Technology Park Malaysia

APP004-4-2-JP-L-3

Java Programming

TEAM: Group 8

INTAKE: UCDF2208ICT(SE)

Group Member		
No.	Name	TP Number
1	LEW WEI HOW	TP071331
2	TAN PO YEH	TP070780
3	TONG JIA CHUEN	TP070484

Table of Contents

1.0 Introduction

Our Group has been tasked to create a simple Clinic management system program which includes a system feature “sign in” and “sign up” page using Java programming languages such as Graphic User Interface for designed, object-oriented programing, and different functionality. The target users of this system mainly focus on admin, doctor and patient. The system is expected to perform some important operations such as user registration, appointment management, medical record management and payment collection. Furthermore, the data created by using an object-oriented software solution must be stored in text file. The first target user system is the role of Admin. The admin role should be able to manage the user registration, manage patients’ walk-in-appointments, track patient’s medical record, track doctor daily appointment and collect patient payment. Secondly, is the role of doctor. The doctor’s role should be able to upload their daily schedule, track their own individual appointment, cancel the appointment, track patients’ medical record and add patients’ medical record. Last but not least, is the role of patients. The Patients role should be able to view their available timeslot, make an appointment, cancel their appointment, track their own personal medical record and track their own personal historical appointment.

2.0 Sample Output and explanation of the Output

2.1 User Credentials

This the distributed login page, user can click on the specific role login button to redirect to the login page.

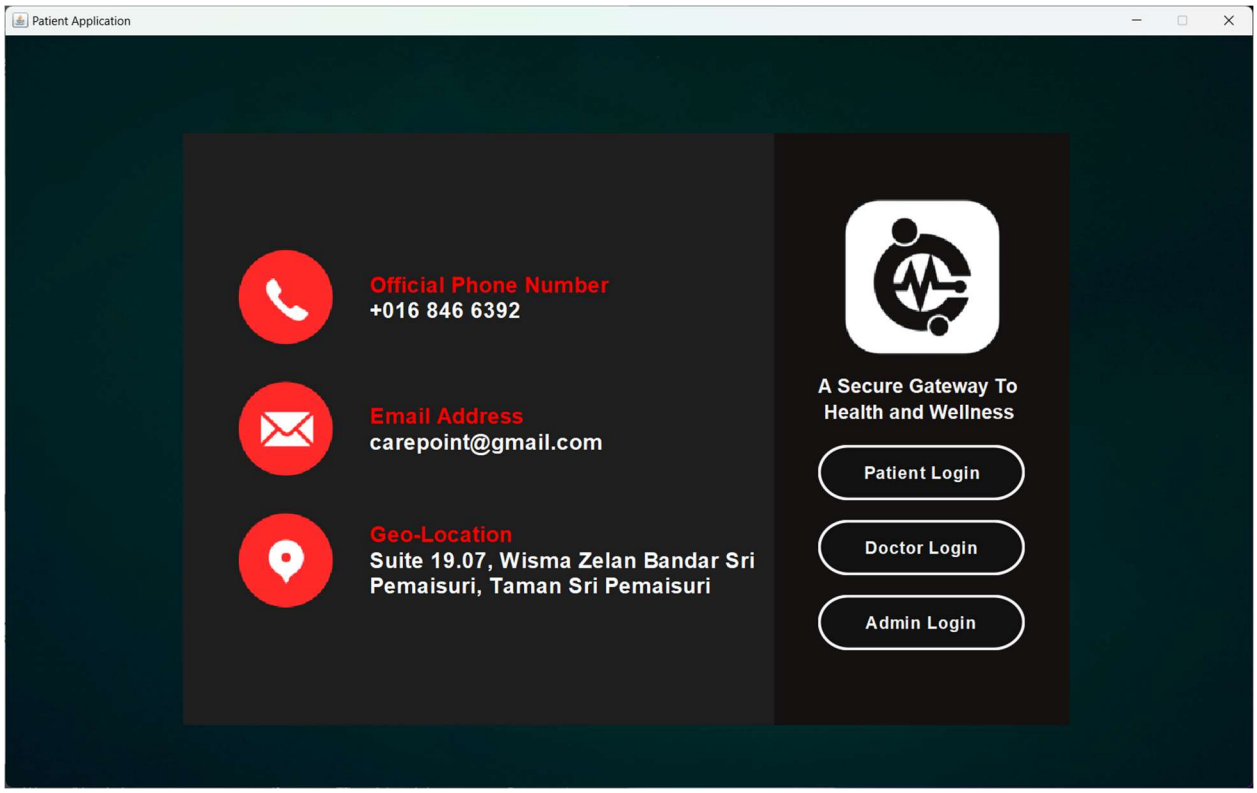


Figure 1 Main Page

Once users pass the authentication, they are allowed to access to specific role of main page. If user input wrong username (IC Number) or password, the system will show “Login Failed” message to the user.

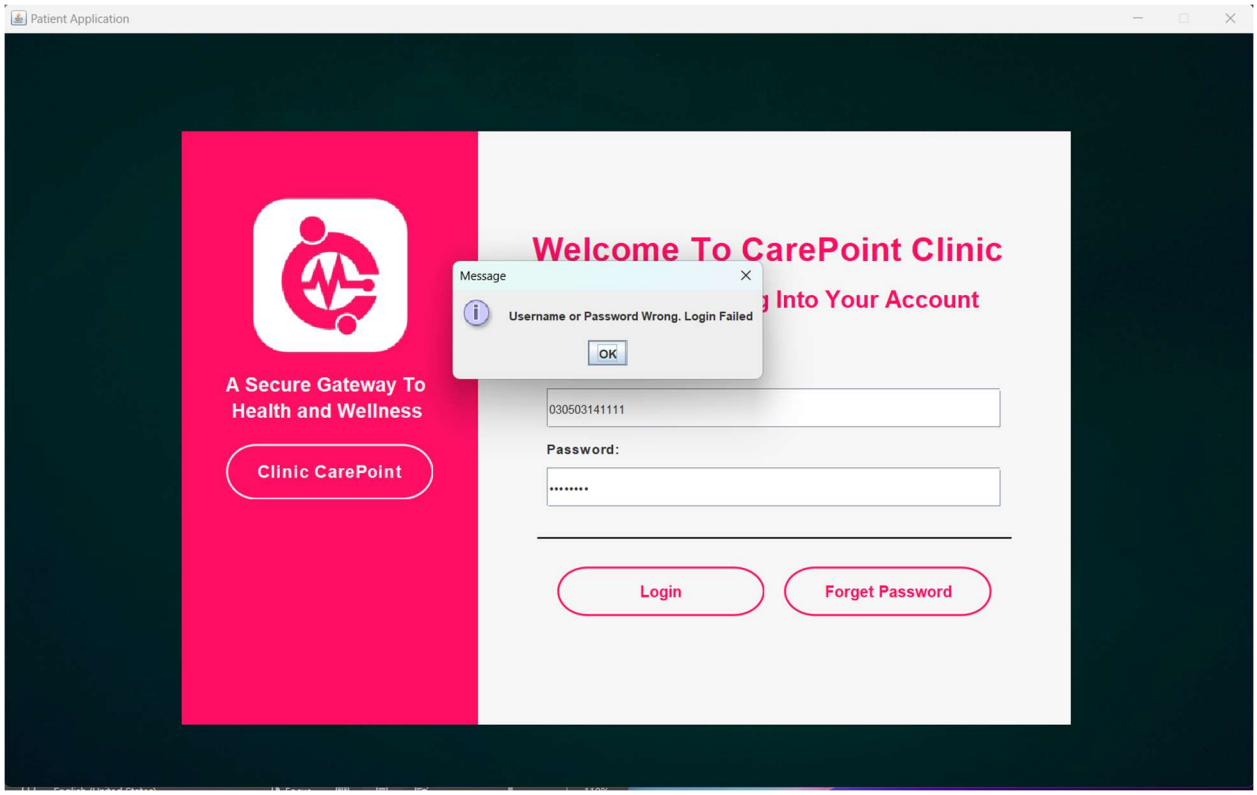


Figure 2 Login page

Users can change their password by clicking on the *Forget Password* button. A Forget Password box will pop up. Users need to key in their IC number and contact number to the account. After that, typing new password and confirm password, then click on the *Reset Password* button to change the password. “Password changed successfully” message will be displayed once the password changed.

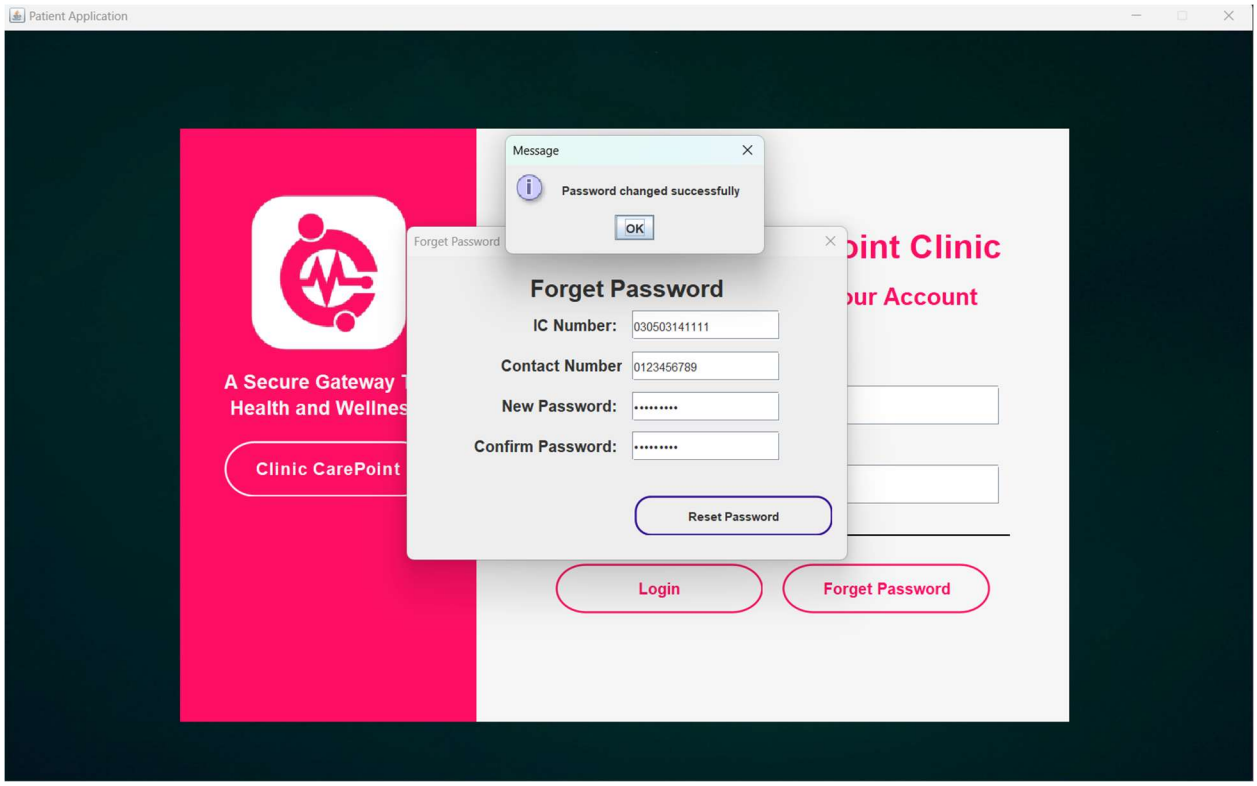


Figure 3 Forget Password Page

2.2 Admin Features

Admin can access the admin main page after authentication passed. The center of this page contains some button that can navigate admin to function page. The bottom of the page shows Upcoming Appointment for current date. Admin can click on the *Refresh* button to refresh the upcoming appointment table.

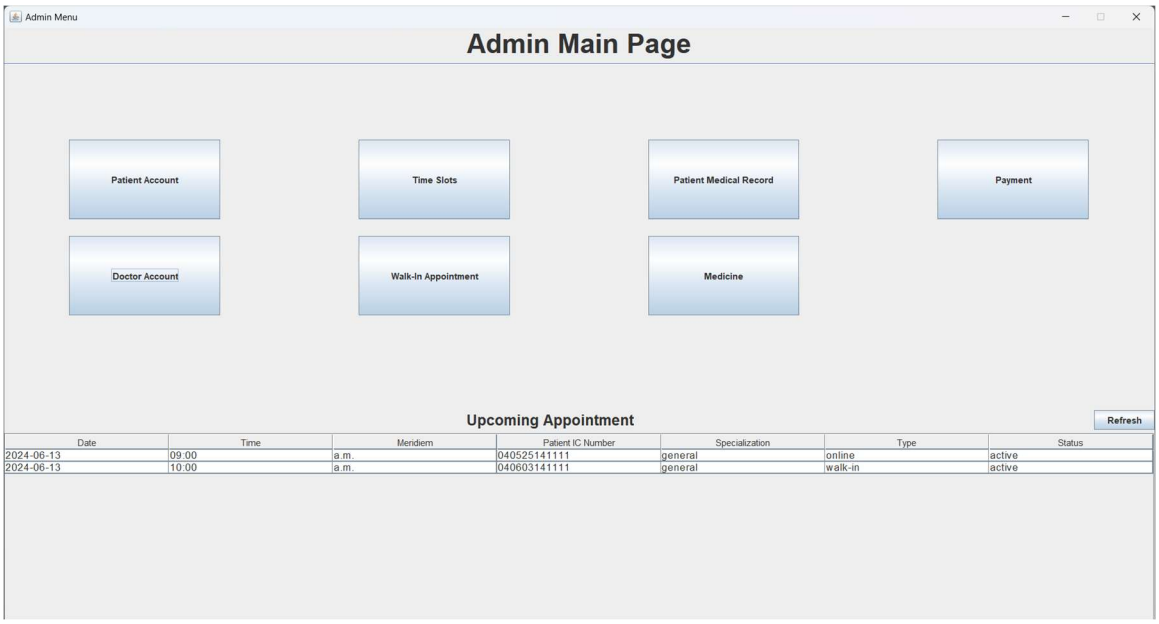


Figure 4 Before refreshing

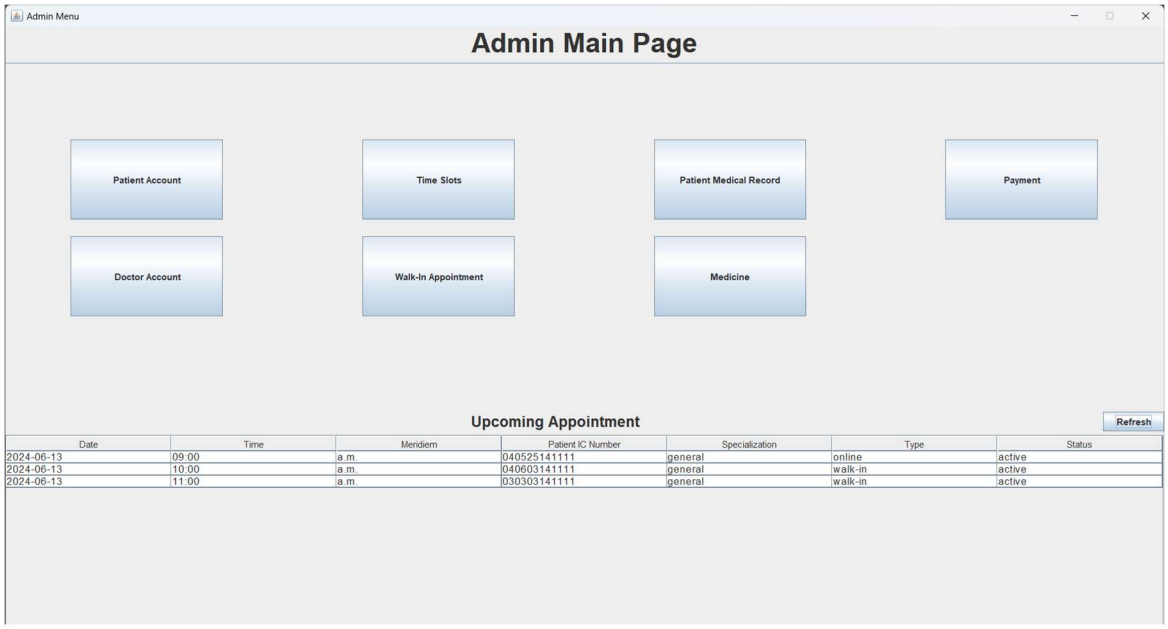


Figure 5 After refreshing

2.2.1 Manage Patient Account

Admin is able to manage patient accounts. Admin will be required to select patient information and click on the Edit button to modify patient accounts.

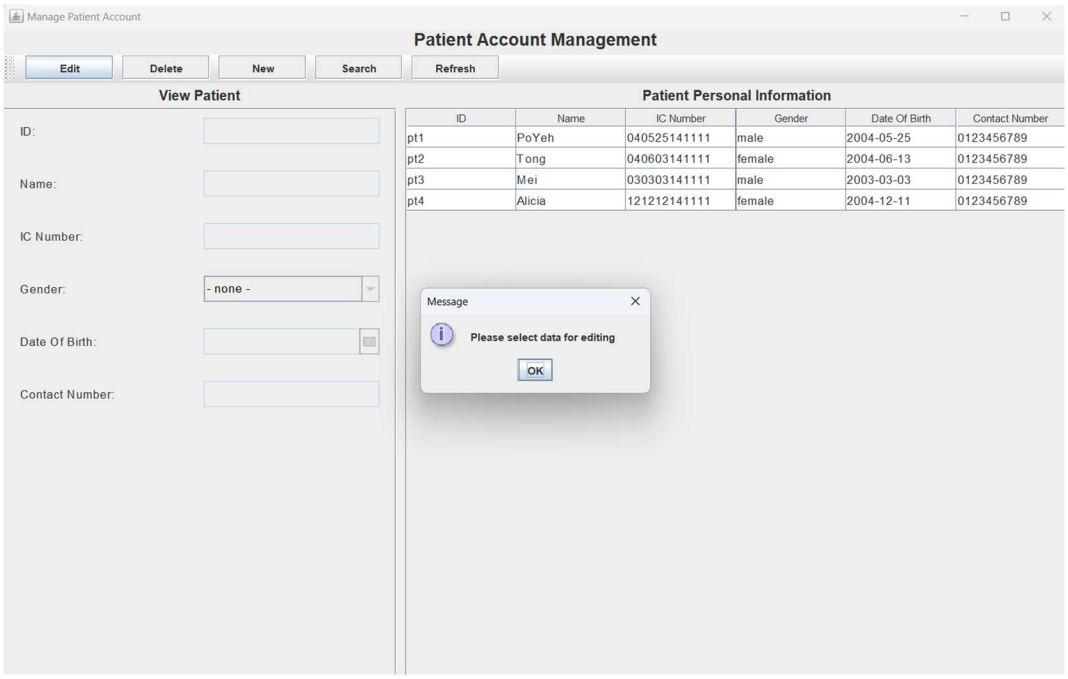


Figure 6 Click on the Edit button before selecting data will show "Please select data for editing" message

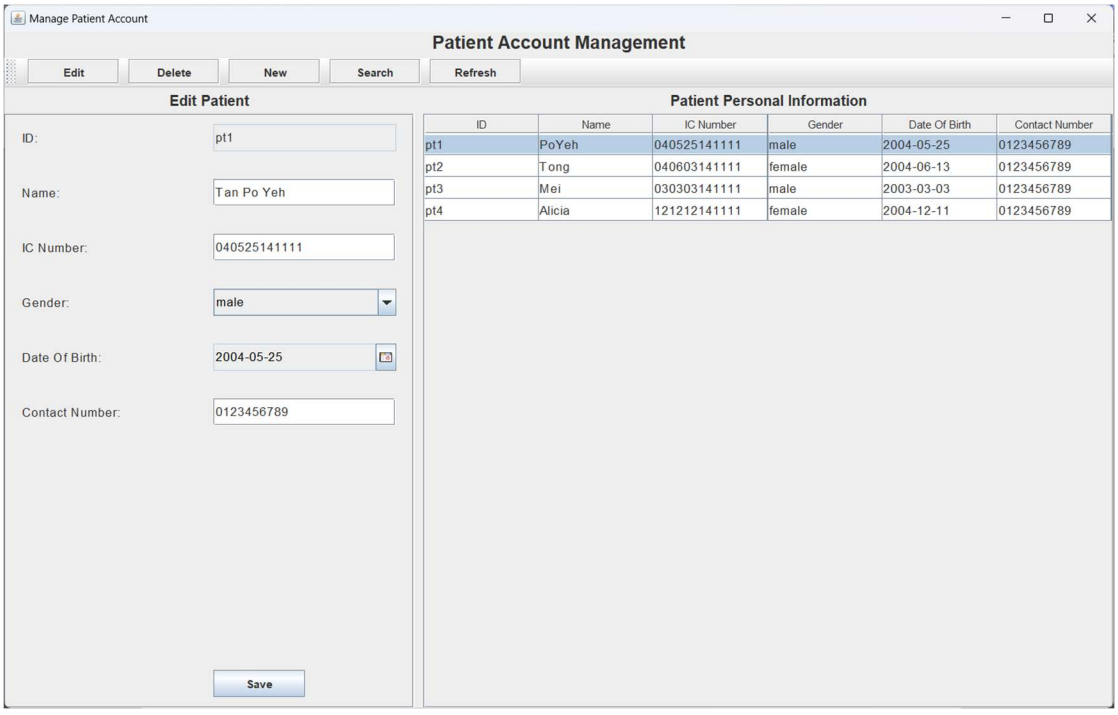


Figure 7 Input Field on left side is editable for admin to modify patient information. For example, the patient name changed to Tan Po Yeh from PoYeh

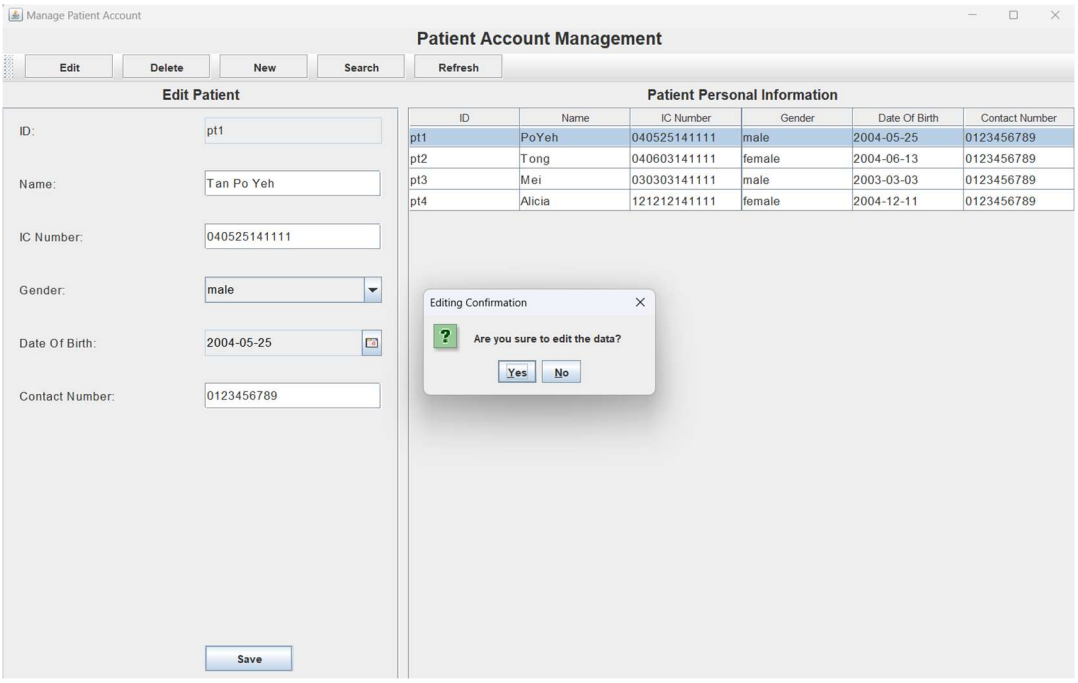


Figure 8 After that, admin can click on the Save button to save the modification. The system will prompt editing comfimation message to admin.

