Topic : Online Medical Portal.

Group no : MLB_9.1_05

Campus : Malabe

Submission Date : 19/05/2022

We declare that this is our own work and this Assignment does not incorporate without acknowledgment any material previously submitted by anyone else in SLIIT or any other university/Institute. And we declare that each one of us equally contributed to the completion of this Assignment.

Registration No	Name	Contact Number
IT21176388	H.C.K Ariyarathna	0761518756
IT21175602	H.A.N Nilakshana	0719484668
IT21173486	H.M.S Migara	0778838567
IT21174780	D.M.M.I.T Dissanayaka	0758236348
IT21174308	Kumarasinghe O.A	0757107968

Table of Contents.

1. User Requirements

1. There are four types of users.
2. Those are, admin who control the system and manager who manage the services.
3. Others are patients and doctors.
4. Patients and doctor are mainly connected with the system.
5. Patients are divided into registered patients and unregistered patients.
6. Unregistered patients can only get doctor details.
7. If they want more details, they must create an account on this website.
8. After users register and login into the website, they can get more facilities.
9. Admin can view user profiles and verify them, and admin can also add/remove users.
10. Registered patients can be booked the desired doctor on the doctor's page.
11. And patients can remove the appointments they made earlier.
12. Patients can be made a payment through an appropriate payment method as a credit or debit.
13. Once the patient has made an appointment, it will appear to the admin, he can check it and accept or cancel it.
14. After that doctor can see those appointments and he can accept.
15. User receives a notification after a successful appointment.
16. Admin can generate those transaction reports.
17. After receiving the service, the patient can add their review in the customer feedback forum.
18. If the patient has questions, they can contact custom services.
19. Admin or manager can reply to those questions because only they have access to it.
20. System admin can maintain doctor profiles, patient profiles and history of appointment details.
21. Manager can manage all staff and manage payments.

2. Noun/Verb Analysis

1. There are four types of users.
2. Those are, admin who control the system and manager who manage the services.
3. Others are patients and doctors.
4. Patients and doctor are mainly connected with the system.
5. Patients are divided into registered patients and unregistered patients.
6. Unregistered patients can only get doctor details.
7. If they want more details, they must create an account on this website.
8. After users register and login into the website, they can get more facilities.
9. Admin can view user profiles and verify them, and admin can also add/remove users.
10. Registered patients can be booked the desired doctor on the doctor's page.
11. And patients can remove the appointments they made earlier.
12. Patients can be made a payment through an appropriate payment method as a credit or debit.
13. Once the patient has made an appointment, it will appear to the admin, he can check it and accept or cancel it.
14. After that doctor can see those appointments and he can accept.
15. User receives a notification after a successful appointment.
16. Admin can generate those transaction reports.
17. After receiving the service, the patient can add their review in the customer feedback forum.
18. If the patient has questions, they can contact custom services.
19. Admin or manager can reply to those questions because only they have access to it.
20. System admin can maintain doctor profiles, patient profiles and history of appointment details.
21. Manager can manage all staff and manage payments.

3. Identify classes using Noun Analysis.

Identified classes: –

- Admin
- Manager
- Registered Patient
- Unregistered Patient
- Doctor
- Appointments
- Reports
- Feedback
- Custom service
- Payment

Reasons for rejecting other nouns.

- Redundant –
  - Users refer to admin, manager, patients, and doctors.
  - Staff members and staff refer to all the users.
  - Patients – divided into unregistered and registered users.
  - System admin refers to admin.

- Outside the scope –
  - System.
  - Website.
- An event or an operation –
  - Services, medical services.
  - Facilities.
  - Doctor details.
  - Review.
  - Questions.
  - Booking details.

- An attribute –
  - Account.
  - Profiles.
  - Payment method.
  - Doctor profiles.
  - Patient profiles.
  - Transaction reports.

