CPIT252 Project

iCare

Supervised by: Fatmah Bamashmoos

Prepared by:

Group project members:

Name	ID	Section
Shuruq Hassan Baabdullah	1906284	CAR
Rahaf Mutaz Dawoud	1911088	CAR
Elaf Yousef Aloufi	1911265	CAR
Manar Mutlaq Altaiary	1906775	CAR
Wejdan Fawaz Aljabarti	1905144	EAR

Table Of Content

Abstract 2

Introduction 3

Business Scenario 4

Functional and non- Functional requirements 4

Functional requirements 4

User requirements **4**

System Requirements **5**

Non-functional requirements 5

Users view in iCare system 6

Nurses and Specialists View 6

Patients View 6

Dynamic class diagram of iCare application 7

The implementation parts 8

Singleton design pattern 8

Builder design pattern 9

Decorator design pattern 10

Facade design pattern 11

Observer design pattern 12

Conclusion 13

Appendix 14

Use case diagram 14

Prototype of your system (GUI) 15

Codes of the classes 25

Classes package 25

User class **25**

Patient class **28**

Medical employee class **28**

Appointment class **29**

Admin class **31**

Design pattern package 33

Access interface **33**

Appointment builder class **33**

Decorator class(“BMI Feature”) **34**

Database class (“Singleton”) **35**

Observer interface **36**

subject interface **36**

Abstract

Due to the need of a large group of people, including the elderly, autistic patients, the disabled, and people with chronic diseases for daily care, we designed this application that contributes to solving this problem by allowing the user to book home care appointments or an assistant to help with achieving daily tasks by a nurse or specialist for a period of time in hours or days or weeks.

In this report, we will show how we implemented the solution by using NetBeans and java programming language and how we applied some design patterns in the application. In fact, the design patterns help promote easier program changes and object reusability.

Introduction

Nowadays, Older adults or people with autism, or people with chronic illnesses need regular home healthcare or assistance with daily tasks for a specified period, hours, days, weeks, or other by a nurse or specialist.

In addition, the appropriate solution is to create an application that aims to assist a large population, such as the elderly, children, and people with disabilities, by requesting a nurse or specialized assistant to complete daily tasks for the user for hours or days.

The application is implemented using NetBeans and java programming language and graphical user interfaces. Also, the coding parts were implemented by using some design patterns such as singleton for the database object.

Business Scenario

Initial assumption: the patient signs up for the system by entering his /her (first name, last name, phone number, SSN, Gender, Email, City, Age, Password, and choosing his or her type). Also, the medical employee can book an appointment.
Normal: The patient can book an appointment by choosing the type of medical employee he/she wants either nurse or a specialist.

Functional and non- Functional requirements

Functional requirements

User requirements

- Users can log in to the system if they do not have an account.
- Users can sign up for the system if they don't have an account.
- Patient can make his appointment upon his choices.
- Patients can choose the type of medical employees such as nurses and specialists and determine the appointment details.
- Patients can pay only by a credit card.
- The medical employees can add appointments.
- The medical employees and the patients can communicate with each other by an online chat.

System Requirements

- Users can browse the system only if they have an account.
- Patient must have an account to pay for his order.
- Account linked with the customer's email.
- System offers online payment.
- The email shall be like the correct format for an email
- The users first name and the last name shall not be empty
- The system shall check if the email is in the database. If the email is not the database, the system shall not let the user use the system but if the email is in the system, then it will check the password.
- The phone number must be only digits and 10 digits only
- The card number length for the credit card shall be in 16 digits only
- The month and year and CCV of the credit card shall be digits
- The cardholder's name shall not be empty

Non-functional requirements

- The system should be secure from any threads. The system shall protect the users information from any unauthorized access to the system.
- The system shall be available in 24 hours.
- The system is flexible and easy to use so that all types of customers can use the system without having any problems.
- The system should be able to maintain and update during the years.
- System should be capable of handling an infinite number of customers.

Users view in iCare system

After collecting the requirements of the application, here is the system functionality and the actors down below:

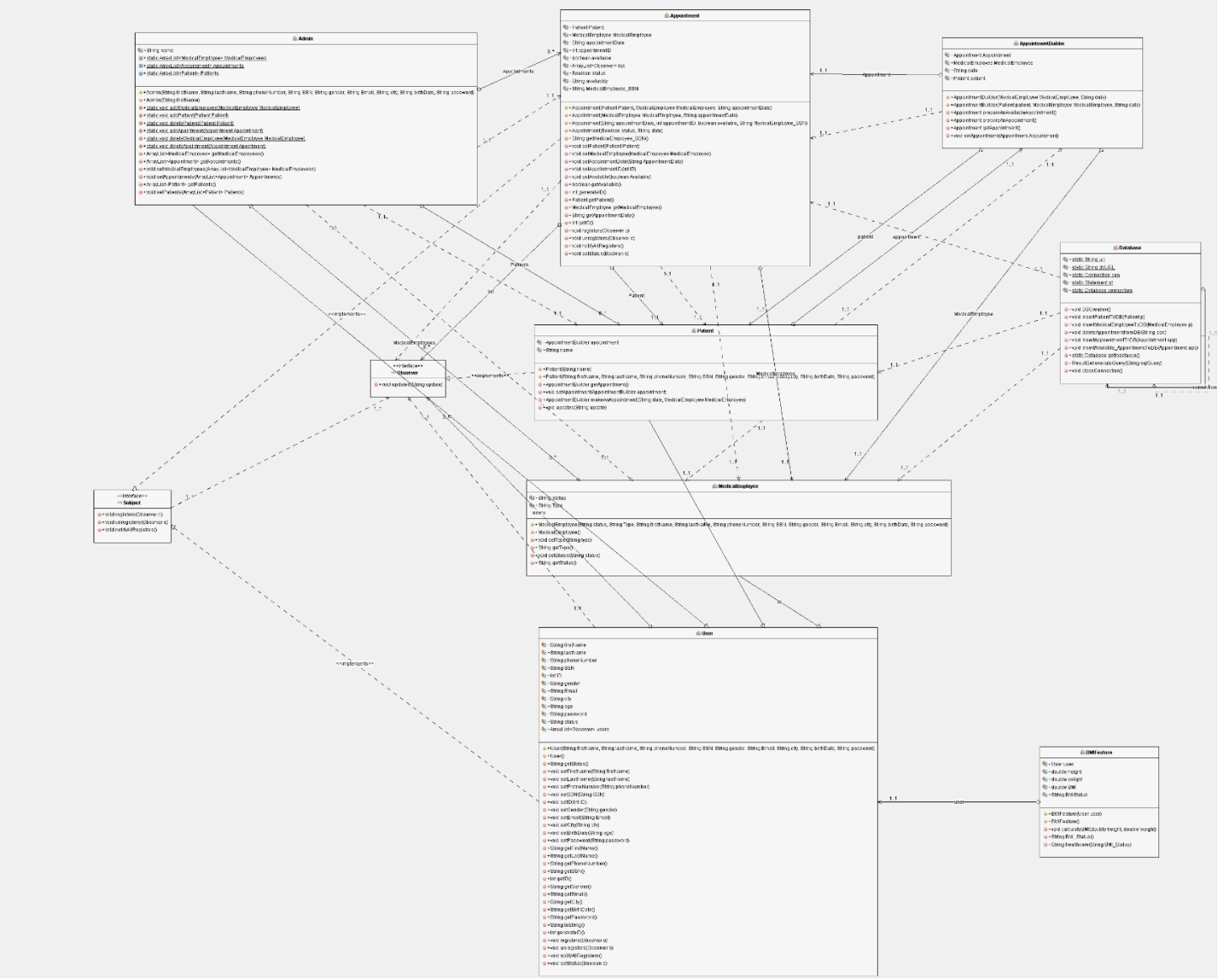
Nurses and Specialists View

1. Add available appointments
2. Accept appointment requests
3. Direct contact with patients (Chatting)

Patients View

1. Book appointments
2. Search for a specific assistant
3. Direct contact with assistants (Chatting)

Dynamic class diagram of iCare application



The implementation parts

In the iCare application, we used design patterns in the back end of the application. In fact, each design pattern is organized and made the coding easier and more reusable, and more efficient to work with. In fact, you can find below each definition of the design patterns that we used in the iCare application and the purpose of using it.

