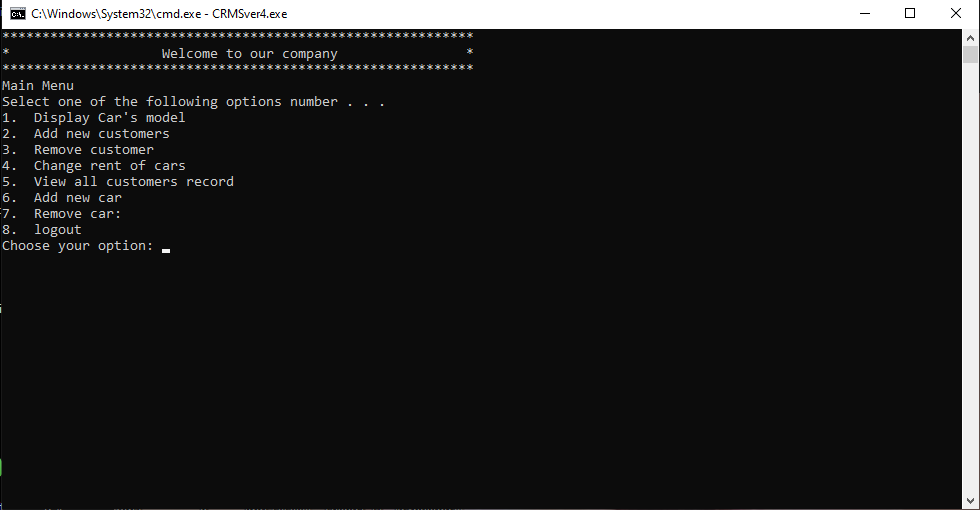
* As a customer I can access login page for CRMS
* As a customer I can Select my Status (Admin, Customer)
* As a customer I can see rent of all cars
* As a customer I can see cars under my budget
* As a customer I can display cars from higher to lower rent (sorting)
* As a customer I can see my Invoice

**Wireframes:**

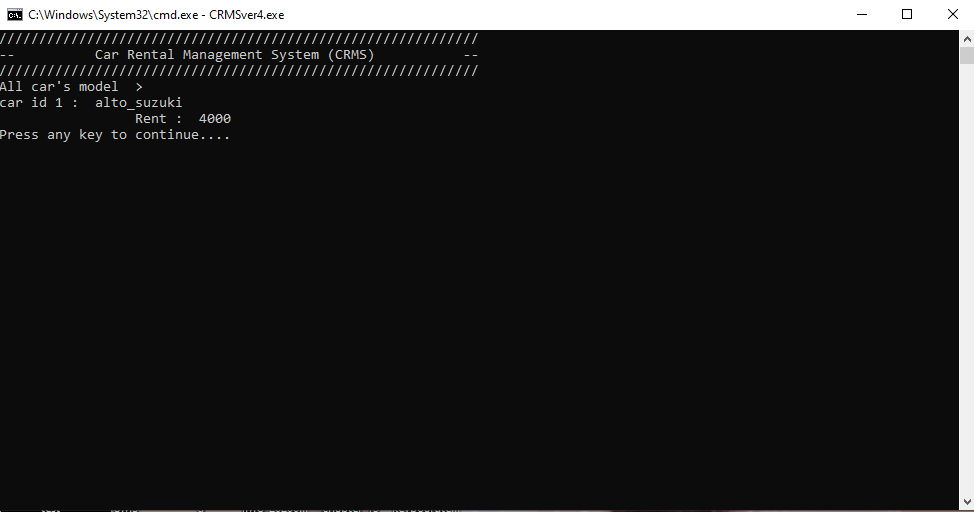
**Status selection**

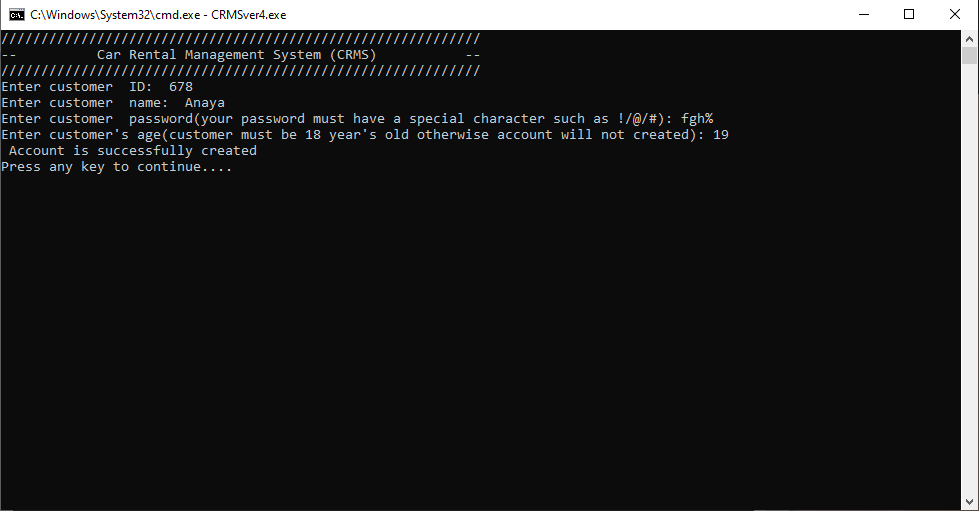
**After selecting option ‘a’ (admin) this login portal will display**