4. CRC Cards.

Class: Admin	
Responsibility	**Collaborators**
View user profiles and verify them.	Registered Patient, Doctor
Check appointment details	Appointment
Add/remove users	Register patient, Doctor
Reply to the custom questions	Custom support
Maintain profiles	Registered Patient, Doctor

Class: Manager	
Responsibility	**Collaborators**
Customer support	
Manage payment	Payment
Manage staff	Doctor, Registered Patient, Unregistered Patient
Generate transaction reports	Reports
Reply to the custom questions	Custom support

Class: Registered Patient	
Responsibility	**Collaborators**
Login	
Make an appointment	Appointments
Get doctor details	
Search a doctor	
Cancel appointment	Appointments
Make a payment	Payment
Contact custom support	Custom support

Class: Unregistered Patient	
Responsibility	**Collaborators**
Get doctor details.	Doctor
Create account	
Check feedback	Feedback

Class: Doctor	
Responsibility	**Collaborators**
Login	
Check Appointments	Appointment

Class: Appointments	
Responsibility	**Collaborators**
Store member details about appointments.	Register patient
Provide details of appointment	
Confirm Appointment.	Admin

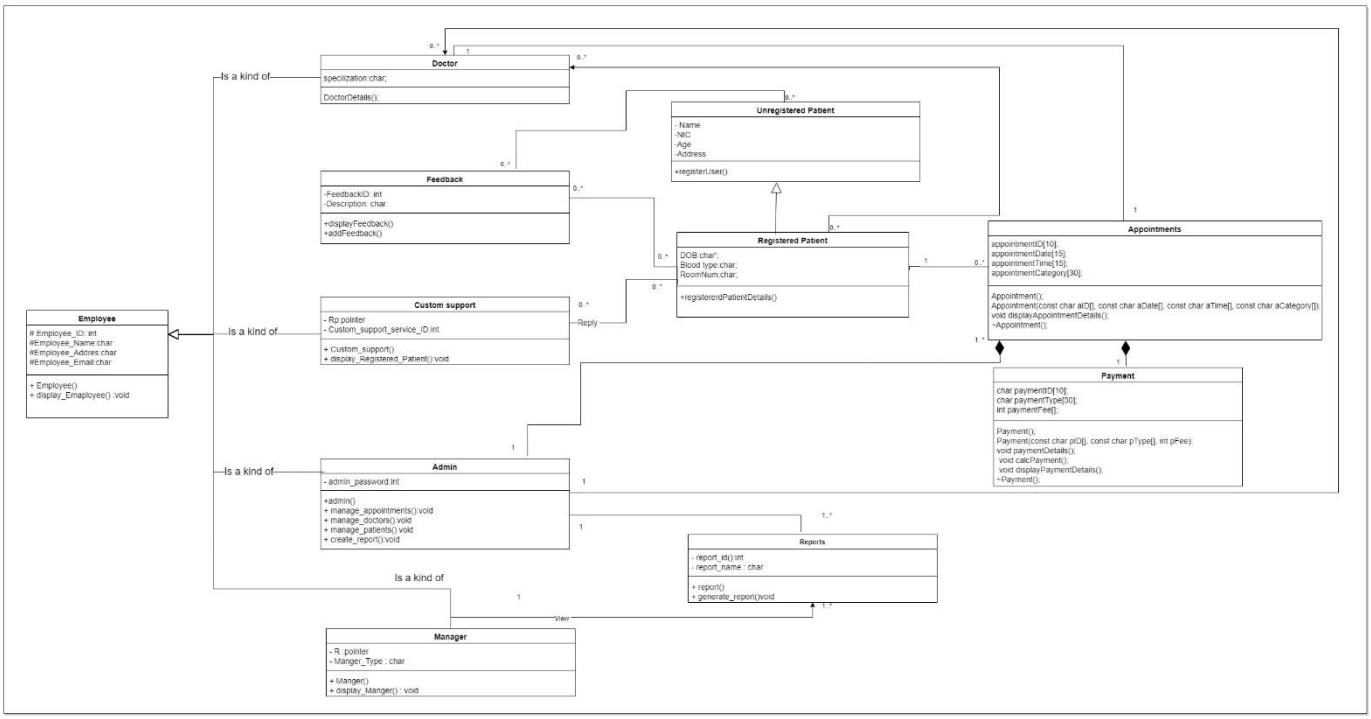
Class: Reports	
Responsibility	**Collaborators**
Generate transaction reports	Admin, Payment
List of appointment history	Appointment