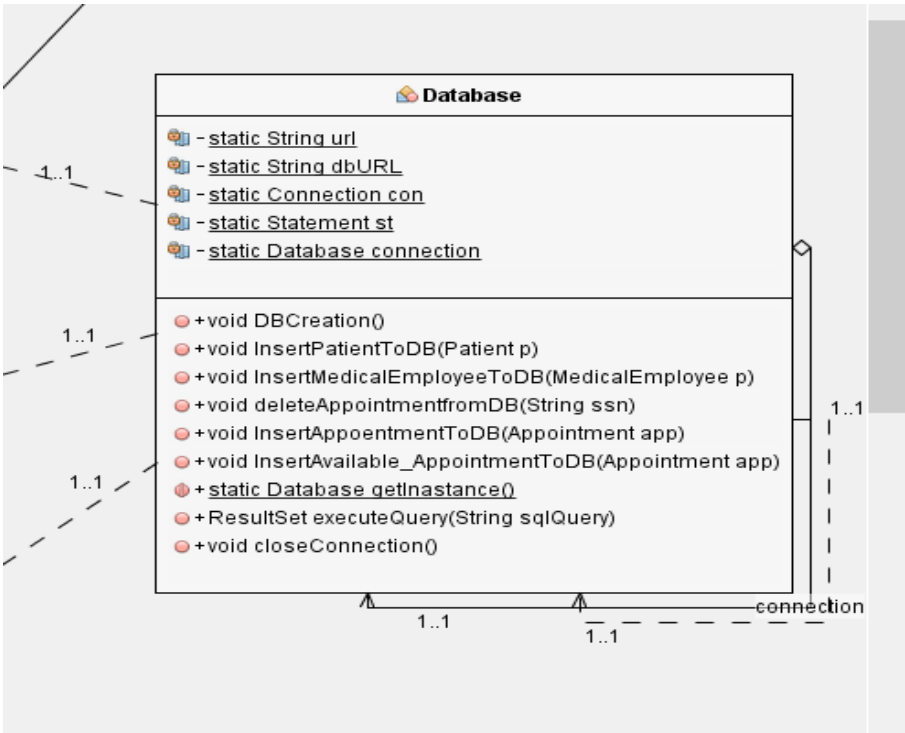
Singleton design pattern

What is the singleton design pattern?

singleton design is one of the simplest design patterns. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

What is the specific purpose of the singleton in iCare application?

In the iCare application, we used the singleton design pattern in creating only an object for accessing the database. This helps to secure the accessing of the database.



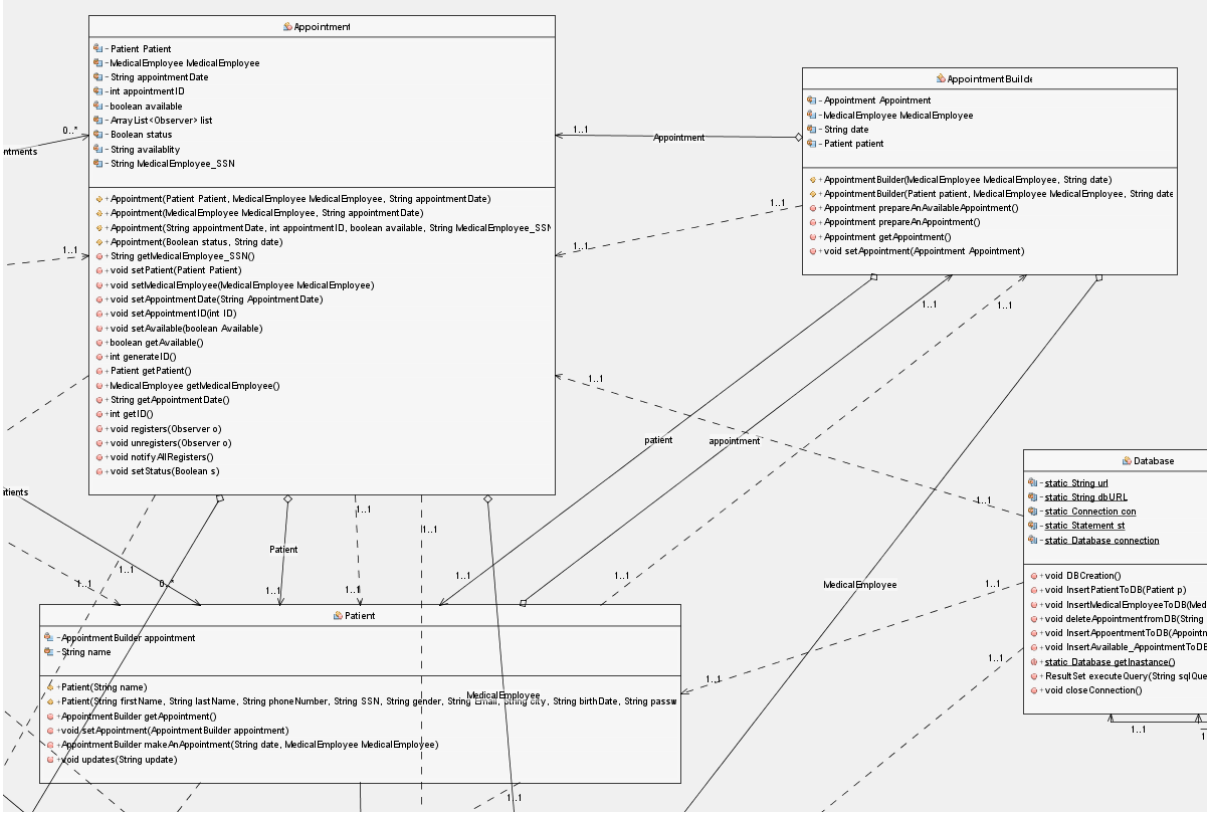
Builder design pattern

What is the builder creational design pattern?

The Builder Design Pattern is another creational pattern designed to deal with the construction of comparatively complex objects.

What is the specific purpose of the Builder in iCare application?

In the iCare application, we used the builder to build an appointment. The patient is responsible to build his appointment with his specific criteria. In fact, the appointment has details which means that the patient should choose his appointment details based on his choices by a builder class.



Decorator design pattern

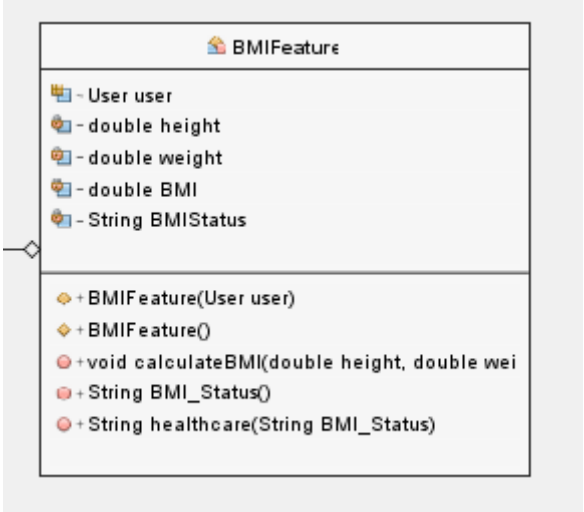
What is the decorator design pattern?

A Decorator pattern can be used to attach additional responsibilities to an object either statically or dynamically. In addition, the decorator provides an enhanced interface to the original object.

What is the specific purpose of the Decorator in iCare application?

Furthermore, there is an additional feature that was added in the iCare application and this feature works as a decorator of the application. Moreover, the feature is calculating body mass index which is an inexpensive and easy screening method for weight category (underweight, healthy weight, overweight, and obesity).

Also, BMI does not measure body fat directly, but BMI is moderately correlated with more direct measures of body fat.



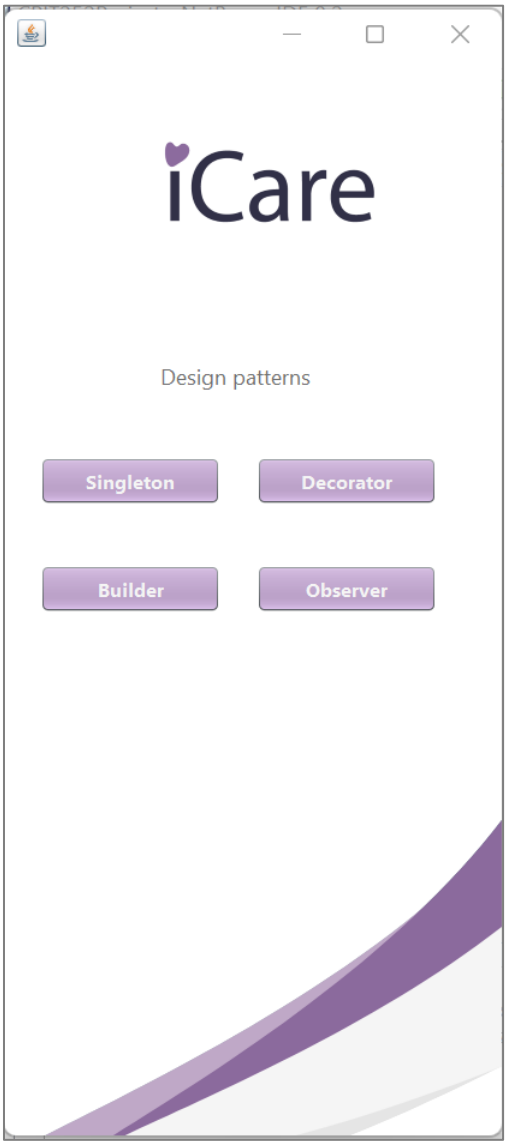
Facade design pattern

What is the facade design pattern?

Facade encapsulates a complex subsystem behind a simple interface.

What is the specific purpose of the facade in iCare application?

It hides much of the complexity and makes the iCare easy to use.



