Topic : Online bill and Event Reminder

Group no : MLB_08.01_05

Campus : Malabe

Submission Date: 2022/05/12

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
IT21165016	Kawya R M S	078 289 4539
IT21164408	Warunika H P R	077 561 4439
IT21165498	Perera W H T H	077 866 2408
IT21166310	Kumara W H T S	077 894 4327
IT21164262	Gamage T G S N	078 992 1868

Requirements Specification

1) In an online bill and event reminder system it has two types of users, one is new user, and the other is registered user.
2) A registered user or new user both can visit online bill and event reminder system by using our website.
3) A new user needs to register to the system by providing details such as name, address, Email, phone number, DOB, country, username, and the password.
4) A registered user can add event reminders for their special occasions after providing date, time, email, and description.
5) They can also add bill reminders for their bill payment by adding date, bill type, bill amount, email, and description.
6) A registered user can add, edit, or delete their reminders and review added reminders.
7) User can get notification after adding reminders.
8) Our system provides 3 subscription plans for different amounts and different time durations.
9) User can purchase one subscription plan and get notification via email and SMS.
10) Users can pay for subscription using debit card, credit card and PayPal.
11) Users can rate and write feedbacks about our system.
12) Users can check for the latest and upcoming public events.
13) If users need any other help, they can refer FAQs.
14) Users have any doubt of our webpage they can contact us.
15) The system will record event reminders, bill reminders and user details.
16) The system admin can update the upcoming public events, validate payment, maintain, and update the system.

Identified classes

1. Guest user
2. Registered user
3. Event reminder
4. Bill reminder
5. Public event
6. Payment
7. Feedback
8. Subscription
9. Report

EXERCISE 2

CRC Cards

Class name: Guest user	
Responsibilities:	**Collaborations:**
Register details	
Search event	Public Event

Class name: Registered user	
Responsibilities:	**Collaborations:**
Store User Details	
Modify details	
Subscribe to the system	Subscription
Make payment	Payment
Send feedback	Feedback
Set reminder	Event reminder, bill reminder
Delete reminder	Event reminder, bill reminder
Review added reminders	Report
Search event	Event reminder, bill reminder, public event

Class name: Event Reminder	
Responsibilities:	**Collaborations:**
Add event reminder	

Update event reminder	
Store reminder details	

Class name: Bill Reminder	
Responsibilities:	**Collaborations:**
Add bill details	
Update bill details	
Store reminder details	

Class name: Payment	
Responsibilities:	**Collaborations:**
Store payment details	
Validate	

Class name: Feedback	
Responsibilities:	**Collaborations:**
Send feedbacks	Registered user
Store feedback details	

Class name: Public Event	
Responsibilities:	**Collaborations:**

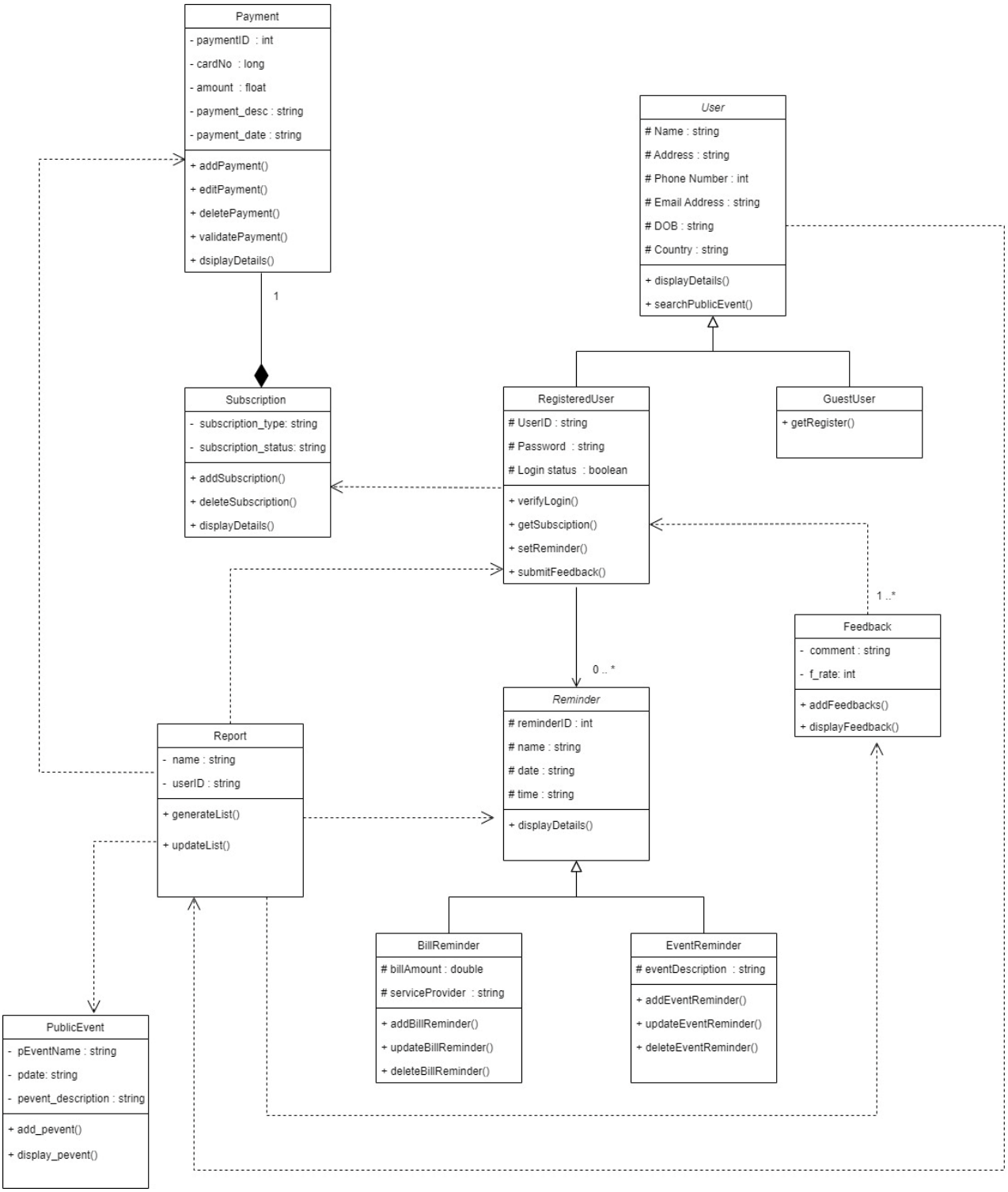
Store public event details	
Add public event	
Update public event	
Delete public event	