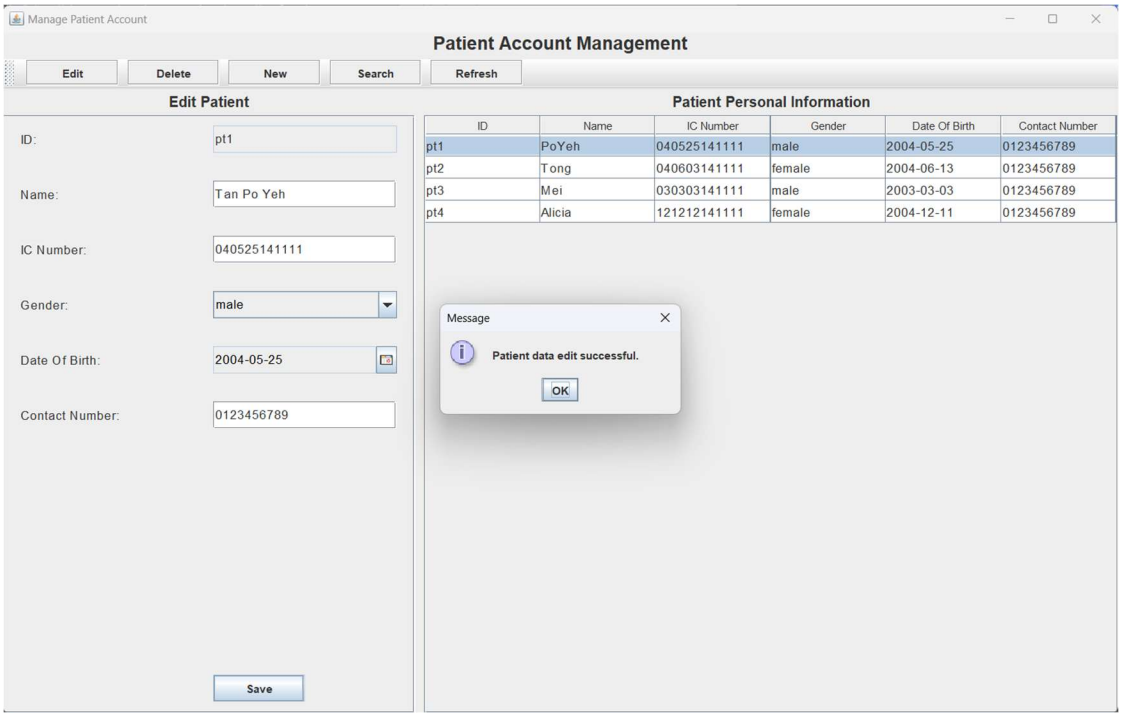


Figure 9 After clicking on Yes, the system saves the modification record into storage and show a message for the process.

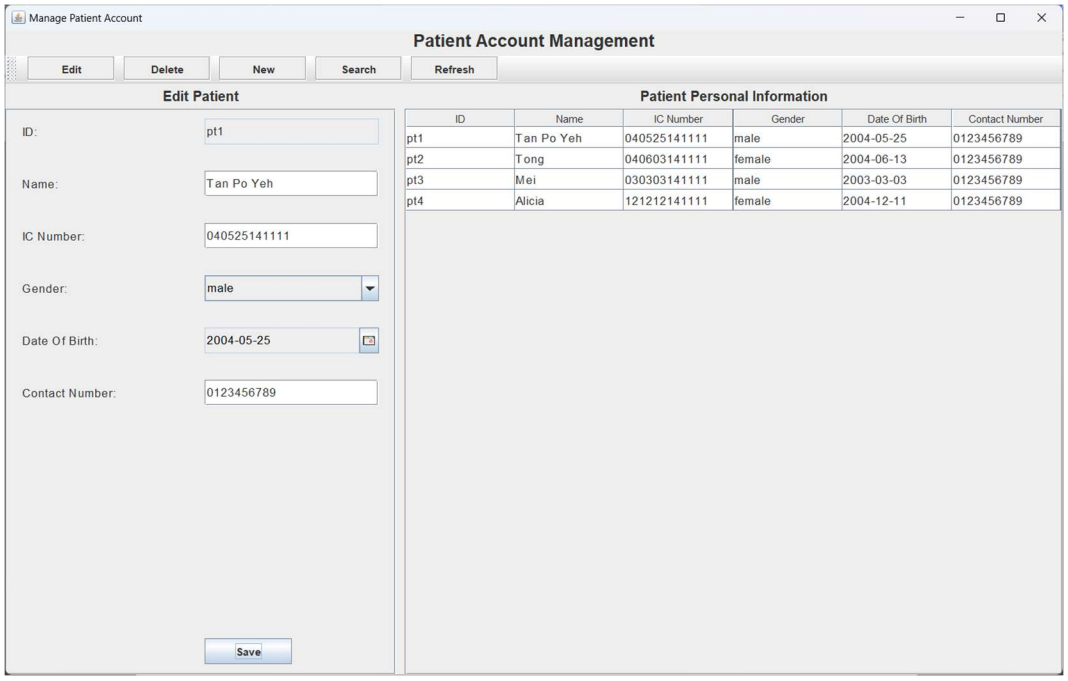


Figure 10 Name of patient pt1 have changed to Tan Po Yeh

Admin also can create new patient accounts. Click on the *New* button, the input field on left side of the screen will be clear to enable admin input new patient information.

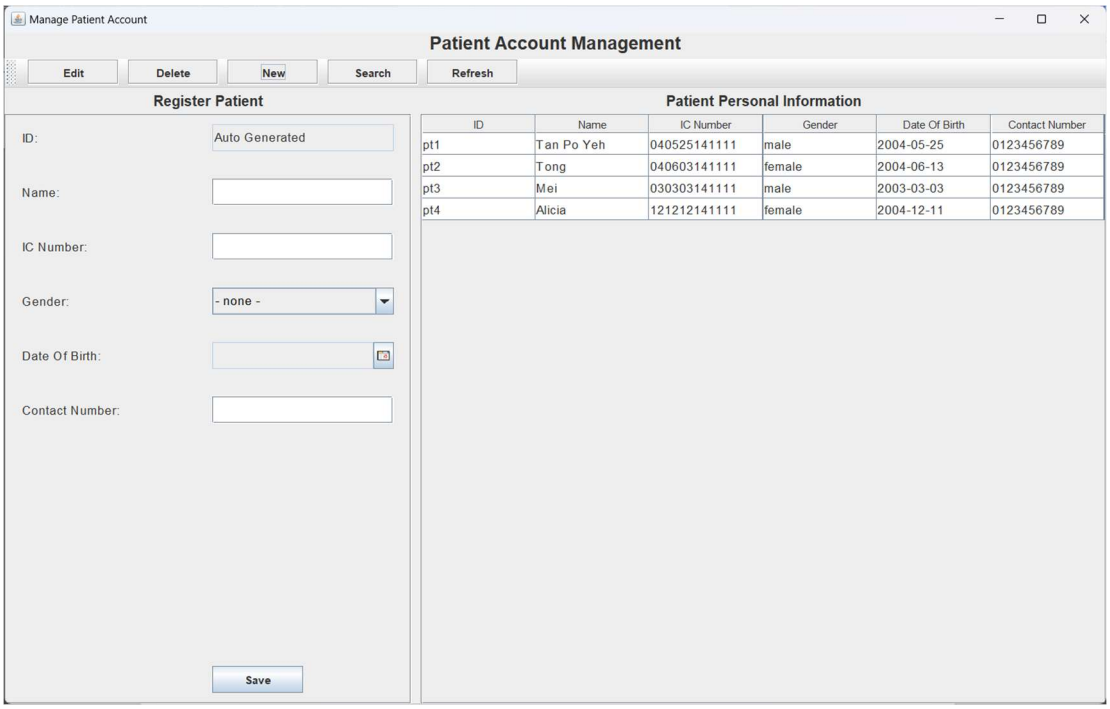


Figure 11 After clicking on New button

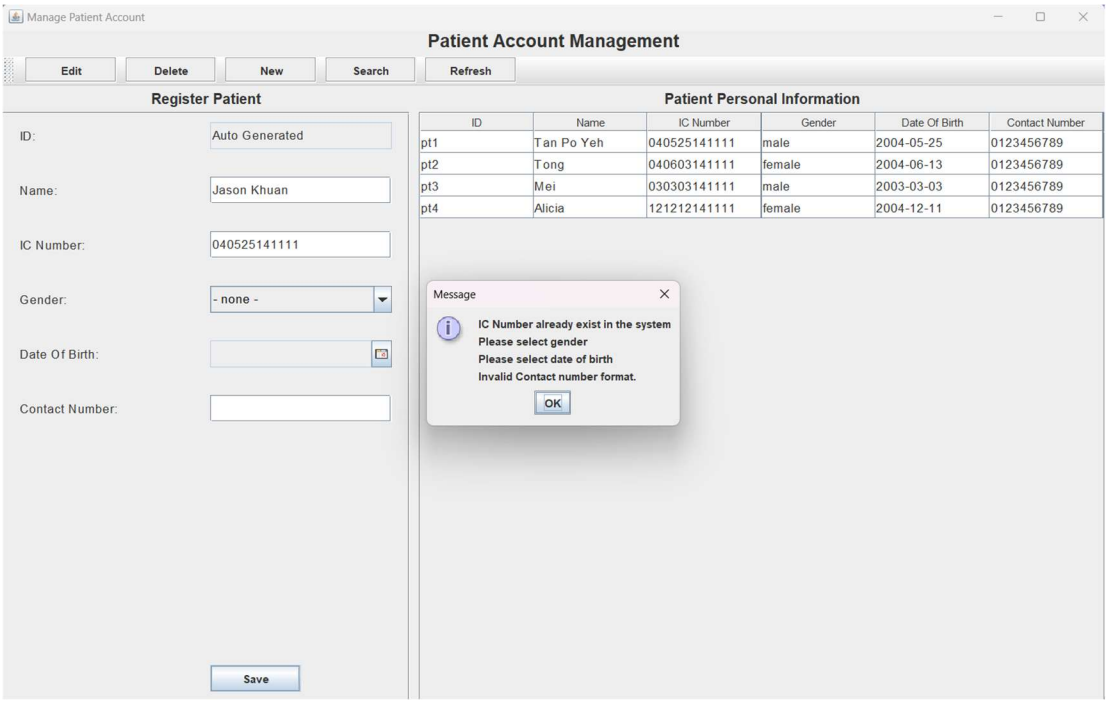


Figure 12 Input Validation

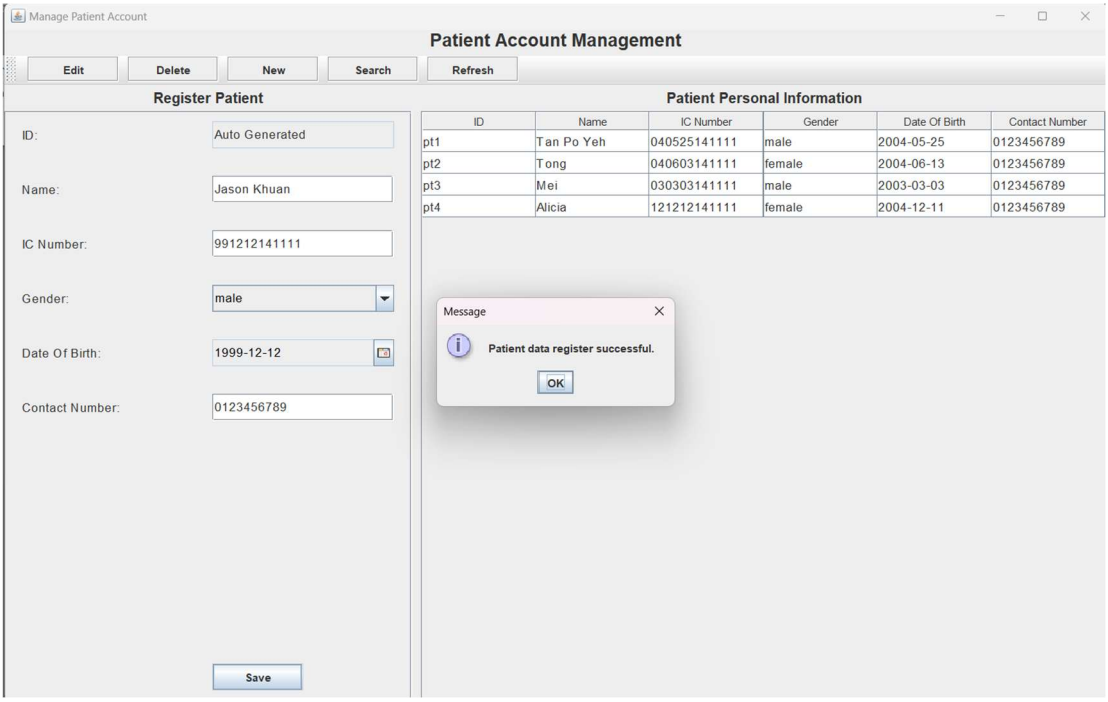


Figure 13 Click on Save button to create new patient account. When the input for the new patient information is valid, the system display registration success message

Deletion of accounts can provide better management to the system. Admin can perform this function to delete existing patient accounts.

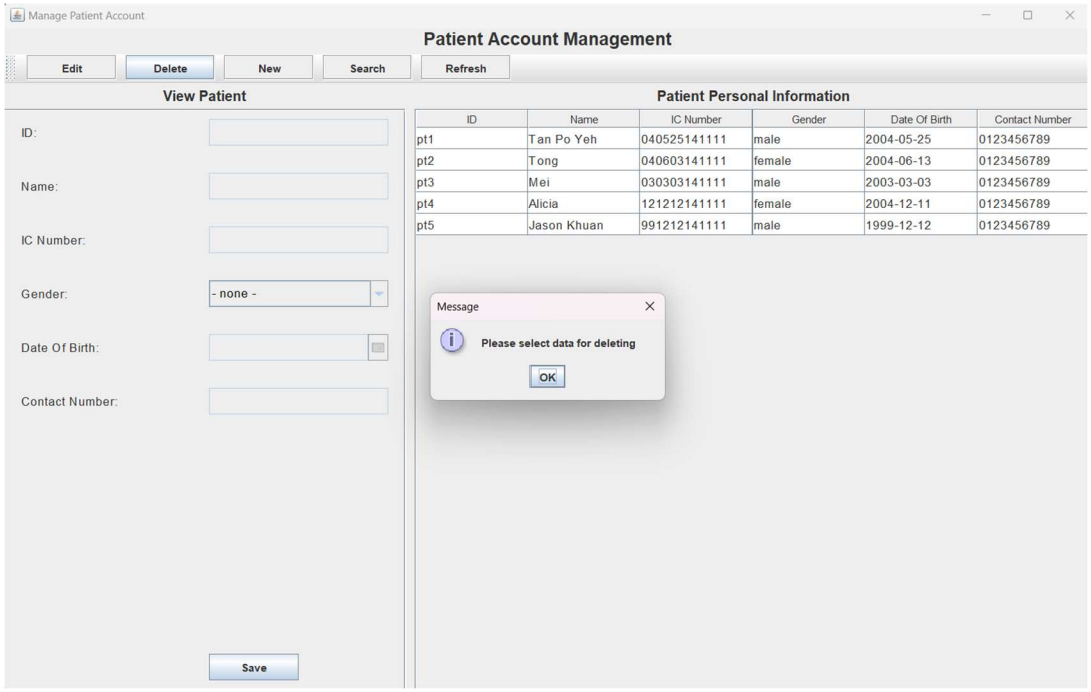


Figure 14 Before selecting patient account

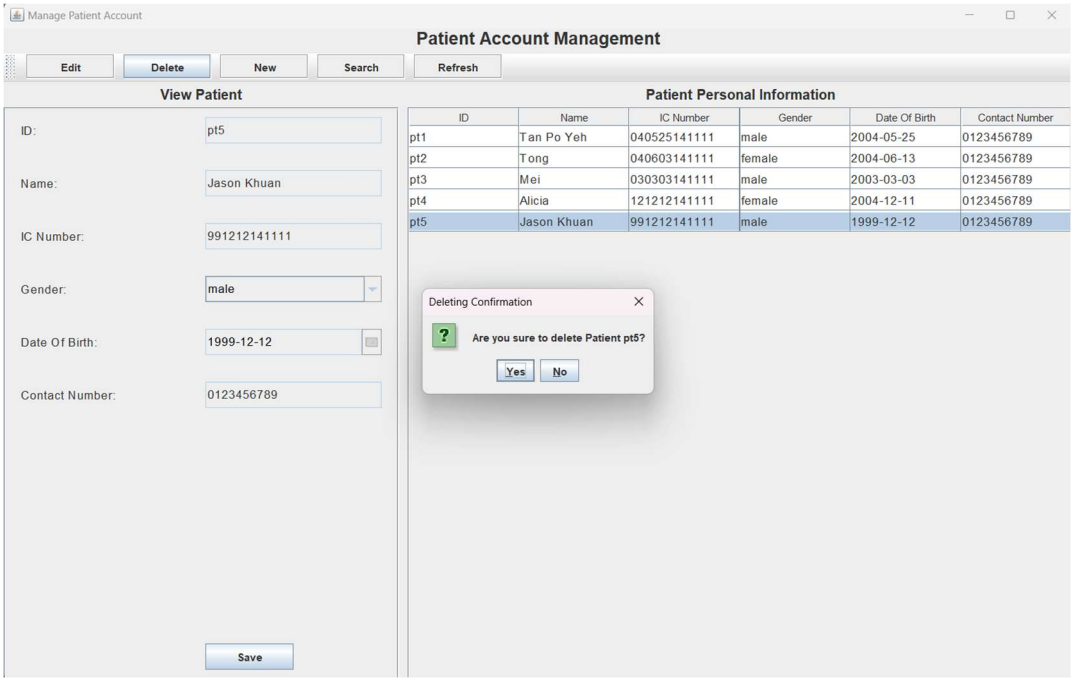


Figure 15 After selecting a patient account, admin can click on the Delete button. Following it, the system prompt admin to confirm the deleting process.

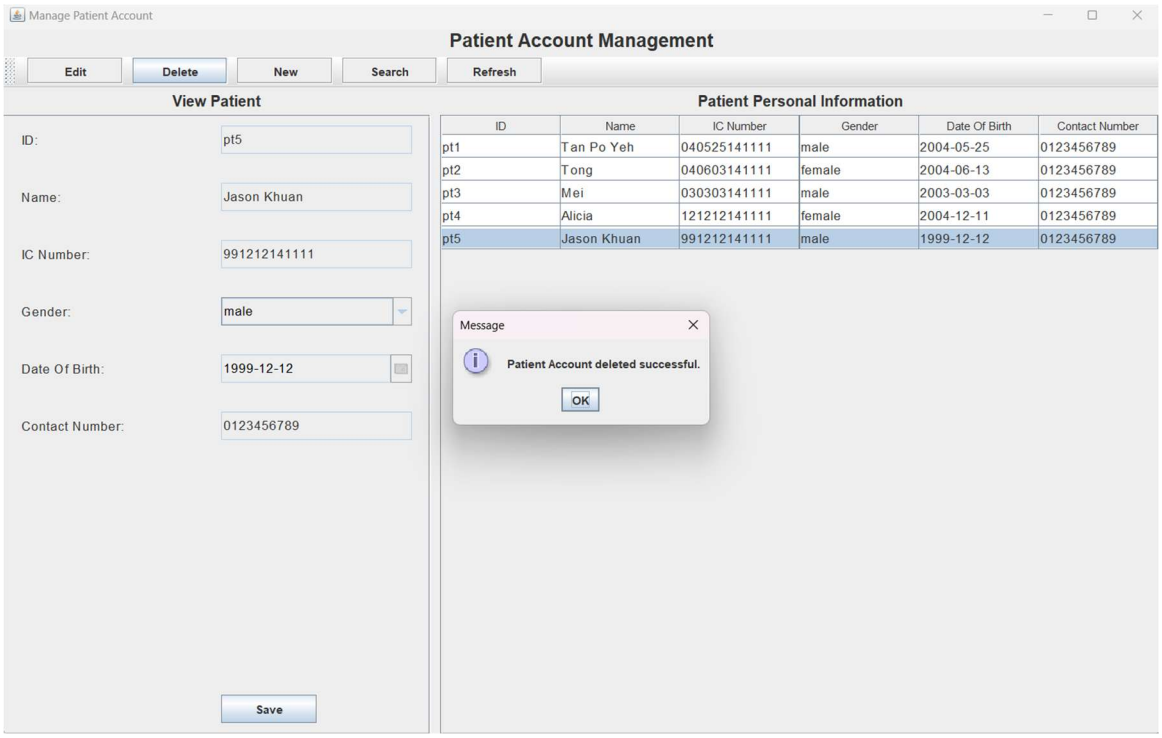


Figure 16 Upon clicking on Yes button, the system removes the patient account from storage and display a message

Admin can also query specific patient information from the system. Click on the *Search* button to access search mode.

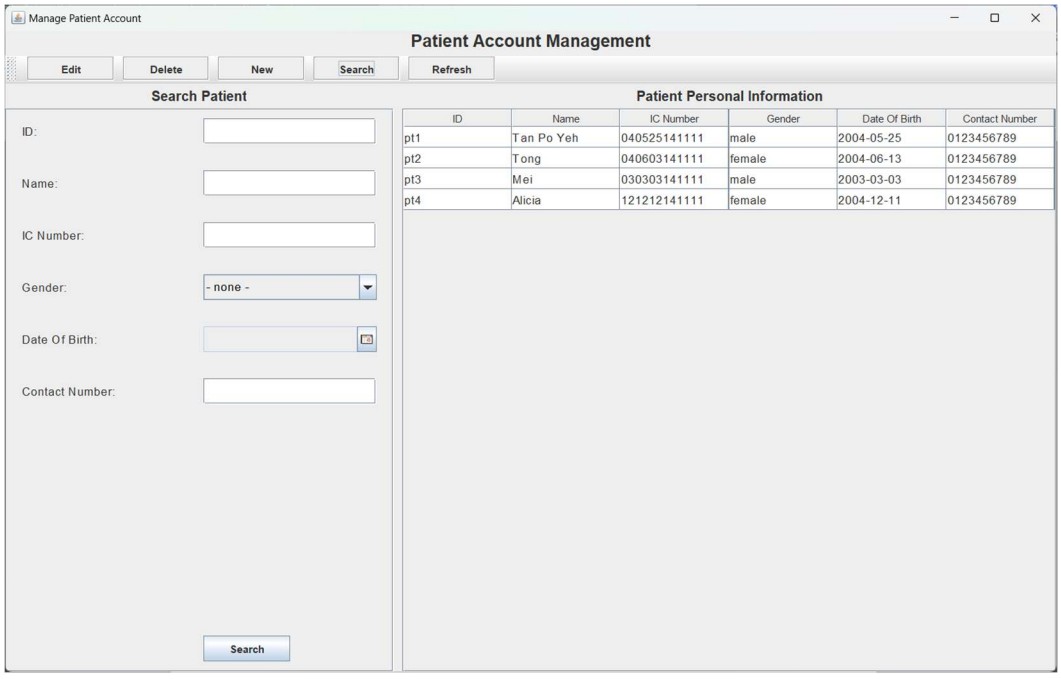


Figure 17 Search Patient

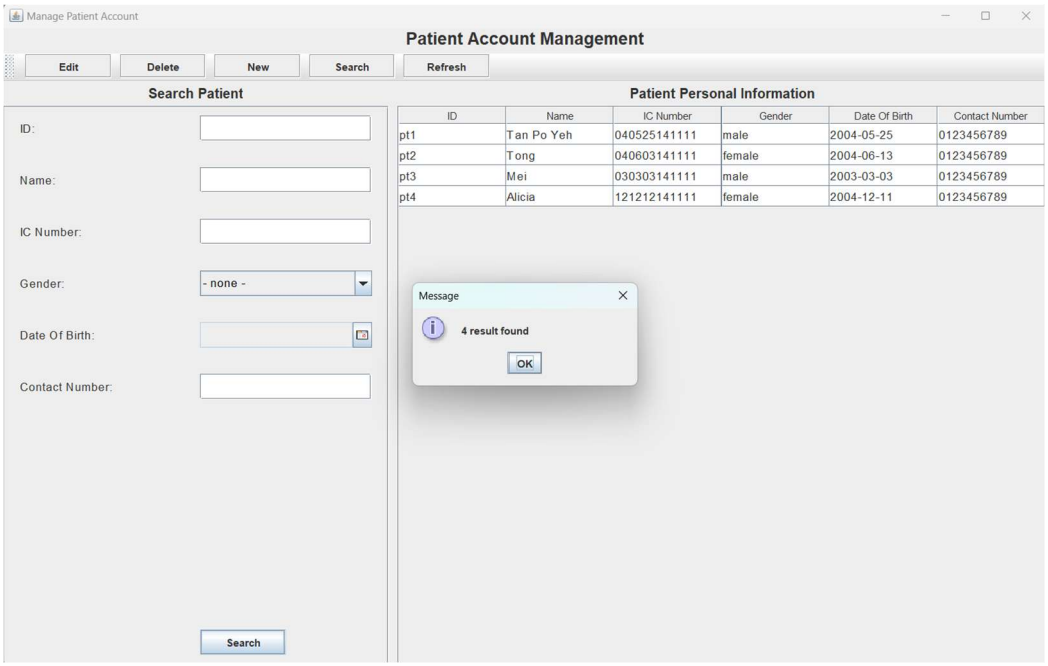


Figure 18 The searching input field left empty represent retrieving any record for this type of information. Therefore, all result will be showed.