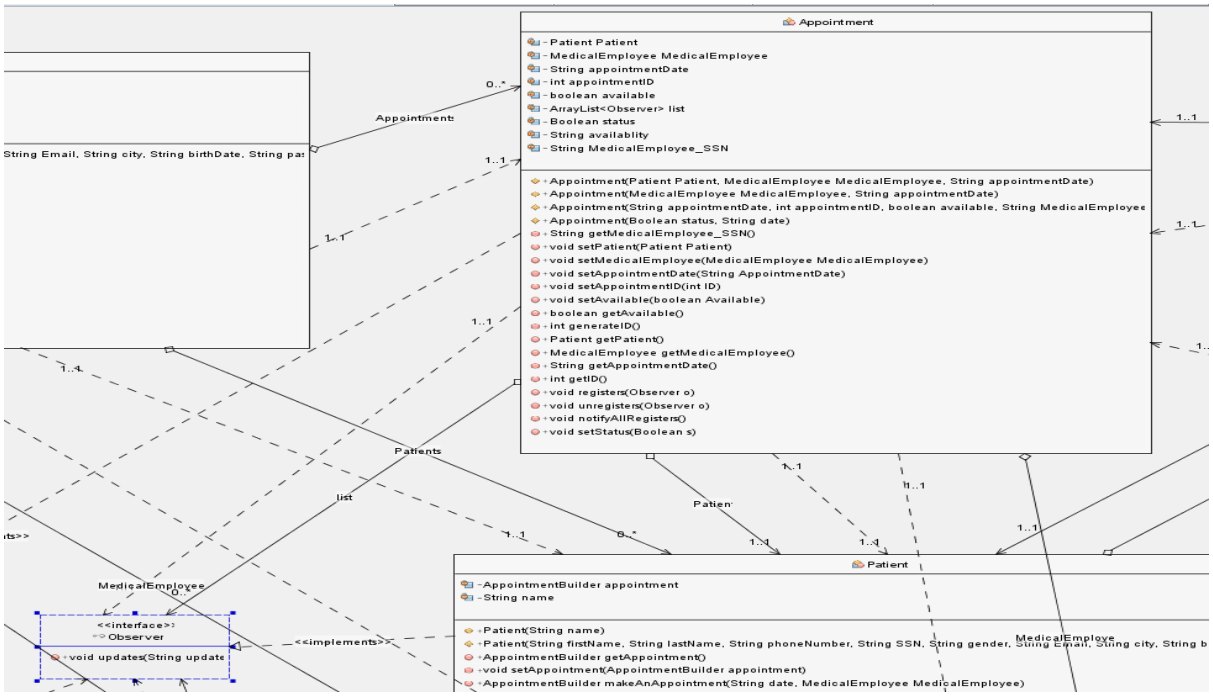
Observer design pattern

What is the observer design pattern?

Observer is a behavioural design pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

What is the specific purpose of the observer in iCare application?

iCare application can notify the patients about the changes in its states.



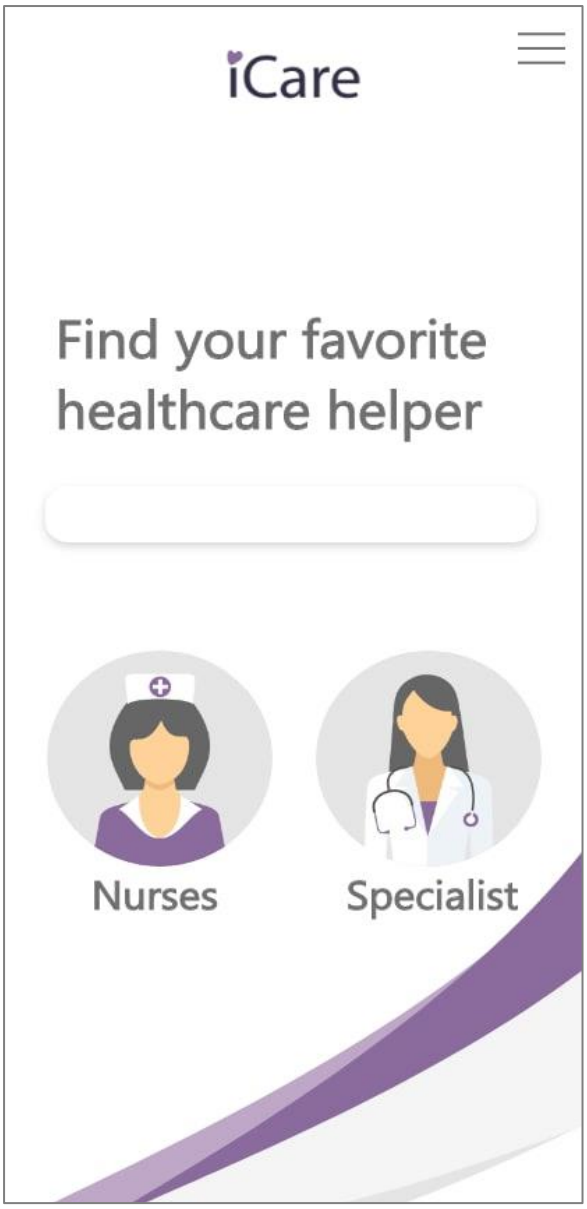
Conclusion

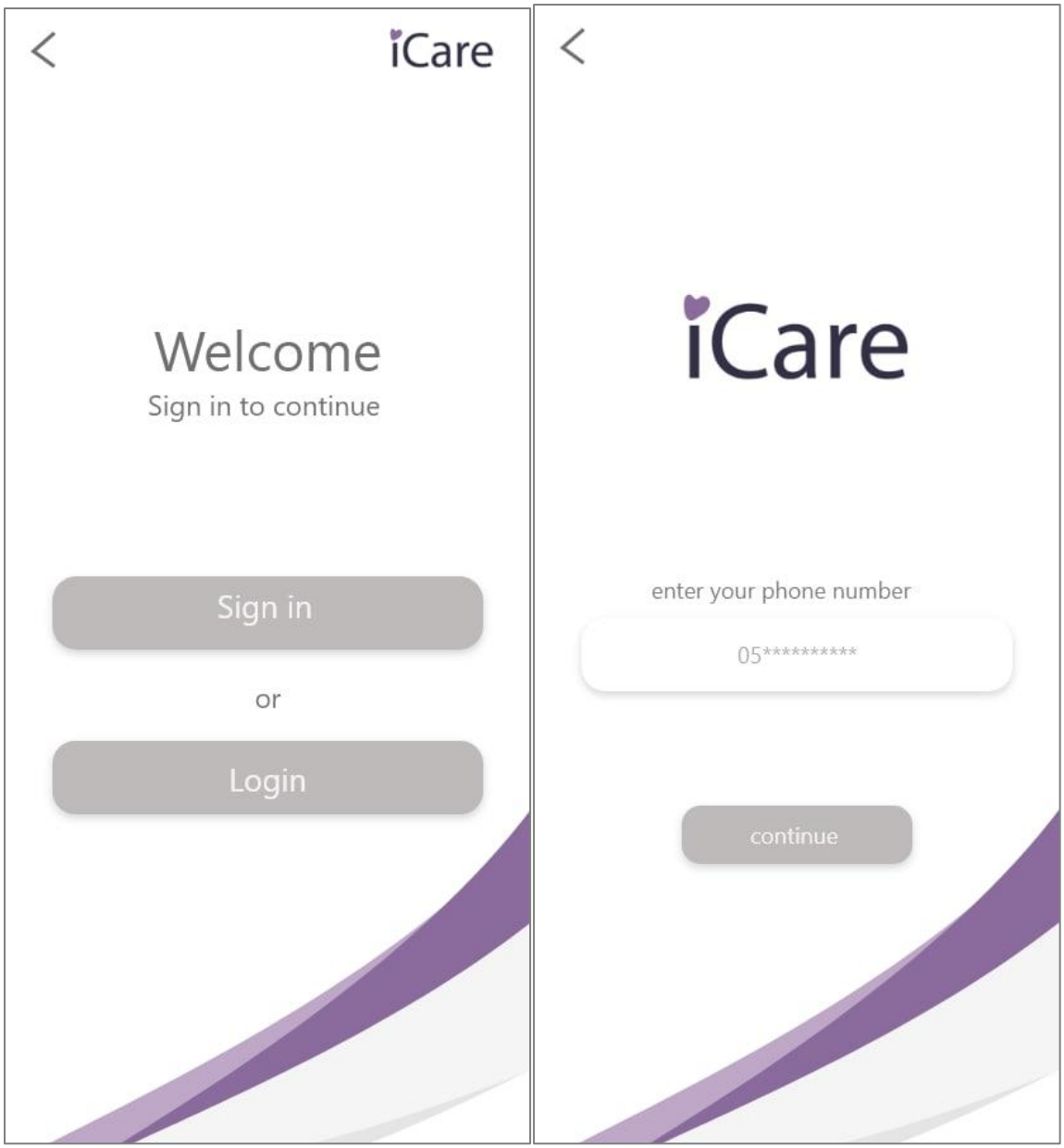
Finally, this report was written and designed to introduce our solution “iCare” to the problem of people who need frequent home healthcare or assistance with everyday duties from an expert for a certain length of time, hours, days, weeks, or months, which aims to provide an experience that saves user’s time and effort. We also seek in the future to add more ideas and develop this project by implementing the entire application instead of running only a few functions and adding new features.