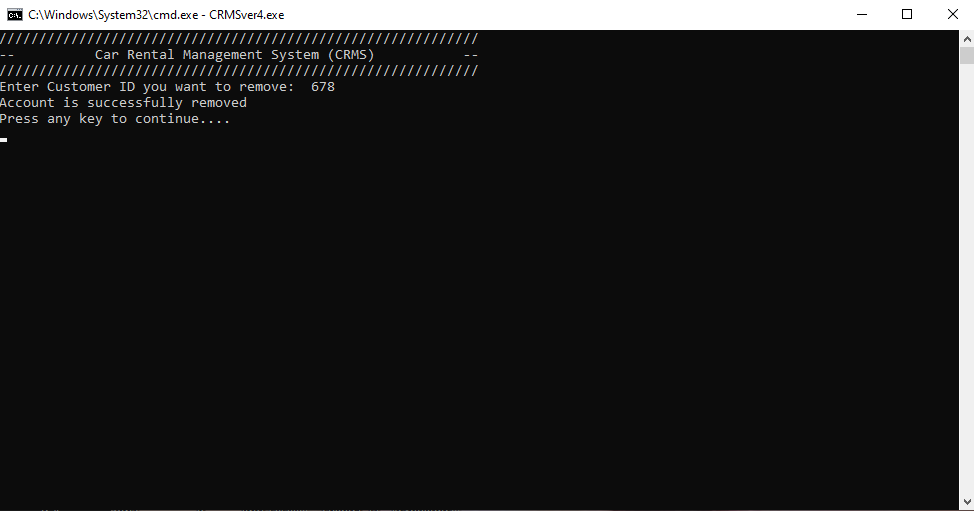
****

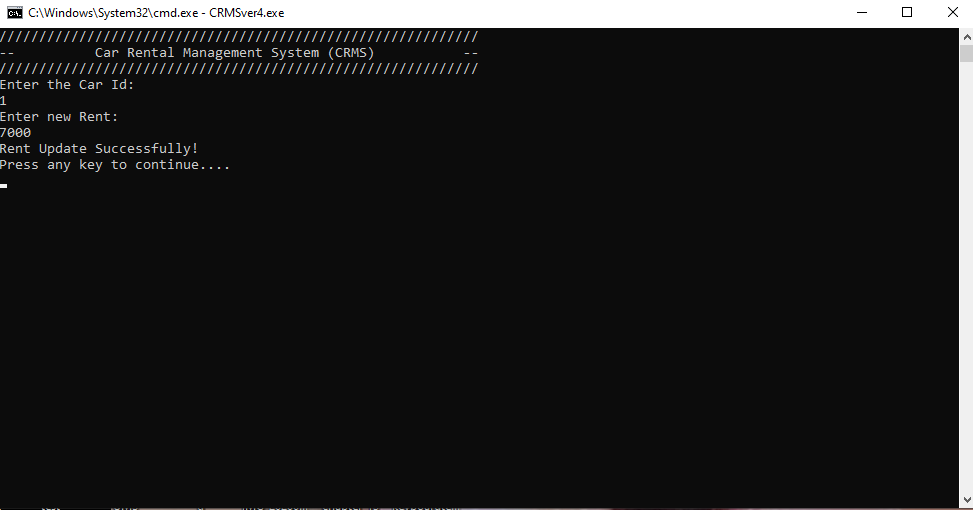
**After enter id and password Admin’s menu will display**

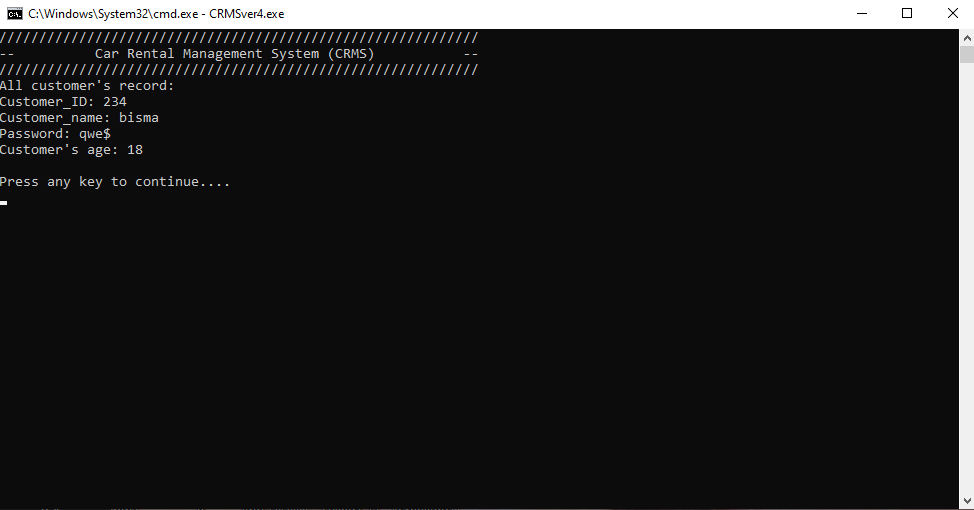
**When user choose option 1 all cars model will display**