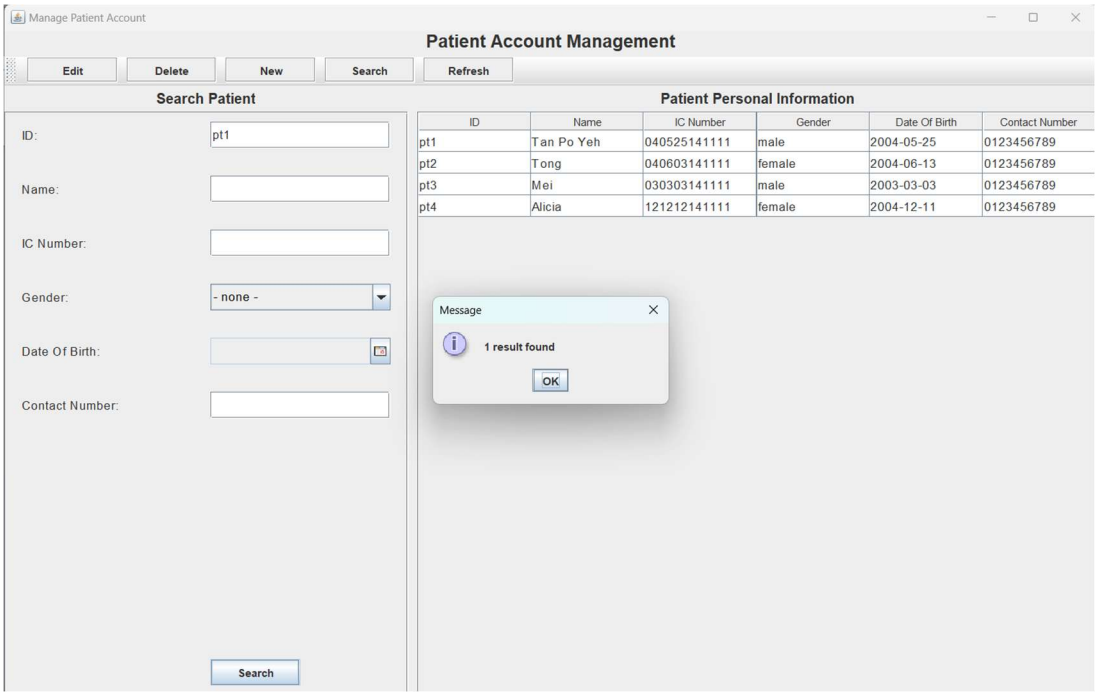


Figure 19 For example, admin input pt1 in ID input field. The system display one result found.

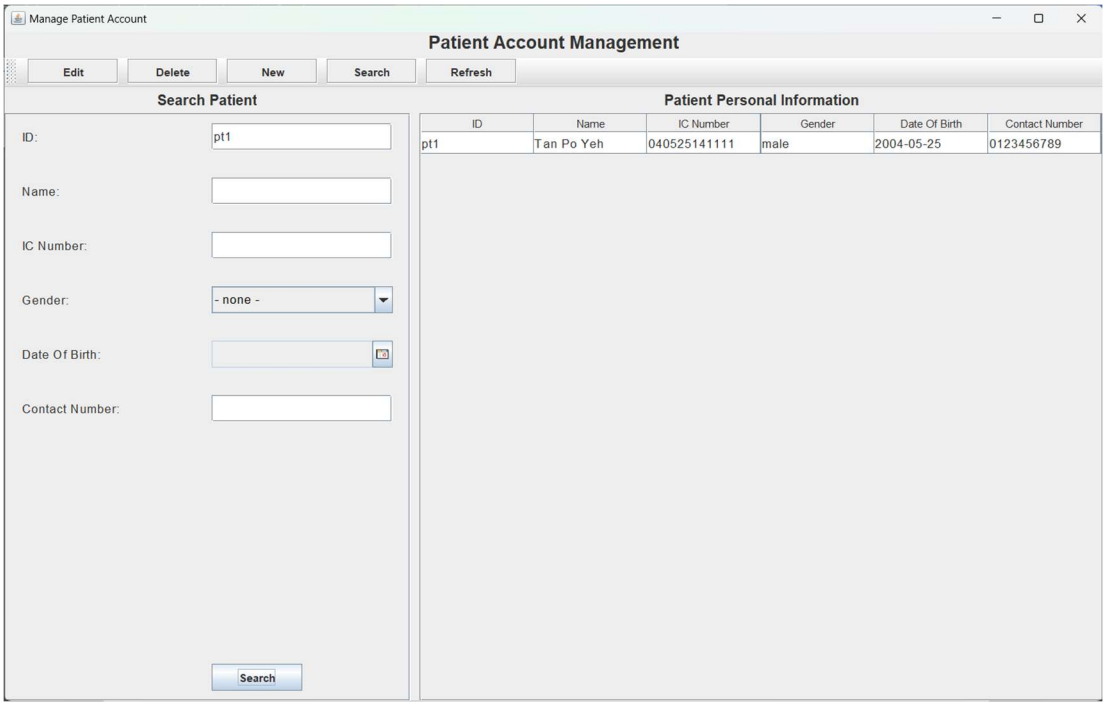


Figure 20 Only patient which contain pt1 ID is displayed in the table

Functions such as Doctor Account management, Walk-In Appointment management, Medicine management, Patient Medical Record Tracking and Payment management also have the features of Create, Read, Update and Delete (CRUD). The operation of CRUD includes input validation and data redundancy detection to ensure the consistency and accuracy of data. However, admin has view-only authorization for available time slots.