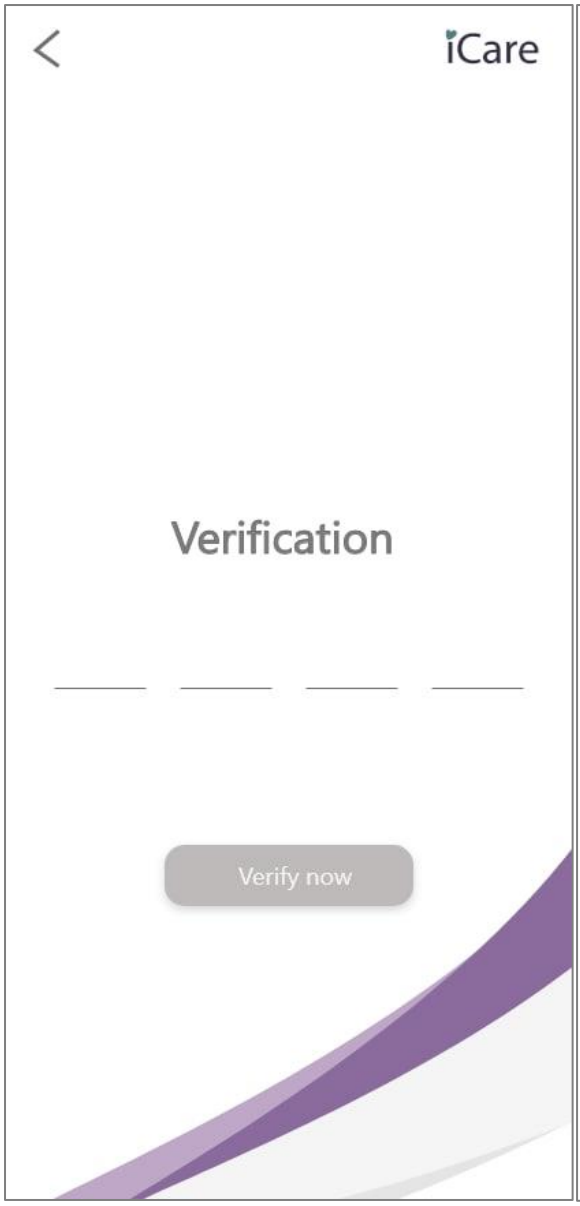
Appendix

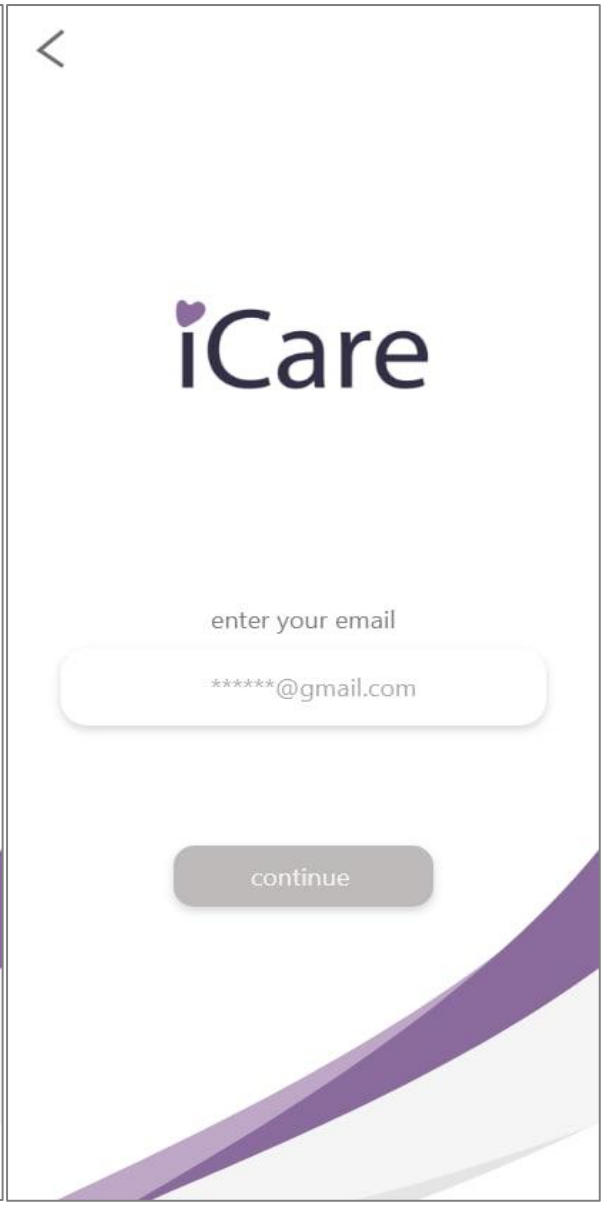
Use case diagram

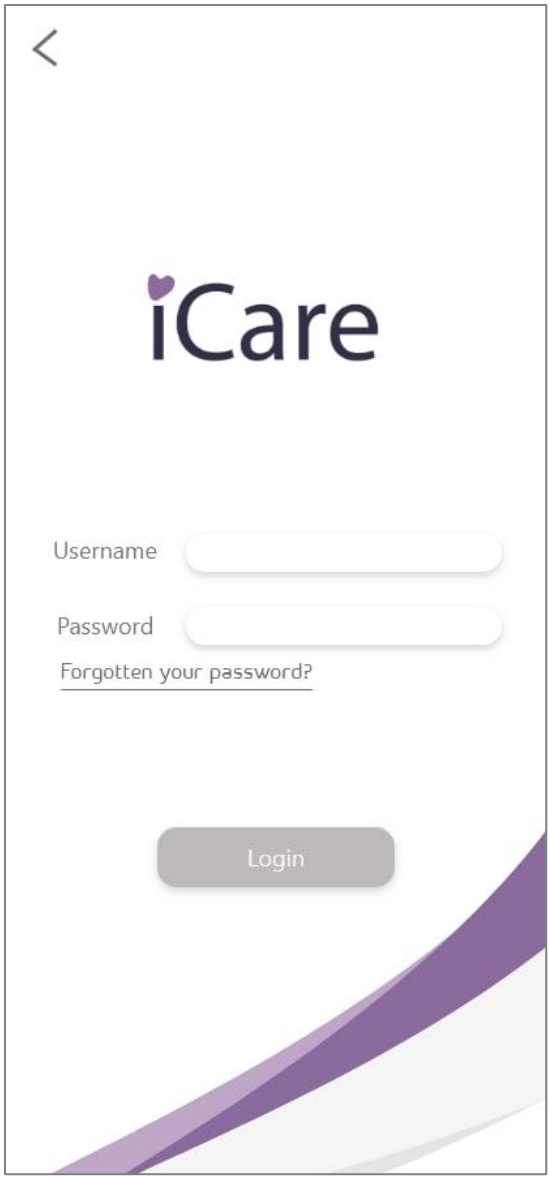
Prototype of your system (GUI)