Class name: Subscription	
Responsibilities:	**Collaborations:**
Add subscription	
Confirm subscription	Payment
Expire subscription	

Class name: Report	
Responsibilities:	**Collaborations:**
List of added reminders	Event reminder, bill reminder, registered user
List of public events	Public event
List of previous subscriptions	Subscription
List of feedbacks	Feedback
List of payments	Payment

EXERCISE 3

Class diagram



EXERCISE 4

Coding

```
/*-----------------User Class------------*/

using namespace std;

#include <string>


//base class

class User{

protected:

  string name;

  string address;

  int mobileNo;

  string email;

  string dob;

  string country;


public:

  User();


  User(string uname,string uaddress,int umobileNo,string uemail,string
udob,string ucountry);


  //this is a abstract class.
```

```
  virtual void displayDetails() = 0;

  //dependecy relationship with Report Class
  void searchPublicEvent(string name, Report *r);

};



/*-----------------User Class Implementation------------*/
#include "User.h"
#include <string>

User::User(){
    name = "No value";
    address = "No value";
    mobileNo = 0000000000;
    email = "No value";
    dob = "No value";
    country = "No value";
  }

User::User(string uname,string uaddress,int umobileNo,string uemail,string
udob,string ucountry){
  name = uname;
```

```
  address = uaddress;
  mobileNo = umobileNo;
  email = uemail;
  dob = udob;
  country = ucountry;
}

void User::searchPublicEvent(string name, Report *r){

}



/*-----------------Guest_user Class------------*/

//derived class from User class
#include "User.h"

class Guest_user : public User{
public:
Guest_user();

Guest_user(string gname,string gaddress,int gno,string gmail,string gdob,string
gcountry);
```

```
void displayDetails();

~Guest_user();

};

/*-----------------Guest_user Class Implementation------------*/
#include <iostream>
using namespace std;
#include "Guest_user.h"

Guest_user::Guest_user(){
  name = "No value";
  address = "No value";
  mobileNo = 0000000000;
  email = "No value";
  dob =  "No value";
  country = "No value";
}

Guest_user::Guest_user(string gname,string gaddress,int gno,string
gmail,string gdob,string gcountry){
    name = gname;
    address = gaddress;
```

```
    mobileNo = gno;
    email = gmail;
    dob = gdob;
    country = gcountry;
}

//overriding displayDetails function
void displayDetails(){
  cout << name <<endl << address <<endl << mobileNo <<endl << email <<endl
<< dob <<endl << country << endl;
  }

Guest_user::~Guest_user(){

  cout << "Destructor called for Guest_user object" << endl;
}

/*-----------------Registered_user Class------------*/

//derived class from User class
using namespace std;
#include <string>
#include "User.h"
#define SIZE 5
```

```
class Registered_user : public User{

protected:

  string userID;

  string password;

  bool login_status;

  Reminder *r[SIZE]; //Uni-directional association relationship with Reminder class

public:

  Registered_user();


 Registered_user(string rname,string raddress,int rno,string rmail,string rdob,string rcountry,string id, string pw,bool logstatus);


  void verify_login(string userID,string password);


  void getSubscription(Subscription *s);  //dependecy relationship with Subscription Class


  void displayDetails();


 ~Registered_user();

};
```

```
/*----------------- Registered_user Class Implementation----------*/

#include <iostream>

using namespace std;

#include "Registered_user.h"

#include <string>



Registered_user::Registered_user()

{

   name = "No value";

   address = "No value";

   mobileNo =  0000000000;

   email = "No value";

   dob = "No value";

   country = "No value";

   userID = "No value";

   password = "No value";

   login_status = true;


}

//implementation

void displayuserdetails(){};

Registered_user::Registered_user(string rname,string raddress,int rno,string
rmail,string rdob,string rcountry,string id, string pw,bool logstatus){
```

```
   name = rname;
   address = raddress;
   mobileNo = rno;
   email = rmail;
   dob = rdob;
   country = rcountry;
   userID = id;