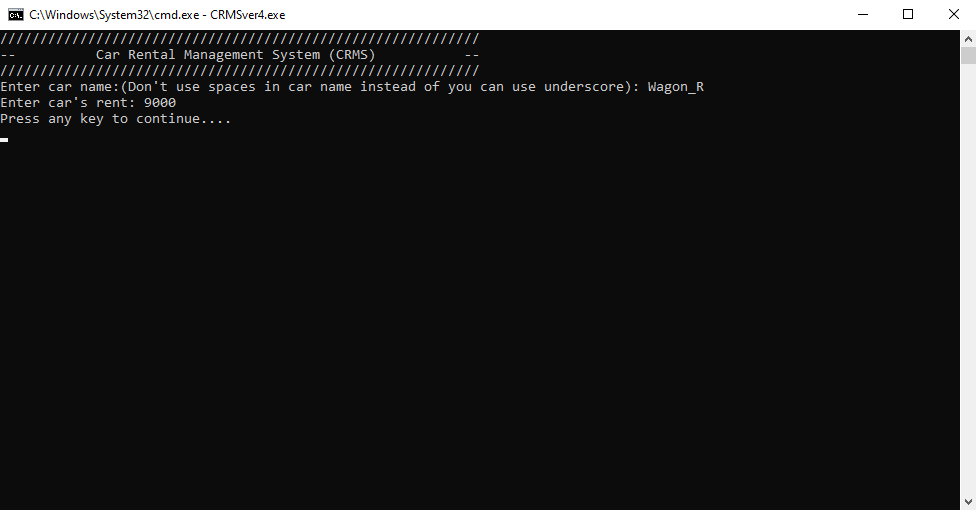
****

**When user choose option 2. A new customer’s account can be creat**

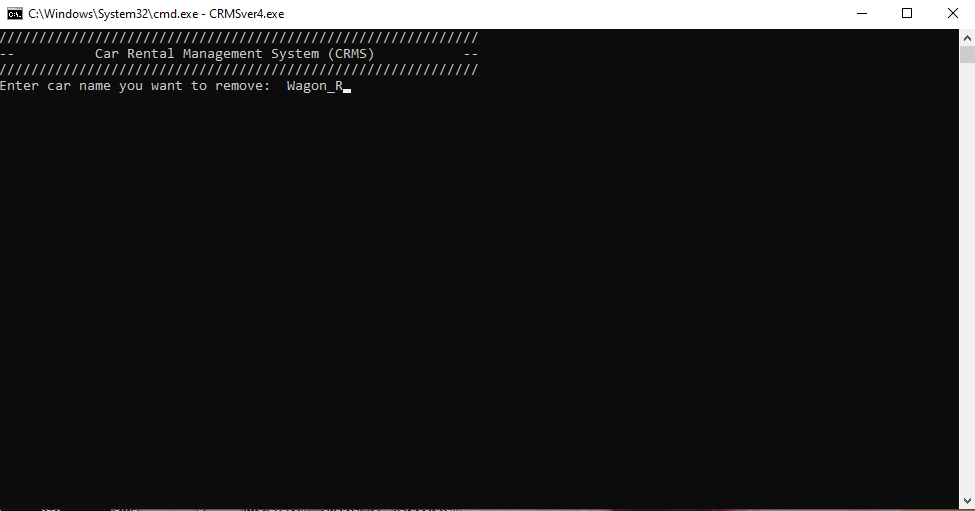
**When user choose option 3. A customer can be removed**

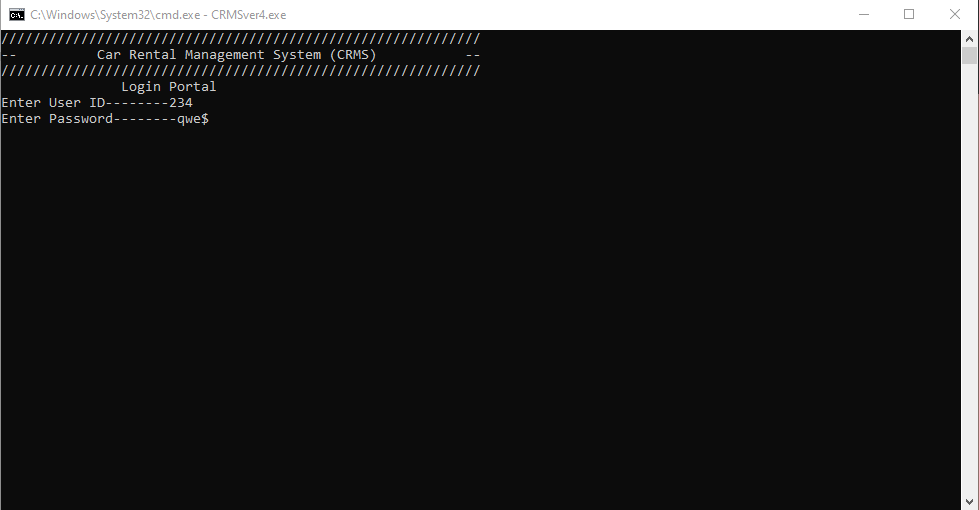
**When user press 4. Rent of car can be changed.**

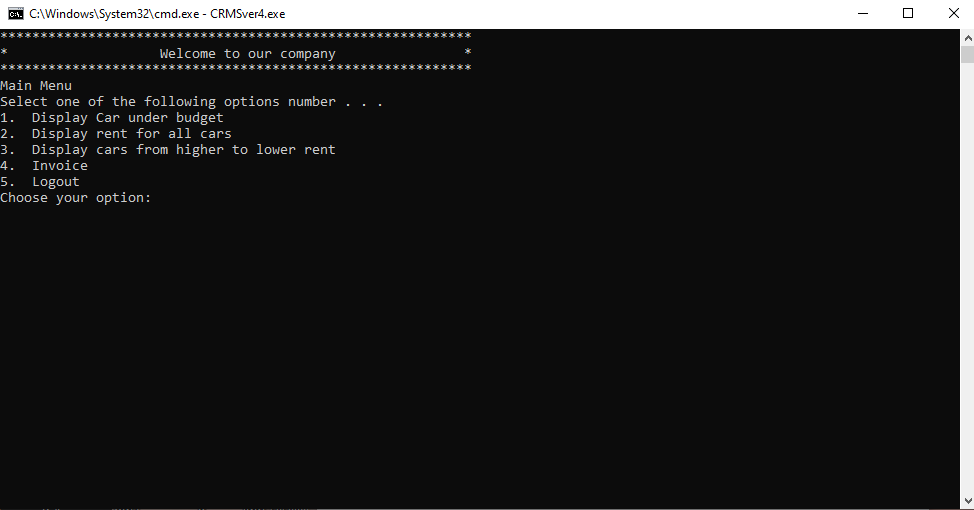
**When user press 5. All customer record will display.**