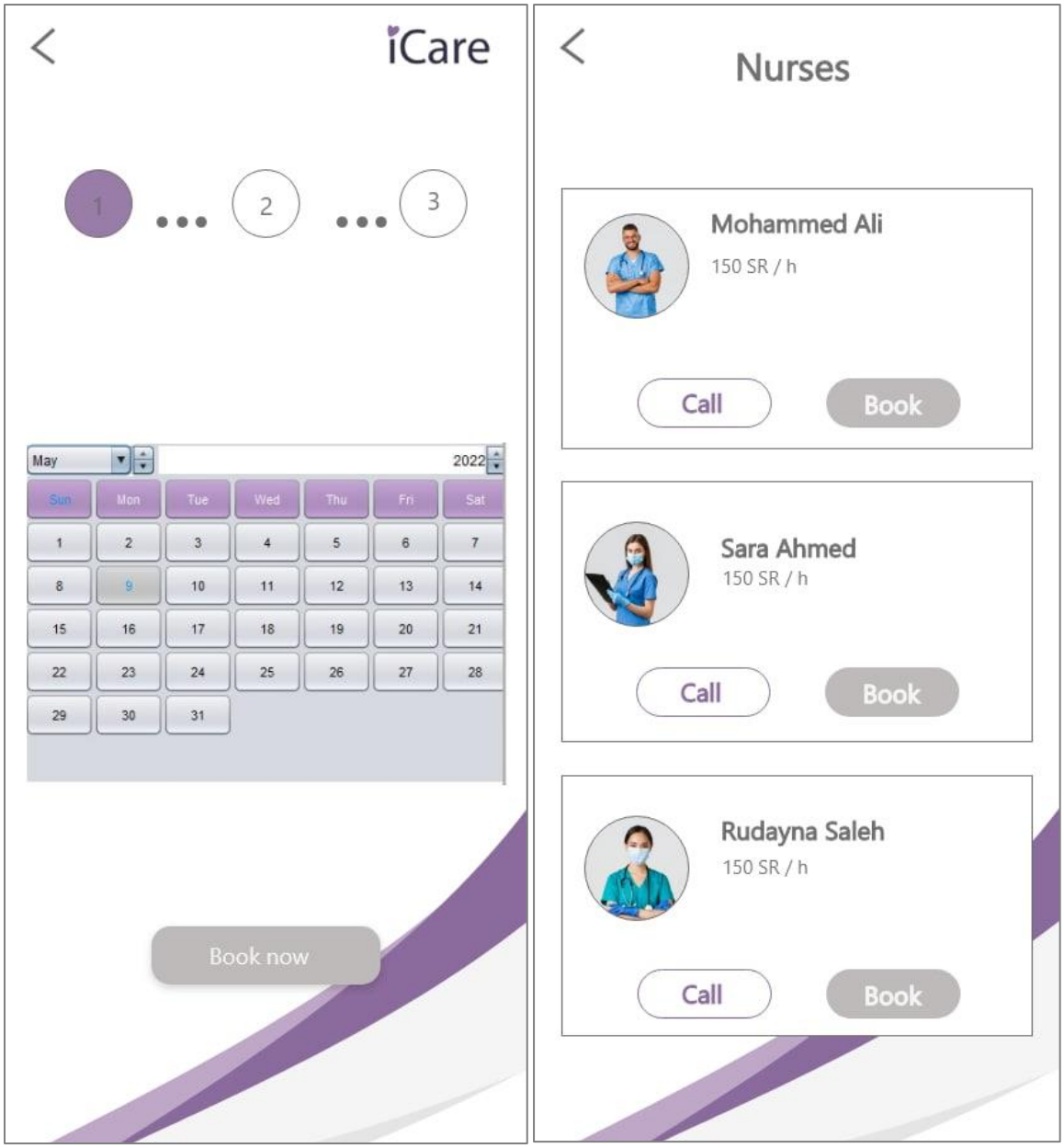

iCare
Welcome
Sign in to continue
Sign in
or
Login
iCare
enter your phone number
05**********
continue

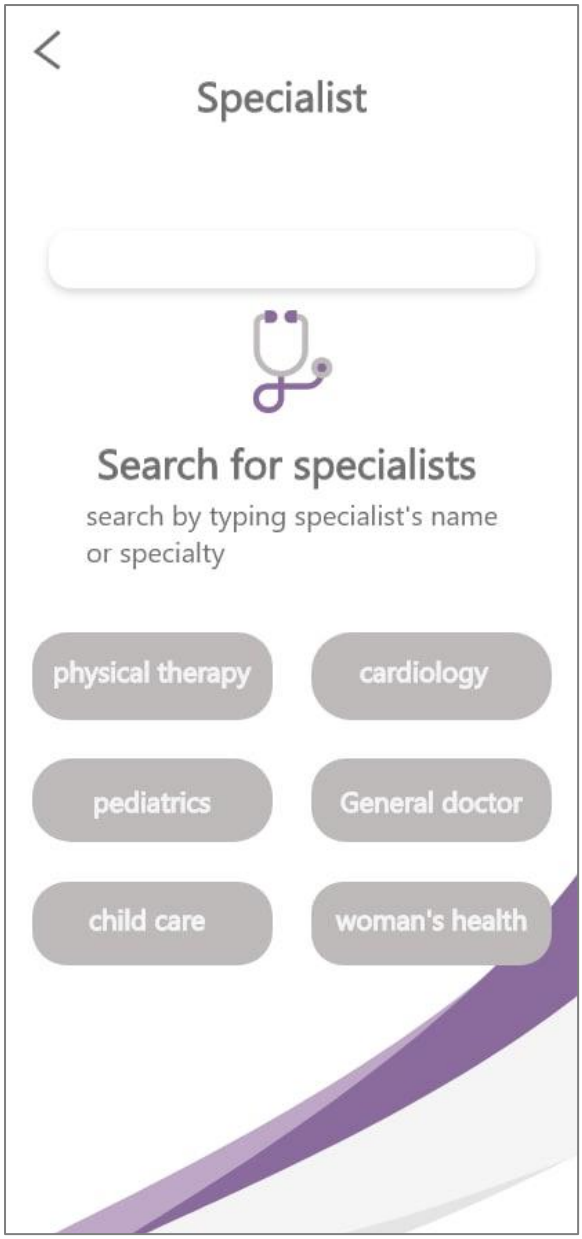

iCare
Verification
Verify now


iCare
enter your email
******@gmail.com
continue


iCare
Username
Password
Forgotten your password?
Login

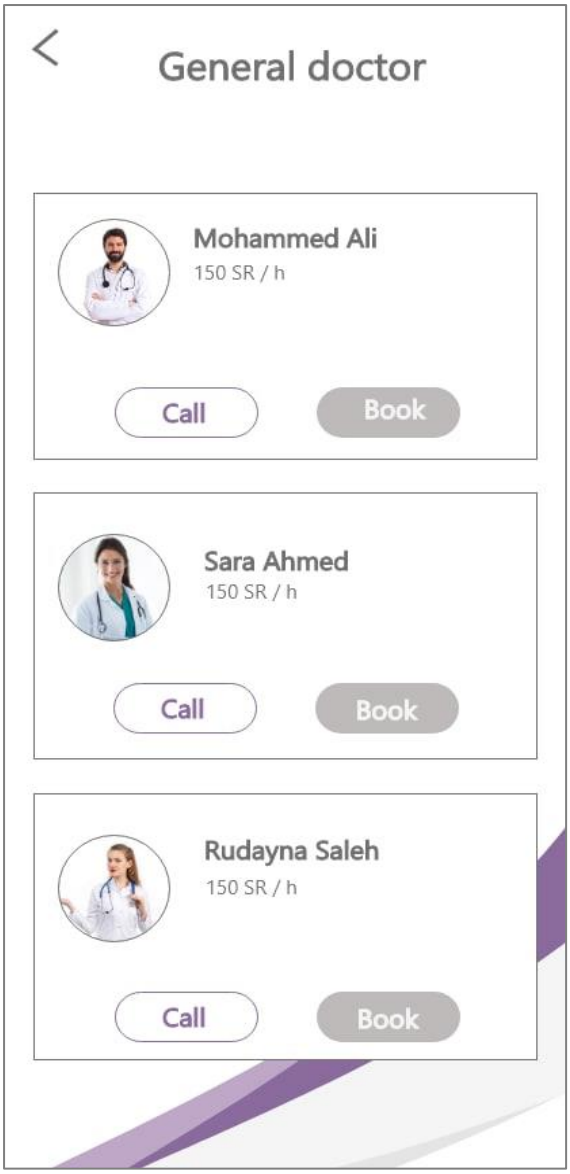
iCare
New password:
Confirm password:
update password

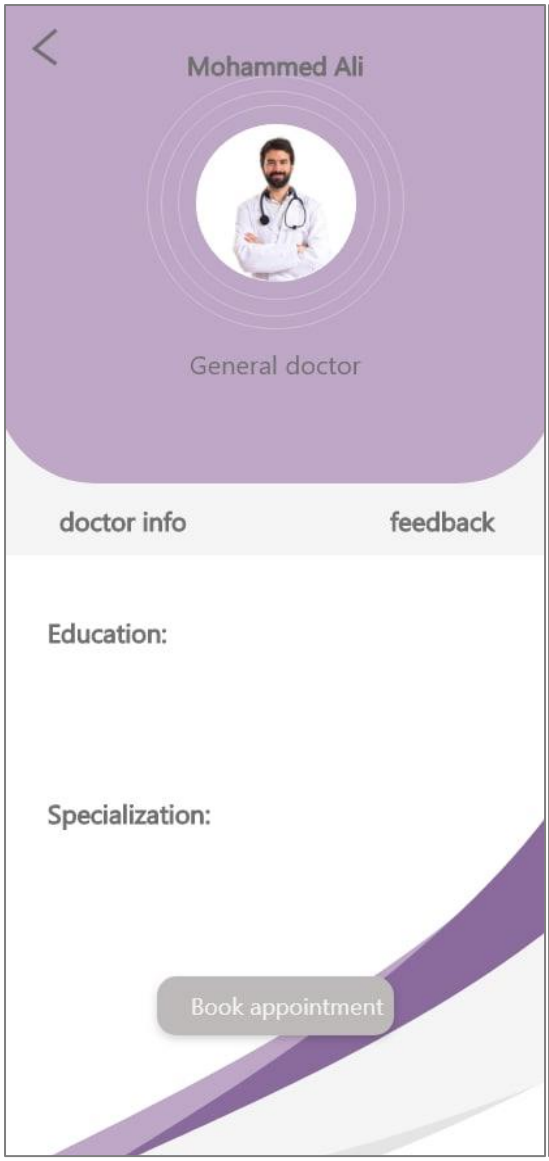

iCare
1
2
3
May
2022
Sun
Mon
Tue
Wed
Thu
Fri
Sat
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
Book now
Nurses
Mohammed Ali
150 SR / h
Call
Book
Sara Ahmed
150 SR / h
Call
Book
Rudayna Saleh
150 SR / h
Call
Book