Class: Feedback	
Responsibility	**Collaborators**
Get customer review	Registered Patient
Show feedbacks	

Class: Custom service	
Responsibility	**Collaborators**
Manage all customer questions	

Class: Payment	
Responsibility	**Collaborators**
Payment details	
Validate the payment	Admin
Display bill	

5. Class Diagram.



Draw.io Link – https://drive.google.com/file/d/183o1Im9v0C_2e8ojwSO-H2useBwBlk-F/view?usp=sharing

6. Codes.

Admin.h

```
#pragma once
#include<iostream>
using namespace std;

class ADMIN: Employee {
private:
	char AdminType[50];

public:
	void Admin();
	void Admin(char* Ename, int Eid, char* Eadd, char Email, char* atype);
	void manage_appointments();
	void manage_doctors();
	void manage_patients();
	void create_reports();
};
```

Admin.cpp

```
#include <iostream>
#include <cstring>
#include "Admin1.h"
#include "Employee.h"
#include "Appointment.h"
```

```
void ADMIN::Admin()
{
	strcpy_s(AdminType, "");
}

void ADMIN::Admin(char* Ename, int Eid, char* Eadd, char Email, char* atype)
{
	strcpy_s(AdminType, "atype");

}

void ADMIN::manage_appointments()
{
	cout << "Appointment ID : " << endl;
	cout << "Appointment category : " << endl;
}

void ADMIN::manage_doctors()
{
	cout << "Doctor ID : " << endl;
	cout << "Doctor Name : " << endl;
}

void ADMIN::manage_patients()
{
	cout << "Patient NIC : " << endl;
	cout << "Patient Name : " << endl;
}

void ADMIN::create_reports()
{
```

```
	cout << "Report ID : " << endl;
	cout << "Report Name : " << endl;
}
```

Admin_main.cpp

```
#include <iostream>
#include "Admin1.h"
#include "Appointment.h"
#include "Employee.h"
#include "registeredPerson.h"
#include "Report.h"
#include "Doctor.h"


using namespace std;

int main()
{
	ADMIN A1;

	A1.Admin("Chinthaka", "010", "Athurugiriya", "admin10@gmail.com",
"Securityt System Admin");
	cout << " *************************" << endl;

	Appointment Ap1;

	Ap1.manage_appointments();

	Doctor Do1;
```

```
	Do1.Doctor()


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


	RegisteredPerson rg1;


	rg1.RegisteredPerson();


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


	Report rp1;


	rp1.Report();


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


}
```

Report.h

```
#pragma once
#include <iostream>
class Report1 {
private:
	int Report_id;
	char Report_name[20];
```

```
public:
	void Report();
	void Report(int r_id, char r_name[]);
	void generate_Report();
};
```

Report.cpp

```
#include <iostream>
#include "Report1.h"
#include <cstring>
using namespace std;

void Report1::Report()
{

}

void Report1::Report(int r_id, char r_name[])
{
	int Report_id = r_id;
	strcpy(Report_name, r_name);
}

void Report1::generate_Report()
{
	cout << "Report ID : " << Report_id << endl;
	cout << "Report Name : " << Report_name << endl;
}
```