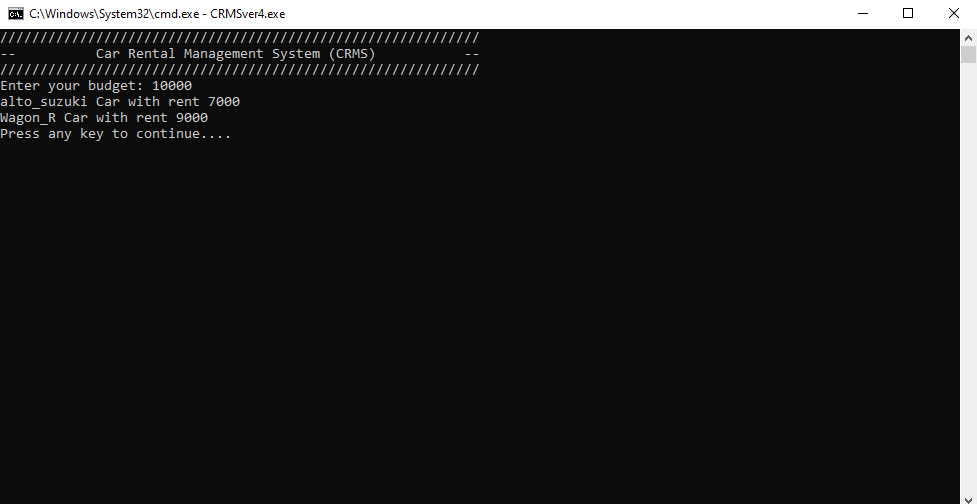
**When user press 6 one car can be add**

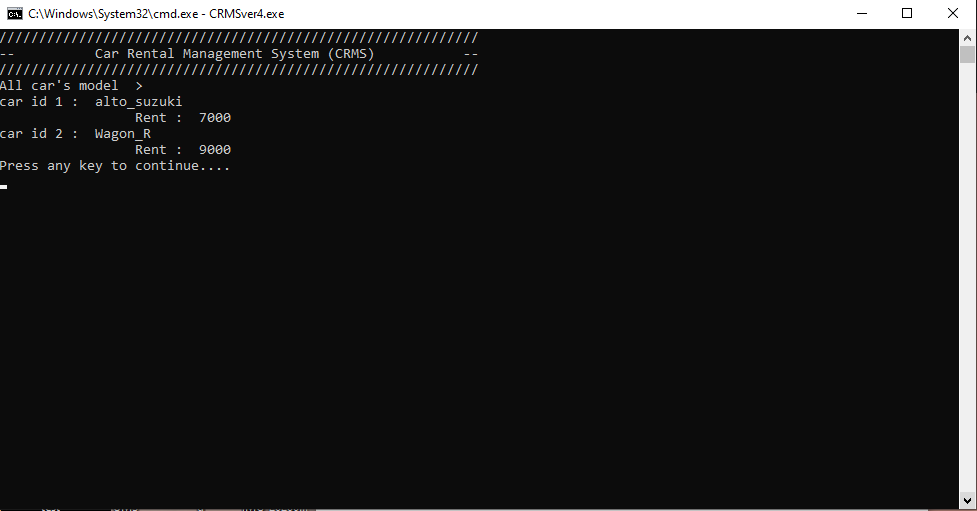
**When user press 7 one car can be removed**

****

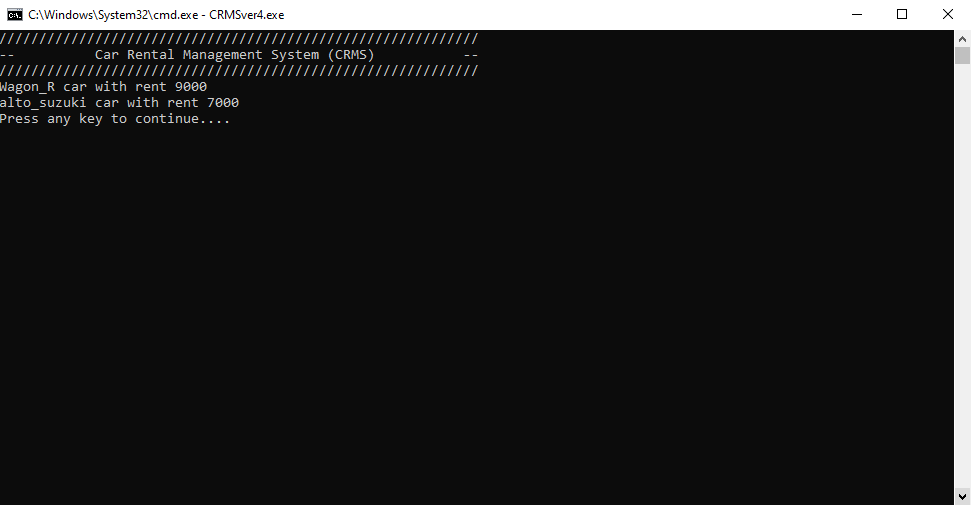
**After logout admin user login portal for customer**