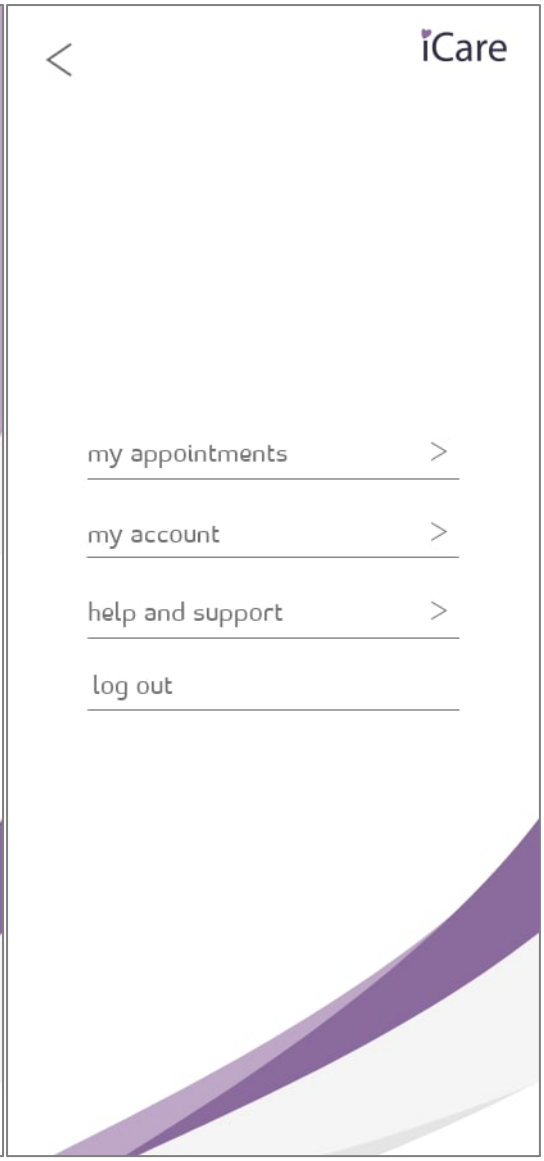

Specialist
Search for specialists
search by typing specialist's name or specialty
physical therapy
cardiology
pediatrics
General doctor
child care
woman's health

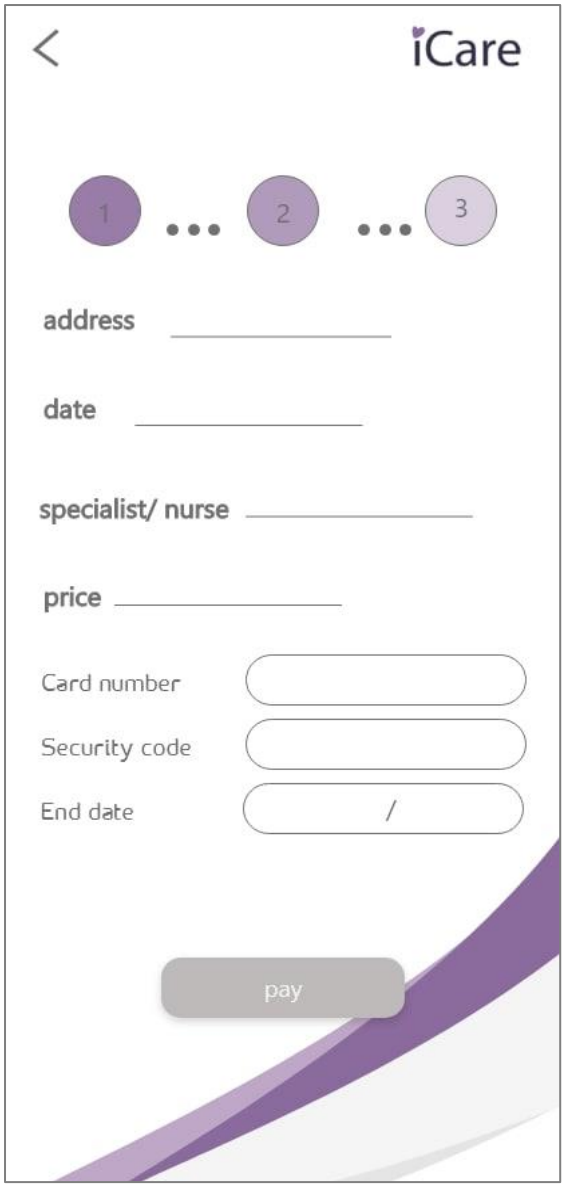
iCare support
iC
Welcome, how can I help you?
iCare
type a message...
send

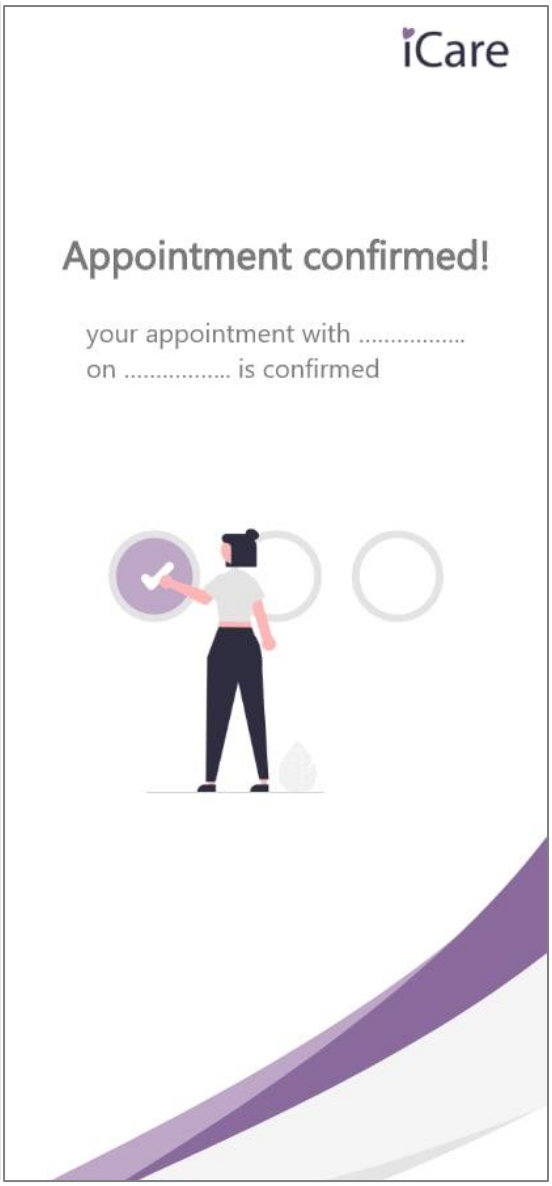
iCare support
direct conversation
call us

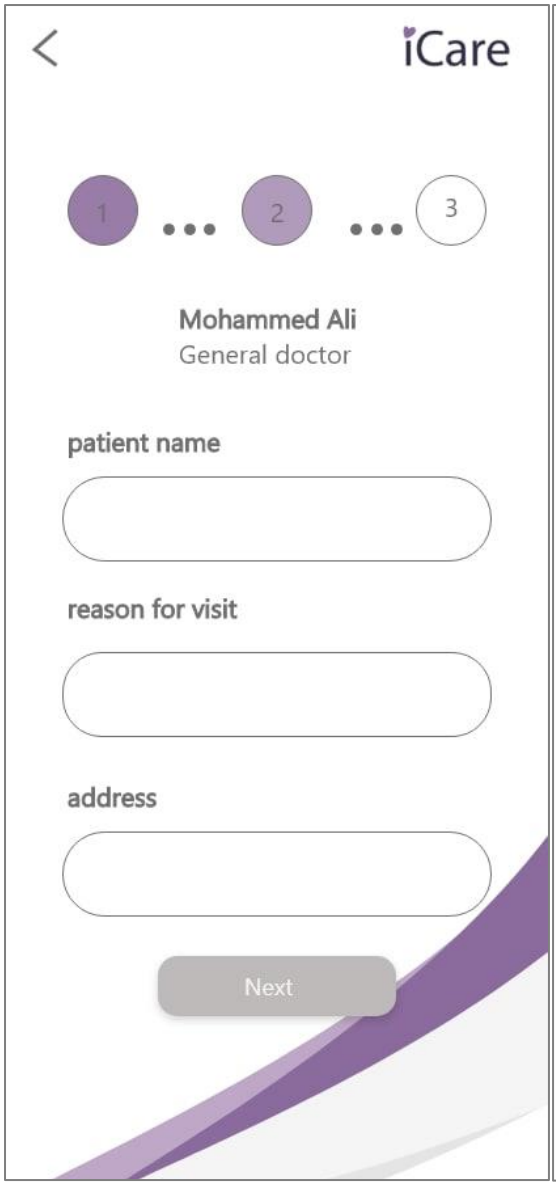

General doctor
Mohammed Ali
150 SR / h
Call
Book
Sara Ahmed
150 SR / h
Call
Book
Rudayna Saleh
150 SR / h
Call
Book