Report_main.cpp

```
#include <iostream>

#include <cstring>

#include "Report1.h"

#include "Report.h"

#include "Admin.h"

using namespace std;


int main()

{

	Report1 r1;


	r1.Report(10, (char*)"Payment Reports");

	r.generate_Report();


	cout << "\n" << endl;

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


}
```

unregistered.h

```
#pragma once
#include<iostream>
#include<cstring>

class UnregisterPatient {
	protected:
		char Name[30];
		int NIC;
		int Age;
		char Address[30];
	public:
		void UnregisteredPatient(){}
		void UnregisteredPatient(char* P_Name, int P_NIC, int P_Age, char*
P_Address);
		void registerUser();
};
```

unregistered.cpp

```
#include"unregistered.h"
#include<iostream>
#include<cstring>
using namespace std;

void UnregisterPatient::UnregisteredPatient() {
	strcpy_s(Name,"");
	NIC = 0;
	Age = 0;
	strcpy_s(Address,"");
}

void UnregisterPatient::UnregisteredPatient(char* P_Name, int P_NIC, int P_Age,
char* P_Address) {
	strcpy_s(Name, P_Name);
	NIC = P_NIC;
	Age = P_Age;
	strcpy_s(Address,P_Address);
}
void UnregisterPatient::registerUser() {
	cout << "Name" << endl;
	cout << "NIC" << endl;
	cout << "Age" << endl;
	cout << "Address" << endl;
}
```

Unregistered_main.cpp

```
#include"unregistered.cpp"
#include<iostream>
#include<cstring>
using namespace std;

int main() {
	UnregisterPatient U1;
	U1.UnregisteredPatient("Oshadhi", 200074000423, 24, "113/18,Peradeniya
road,Kandy");
	U1.registerUser();

		return 0;
};
```

feedback.h

```
#pragma once
#include<iostream>
#include<cstring>
using namespace std;

class Feed {
private:
	int FeedbackID;
	char Description[50];
	Feed* feedback[SIZE];
public:
	void Feedback();
	void Feedback(int F_ID, char* Des);
	void displayFeedback();
};
```

feedback.cpp

```
#include"feedback.h"
#include<iostream>
#include<cstring>
using namespace std;

void Feed::Feedback() {
	FeedbackID = 0;
	strcpy_s(Description, "");
}
void Feed::Feedback(int F_ID, char* Des) {
	FeedbackID = F_ID;
	strcpy_s(Description, Des);
}
void displayFeedback(){}
```

Feedback_Main.cpp

```
#include"feedback.cpp"
#include<iostream>
#include<cstring>
using namespace std;
```

Employee.h

```
#pragma once
#include <iostream>
#include <cstring>

class Employee
{
protected:
    int Employee_ID;
    char Employee_Name[30];
    char Employee_Addres[40];
    char Employee_Email[50];

public:
    Employee() {}
    Employee(int Eid, char *Ename, char *Eadd, char Email);
    void display_Emaployee();
};
```

Employee.cpp

```
#include <iostream>
#include <cstring>
#include "Employee.h"
using namespace std;

Employee::Employee()
{
    Employee_ID = 0;
    strcpy_s(Employee_Name, "");
    strcpy_s(Employee_Addres, "");
    strcpy_s(Employee_Email, "");
}

Employee::Employee(int Eid, char *Ename, char *Eadd, char Email)
{
    Employee_ID = Eid;
    strcpy_s(Employee_Name, "Ename");
    strcpy_s(Employee_Addres, "Eadd");
    strcpy_s(Employee_Email, "Email");
}

void Employee::display_Emaployee()
{
    cout << "Employee_ID" << endl;
    cout << "Employee_Name" << endl;
    cout << "Employee_Addres" << endl;
    cout << "Employee_Email" << endl;
}
```

Manger.h

```
#pragma once
#include <iostream>
#include <cstring>
using namespace std;


class Manger : public Employee
{
private:
    Report *R;
    char Manger_Type[20];

public:
    Manger(){};
    Manger(int Eid, char *Ename, char *Eadd, char Email, Report *repo, char
*M_type);
    void display_Manger();
};
```

Manger.cpp

```
#include <iostream>
#include <cstring>
#include "Employee.h"
#include "Manger.h"
#include "Report.h"
using namespace std;



Manger::Manger() : Employee()
{
    strcpy_s(Manger_Type, "");
    R = 0;
}

Manger::Manger(int Eid, char *Ename, char *Eadd, char Email, Report *repo, char
*M_type) : Employee(Eid, Ename, Eadd, Email)
{
    strcpy_s(Manger_Type, "M_type");
    R = repo;
}

void Manger::display_Manger()
{
    display_Emaployee();
    R->displayReport();
    cout << Manger_Type << endl;
}
```