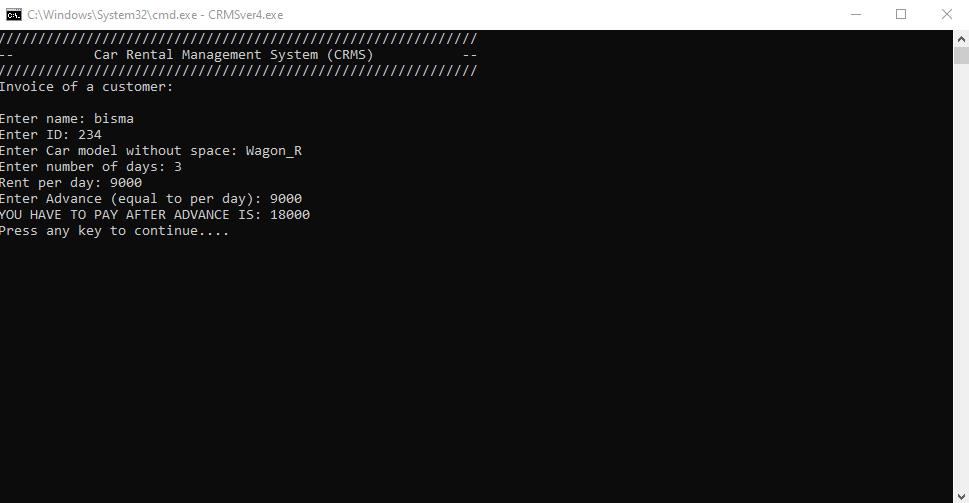
**After login customer menu will display**

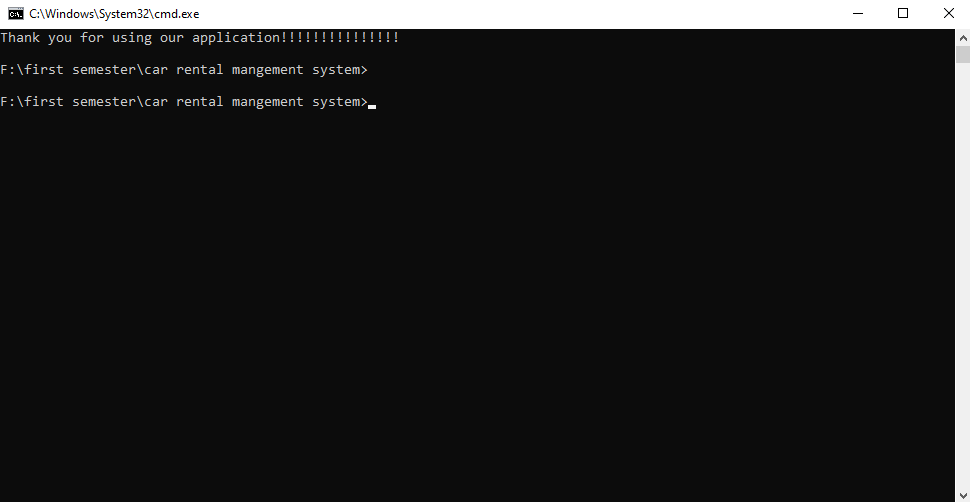
**When user press 1 cars will display under the budget of the customer**

**When user press 2 rent of all cars will display with their model**

**When user press 3 cars will display from higher to lower rent**

****

**When user press 4 invoice of the customer will create**

** When user press ‘c’ (Exit)**

**Data Structures:**

string car\_nameA[MAX\_RECORDS];

int car\_rentA[MAX\_RECORDS];

int car\_count = 0;

string cust\_nameA[MAX\_RECORDS];

int cust\_idA[MAX\_RECORDS];

string C\_passwordA[MAX\_RECORDS];

int cust\_age[MAX\_RECORDS];

int count = 0;

**Function Prototypes:**

// ------------------ Prototypes---------------------

void header();

void header1();

void system\_cls();

char status\_menu();

char login\_portal();

char admin\_menu(char option);

char cust\_menu(char option);

void cust\_display();

void models();

void invoice(string car\_name, int day, int rental\_amount);

void Budget(int budget);

void edit\_rent();

void add\_C();

void remove\_C();

void add\_Car();

void remove\_Car();

void addCarIntoArray(string car, int rent);

void addCustIntoArray(int c\_id, string cust\_name, string cust\_p, int age);

int sorting(int s);

bool checkUser(int id, string password);

void addCarIntoFile();

void addCustIntoFile();

void loadCarIntoArray();

void loadCustomerIntoArray();

string parseRecord(string record, int field);

bool validationOfPassword(string pass);

//------------------- prototypes end-------------------------------

**Function Working Flow:**

**Code:**

#include <iostream>

#include <conio.h>

#include <stdlib.h>

#include <fstream>

#include <sstream>

#include <string.h>

using namespace std;

char status;

int id = 0;

string password;

// ------------------ Prototypes---------------------

void header();

void header1();

void system\_cls();

char status\_menu();

char login\_portal();

char admin\_menu(char option);

char cust\_menu(char option);

void cust\_display();

void models();

void invoice(string car\_name, int day, int rental\_amount);

void Budget(int budget);

void edit\_rent();

void add\_C();

void remove\_C();

void add\_Car();

void remove\_Car();

void addCarIntoArray(string car, int rent);

void addCustIntoArray(int c\_id, string cust\_name, string cust\_p, int age);

int sorting(int s);

bool checkUser(int id, string password);

void addCarIntoFile();

void addCustIntoFile();

void loadCarIntoArray();

void loadCustomerIntoArray();

string parseRecord(string record, int field);

bool validationOfPassword(string pass);