2.2.2 Manage Doctor Account

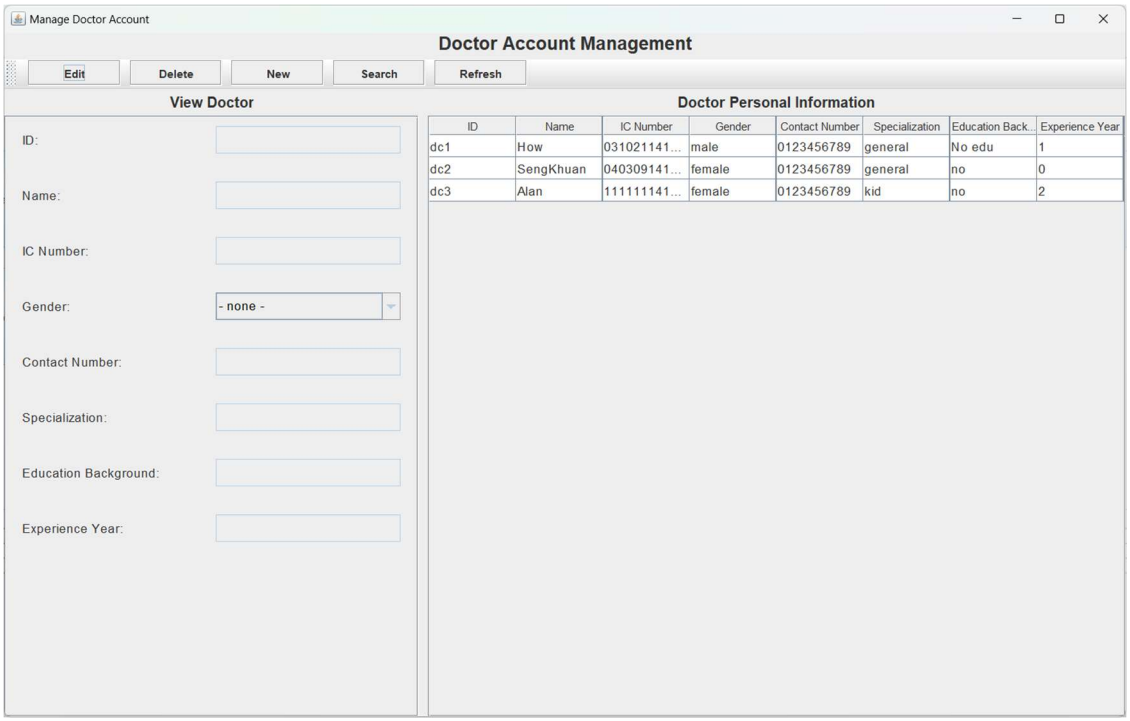


Figure 21 Doctor Account Management page

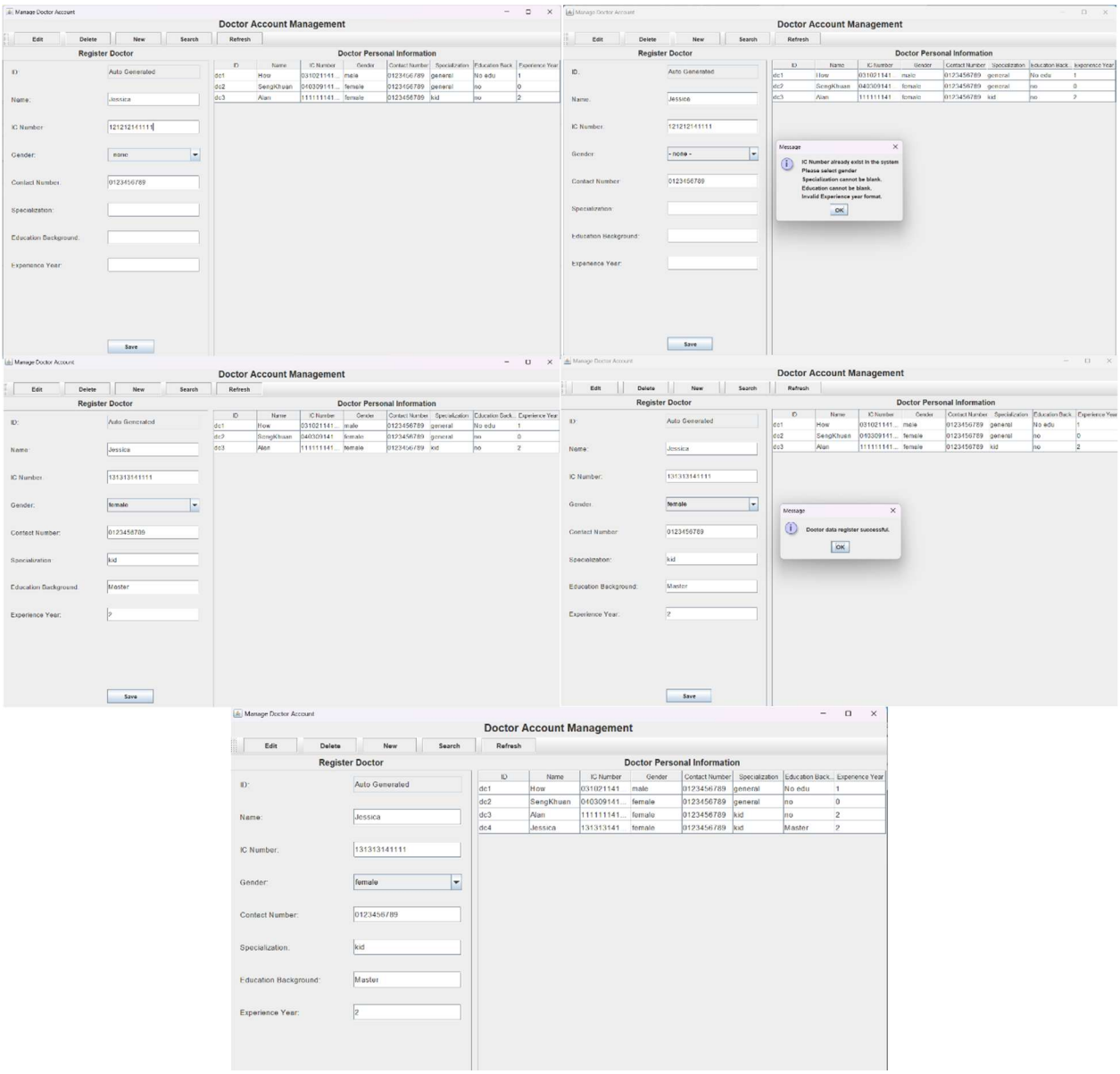


Figure 22 create new doctor account

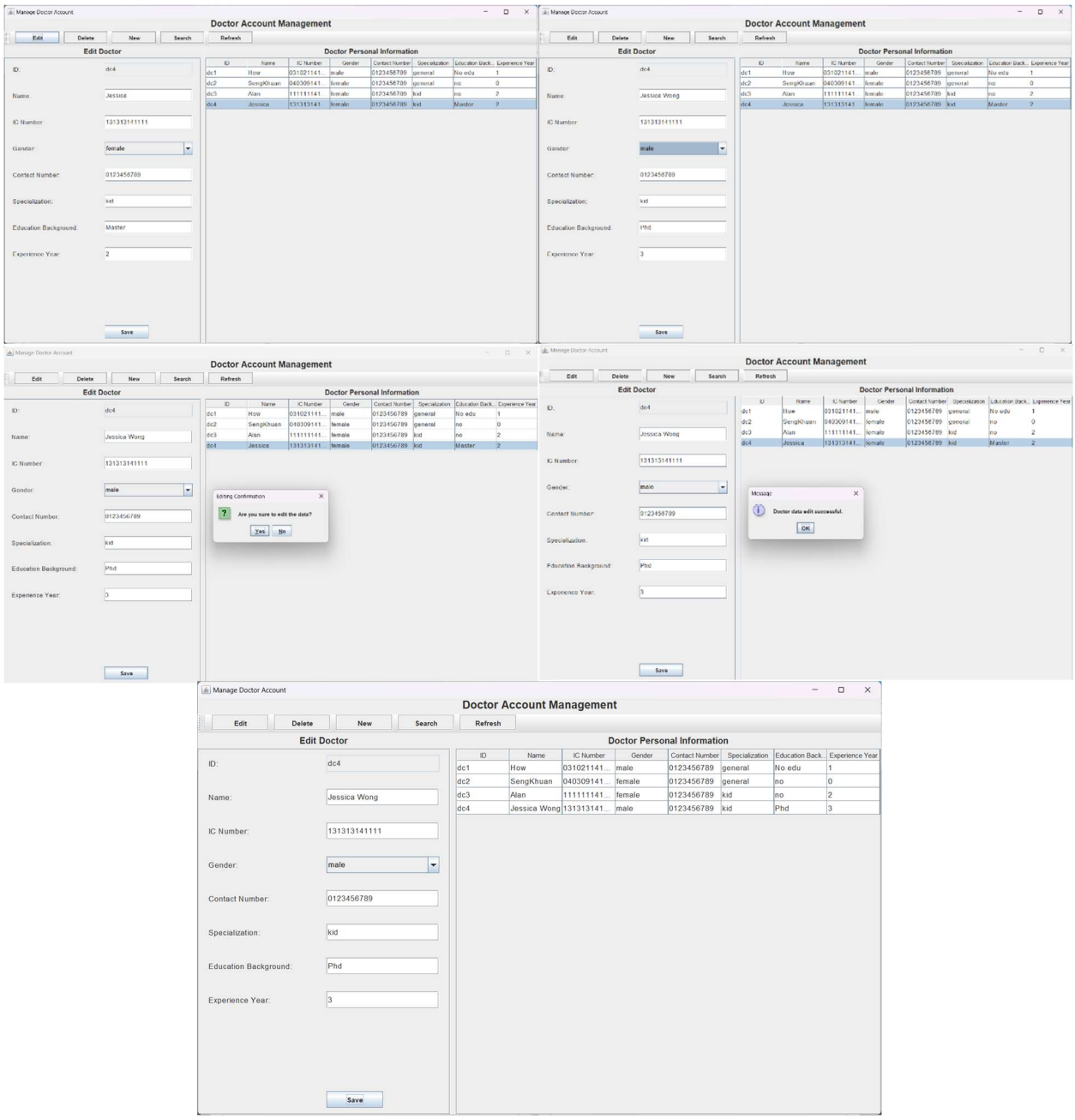


Figure 23 Edit doctor's account

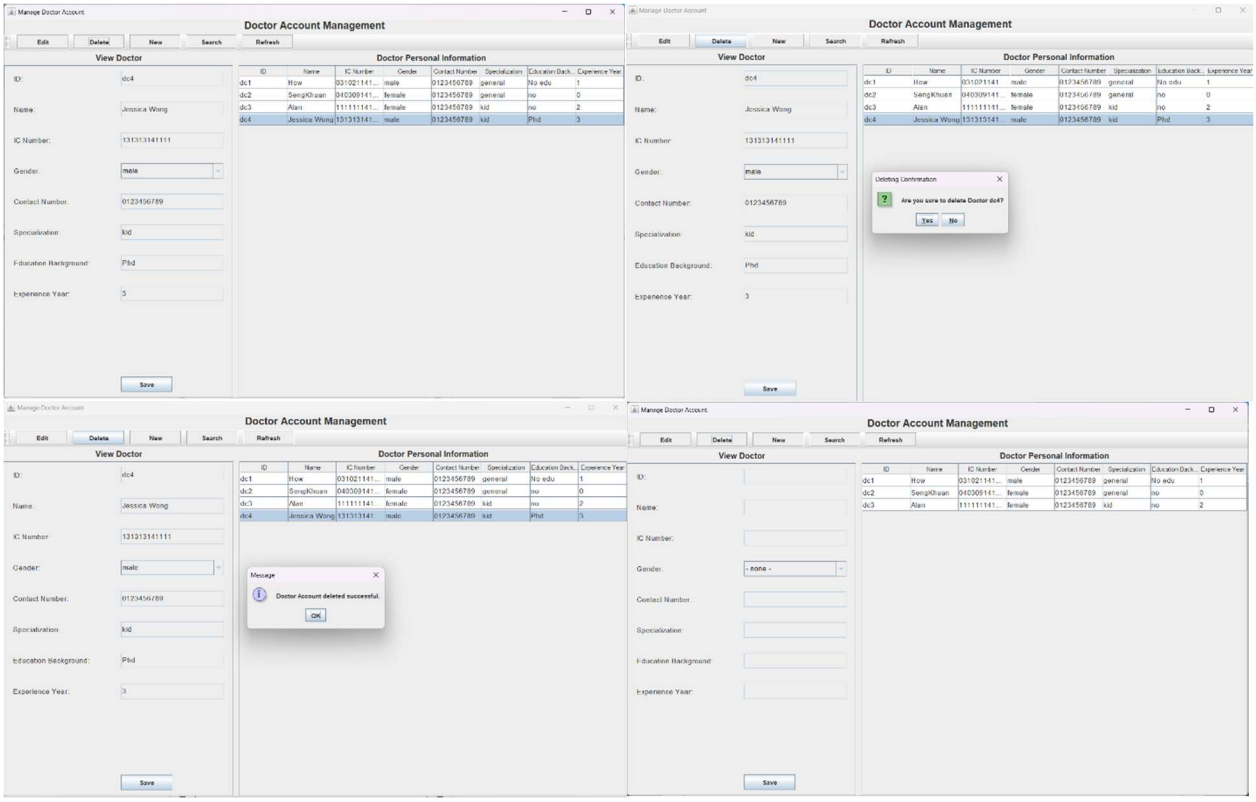


Figure 24 Delete doctor's account

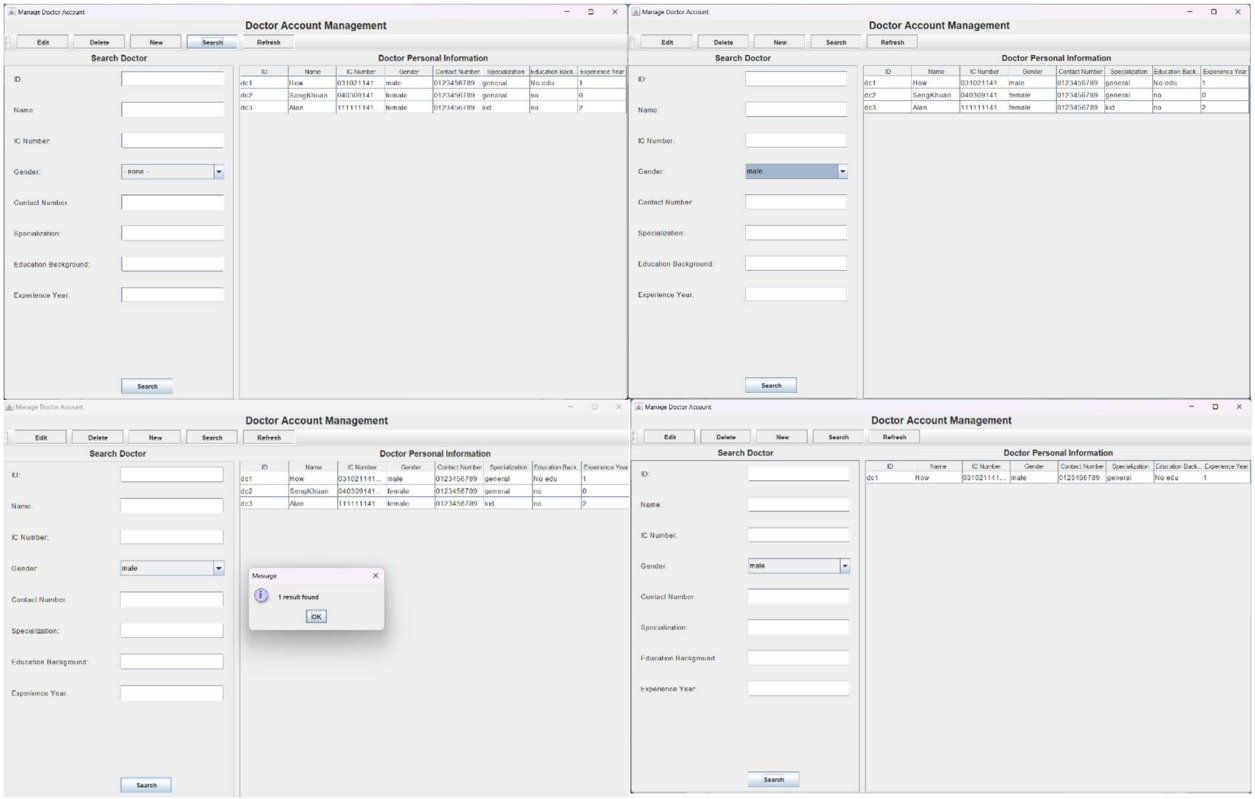


Figure 25 Search Doctor's account

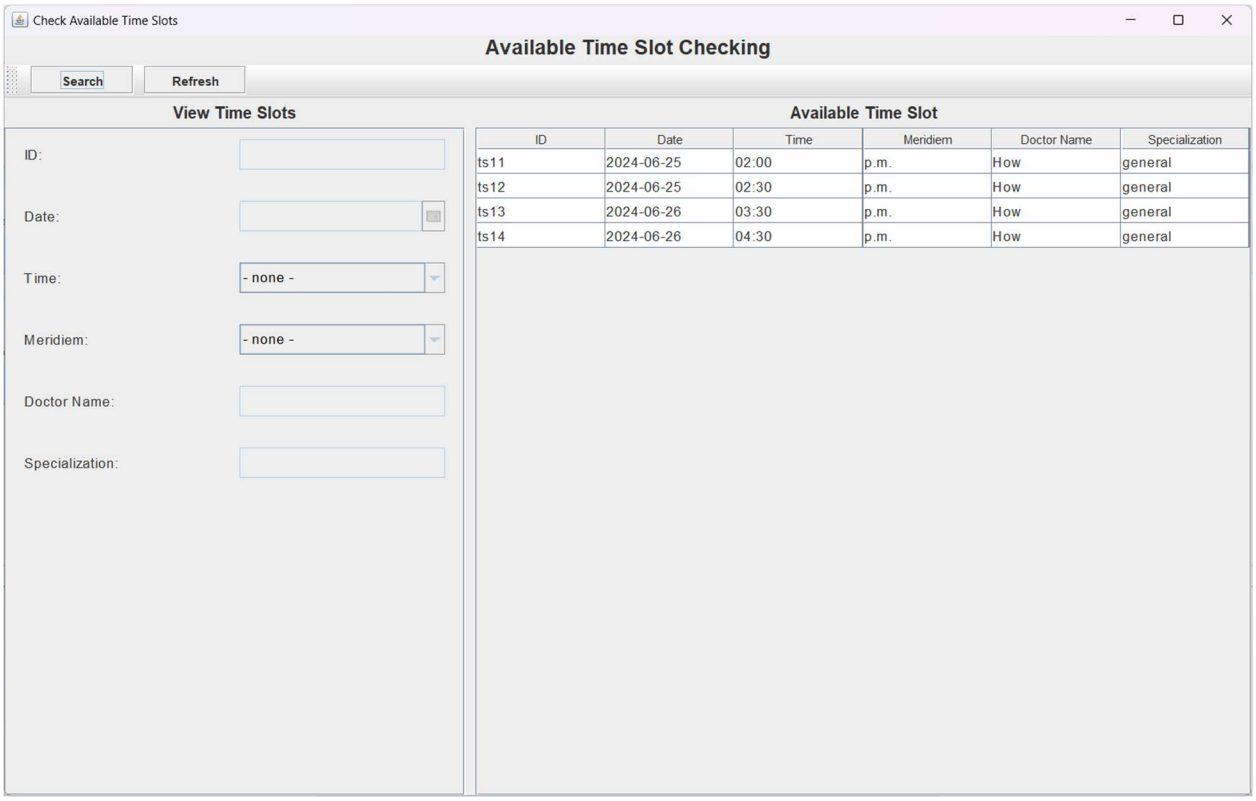


Figure 26 Available Time Slot Page

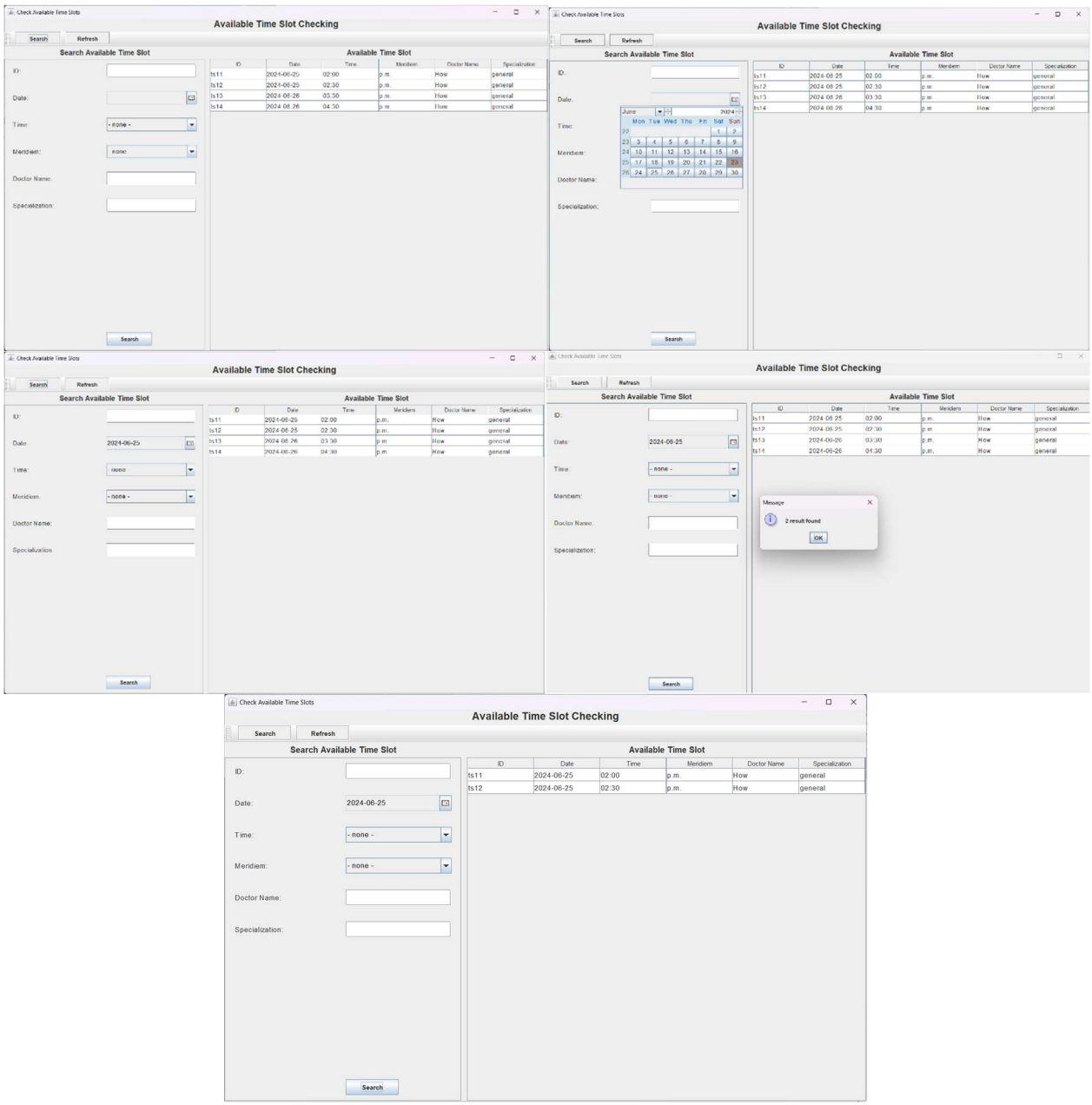


Figure 27 Search available time slots

2.2.3 Manage Walk-In Appointments

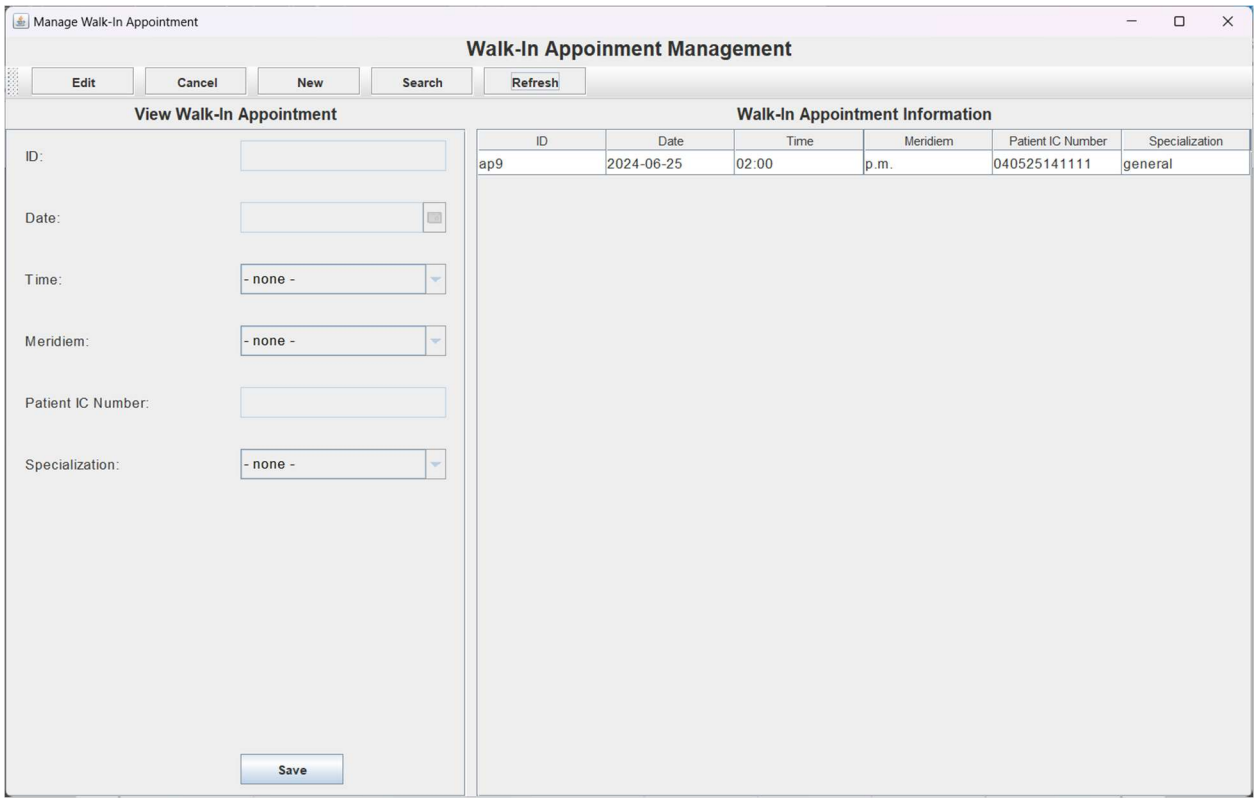


Figure 28 Walk-In Appointment Page

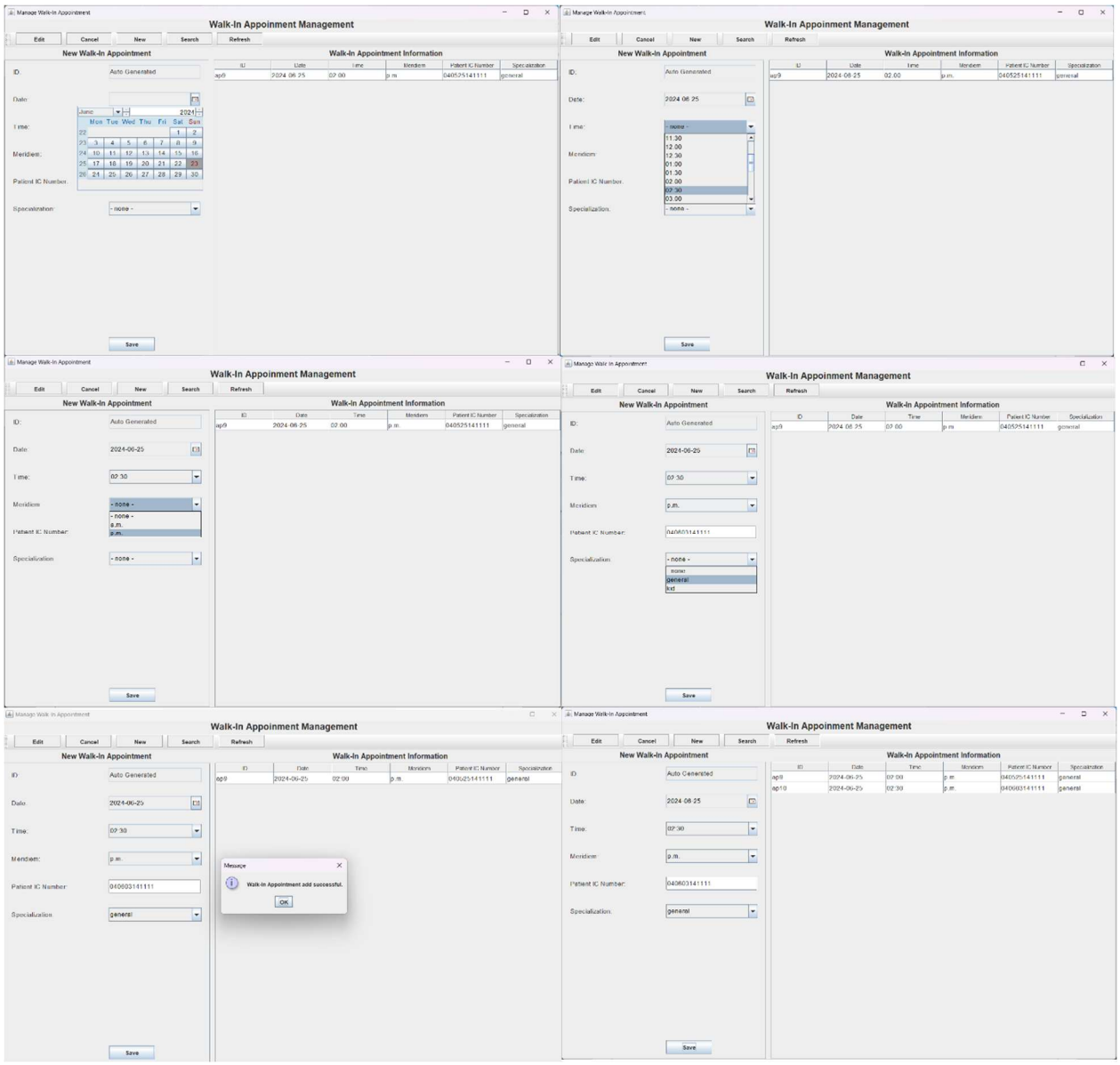


Figure 29 Make new walk-in appointment

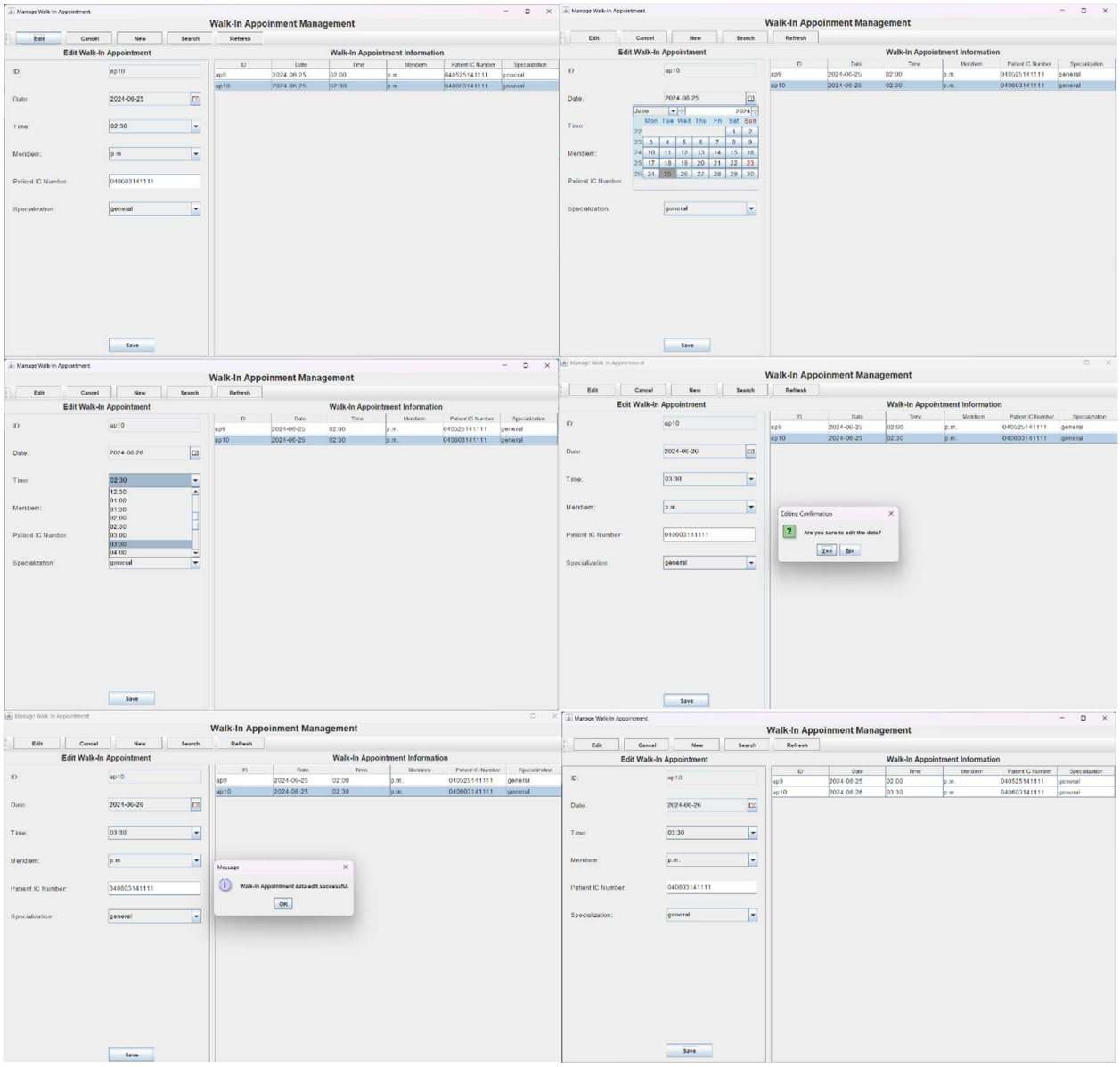


Figure 30 Edit walk-in appointment

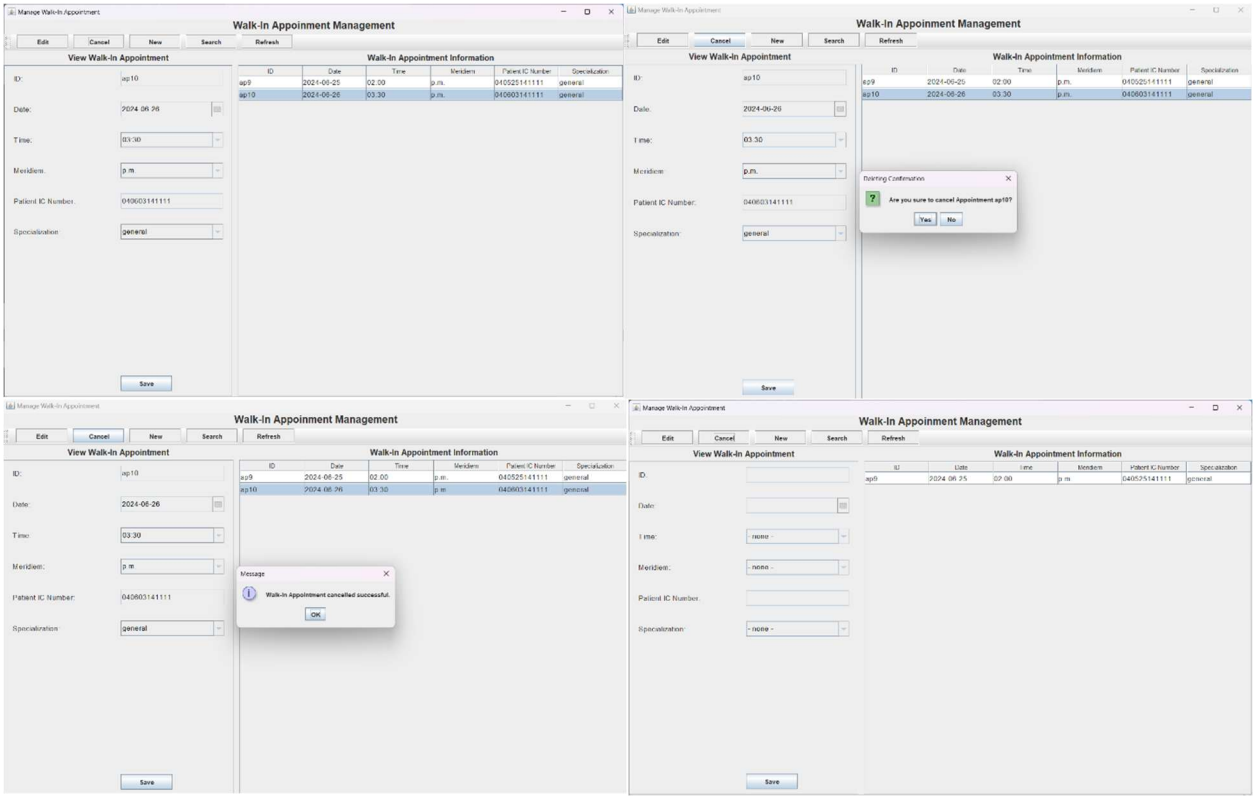


Figure 31 Cancel walk-in appointment

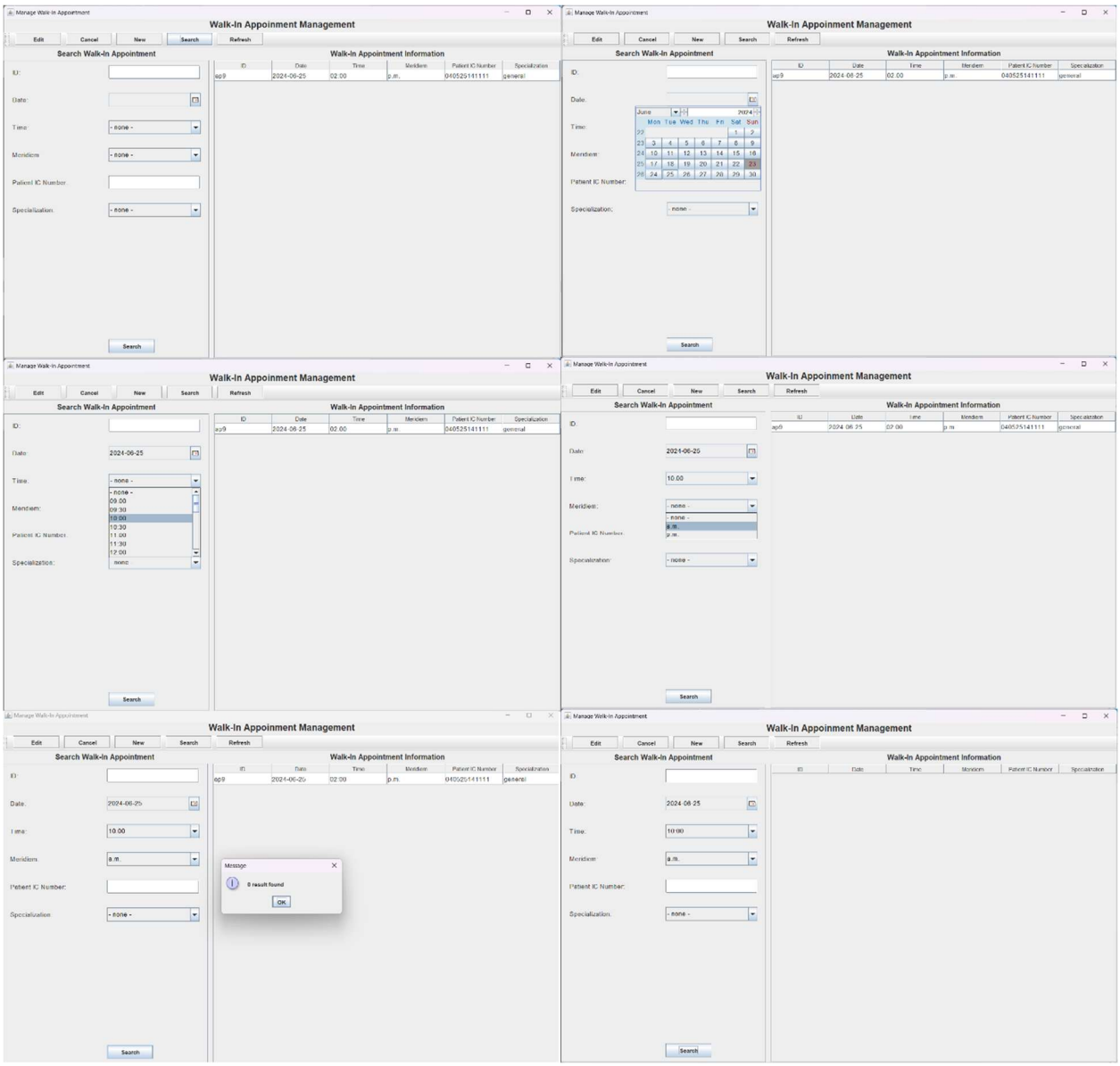


Figure 32 Search walk-in appointment

2.2.4 Track Patient Medical Record

Figure 33 Patient Medical Record Tracking Page

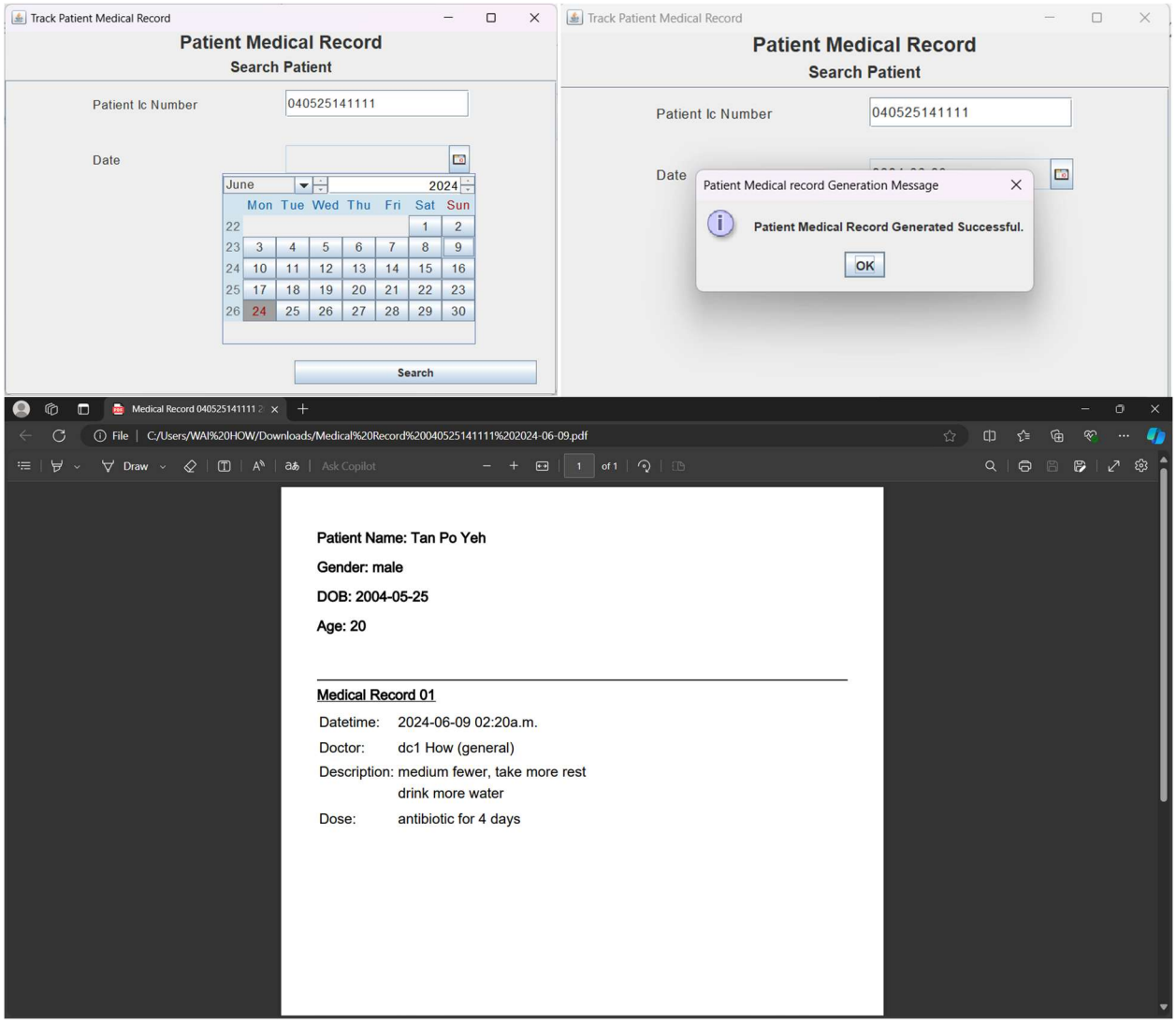