Custom_support.h

```
#pragma once
#include <iostream>
#include <cstring>


class Custom_support : public Employee
{
private:
    Registered_Patient *Rp;
    int Custom_support_service_ID;

public:
    Custom_support();
    Custom_support(int Eid, char *Ename, char *Eadd, char Email,
Registered_Patient *reji_p, int cssi);
    void display_Registered_Patient();
};
```

Custom_support.cpp

```
#include <iostream>
#include <cstring>
#include "Employee.h"
#include "Custom_support.h"
#include "Registered_Patient.h"
using namespace std;

Custom_support::Custom_support() : Employee()
{
    Custom_support_service_ID = 0;
    Rp = 0;
}

Custom_support::Custom_support(int Eid, char *Ename, char *Eadd, char Email,
Registered_Patient *reji_p, int cssi) : Employee(Eid, Ename, Eadd, Email)
{
    Custom_support_service_ID = cssi;
    Rp = reji_p;
}

void Custom_support::display_Registered_Patient()
{
    display_Emaployee();
    Rp->displayReport();
    cout << Custom_support_service_ID << endl;
}
```

```
#include <iostream>
#include <cstring>
#include "Employee.h"
#include "Custom_support.h"
#include "Registered_Patient.h"
#include "Manger.h"
using namespace std;

    int main()
    {
        Employee E;
        E.Employee("E1", "Ishara", "Knady", "Ishara@gmail.com");
        E.display_Emaployee;
        cout << " *************************" << endl;
        Manger *M = new Manger("M1", "Ishara", "Knady", "Ishara@gmail.com", R,
"Gread A");
        M->display_Manger();
        cout << " *************************" << endl;
        delete M;
        Custom_support * 01 = new Custom_support("C001", "Tharindu", "Colambo",
"tharindu@gmail.com", RP1, "CSSI01");
        return 0;
    }
```

```
	Feed F1;
	F1.Feedback(001, "Good");
	F1.displayFeedback();

return 0;
};
```

Payment.h

```
class Payment {
private:
	//Payment();
	char paymentID[10];
	char paymentType[30];
	int paymentFee[];

	//commission* com;


public:
	Payment();
	Payment(const char pID[], const char pType[], int pFee);
	void paymentDetails();
	void calcPayment();
	void displayPaymentDetails();

	~Payment();

};
```

Payment.cpp

```
#include <iostream>
#include<cstring>
#include "payment.h"
using namespace std;

Payment::Payment() {

	strcpy_s(paymentID, "");
	strcpy_s(paymentType, "");
	int paymentFee = 0;

}
Payment::Payment(const char pID[], const char pType[], int pFee) {

	strcpy_s(paymentID, pID);
	strcpy_s(paymentType, pType);
	int paymentFee = pFee;
}

void Payment::paymentDetails() {

}
```

```
void Payment::calcPayment() {

}

void Payment::displayPaymentDetails() {

}

Payment:: ~Payment() {
	//Destructor
}
```

Payment_Main.cpp

```
#include "Payment.h"
#include <iostream>
using namespace std;

int main()
{
	Payment *pay;
	pay = new Payment();

	//method calling
	pay -> displayPaymentDetails();
	pay -> calcPayment();

	//delete dynamic object
	delete pay;
return 0;

}
```

Appointment.h

```
class Appointment
{
Private:
	char appointmentID[10];
	char appointmentDate[15];
	char appointmentTime[15];
	char appointmentCategory[30];

public:
	Appointment();
	Appointment(const char aID[], const char aDate[], const char aTime[],
const char aCategory[]);

	void displayAppointmentDetails();
	~Appointment();

};
```