   password = pw;
   login_status = logstatus;
  }

 void Registered_user::verify_login(string userID,string password){

 }

void Registered_user::getSubscription(Subscription *s){

}

//overriding displayDetails function
 void Registered_user::displayDetails(){
  cout << name <<" "<< address <<" "<< mobileNo <<" "<< email <<" "<< dob
<<" "<< country <<" "<< userID     <<" "<< password <<" "<< login_status <<
endl;
```

```
}

 Registered_user::~Registered_user(){
   cout << "Destructor called for Registered_user object" << endl;
 }

//Reminder class
class Reminder{
    protected:
          string name;
          string date;
          string time;

    public:
       //default constructor
         Reminder(){
           name="None";
           date="None";
           time="None";
         }
        //Overloaded constructor
         Reminder(string Rname,string Rdate,string Rtime){
           name=Rname;
           date=Rdate;
```

```
        time=Rtime;
      }
      virtual void displayDetails(){
        displayDetails();
      }

};
//BillReminder class
class BillReminder:public Reminder{
    protected:
      double billAmount;
      string serviceProvider;

    public:
       //default constructor
        BillReminder(){
        billAmount=0;
        serviceProvider="None";
        }
       //Overloaded constructor
        BillReminder(string Rname,string Rdate,string Rtime,double
BbillAmount,string BserviceProvider):Reminder(Rname,Rdate,Rtime){
          billAmount=BbillAmount;
          serviceProvider=BserviceProvider;
```

```
    }
    void displayDetails(){
      cout<<"Bill name          :"<<name<<endl;
      cout<<"Bill Reminder Date   :"<<date<<endl;
      cout<<"Bill reminder time   :"<<time<<endl;
      cout<<"Bil Amount          :"<<billAmount<<endl;
      cout<<"Service provider name:"<<serviceProvider<<endl;
    }
    void addBillReminder();
    void updateBillReminder();
    void deleteBillReminder();
    ~BillReminder();//default destructor
};
//EventReminder class
class EventReminder:public Reminder{
    protected:
      string eventDescription;

    public:
      //default constructor
      EventReminder(){
      eventDescription="None";
    }
```

```
        //Overloaded constructor
      EventReminder(string Rname,string Rdate,string Rtime,string
EeventDescription):Reminder(Rname,Rdate,Rtime){
         eventDescription=EeventDescription;
      }
      void displayDetails(){
        cout<<"Event name        :"<<name<<endl;
        cout<<"Event Reminder Date:"<<date<<endl;
        cout<<"Event reminder time:"<<time<<endl;
        cout<<"Event Description  :"<<eventDescription<<endl;
       }
      void addEventReminder();
      void updateEventReminder();
      void deleteEventReminder();
      ~EventReminder(); //default destructor

};
//public event class
class PublicEvent
{
private:
  string pEventName;
  string pdate;
  string pevent_description;
```

```
public:
PublicEvent () {}
PublicEvent (string U_pEventName, string U_pdate, string
U_peventdescription) {
pEventName=U_pEventName;
pdate=U_pdate;
pevent_description=U_peventdescription;
~PublicEvent ();
}
 //implementation
void displaypublicevent (User*u) {
cout<< "Public Event Default constructor called " << "\n" <<endl;
}
{
 pEventName = User*u.
cout <<pEventName<< endl;
cout << pdate << endl;
cout<<pevent_description<<"\n"<<endl;
}
};

{
private:
 string  Comment;
```

```
  int f_rate;
public:
Feedback(){}
Feedback(string U_comment,int U_frate){
Comment=U_comment;
f_rate=U_frate;
}
void display()
{
cout<<"Feedback Default constructor called"<<"\n"<<endl;
cout << Comment<< endl;
cout << f_rate << endl;
}
};
class RegisteredUser
{
protected:
string userID;
string password;
public:
RegisteredUser(string UID,string U_PW){
userID= UID;
password =U_PW;
```

```
}
void display()
{
cout<<"Registered user Default constructor called"<<"\n"<<endl;
cout << "User ID: " <<userID<< endl;
cout << "User Password:" << password<<"\n"<< endl;
}
};
int main()
{
Feedback *f1;
f1= new Feedback( "Thank you for this experience!!",4);
RegisteredUser*R1;
R1= new RegisteredUser("ID10362678","12345");