Figure 34 Track patient medical record

2.2.5 Manage Medicine

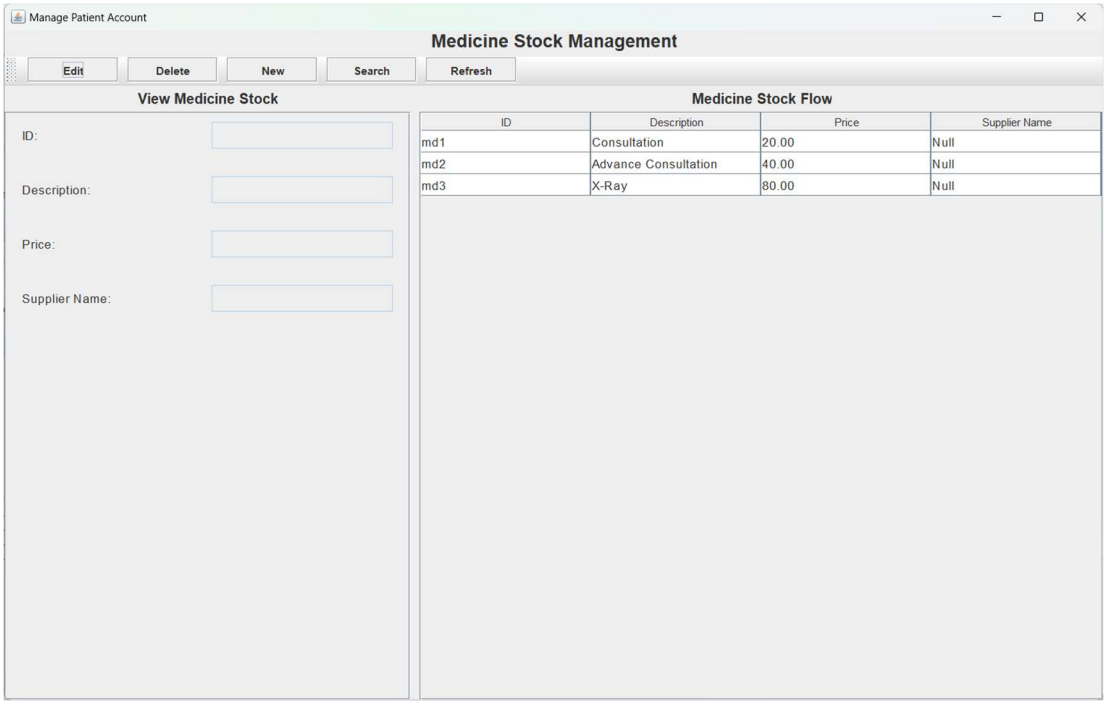


Figure 35 Medicine Management page

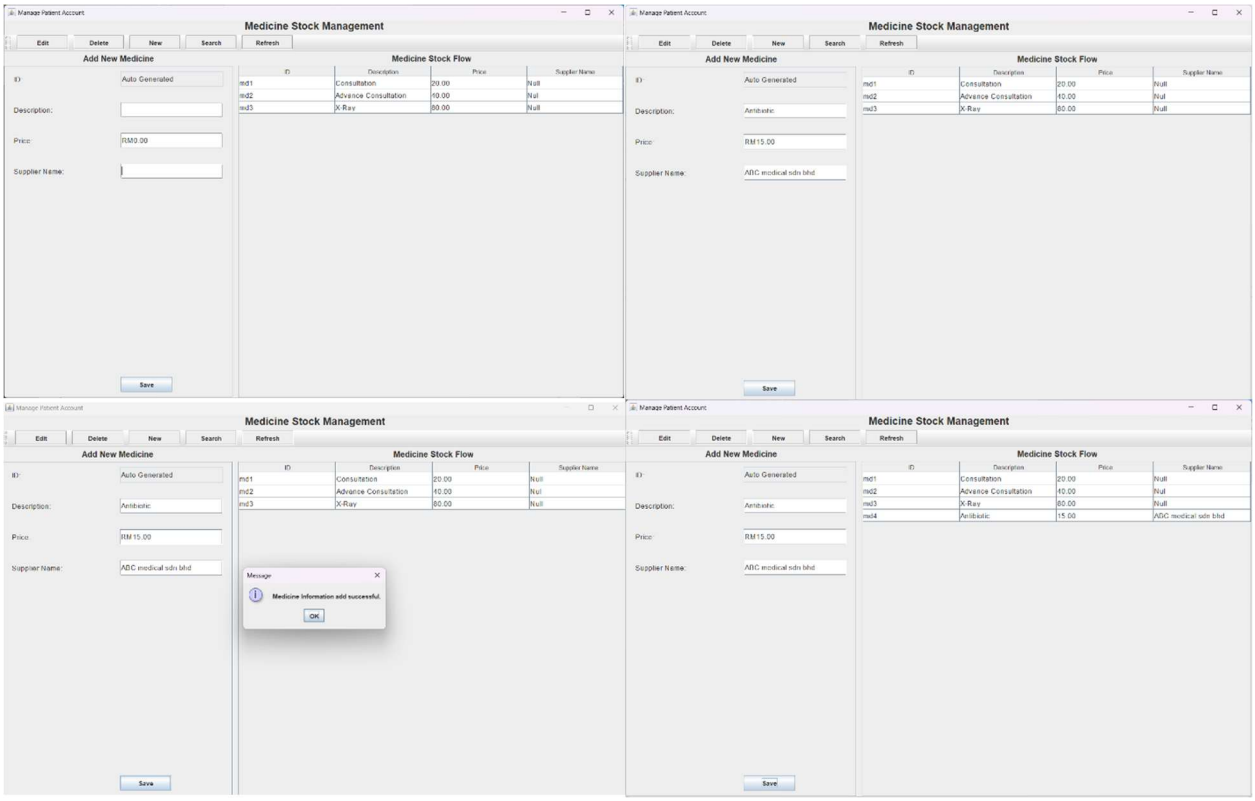


Figure 36 Add new medicine

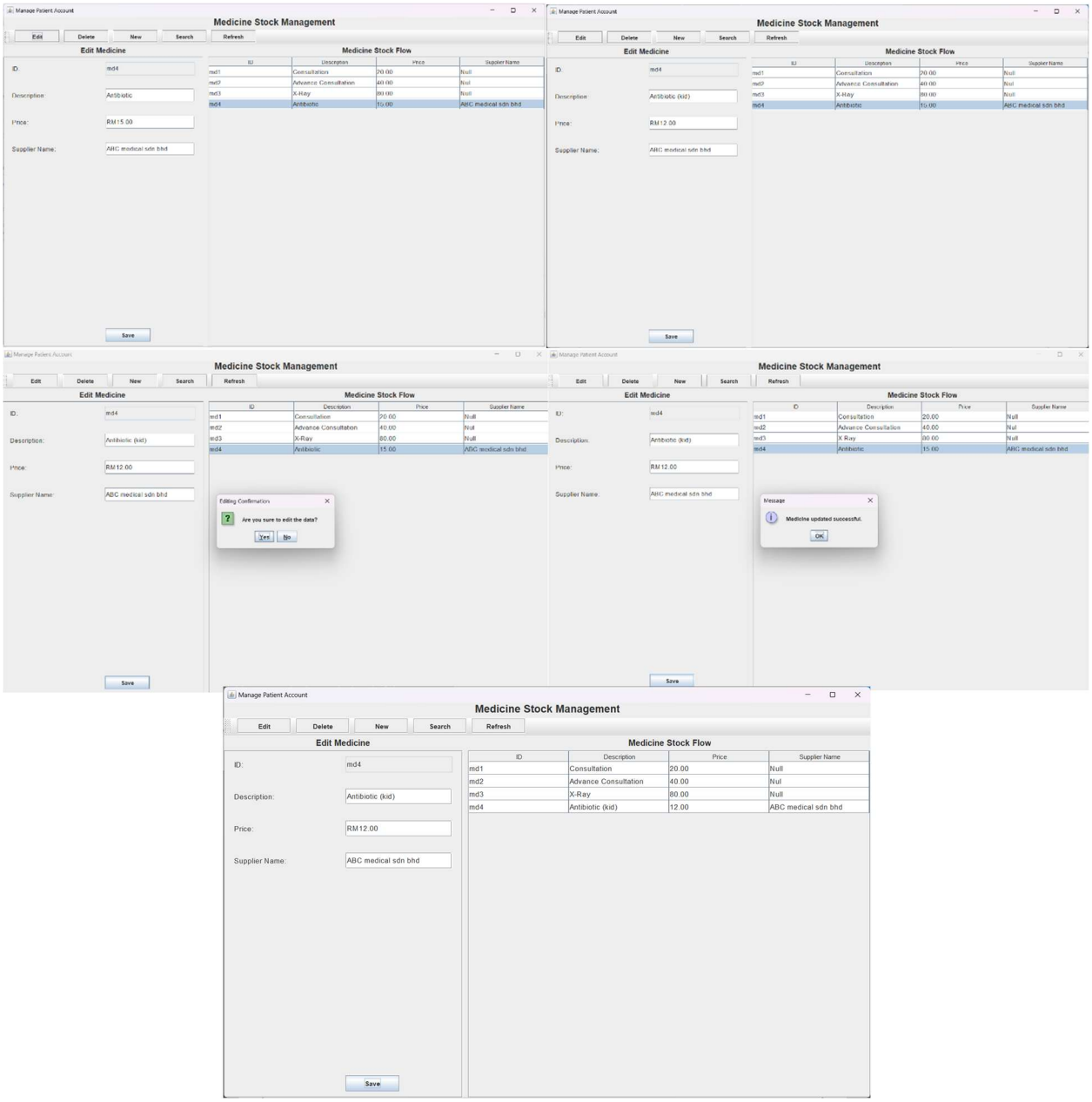


Figure 37 Edit medicine

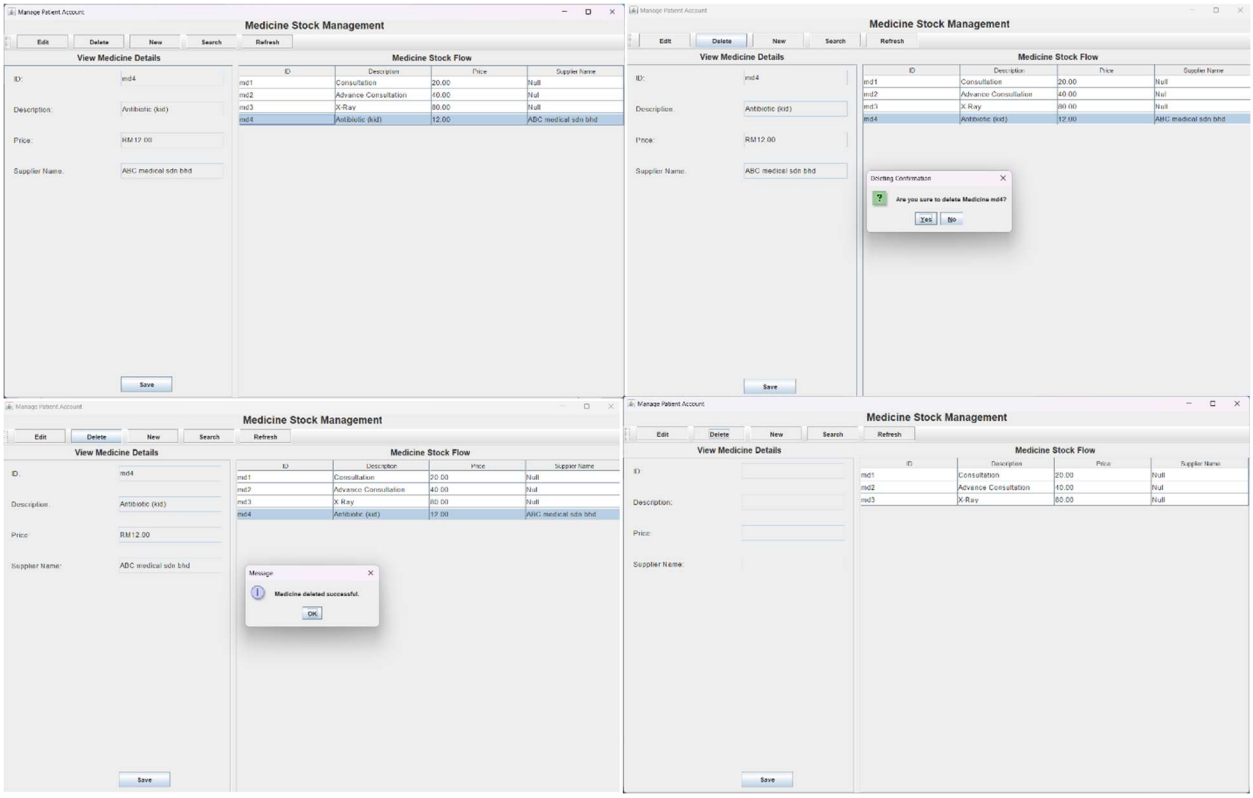


Figure 38 Delete medicine

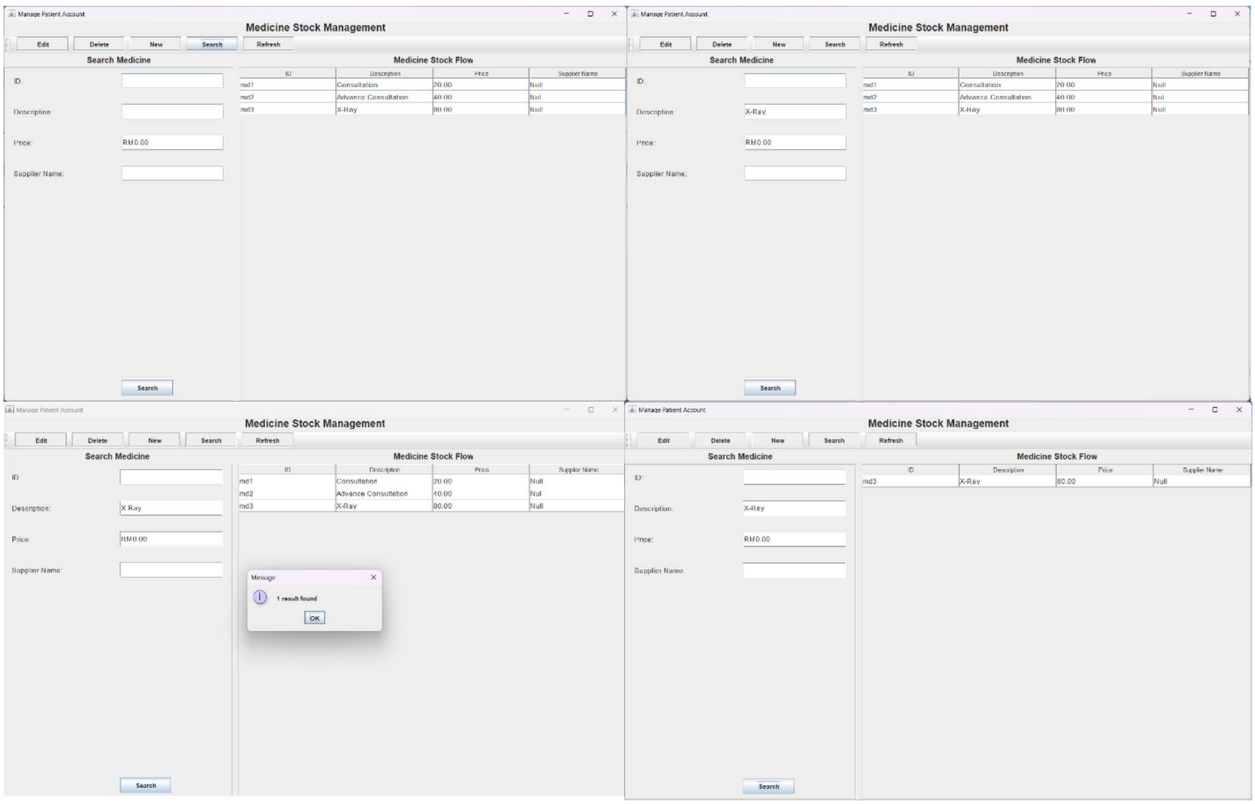


Figure 39 Search medicine

2.2.6 Manage Payment

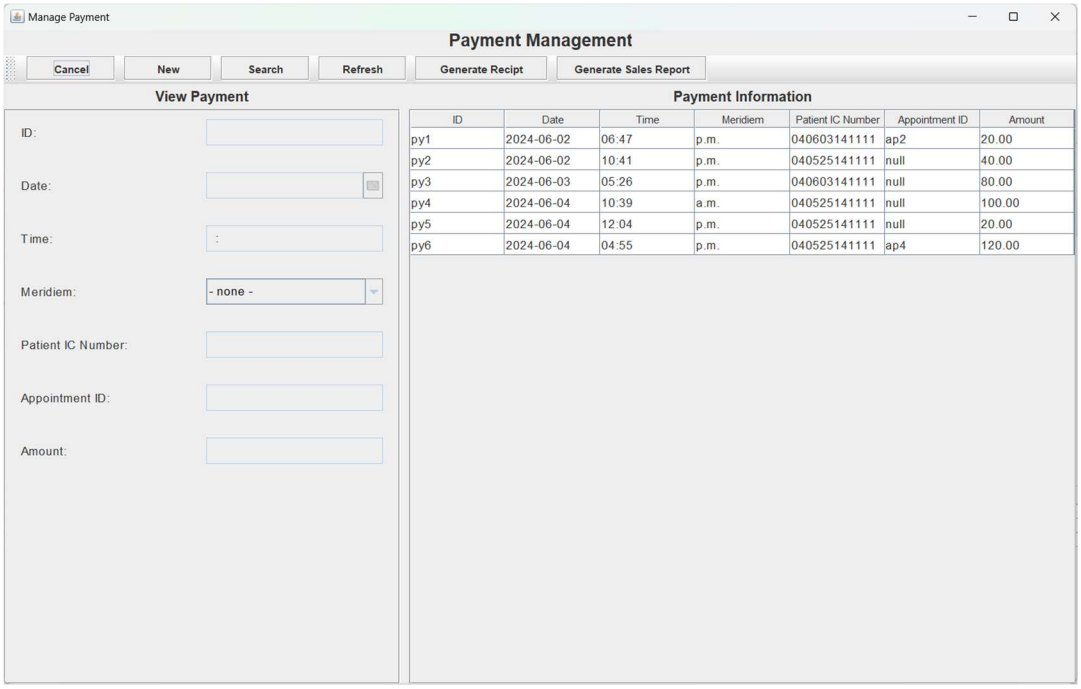


Figure 40 Payment Management page

We integrate some features of Point of Sales system into the payment process:

1. **Enter Payer's IC Number**: Admins need to input the payer's identification number.
2. **Select Appointment**: Choose the appropriate appointment linked to the payment.
3. **Select Items**: Admins can select up to five medicines or items to include in the payment.
4. **Generate Invoice**: Click the *Generate Invoice* button to preview the payment details.
5. **Confirm Payment**: Click *Confirm* button to confirm the preview payment detail.
6. **Enter Paid Amount**: Enter the amount of the payer paid.