//------------------- prototypes end-------------------------------

const int MAX\_RECORDS = 20;

//------------------- Data Structures-----------------------------

string car\_nameA[MAX\_RECORDS];

int car\_rentA[MAX\_RECORDS];

int car\_count = 0;

string cust\_nameA[MAX\_RECORDS];

int cust\_idA[MAX\_RECORDS];

string C\_passwordA[MAX\_RECORDS];

int cust\_age[MAX\_RECORDS];

int count = 0;

/////////////////////// start of main function /////////////////////////////

main()

{

loadCarIntoArray();

loadCustomerIntoArray();

char opt, mainchoice, login, option;

int budget = 0, rental\_amount = 0, rent = 0, cust\_id = 0, day, Advance = 0;

string cust\_name, cust\_p, car\_name;

/\* Hard code Id and password....

Admin id is 123 and password is abc@ \*/

while (true)

{

mainchoice = status\_menu();

login = login\_portal();

if (status == 'a' && id == 123 && password == "abc@")

{

while (true)

{

option = admin\_menu(option);

if (option == '1')

{

models();

} // display model of cars

else if (option == '2')

{

header1();

add\_C();

system\_cls();

} // Add customers

else if (option == '3')

{

remove\_C();

system\_cls();

} // remove customers

else if (option == '4')

{

header1();

edit\_rent();

system\_cls();

} // change rent

else if (option == '5')

{

header1();

cust\_display();

system\_cls();

} // view customer's detail

else if (option == '6')

{

header1();

add\_Car();

system\_cls();

} // Add cars

else if (option == '7')

{

header1();

remove\_Car();

system\_cls();

} // Remove cars

else if (option == '8')

{

addCustIntoFile(); // add customer into file

addCarIntoFile(); // add car into file

break; // log out

}

else

{

cout << "You Chose Wrong option " << endl;

}

} // end of admin while loop

} // start of customers option

else if (status == 'b')

{

if (checkUser(id, password))

{

while (true)

{

option = cust\_menu(option);

if (option == '1')

{

Budget(budget);

system\_cls();

} // Recommend cars under customer's budget

else if (option == '2')

{

models();

} // display rent of all cars

else if (option == '3')

{

header1();

int temp, high\_idx;

string tempS;

for (int s = 0; s < car\_count; s++)

{

high\_idx = sorting(s);

temp = car\_rentA[high\_idx];

car\_rentA[high\_idx] = car\_rentA[s];

car\_rentA[s] = temp;

tempS = car\_nameA[high\_idx];

car\_nameA[high\_idx] = car\_nameA[s];

car\_nameA[s] = tempS;

}

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

{

cout << car\_nameA[i] << " car with rent " << car\_rentA[i] << endl;

}

system\_cls();

} // sort cars on the basis of rent

else if (option == '4')

{

invoice(car\_name, day, rental\_amount);

system\_cls();

} // bill of customer

else if (option == '5')

{

break;

system\_cls(); // log out

}

else

{

cout << "You Choose Wrong option " << endl;

system\_cls();

} // end of customer's while loop

}

}

else

{

cout << "You entered wrond password " << endl;

system\_cls();

}

}

else if (status == 'c')

{

cout << "Thank you for using our application!!!!!!!!!!!!!!!" << endl;

break;

}

else

{

cout << "You entered wrong option......" << endl;

system\_cls();

}

}

}

/////////////////////////////// PROGRAM END...... /////////////////////////////////

// function definition.........

void header()

{

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

cout << "\* Welcome to our company \*" << endl;

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

}

// end of header function

void header1()

{

cout << "////////////////////////////////////////////////////////////" << endl;

cout << "-- Car Rental Management System (CRMS) --" << endl;

cout << "////////////////////////////////////////////////////////////" << endl;

}

// end of 2nd header function

void system\_cls()

{

cout << "Press any key to continue...." << endl;

getch();

system("CLS");

}

// end of system\_cls function

char status\_menu()

{

header();

cout << "Select your Status: >" << endl;

cout << "a.... Admin" << endl;

cout << "b.... Customer" << endl;

cout << "c.... Exit" << endl;

cout << "Choose your option: ";

cin >> status;

system\_cls();

return status;

}

// end of status\_menu

char login\_portal()

{

if (status == 'a' || status == 'b')

{

header1();

cout << " Login Portal " << endl;

cout << "Enter User ID--------";

cin >> id;

cout << "Enter Password--------";

cin.ignore();

getline(cin, password);

}

system\_cls();

return status;

}

// end of login portal function

char admin\_menu(char option)

{

header();

cout << "Main Menu " << endl;

cout << "Select one of the following options number . . ." << endl;

cout << "1. Display Car's model" << endl;

cout << "2. Add new customers" << endl;

cout << "3. Remove customer" << endl;

cout << "4. Change rent of cars" << endl;

cout << "5. View all customers record" << endl;

cout << "6. Add new car" << endl;

cout << "7. Remove car: " << endl;

cout << "8. logout" << endl;

cout << "Choose your option: ";

cin >> option;

system\_cls();

return option;

}

// end of admin's menu

char cust\_menu(char option)

{

header();

cout << "Main Menu " << endl;

cout << "Select one of the following options number . . ." << endl;

cout << "1. Display Car under budget" << endl;

cout << "2. Display rent for all cars" << endl;

cout << "3. Display cars from higher to lower rent" << endl;

cout << "4. Invoice" << endl;

cout << "5. Logout" << endl;

cout << "Choose your option: ";

cin >> option;

system\_cls();

return option;