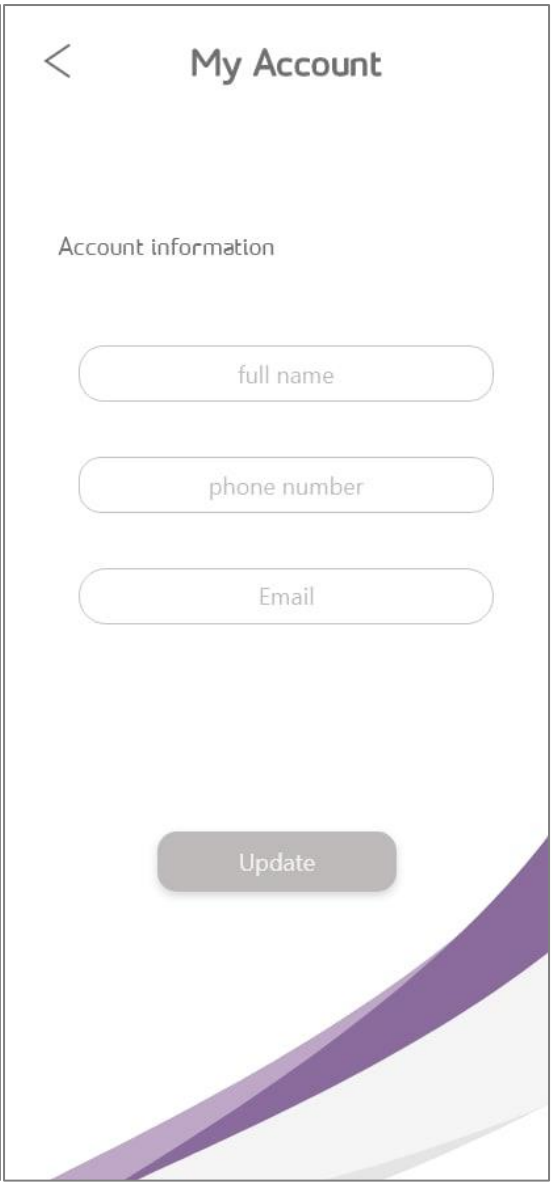

Mohammed Ali
General doctor
doctor info
feedback
Education:
Specialization:
Book appointment


iCare
my appointments
my account
help and support
log out


iCare
1
2
3
address
date
specialist/ nurse
price
Card number
Security code
End date
/
pay


iCare
Appointment confirmed!
your appointment with .....
on ..... is confirmed


iCare
1
2
3
Mohammed Ali
General doctor
patient name
reason for visit
address
Next


My Account
Account information
full name
phone number
Email
Update

Codes of the classes

Classes package

User class

```
package CPIT252Project.Classes;
import DesignPattern.*;
import java.util.ArrayList;
public class User implements Subject {
    private String firstName;
    private String lastName;
    private String phoneNumber;
    private String SSN;
    private int ID;
    private String gender;
    private String Email;
    private String city;
    private String age;
    private String password;
    private String status;
    private ArrayList<Observer> users;
    public String getStatus() {
        return status;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
    public void setID(int ID) {
        this.ID = ID;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }

    public void setEmail(String Email) {
        this.Email = Email;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public void setBirthDate(String age) {
        this.age = age;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getFirstName() {
```

```java
    public String getLastName() {
        return lastName;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public String getSSN() {
        return SSN;
    }

    public int getID() {
        return ID;
    }

    public String getGender() {
        return gender;
    }

    public String getEmail() {
        return Email;
    }

    public String getCity() {
        return city;
    }
```

```java
public String getBirthDate() {
    return age;
}

public String getPassword() {
    return password;
}
public User(String firstName, String lastName, String phoneNumber, String SSN, String gender,
    this.firstName = firstName;
    this.lastName = lastName;
    this.phoneNumber = phoneNumber;
    this.SSN = SSN;
    this.ID = generateID();
    this.gender = gender;
    this.Email = Email;
    this.city = city;
    this.age = birthDate;
    this.password = password;
    this.users = new ArrayList<>();
}

public User() {

}
@Override
public String toString() {
```

```
    public User() {
    }
    @Override
    public String toString() {
        return "The User Information is: \n" + "firstName: " + firstName + ", lastName: " + lastN
    }

    public int generateID() {
        return (int) (Math.random() * 100000);
    }

    @Override
    public void registers(Observer o) {
        users.add(o);
    }

    @Override
    public void unregisters(Observer o) {
        users.remove(o);
    }

    @Override
    public void notifyAllRegisters() {
        for (Observer o : users) {
            o.updates(status);
        }
    }
```

```
    public void unregisters(Observer o) {
        users.remove(o);
    }

    @Override
    public void notifyAllRegisters() {
        for (Observer o : users) {
            o.updates(status);
        }
    }

    public void setStatus(Boolean s) {
        status = this.firstName + (s ? "Available" : "Not Available");
        notifyAllRegisters();
    }

}
```

Patient class

```java
package CPIT252Project.Classes;
import DesignPattern.AppointmentBuilder;
import DesignPattern.Observer;
import javax.swing.JOptionPane;
public class Patient extends User implements Observer {
    private AppointmentBuilder appointment;
    private String name;
    public AppointmentBuilder getAppointment() {
        return appointment;
    }
     public void setAppointment(AppointmentBuilder appointment) {
        this.appointment=appointment;
    }
     public AppointmentBuilder makeAnAppointment(String date,MedicalEmployee MedicalEmployee){
         appointment= new AppointmentBuilder(this,MedicalEmployee, date);
         return appointment;
     }
    public Patient(String name) {
        this.name = name;
    }
    public Patient(String firstName, String lastName, String phoneNumber, String SSN, String gend
        super(firstName, lastName, phoneNumber, SSN, gender, Email, city, birthDate, password);
    }
    @Override
    public void updates(String update) {
        JOptionPane.showMessageDialog(null, name + " has new notification: " + update);
```

Medical employee class

```java
package CPIT252Project.Classes;
import DesignPattern.*;
public class MedicalEmployee extends User {
    private String status;
    private String Type;
    public MedicalEmployee(String status, String Type, String firstName, String lastName, String
        super(firstName, lastName, phoneNumber, SSN, gender, Email, city, birthDate, password);
        this.status = status;
        this.Type = Type;
    }
    public MedicalEmployee() {
    }
    public void setType(String type) {
        this.Type = type;
    }
    public String getType() {
        return Type;
    }
    public void setStatus(String status) {
        this.status = status;
    }
    public String getStatus() {
        return status;
    }
}
```

Appointment class

```
package CPIT252Project.Classes;
import DesignPattern.*;
import java.util.ArrayList;
public class Appointment implements Subject {
    private Patient Patient;
    private MedicalEmployee MedicalEmployee;
    private String appointmentDate;
    private int appointmentID;
    private boolean available=true;
    private ArrayList<Observer> list;
    private Boolean status;
    private String availablity;
    private String MedicalEmployee_SSN;
    public String getMedicalEmployee_SSN() {
        return MedicalEmployee_SSN;
    }
    public Appointment(Patient Patient, MedicalEmployee MedicalEmployee, String appointmentDate)
        this.Patient = Patient;
        this.MedicalEmployee = MedicalEmployee;
        this.appointmentDate = appointmentDate;
        this.appointmentID = generateID();
        list = new ArrayList<>();
    }
    public Appointment(MedicalEmployee MedicalEmployee, String appointmentDate) {
        this.MedicalEmployee = MedicalEmployee;
        this.appointmentDate = appointmentDate;
        this.appointmentID = generateID();
```

```
    public Appointment(String appointmentDate, int appointmentID,boolean available,String Medic
     this.appointmentDate=appointmentDate;
     this. appointmentID=appointmentID;
     this.available=available;
    this.MedicalEmployee_SSN=MedicalEmployee_SSN;
    }
    public Appointment(Boolean status, String date) {
        this.available = status;
        this.appointmentDate = date;
        list = new ArrayList<>();
    }
    public void setPatient(Patient Patient) {
        this.Patient = Patient;
    }
    public void setMedicalEmployee(MedicalEmployee MedicalEmployee) {
        this.MedicalEmployee = MedicalEmployee;
    }
    public void setAppointmentDate(String AppointmentDate) {
        this.appointmentDate = AppointmentDate;
    }
    public void setAppointmentID(int ID) {
        this.appointmentID = ID;
    }
    public void setAvailable(boolean Available) {
        this.available = Available;
    }
    public boolean getAvailable() {
```

```
    public boolean getAvailable() {
        return available;
    }
    public int generateID() {
        return (int) (Math.random() * 100000);
    }
    public Patient getPatient() {
        return Patient;
    }
    public MedicalEmployee getMedicalEmployee() {
        return MedicalEmployee;
    }
    public String getAppointmentDate() {
        return appointmentDate;
    }
    public int getID() {
        return appointmentID;
    }

    @Override
    public void registers(Observer o) {
        list.add(o);
    }
    @Override
    public void unregisters(Observer o) {
        list.remove(o);
    }
```

```
        list.add(o);
    }
    @Override
    public void unregisters(Observer o) {
        list.remove(o);
    }
    @Override
    public void notifyAllRegisters() {
        for (Observer o : list) {
            o.updates(availablity);
        }
    }
    public void setStatus(Boolean s) {
        availablity = s ? "New Appointments Available" : "Appointments Not Available";
        notifyAllRegisters();
    }
}
```