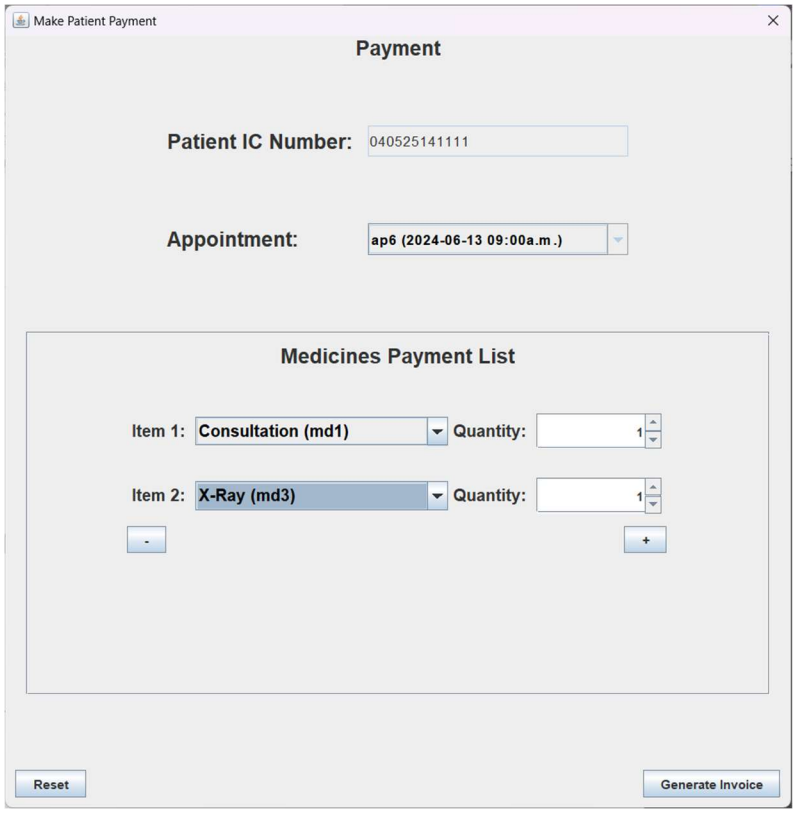


Figure 41 Make new payment

Figure 42 Preview payment details

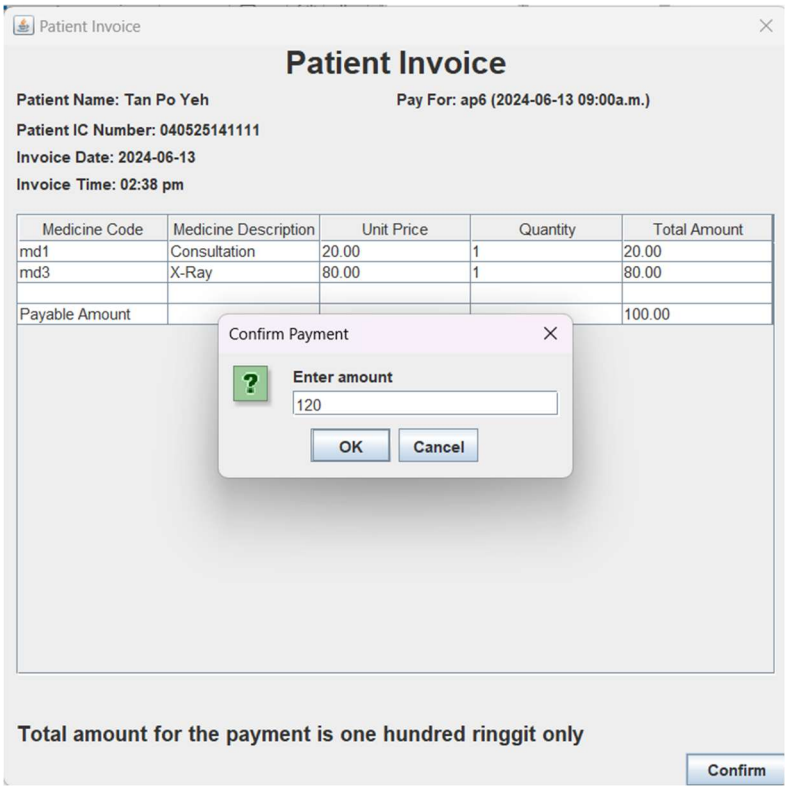


Figure 43 Enter paid amount

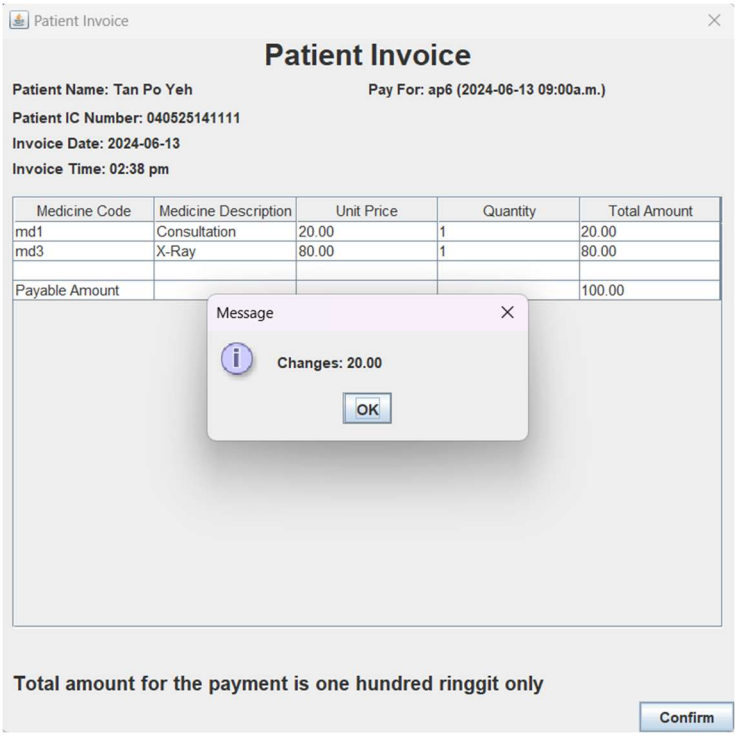


Figure 44 The system display change for the payment

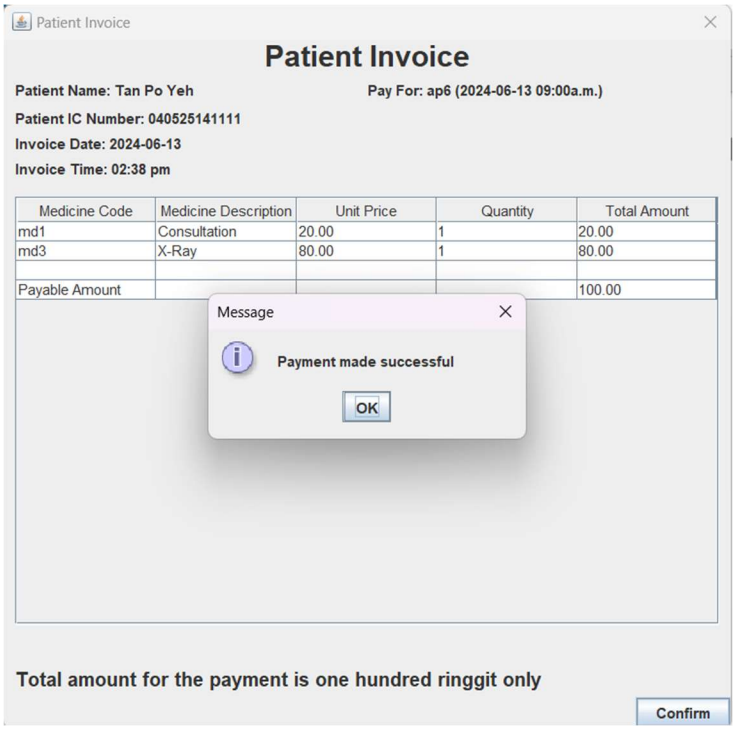


Figure 45 Payment made successfu

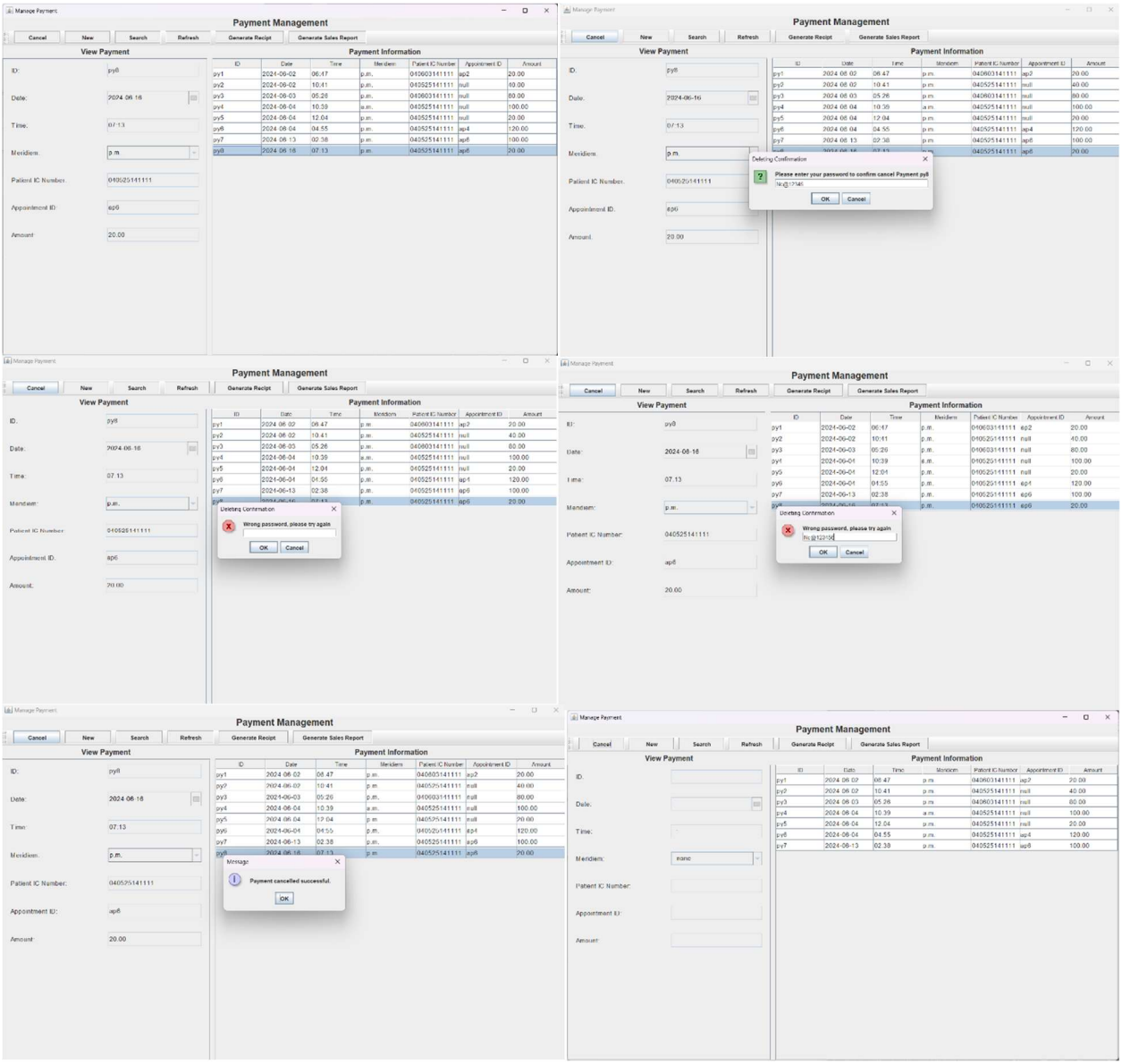


Figure 46 Cancel Payment

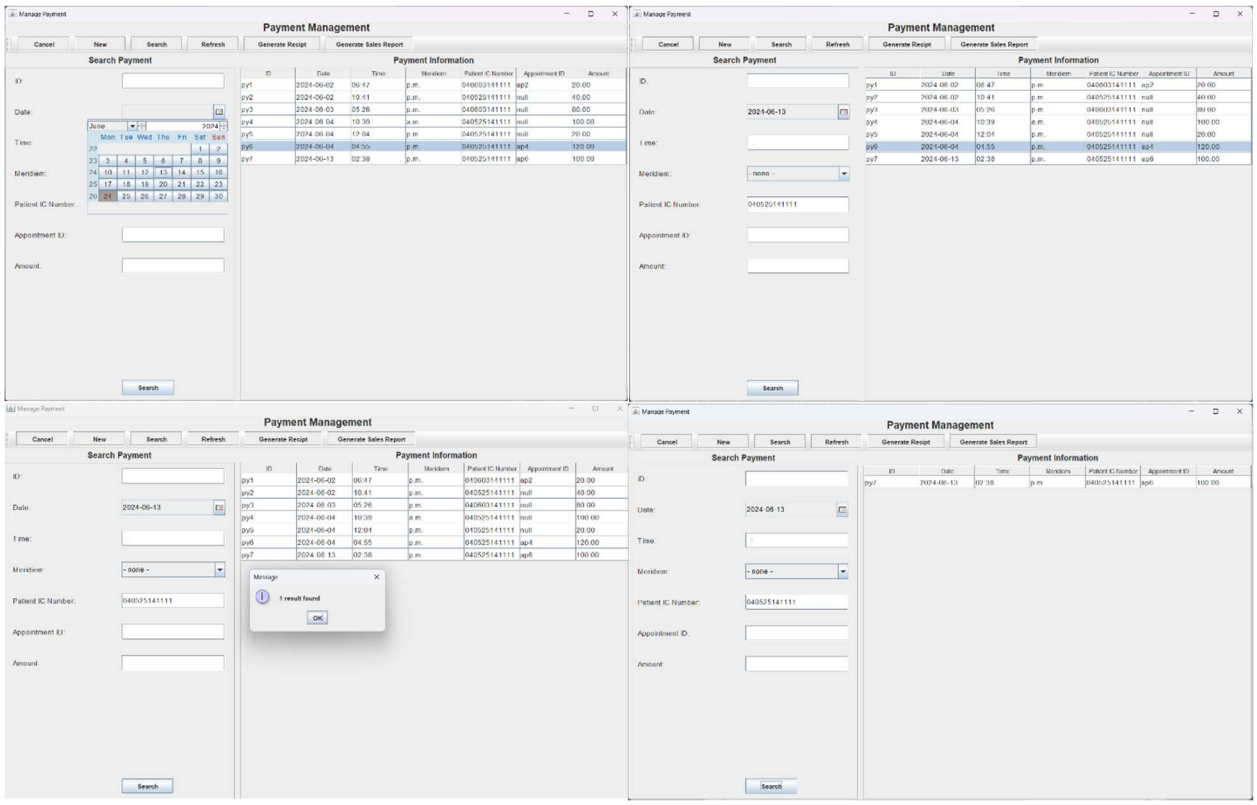


Figure 47 Search payment

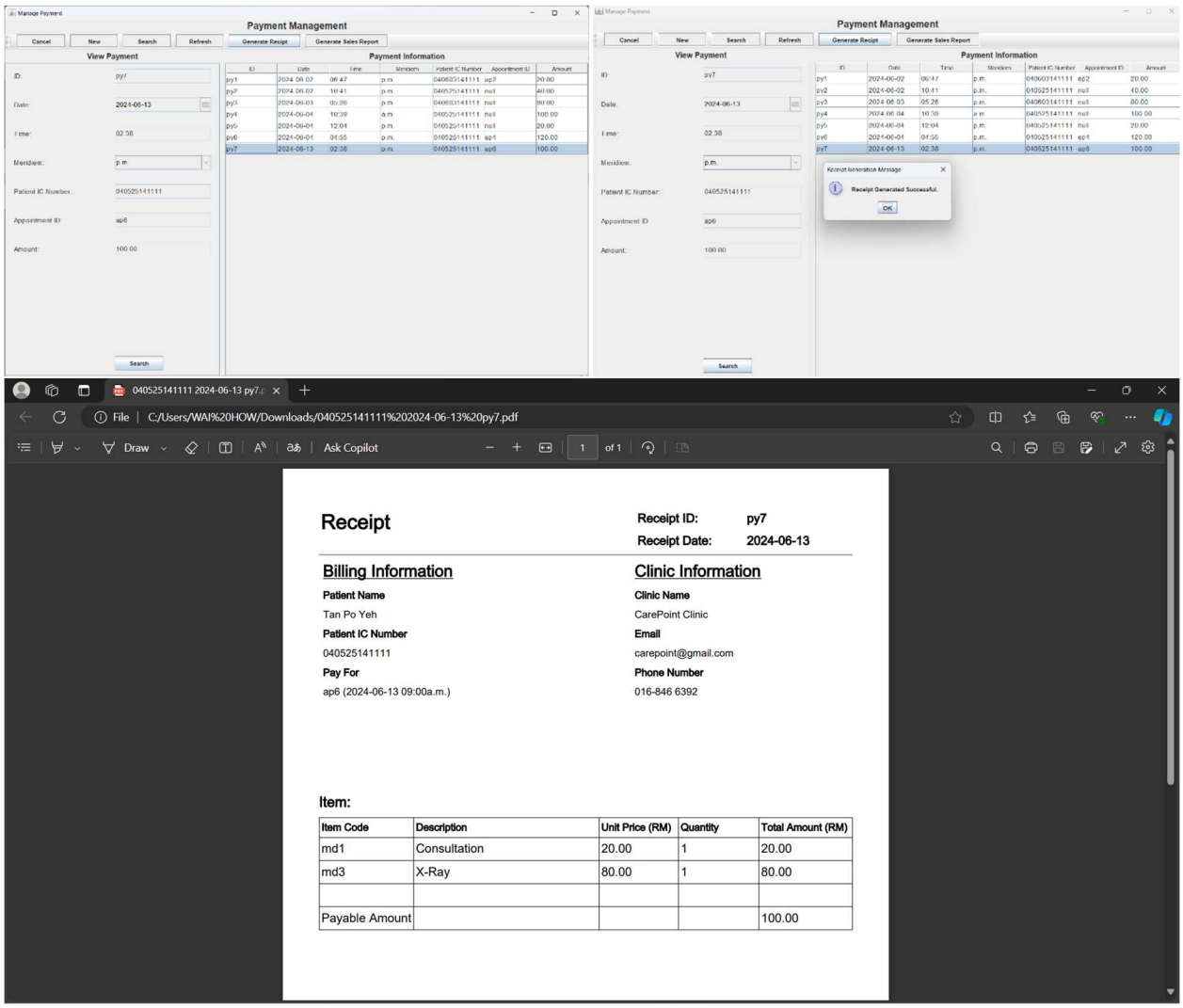


Figure 48 Generate payment receipt

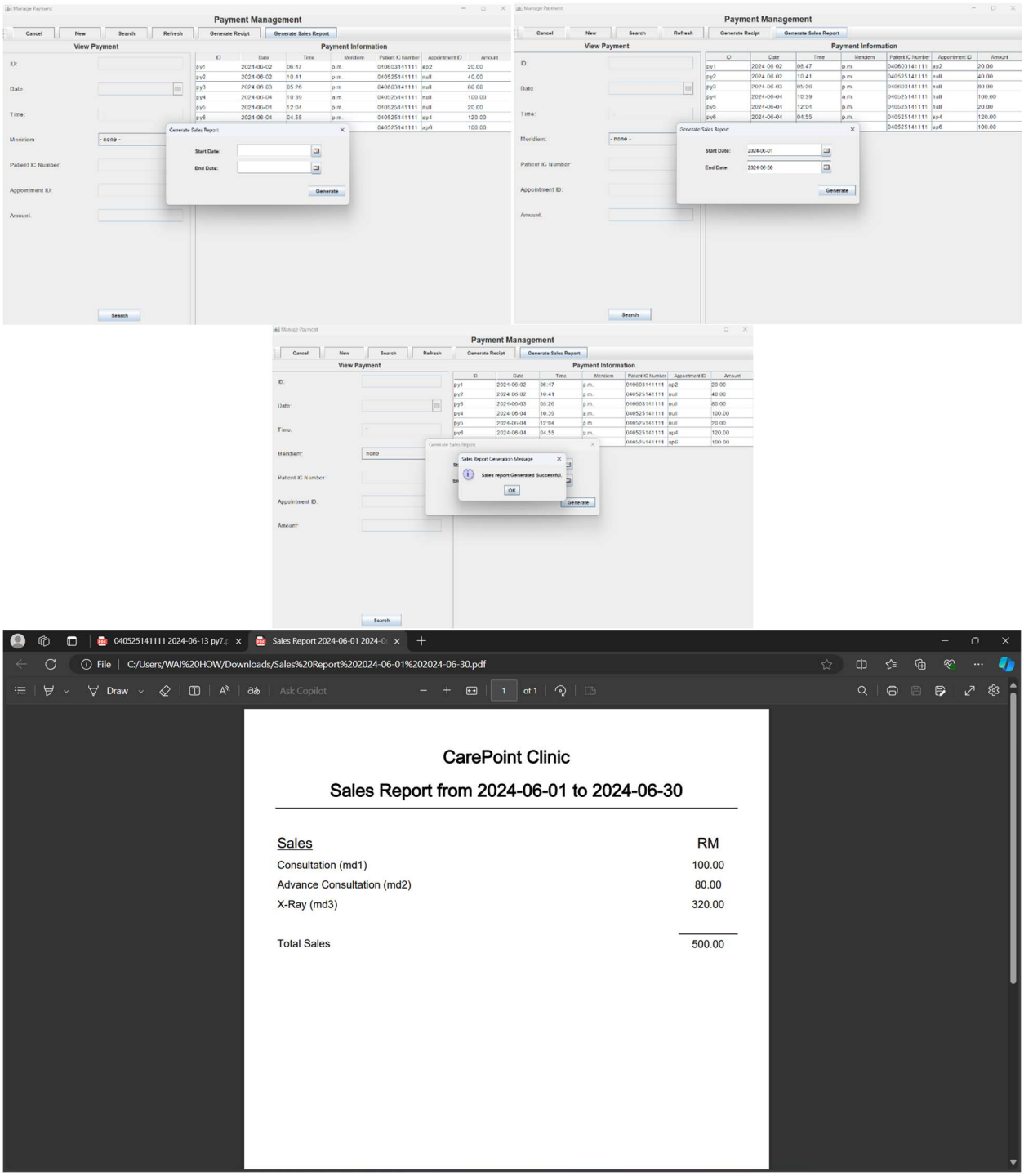


Figure 49 Generate sales report

2.2 Doctor Features

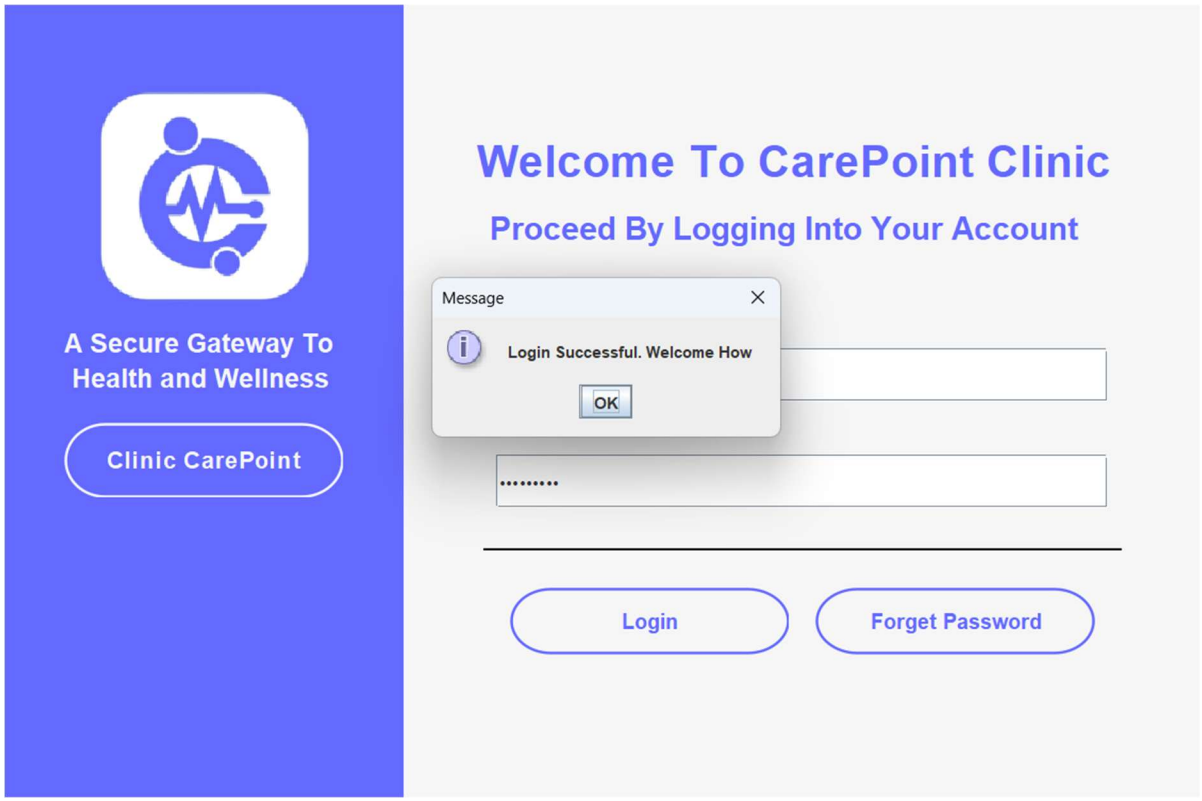


Figure 50 :Doctor sign in page

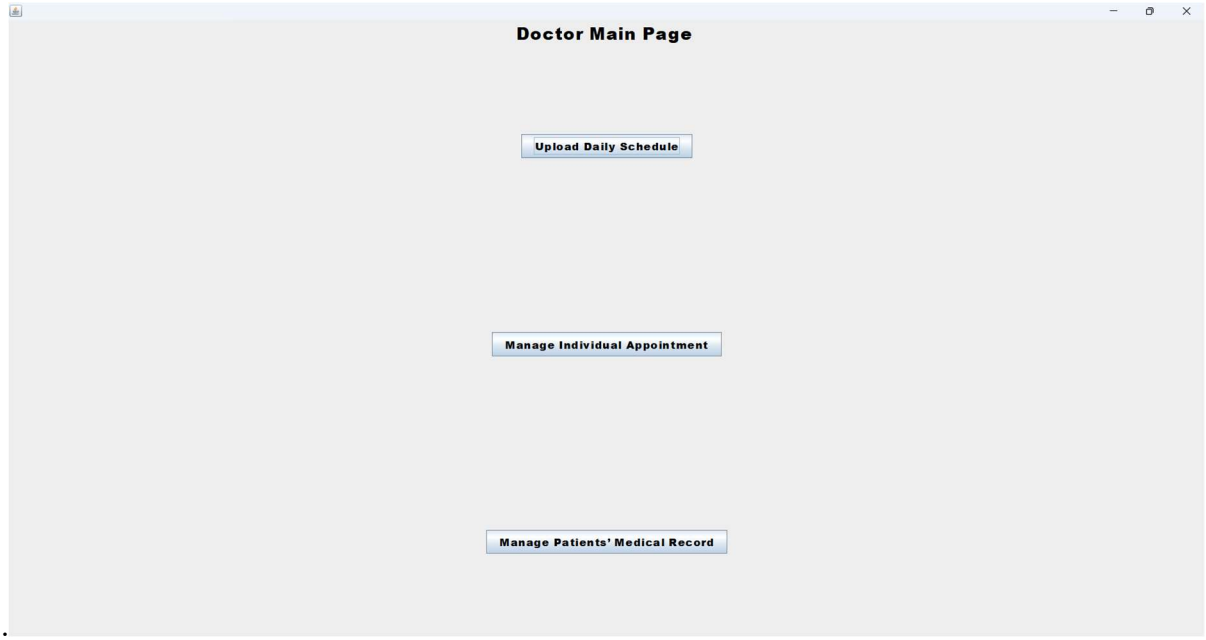


Figure 51 :Main Page for Doctor

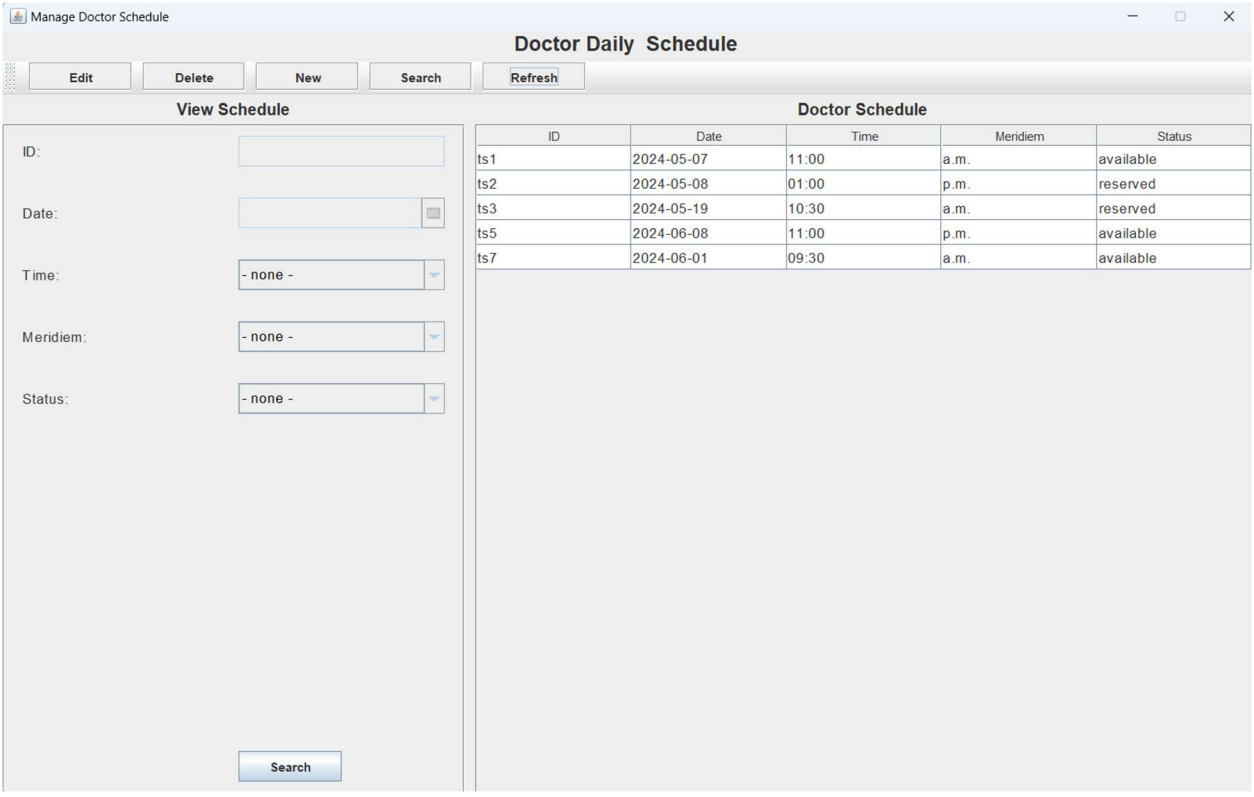


Figure 52:Update Daily Schedule for Doctor

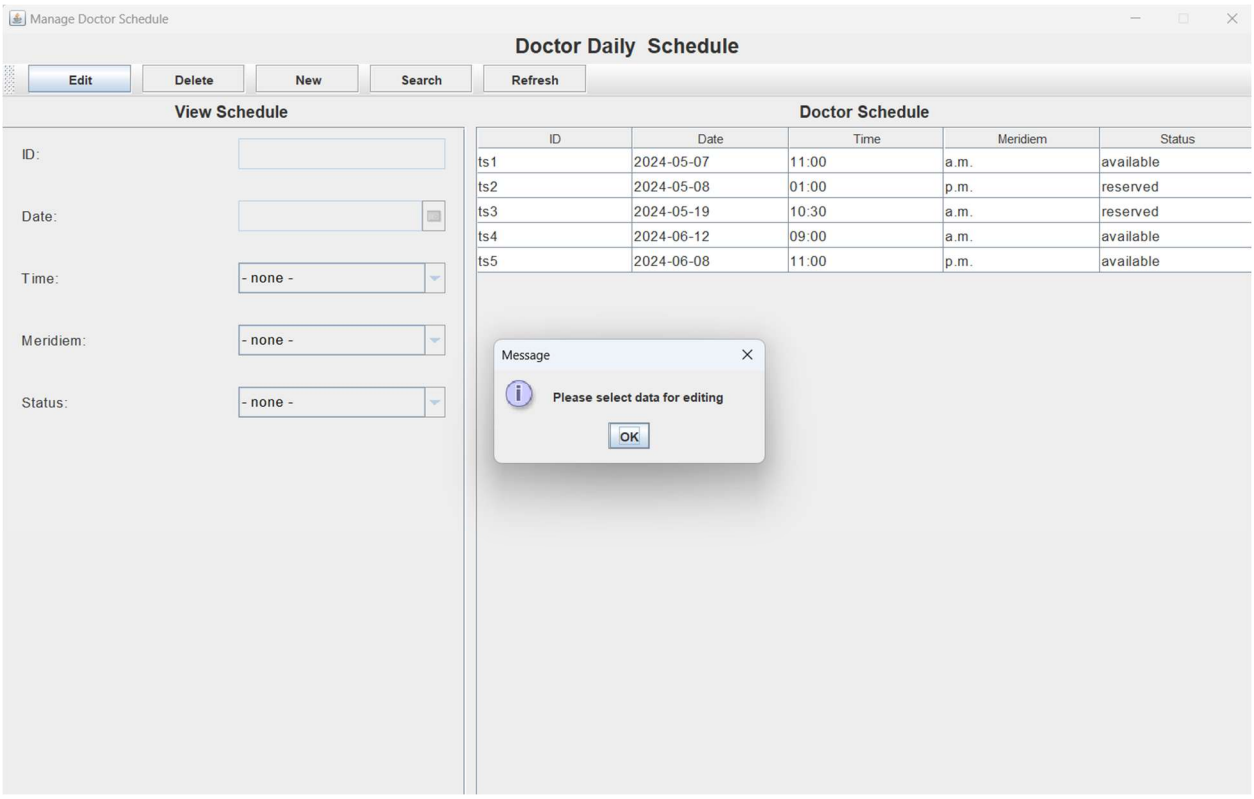


Figure 53:User must Select Data for Editing as shown above