}

// end of customer's menu

void cust\_display()

{

if (count > 0)

{

cout << "All customer's record:" << endl;

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

{

cout << "Customer\_ID: " << cust\_idA[i] << endl;

cout << "Customer\_name: " << cust\_nameA[i] << endl;

cout << "Password: " << C\_passwordA[i] << endl;

cout << "Customer's age: " << cust\_age[i] << endl;

cout << " " << endl;

}

}

else

cout << "No Customer record Available! " << endl;

}

// end of display customers function

void models()

{

header1();

if (car\_count == 0)

{

cout << "Sorry no car available!!!!!!" << endl;

}

else

{

cout << "All car's model >" << endl;

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

{

cout << "car id " << i + 1 << " : " << car\_nameA[i] << endl

<< "\t\t Rent : " << car\_rentA[i] << endl;

}

}

system\_cls();

}

// end of customer display function

void edit\_rent()

{

int \_id;

int rent;

if (car\_count == 0)

{

cout << "Sorry no car available!!!!!!" << endl;

}

else

{

cout << "Enter the Car Id: " << endl;

cin >> \_id;

if (\_id > 0 && \_id <= car\_count)

{

cout << "Enter new Rent: " << endl;

cin >> rent;

car\_rentA[\_id - 1] = rent;

cout << "Rent Update Successfully!" << endl;

}

else

cout << "Invalid Care Id!" << endl;

}

}

// end of change rent function

void invoice(string car\_name, int day, int rental\_amount)

{

header1();

cout << "Invoice of a customer: " << endl;

cout << " " << endl;

cout << "Enter Car model: ";

cin.ignore();

getline(cin, car\_name);

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

{

if (car\_nameA[i] == car\_name)

{

cout<<"If you avail this car more than 4 days you will get 5% discount"<<endl;

cout << "Enter number of days: ";

cin >> day;

rental\_amount = day \* car\_rentA[i];

if(day>=5)

{

rental\_amount=rental\_amount-(rental\_amount\*5/100);

}

cout << "YOUR TOTAL AMOUNT: " << rental\_amount << endl;

}

}

if (rental\_amount == 0)

{

cout << "Car not found!!!!" << endl;

}

}

// end of invoice function

void Budget(int budget)

{

int c = 0;

header1();

cout << "Enter your budget: ";

cin >> budget;

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

{

if (car\_rentA[i] <= budget)

{

cout << car\_nameA[i] << " Car with rent " << car\_rentA[i] << endl;

c++;

}

}

if (c == 0)

{

cout << "Sorry no car available under your budget......" << endl;

}

}

// end of function that recommend cars under customer's budget

void add\_C()

{

int n = 0;

int c\_id;

string cust\_name;

string cust\_p;

int age;

cout << "Enter customer ID: ";

cin >> c\_id;

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

{

if (cust\_idA[i] == c\_id)

{

cout << "You already assign this id to another customer Use another id" << endl;

n = 1;

break;

}

}

if (n == 0)

{

cout << "Enter customer name: ";

cin.ignore();

getline(cin, cust\_name);

cout << "Enter customer password(your password must have a special character such as !/@/#): ";

getline(cin, cust\_p);

if (validationOfPassword(cust\_p) == 1)

{

cout << "Enter customer's age(customer must be 18 year's old otherwise account will not created): ";

cin >> age;

if (age >= 18 && age <= 65)

{

addCustIntoArray(c\_id, cust\_name, cust\_p, age);

cout << " Account is successfully created" << endl;

}

else

{

cout << "To create an account your age must be 18!!!!!!!!!!!!" << endl;

}

}

else

cout << "please enter a valid password" << endl;

}

}

// add customers function end

void addCustIntoArray(int c\_id, string cust\_name, string cust\_p, int age)

{

cust\_idA[count] = c\_id;

cust\_nameA[count] = cust\_name;

C\_passwordA[count] = cust\_p;

cust\_age[count] = age;

count++;

}

// storing customer details in array

void addCustIntoFile()

{

fstream cust;

cust.open("customer.txt", ios::out);

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

cust << cust\_idA[i] << "," << cust\_nameA[i] << "," << C\_passwordA[i] << "," << cust\_age[i] << endl;

cust.close();

}

// add customer's into file function

void remove\_C()

{

int \_id;

int index = -1;

if (count == 0)

cout << "No Record Available to Remove!" << endl;

else

{

header1();

cout << "Enter Customer ID you want to remove: ";

cin >> \_id;

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

{

if (cust\_idA[i] == \_id)

{

index = i;

break;

}

}

if (index >= 0 && index < MAX\_RECORDS)

{

for (int i = index; i < count - 1; i++)

{

cust\_idA[i] = cust\_idA[i + 1];

cust\_nameA[i] = cust\_nameA[i + 1];

C\_passwordA[i] = C\_passwordA[i + 1];

}

cust\_idA[count] = 0;

cust\_nameA[count] = "";

C\_passwordA[count] = "";

count--;

cout << "Account is successfully removed" << endl;

}

else

cout << "Invalid Customer ID!" << endl;

}

}

// remove customers function

void add\_Car()

{

string car;

int rent;

cout << "Enter car name: ";

cin.ignore();

getline(cin, car);

cout << "Enter car's rent: ";

cin >> rent;

addCarIntoArray(car, rent);

}

void addCarIntoArray(string car, int rent)

{

car\_nameA[car\_count] = car;

car\_rentA[car\_count] = rent;

car\_count++;