R1->display();
f1->display();
}
```

```
#include <iostream>
#include <string>
using namespace std;

class Payment //creating Payment class
```

```
{
  private:
    int paymentID, cardNo; //declare variables
    float amount;
    string payment_desc, payment_date;

  public:
    Payment() ;//default constructor
    void addPayment() ; //define functions
    void editPayment() ;
    void deletePayment() ;
    void validatePayment() ;
    void displayDetails() ;
    ~Payment() ; //default destructor
};

class Subscription //creating Subscription class
{
  private:
    Payment * payment; //composition relationship with Payment class
    string subscription_type, subscription_status; //declare variables

  public:
```

```
    Subscription() ; //default constructor
    void addSubscription() ; //define functions
    void deleteSubscription() ;
    void displayDetails() ;
    ~Subscription(); //default destructor
};

Payment :: Payment() //default constructor of Payment class
{
          paymentID = 0;
  payment_date = "Not Entered";
  payment_desc = "Not Entered";
  amount = 0;
}

void Payment :: displayDetails()
{
  cout << "Payment ID  : " << paymentID << endl;
  cout << "Date        : " << payment_date << endl;
  cout << "Description : " << payment_desc << endl;
  cout << "Amount      : " << amount << endl;
}
```

```
Payment :: ~Payment() //default destructor of Payment class
{
  cout << "Deleting payment " << paymentID << endl;
}

Subscription :: Subscription() //default constructor of Subscription class
{
        payment = new Payment;
        subscription_type = "Not Entered";
        subscription_status = "Not Entered";
}

void Subscription :: displayDetails()
{
  cout << "Subscription type   : " << subscription_type << endl;
  cout << "Subscription status : " << subscription_status << endl;
}

Subscription :: ~Subscription() //default destructor of Subscription class
{
  cout << "Deleting subscription" << endl;
        delete payment;
}
```

```
/*----------------- Report_ Class ------------*/

#include<iostream> //header file
#include<string.h> //string header file
using namespace std;

//class declaration
class Report {
   //attributes
private:
   string name;
   string userID;

   //methods
public:
   Report ();
   Report (string Rname, string RuserID);
   void generateList();
   void updateList();

};
```

```
void listOfaddreminders(){}
void ListOfPublicEvent(){}
void listofFeedback(){}
void listofPayment(){}

//default constructor
Report::Report()
{
    name = "";
    userID = "";

}

//overload constructor
Report::Report(string nname, string uuserID)
{
   name=nname;
   userID=uuserID;

}
void Report::generateList() {}
void Report::updateList() {}
```

```
//begin of the main program


int main()
{
        Registered_user *r = new Registered_user();
        r->searchPublicEvent("Vesak", rep);
        r->displayDetails();
        delete r;


        Reminder *b1,*b2;
        b1=new BillReminder();
        b1->displayDetails();
        cout<<"================================================"<<en
        dl;
        b2=new BillReminder("Electricity
        bill","2022/02/22","07.50pm",7500,"CEB");
        b2->displayDetails();
        cout<<"================================================"<<en
        dl;
        Reminder *e1,*e2;
        e1=new EventReminder();
        e1->displayDetails();
        cout<<"================================================"<<en
        dl;
```

```
e2=new EventReminder("Birthday
Party","2022/11/06","12.00am","Hiruni's 21st birth day");
e2->displayDetails();

 delete b1;
 delete e1;
 delete b2;
 delete e2;

Subscription subscription1; //create objects for Subscription class
Payment payment1; //create objects for Payment class

payment1.displayDetails();
 subscription1.displayDetails();

   //create objects
   Report r1, r2, r3;

   //calling generateList() method
   r1.generateList();
   r1.generateList();
   r3.generateList();
```

```
        //calling updateList() method
        r1.updateList();
        r2.updateList();
        r3.updateList();


        return 0; // end of the main program
}
```