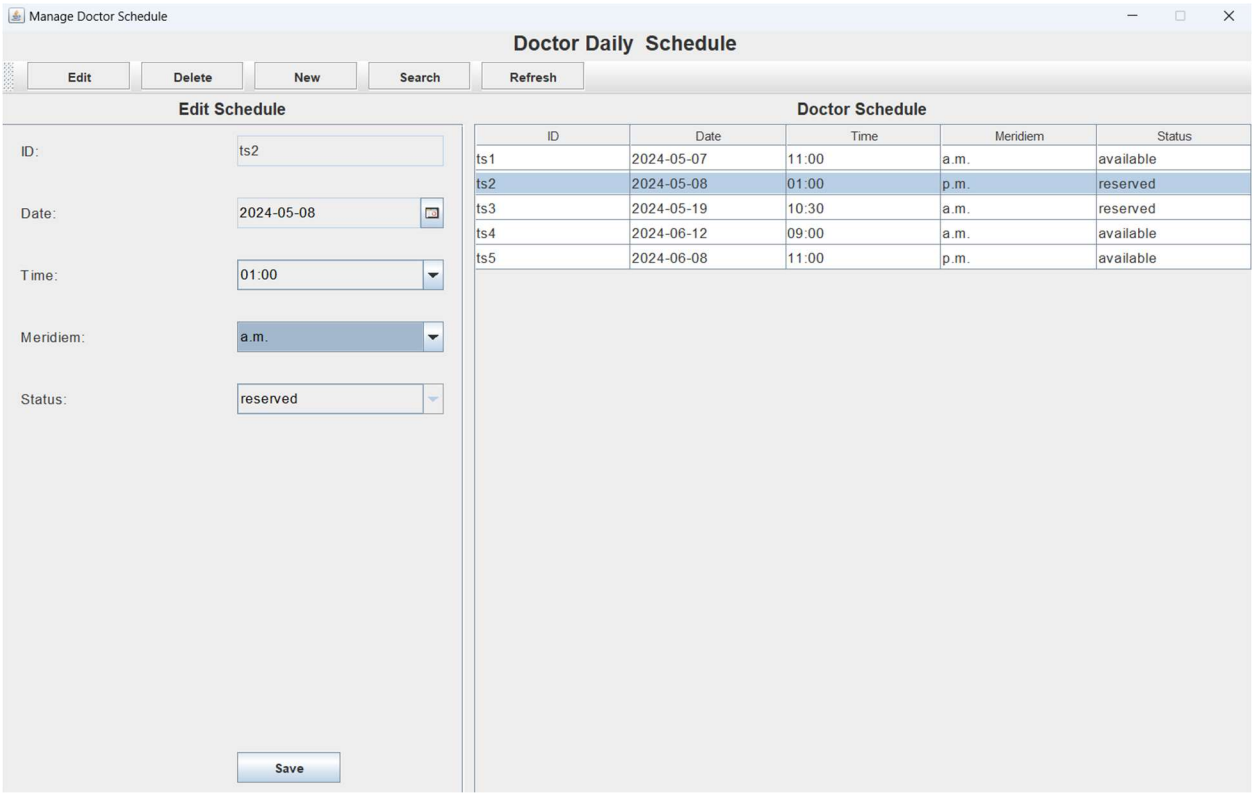


Figure 54:After selecting the data, it will then display on the left-hand side of the table

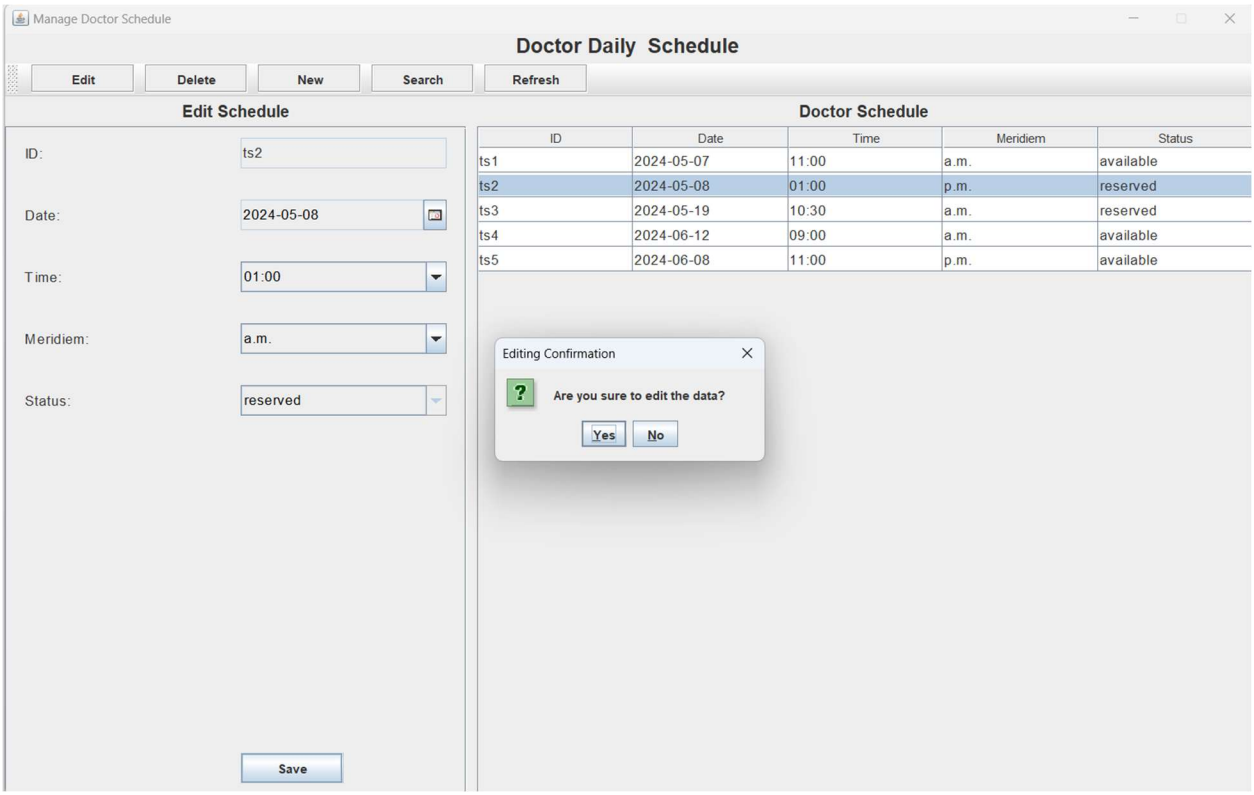


Figure 555:A pop out Message Box asking for clarification on the selected data

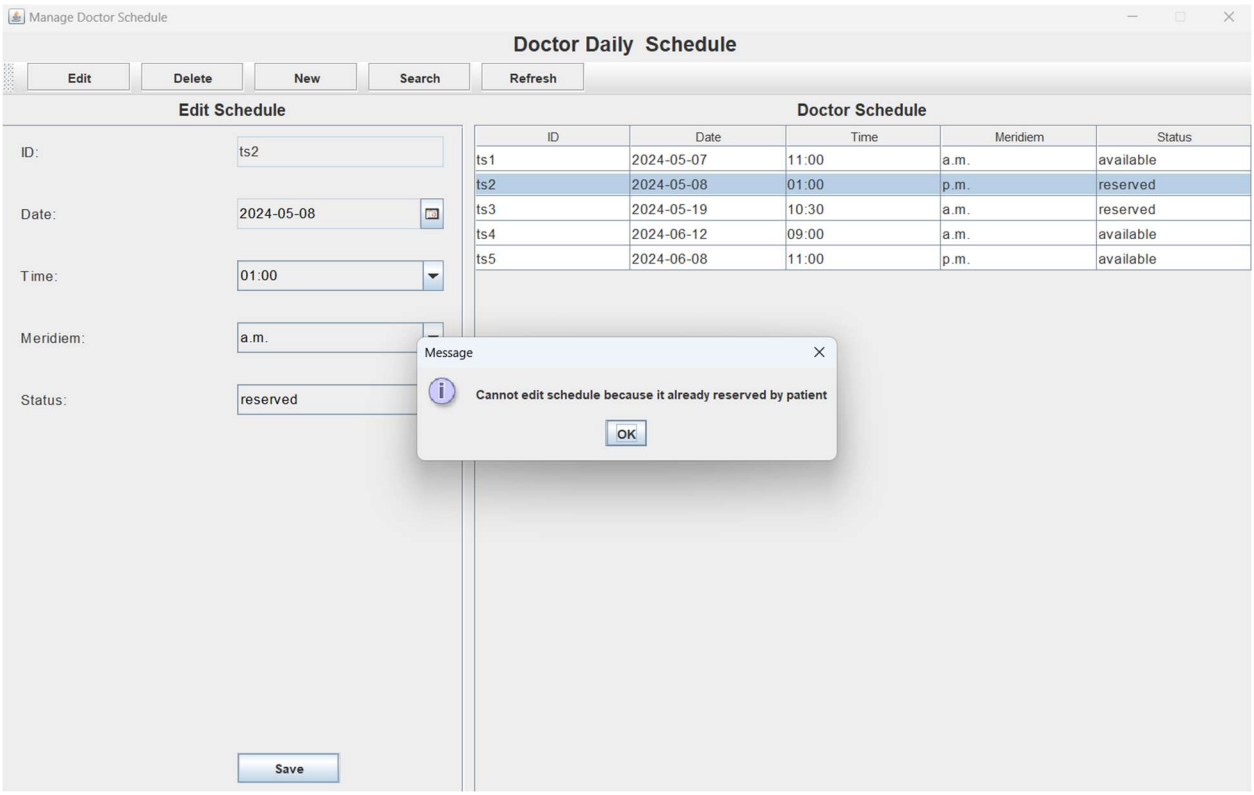


Figure 56:As u can see in the figure above, a patient who has reserved cannot be editable

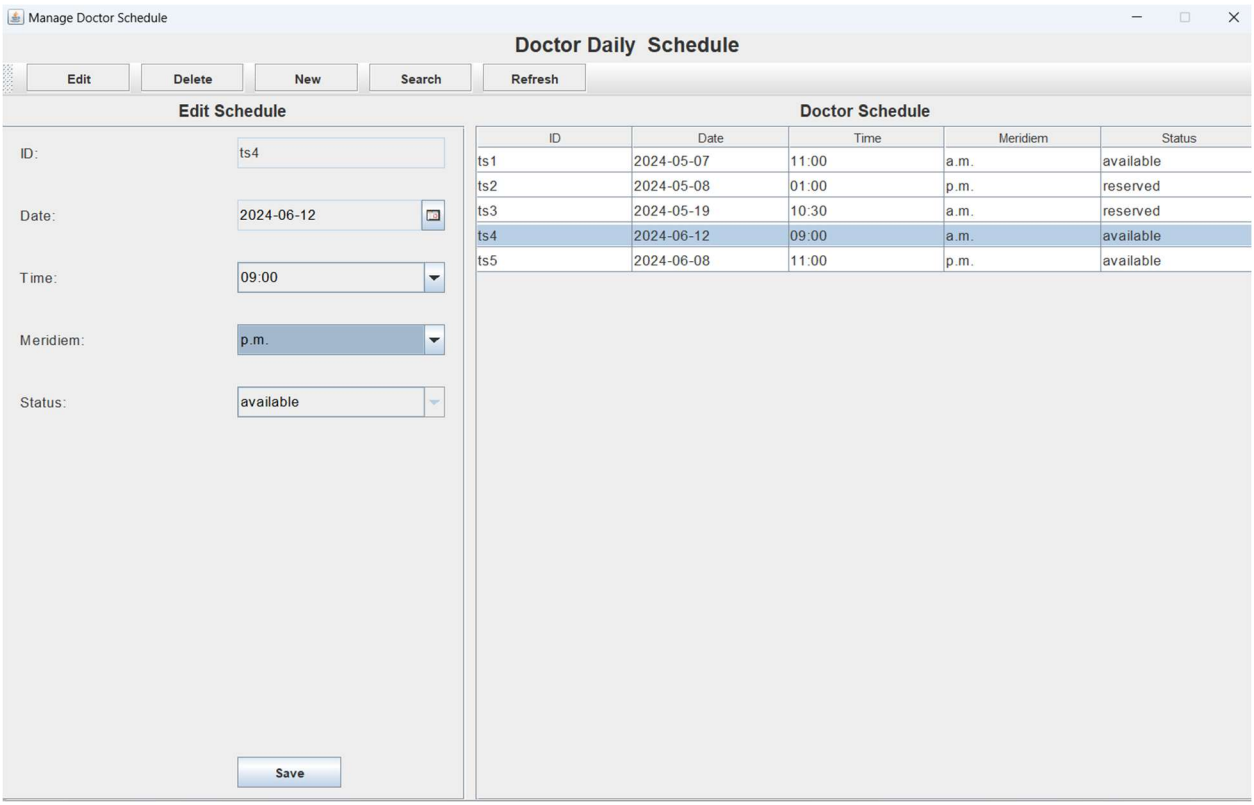


Figure 57: As u can see, the selected data which is not reserved by a patient is editable

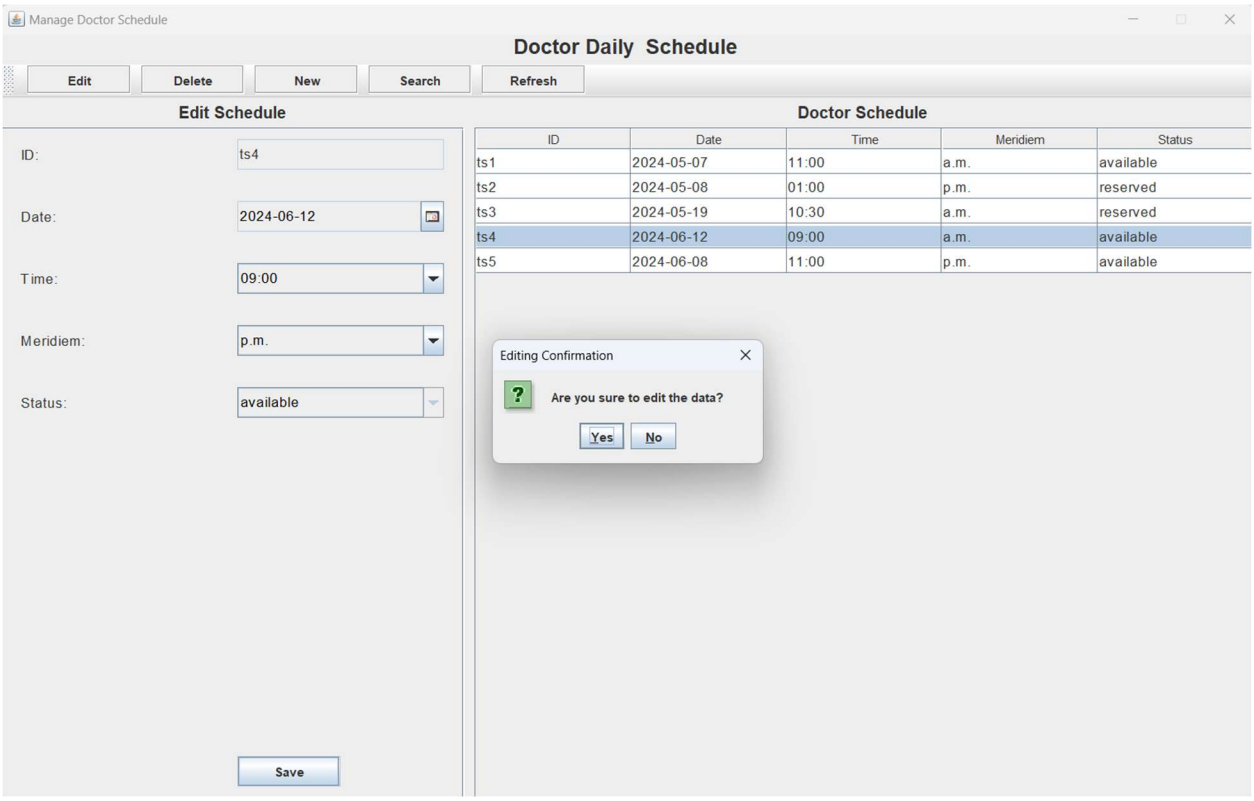


Figure 58: :A pop out Message Box asking for clarification on the selected data

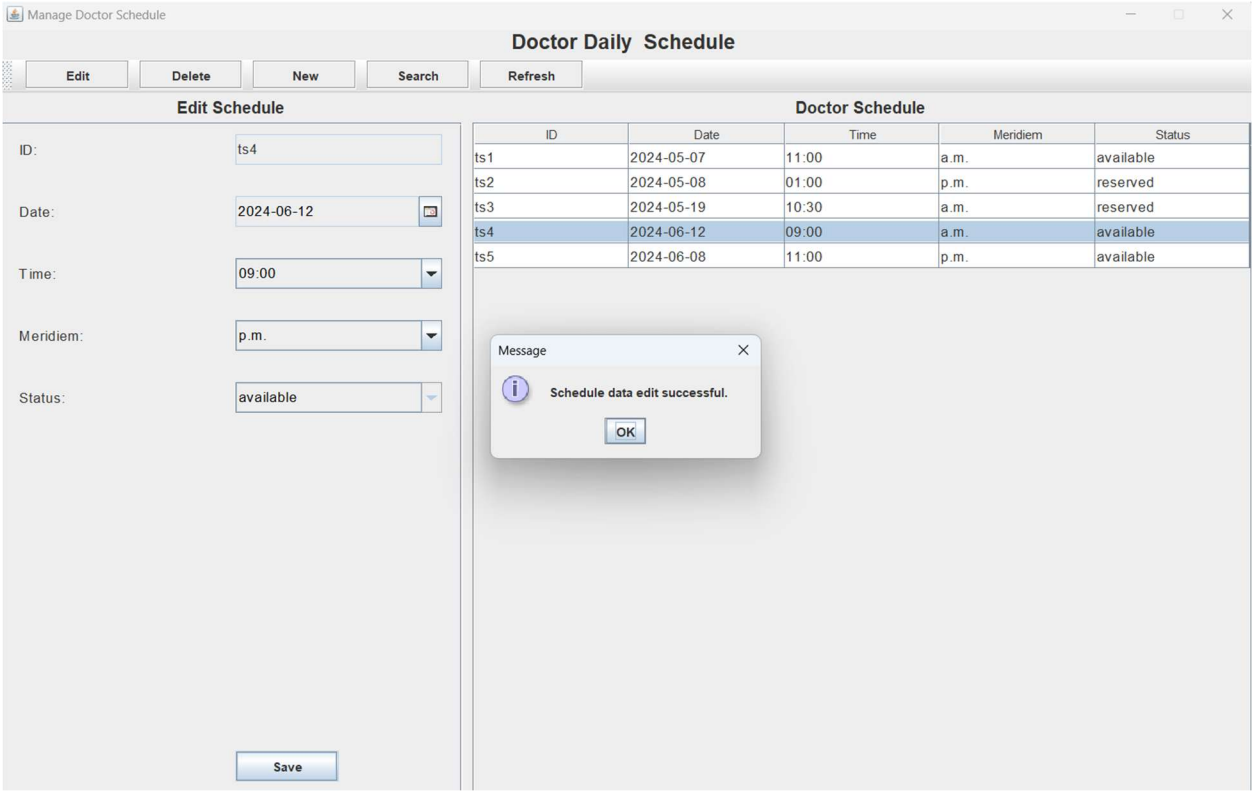


Figure 59:After selecting ok, a pop out message will appear indicating the data is edit successfully

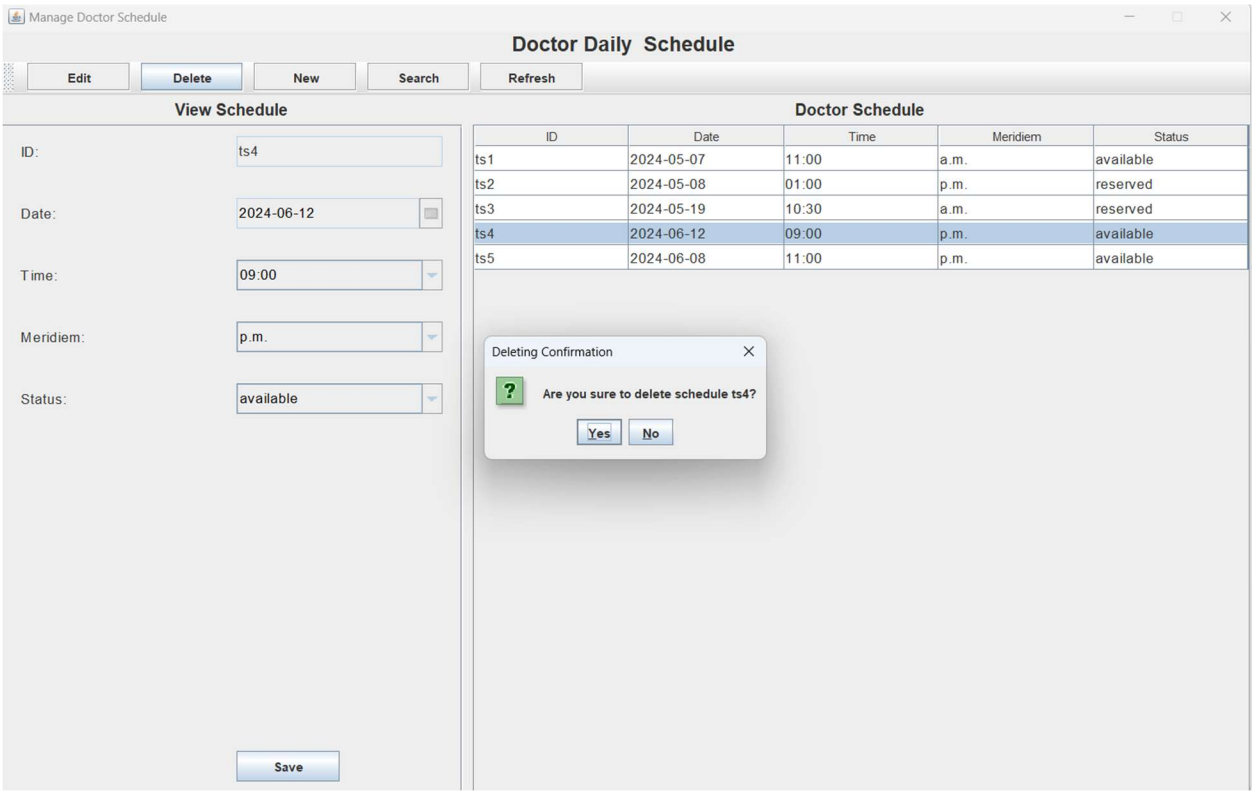


Figure 60:Delete function, the user must select a data to delete .After selecting the data a pop up message box will appear ,asking for confirmation to delete on the selected data