}

// storing cars details into array

void addCarIntoFile()

{

fstream car;

car.open("car.txt", ios::out);

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

car << car\_nameA[i] << "," << car\_rentA[i] << endl;

car.close();

}

// add car into file

void remove\_Car()

{

string car\_name;

int index = -1;

if (car\_count == 0)

cout << "No Record Available to Remove!" << endl;

else

{

cout << "Enter car name you want to remove: ";

cin.ignore();

getline(cin, car\_name);

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

{

if (car\_nameA[i] == car\_name)

{

index = i;

break;

}

}

if (index >= 0 && index < MAX\_RECORDS)

{

for (int i = index; i < car\_count - 1; i++)

{

car\_nameA[i] = car\_nameA[i + 1];

car\_rentA[i] = car\_rentA[i + 1];

}

car\_nameA[car\_count] = "";

car\_rentA[car\_count] = 0;

car\_count--;

cout << "Car is successfully removed" << endl;

}

else

cout << "Invalid Car !" << endl;

}

}

// function for removing car

int sorting(int s)

{

int largest, idx;

largest = -1;

for (int i = s; i < car\_count; i++)

{

if (largest < car\_rentA[i])

{

largest = car\_rentA[i];

idx = i;

}

}

return idx;

}

// function for sorting car

bool checkUser(int id, string password)

{

for (int i = 0; i < MAX\_RECORDS; i++)

{

if (id == cust\_idA[i] && password == C\_passwordA[i])

{

return true;

}

}

return false;

}

// function for check user wether they are valid or not

string parseRecord(string record, int field)

{

int commaCount = 1;

string item;

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

{

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

{

commaCount++;

}

else if (commaCount == field)

{

item = item + record[x];

}

}

return item;

}

// function to seprate data into files

void loadCustomerIntoArray()

{

int idx = 0;

string word;

fstream cust;

cust.open("customer.txt", ios::in);

while (!cust.eof())

{

getline(cust, word);

if (cust.eof())

break;

cust\_idA[count] = stoi(parseRecord(word, 1));

cust\_nameA[count] = parseRecord(word, 2);

C\_passwordA[count] = parseRecord(word, 3);

cust\_age[count] = stoi(parseRecord(word, 4));

count++;

}

cust.close();

}

// function for storing data from file to arrays

void loadCarIntoArray()

{

string word;

fstream car;

car.open("car.txt", ios::in);

while (!car.eof())

{

getline(car, word);

if (car.eof())

break;

car\_nameA[car\_count] = parseRecord(word, 1);

car\_rentA[car\_count] = stoi(parseRecord(word, 2));

car\_count++;

}

car.close();

}

// function for storing cars from file to arrays

bool validationOfPassword(string pass)

{

for (int x = 0; pass[x] != '\0'; x++)

{

if (pass[x] == '@' || pass[x] == '!' || pass[x] == '#' || pass[x] == '%' || pass[x] == '^' || pass[x] == '&' || pass[x] == '\*' || pass[x] == '$')

return 1;

}

return 0;

}

// function for checking that password is valid or not

**Test cases:**

Same as wireframes.

**Student Reg. No. :**  2021-CS-170  **Student Name.**  Bisma Muhammad Ali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 are 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 consistent and according to given **guidelines**. The project **Poster** is professionally designed and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meets more than 80% of the criteria given. | Documentation meets more than 50% of the criteria. | When the documentation meets less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What you **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | The project has at least 2 user 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 are followed | All code style criteria followed but some improvements required | A lot of improvements are 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 are synchronized. | Code and documentation do not synchronize at **some** places | Code and documentation do not synchronize in **many** places | Code and documentation **do not** synchronize. |
| Data Structure (Arrays)  **Grade:** | The data structure is sufficient for the project requirements | Data Structure is sufficient but requires improvement to meet project requirements. | The data structure is not sufficient and needs a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating a useful report | The sorting feature is working but sorted data is not useful for the project. | The sorting feature is partial implemented | The project does not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places, it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places, it is missing. | Validations are missing a lot of places | No Validations are used |
| Recommendation Feature | The proper meaning full recommendation is present in the system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | The presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | The student has a complete understanding of how the code is working and knows the concept. | The student has good understanding but in some places, he does not know the concepts | The student has very little understanding and lacks the major concepts. | The student does not have any level of understanding of the code. |

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

**Student Reg. No. :**  2021-CS-170  **Student Name.**  Bisma Muhammad Ali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 are 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 consistent and according to given **guidelines**. The project **Poster** is professionally designed and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meets more than 80% of the criteria given. | Documentation meets more than 50% of the criteria. | When the documentation meets less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What you **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | The project has at least 2 user 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 are followed | All code style criteria followed but some improvements required | A lot of improvements are 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 are synchronized. | Code and documentation do not synchronize at **some** places | Code and documentation do not synchronize in **many** places | Code and documentation **do not** synchronize. |
| Data Structure (Arrays)  **Grade:** | The data structure is sufficient for the project requirements | Data Structure is sufficient but requires improvement to meet project requirements. | The data structure is not sufficient and needs a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating a useful report | The sorting feature is working but sorted data is not useful for the project. | The sorting feature is partial implemented | The project does not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places, it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places, it is missing. | Validations are missing a lot of places | No Validations are used |
| Recommendation Feature | The proper meaning full recommendation is present in the system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | The presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | The student has a complete understanding of how the code is working and knows the concept. | The student has good understanding but in some places, he does not know the concepts | The student has very little understanding and lacks the major concepts. | The student does not have any level of understanding of the code. |

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