Appointment.cpp

```
#include "Appointment.h"
#include <cstring>
#include <iostream>
using namespace std;

Appointment::Appointment() {

	strcpy_s(appointmentID, "");
	strcpy_s(appointmentDate, "");
	strcpy_s(appointmentTime, "");
	strcpy_s(appointmentCategory, "");

}

Appointment::Appointment(const char aID[], const char aDate[], const char
aTime[], const char aCategory[]) {

	strcpy_s(appointmentID, aID);
	strcpy_s(appointmentDate, aDate);
	strcpy_s(appointmentTime, aTime);
	strcpy_s(appointmentCategory, aCategory);

}

void Appointment::displayAppointmentDetails() {

}

Appointment :: ~Appointment() {
	//destructer
}
```

Appointment _Main.cpp

```
#include "Appointment.h"

#include <iostream>
using namespace std;

int main()
{
	Appointment* appointment;
	appointment = new Appointment();

	//method calling
	appointment->displayAppointmentDetails();

	//delete dynamic object
	delete appointment;

return 0;
}
```

Doctor.h

```
#include "Employee.h"

#include <string>
class Doctor:public Employee {
private:
	string specialization;
public:
	Doctor();
	Doctor(int Eid, string Ename, string Eadd, string Email, string spec);
	void displayDetails();
};
```

Doctor.cpp

```
#include <iostream>
#include "Doctor.h"
#include <string>
Using namespace std;

Doctor::Doctor(){
	Employee_Name = "default";
	Employee_Addres = "default";
}
Doctor::Doctor(int Eid, string Ename, string Eadd, string Email, string spec) {

	Employee_ID = Eid;
	Employee_Name = Ename;
	Employee_Addres = Eadd;
	Employee_Email = Email;
	specialization = spec;
}
void Doctor::displayDetails() {

	cout << "Doctor ID: " << Employee_ID << endl;
	cout << "Doctor Name: " << Employee_Name << endl;
	cout << "Doctor Address: " << Employee_Addres << endl;
	cout << "Doctor Email: " << Employee_Email << endl;
	cout << "Doctor Specialization: " << specialization << endl;
}
```

Doctor_Main.cpp

```
#include <iostream>
#include "Doctor.h"
#include <string>

int main() {

	Doctor* d1;
	d1 = new Doctor(001, "max", "colombo", "max@gmail.com", "dentist");
	d1 -> displayDetails();
	delete d1;
	return 0;
}
```

RegisteredPatient.h

```
class RegisteredPatient {
protected:
	string DOB;
	string bloodType;
	string roomNum;
```

```
	Doctor* dr;
public:
	RegisteredPatient();
	RegisteredPatient(string date, string btype, string room);
	void displayAll();
};
```

RegisteredPatient.cpp

```
#include "RegisteredPatient.h"
#include <string>
using namespace std;


RegisteredPatient::RegisteredPatient() {

	DOB = "12-12-2012";
	bloodType = "O positive";
	roomNum = "B202";
}

RegisteredPatient::RegisteredPatient(string date, string btype, string room) {

	DOB = date;
	bloodType = btype;
	roomNum = room;
}
void RegisteredPatient::displayAll() {

	cout << DOB << endl;
	cout << bloodType << endl;
	cout << roomNum << endl;

}
```

RegisteredPatient_Main.cpp

```
#include "RegisteredPatient.h"

#include <string>



int main() {

	Doctor* d1;
	d1 = new Doctor(001, "max", "colombo", "max@gmail.com", "dentist");
	d1->displayDetails();
	delete d1;

	cout << endl;

	RegisteredPatient* rp;
	rp = new RegisteredPatient("2-03-2021", "O+", "002");
	rp->displayAll();
```

```
 return 0;
}
```

7. Contribution.

H.C.K.Ariyarathna IT21176388 – Admin & Report Parts

H.A.N Nilakshana IT21175602 - Appointment , Payment Parts

H.M.S Migara IT21173486 - Registered Patients, Doctor Parts

D.M.M.I.T Dissanayaka IT21174780 – Employee , Custom Support, Manager Parts

Kumarasinghe O.A IT21174308 - Feedback , Unregistered patients Parts