Admin class

```java
package CPIT252Project.Classes;
import java.util.ArrayList;
public class Admin extends User  {
    private String name;
    public static ArrayList<MedicalEmployee> MedicalEmployees = new ArrayList<MedicalEmployee>();
    public static ArrayList<Appointment> Appointments = new ArrayList<Appointment>();
    public static ArrayList<Patient> Patients = new ArrayList<Patient>();
    public Admin(String firstName, String lastName, String phoneNumber, String SSN, String gender
        super(firstName, lastName, phoneNumber, SSN, gender, Email, city, birthDate, password);
    }
    public Admin(String firstName) {
        name = firstName;
    }
    public static void addMedicalEmployee(MedicalEmployee MedicalEmployee) {
        MedicalEmployees.add(MedicalEmployee);
    }

    public static void addPatient(Patient Patient) {
        Patients.add(Patient);
    }

    public static void deletePatient(Patient Patient) {
        for (int i = 0; i < MedicalEmployees.size(); i++) {
            if (Patients.get(i).getID() == Patient.getID()) {
                Patients.remove(Patient);
            }
        }
```

```java
public static void addApointment(Appointment Appointment) {
    Appointments.add(Appointment);
}

public static void deleteMedicalEmployee(MedicalEmployee MedicalEmployee) {
    for (int i = 0; i < MedicalEmployees.size(); i++) {
        if (MedicalEmployees.get(i).getID() == MedicalEmployee.getID()) {
            MedicalEmployees.remove(MedicalEmployee);
        }
    }
}

public static void deleteApointment(Appointment Apointment) {
    for (int i = 0; i < Appointments.size(); i++) {
        if (Appointments.get(i).getID() == Apointment.getID()) {
            Appointments.remove(Apointment);
        }
    }
}

public ArrayList<MedicalEmployee> getMedicalEmployees() {
    return MedicalEmployees;
}

public ArrayList<Appointment> getAppointments() {
    return Appointments;
}
```

```
        return MedicalEmployees;
    }

    public ArrayList<Appointment> getAppointments() {
        return Appointments;
    }

    public void setMedicalEmployees(ArrayList<MedicalEmployee> MedicalEmployees) {
        this.MedicalEmployees = MedicalEmployees;
    }

    public void setAppointments(ArrayList<Appointment> Appointments) {
        this.Appointments = Appointments;
    }

    public ArrayList<Patient> getPatients() {
        return Patients;
    }

    public void setPatients(ArrayList<Patient> Patients) {
        this.Patients = Patients;
    }

}
```

Design pattern package

Access interface

```java
package DesignPattern;

public interface Access {

    public void access(String email, String password);

    public void checkAccess();
}
```

Appointment builder class

```java
package DesignPattern;
import CPIT252Project.Classes.*;
public class AppointmentBuilder {
    private Appointment Appointment;
    private MedicalEmployee MedicalEmployee;
    private String date;
    private Patient patient;
    public AppointmentBuilder(MedicalEmployee MedicalEmployee, String date) {
        this.MedicalEmployee = MedicalEmployee;
        this.date = date;
    }
    public AppointmentBuilder(Patient patient, MedicalEmployee MedicalEmployee, String date) {
        this.MedicalEmployee = MedicalEmployee;
        this.date = date;
        this.patient = patient;
    }
    public Appointment prepareAnAvailableAppointment() {
        this.Appointment = new Appointment(MedicalEmployee, date);
        Appointment.setAvailable(true);
        return Appointment;
    }
    public Appointment prepareAnAppointment() {
        this.Appointment = new Appointment(patient, MedicalEmployee, date);
        return Appointment;
    }
    public Appointment getAppointment() {
        return Appointment;
    }public void setAppointment(Appointment Appointment) {
```

```
    private String date;
    private Patient patient;
    public AppointmentBuilder(MedicalEmployee MedicalEmployee, String date) {
        this.MedicalEmployee = MedicalEmployee;
        this.date = date;
    }
    public AppointmentBuilder(Patient patient, MedicalEmployee MedicalEmployee, String date) {
        this.MedicalEmployee = MedicalEmployee;
        this.date = date;
        this.patient = patient;
    }
    public Appointment prepareAnAvailableAppointment() {
        this.Appointment = new Appointment(MedicalEmployee, date);
        Appointment.setAvailable(true);
        return Appointment;
    }
    public Appointment prepareAnAppointment() {
        this.Appointment = new Appointment(patient, MedicalEmployee, date);
        return Appointment;
    }
    public Appointment getAppointment() {
        return Appointment;
    }public void setAppointment(Appointment Appointment) {
        this.Appointment=Appointment;
    }
}
```

Decorator class(“BMI Feature”)

```
package DesignPattern;
import CPIT252Project.Classes.User;
public class BMIFeature {
    User user;
    private double height;
    private double weight;
    private double BMI;
    private String BMIStatus;
    public BMIFeature(User user) {
        this.user = user;
    }
     public BMIFeature() {
    }
    public void calculateBMI(double height, double weight) {
        this.weight = weight;
        this.height = height;
        this.BMI = (weight / (height * height));
    }
    public String BMI_Status() {
        if (BMI < 18.5) {
            BMIStatus = "Underweight";
        } else if (BMI >= 18.5 && BMI <= 24.9) {
            BMIStatus = "Normal weight";
        } else if (BMI >= 25 && BMI <= 29.9) {
            BMIStatus = "Overweight";
        } else {
            BMIStatus = "Obesity";
```

Users\hp\Desktop\Final_252 (3)\Final_252\Final_252\CPIT305Project\CPIT252Project\src\DesignPattern\BMIFeature.java (modified)

```
    public BMIFeature(User user) {
        } else {
            BMIStatus = "Obesity";
        }
        return BMIStatus;
    }

    public String healthcare(String BMI_Status) {
        if (BMIStatus.equalsIgnoreCase("Underweight")) {
            return "Your BMI and body weight are low,\n you should consider gaining weight throug
                    + "\ngood diet and exercise habits, to increase your muscle mass";

        } else if (BMIStatus.equalsIgnoreCase("Normal weight")) {
            return "Your BMI considered an acceptable range, and is associated with good health.\
        } else if (BMIStatus.equalsIgnoreCase("Overweight")) {
            return "You are considered overweight and should finds ways to lower your weight,\n t
                    + " You should lose weight by changing your diet and exercising more.";
        } else {
            return "You are in a dangourse level!\n"
                    + " This indicates an unhealthy condition!\n your excess mass is putting you
                    + " \nYou should lose weight by changing your diet and exercising more.";
        }
    }

```

Database class (“Singleton”)

```
package DesignPattern;
import CPIT252Project.Classes.*;
import java.sql.*;
import javax.swing.JOptionPane;
public class Database {
    private static String url;
    private static String dbURL;
    private static Connection con;
    private static Statement st;
    private static Database connection = null;
    public void DBCreation() throws SQLException {

//                st.executeUpdate("CREATE TABLE Patient "
//                        + "(FirstName VARCHAR(100) not NULL, "
//                        + " LastName  VARCHAR(100) not NULL, "
//                        + "PhoneNumber VARCHAR(10) not NULL,"
//                        + "SSN VARCHAR(10) not NULL,"
//                        + "Gender VARCHAR(7) not NULL,"
//                        + "Email VARCHAR(100) not NULL,"
//                        + "City VARCHAR(100) not NULL,"
//                        + "Age VARCHAR(10) not NULL,"
//                        + "PassWord VARCHAR(100) not NULL, "
//                        + "PRIMARY KEY (SSN))");
//
//                st.executeUpdate("CREATE TABLE Medical_Employee "
//                        + "(FirstName VARCHAR(100) not NULL, "
//                        + " LastName  VARCHAR(100) not NULL, "
//                        + "PhoneNumber VARCHAR(10) not NULL,"
```

```
//                st.executeUpdate("CREATE TABLE Medical_Employee "
//                        + "(FirstName VARCHAR(100) not NULL, "
//                        + " LastName  VARCHAR(100) not NULL, "
//                        + "PhoneNumber VARCHAR(10) not NULL,"
//                        + "SSN VARCHAR(10) not NULL,"
//                        + "Gender VARCHAR(7) not NULL,"
//                        + "Email VARCHAR(100) not NULL,"
//                        + "City VARCHAR(100) not NULL,"
//                        + "Age VARCHAR(10) not NULL,"
//                        + "PassWord VARCHAR(100) not NULL, "
//                        + "User_Type VARCHAR(100) not NULL, "
//                        + "PRIMARY KEY (SSN))");
//                st.executeUpdate("CREATE TABLE Appointment "
//                        + "(appointmentID INT not NULL, "
//                        + " appointmentDate VARCHAR(100) not NULL, "
//                        + " available BOOLEAN not NULL,"
//                        + " MedicalEmployee_SSN VARCHAR(10) not NULL,"
//                        + " Patient_SSN VARCHAR(10) not NULL, "
//                        + " PRIMARY KEY (appointmentID))");
//
//st.executeUpdate("CREATE TABLE Available_Appointment "
//                        + "(appointmentID INT not NULL, "
//                        + " appointmentDate VARCHAR(100) not NULL, "
//                        + " available BOOLEAN not NULL,"
//                        + " MedicalEmployee_SSN VARCHAR(10) not NULL,"
//                        + " PRIMARY KEY (appointmentID))");
    }
```

Observer interface

```
package DesignPattern;

public interface Observer {

    public void updates(String update);
}
```

subject interface

```
package DesignPattern;
public interface Subject {

    public void registers(Observer o);

    public void unregisters(Observer o);

    public void notifyAllRegisters();
```