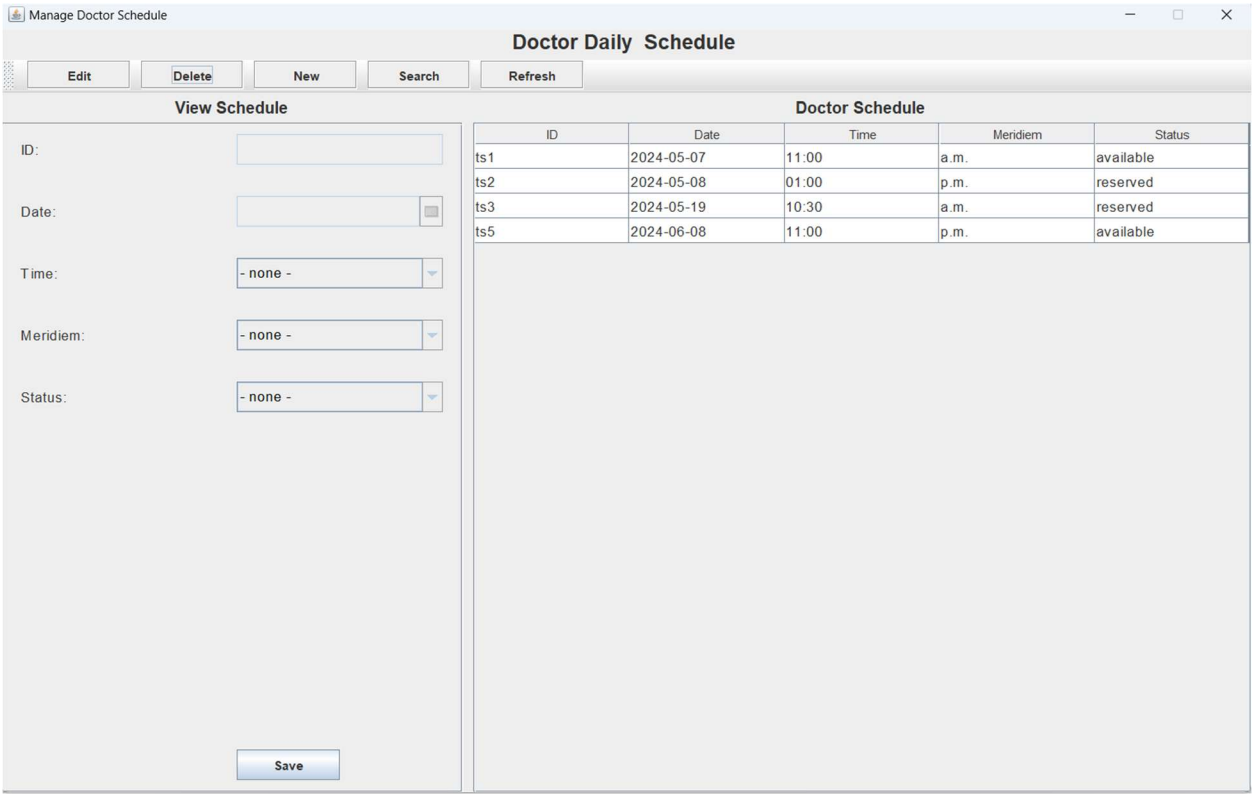


Figure 61 :The selected data by the user is deleted as shown above

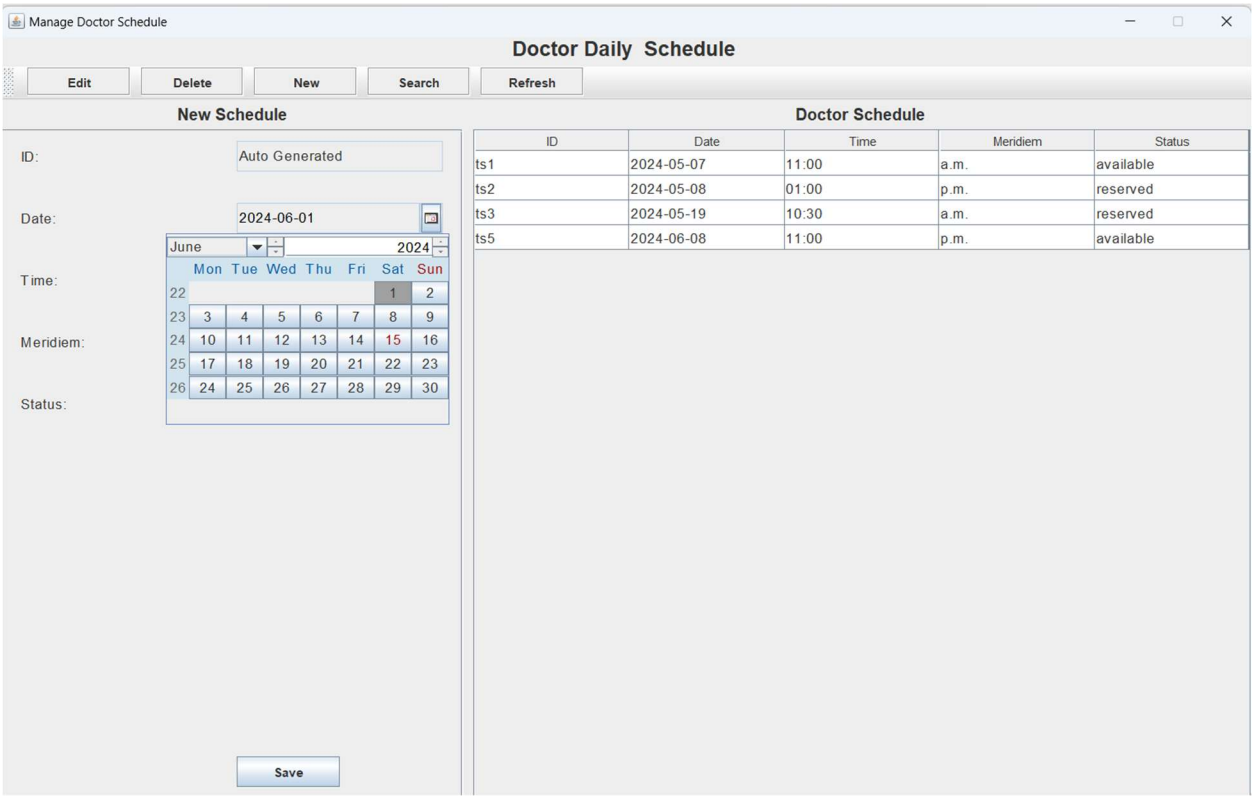


Figure 62 :If the user want to create a new Schedule ,he or she must choose a date

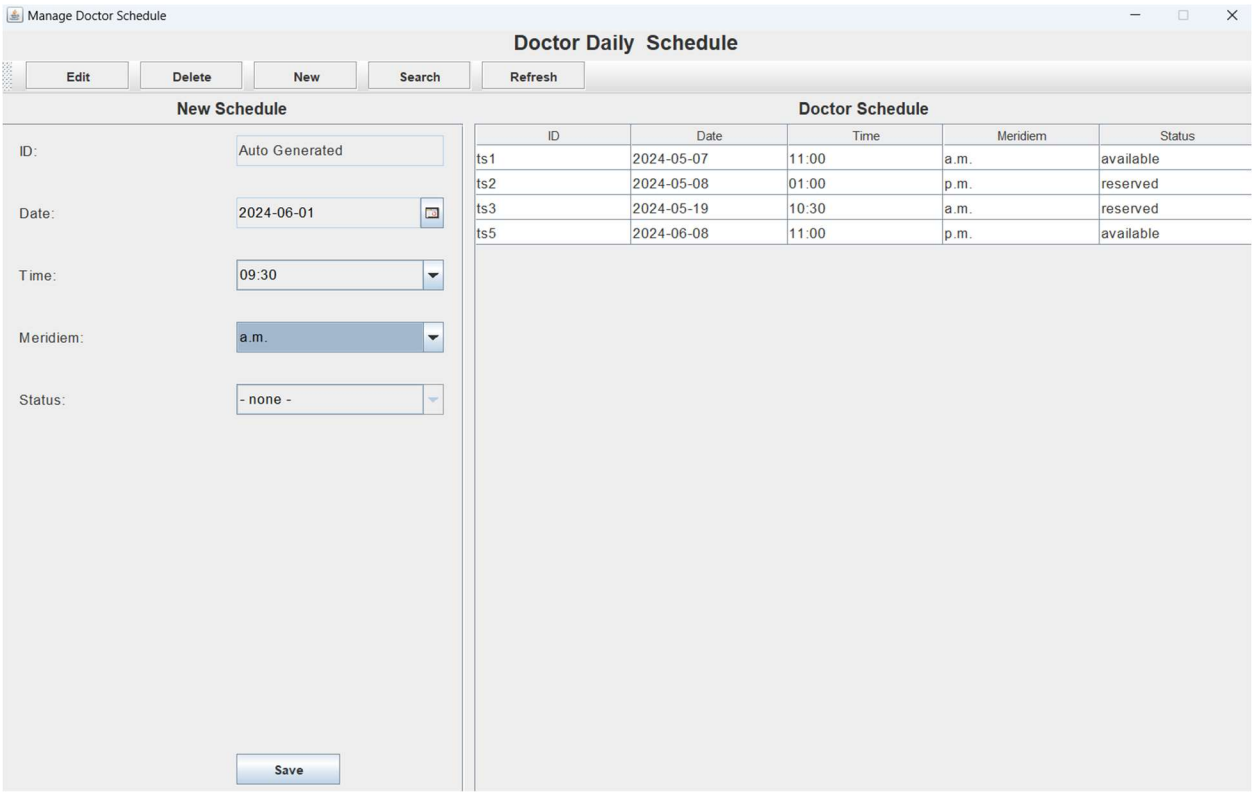


Figure 63:After user select the date and time , he or she can proceed to press the safe button

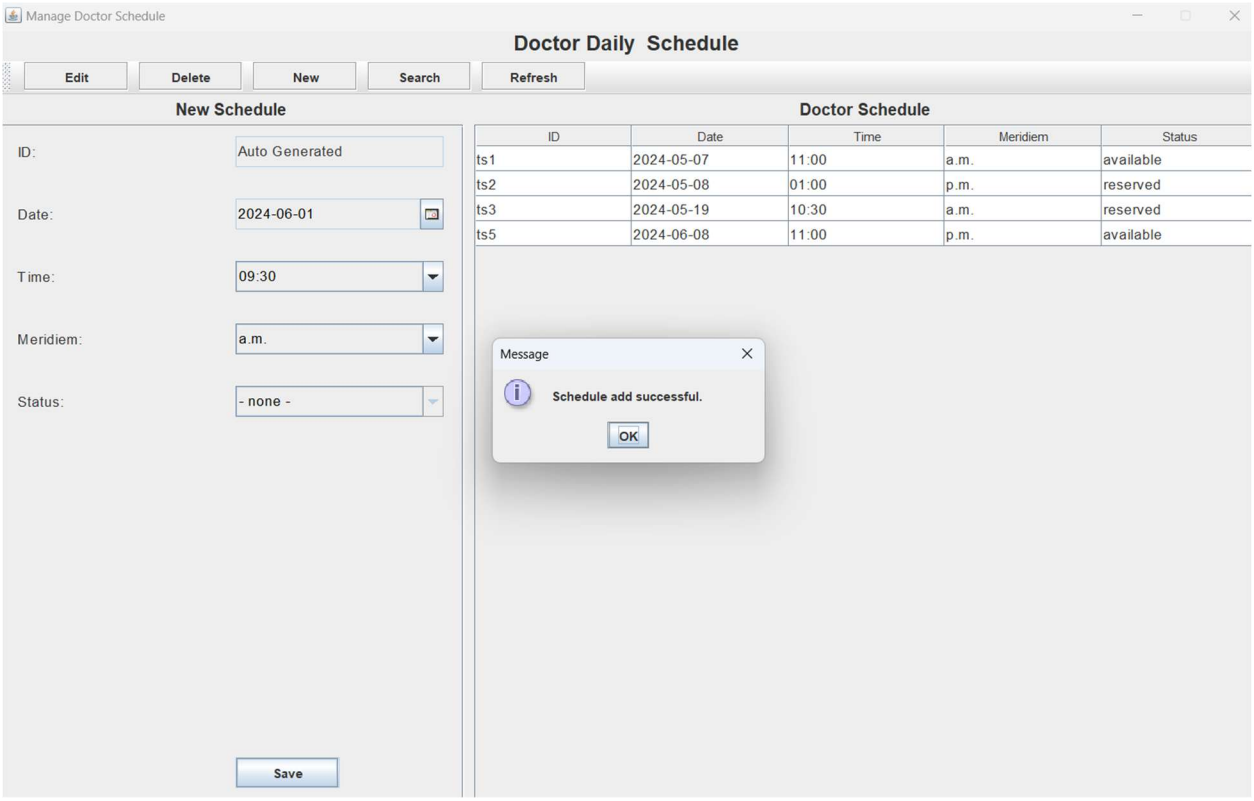


Figure 64:As shown above the schedule is added to the Doctor Schedule successfully

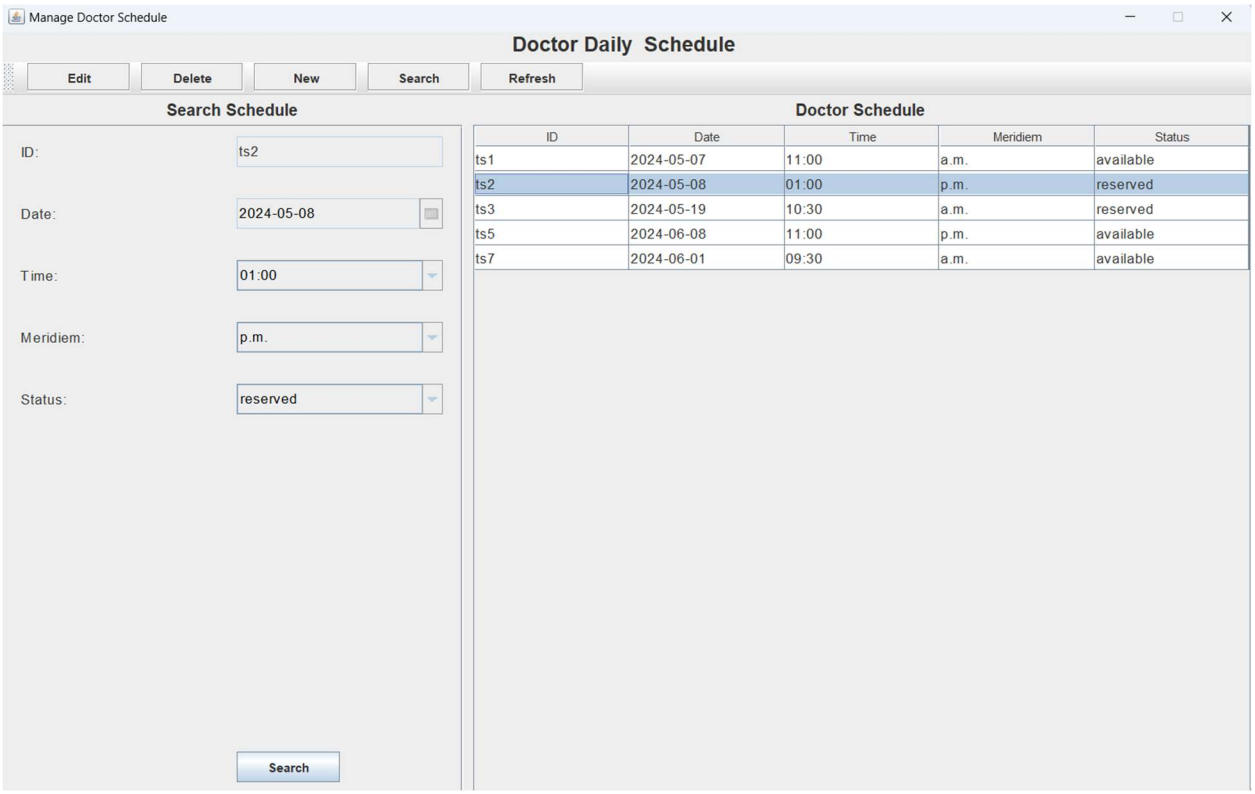


Figure 65 :If the user want to search a schedule ,he or she can press the search button

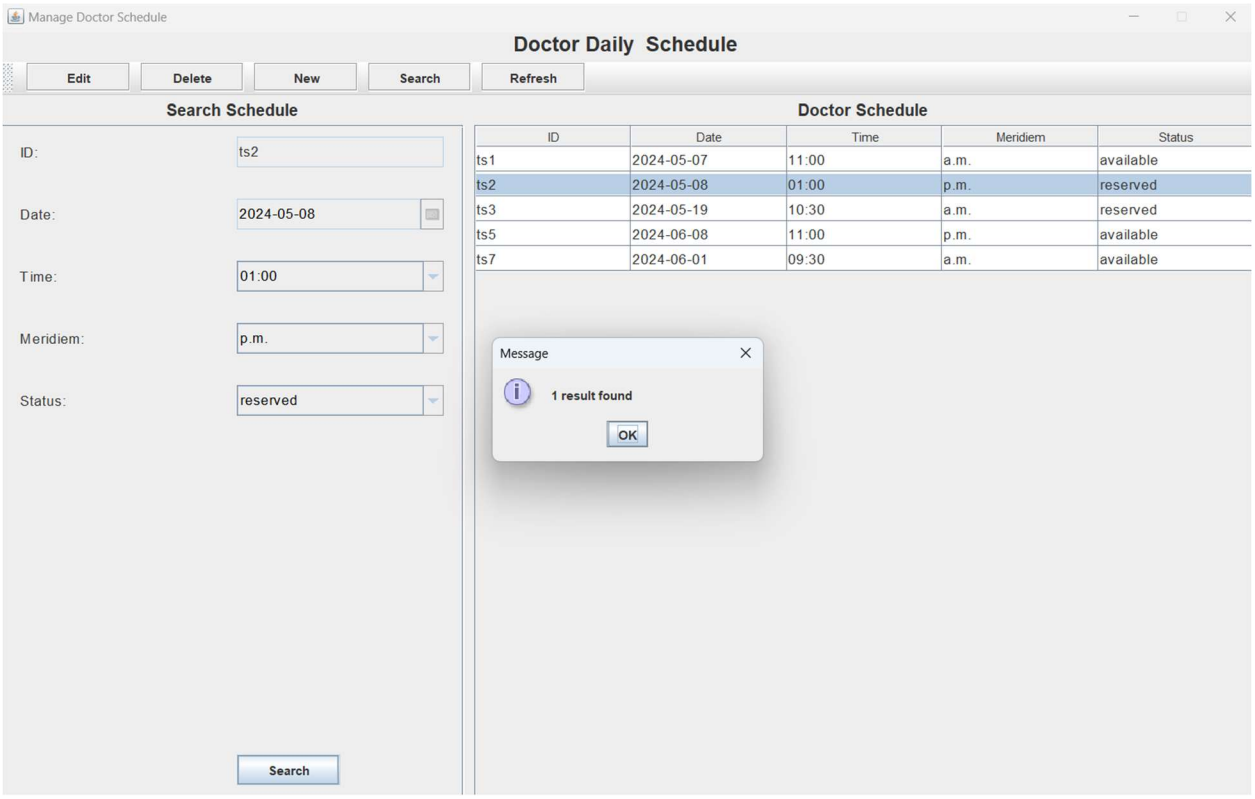


Figure 66 :As Shown above ,the user must enter the information then she want to search .After that ,it will display a pop out message box that indicating 1 result found

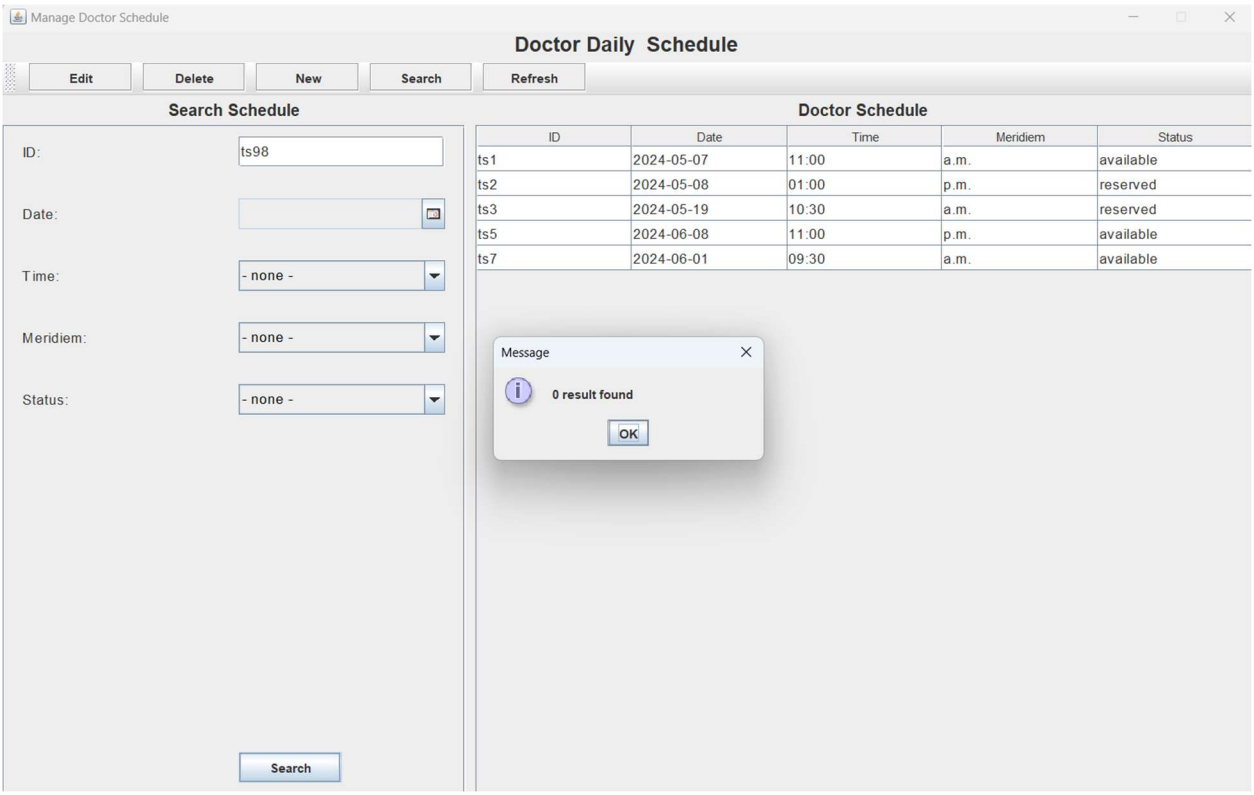


Figure 67:As Shown above , if the user did not enter the information then she or he want to search ,it will display a pop out message box that indicating 0 result found

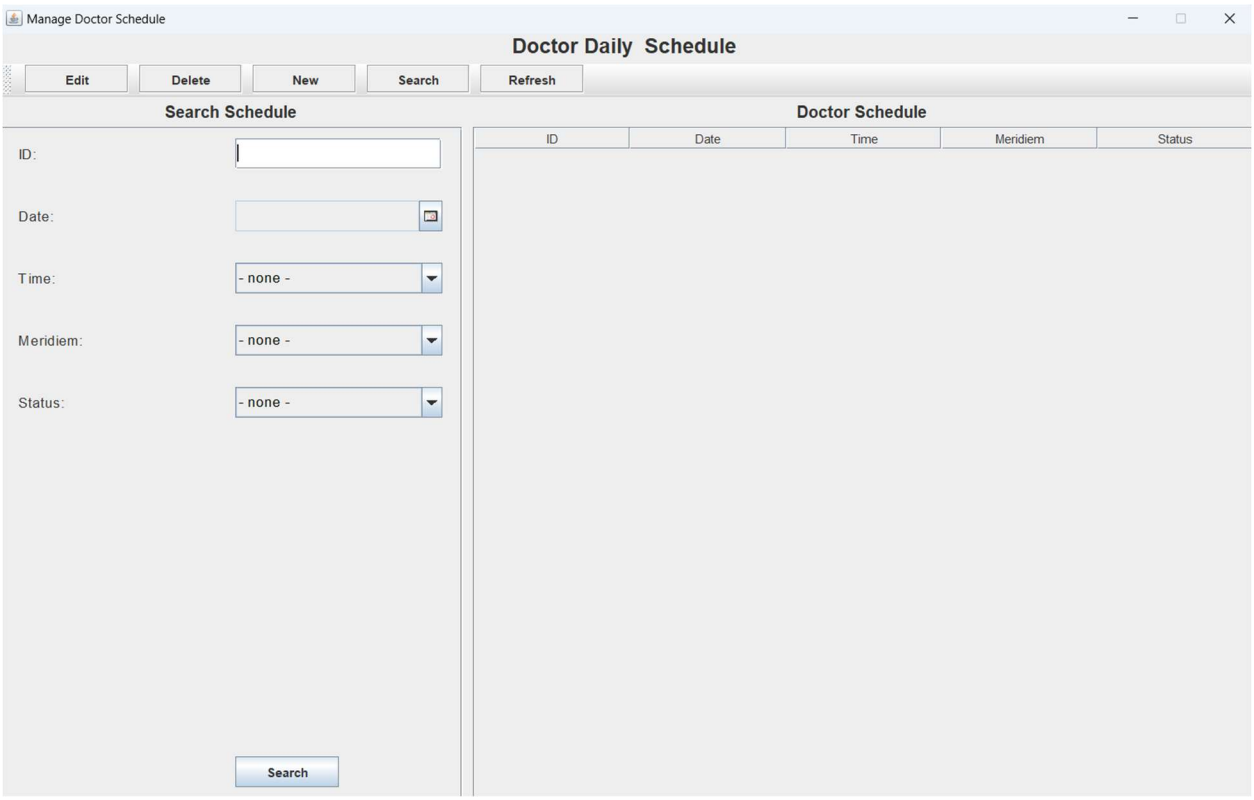


Figure 68:As Shown above , if the user want to refresh the page after adding new schedule ,he or she can press the refresh button

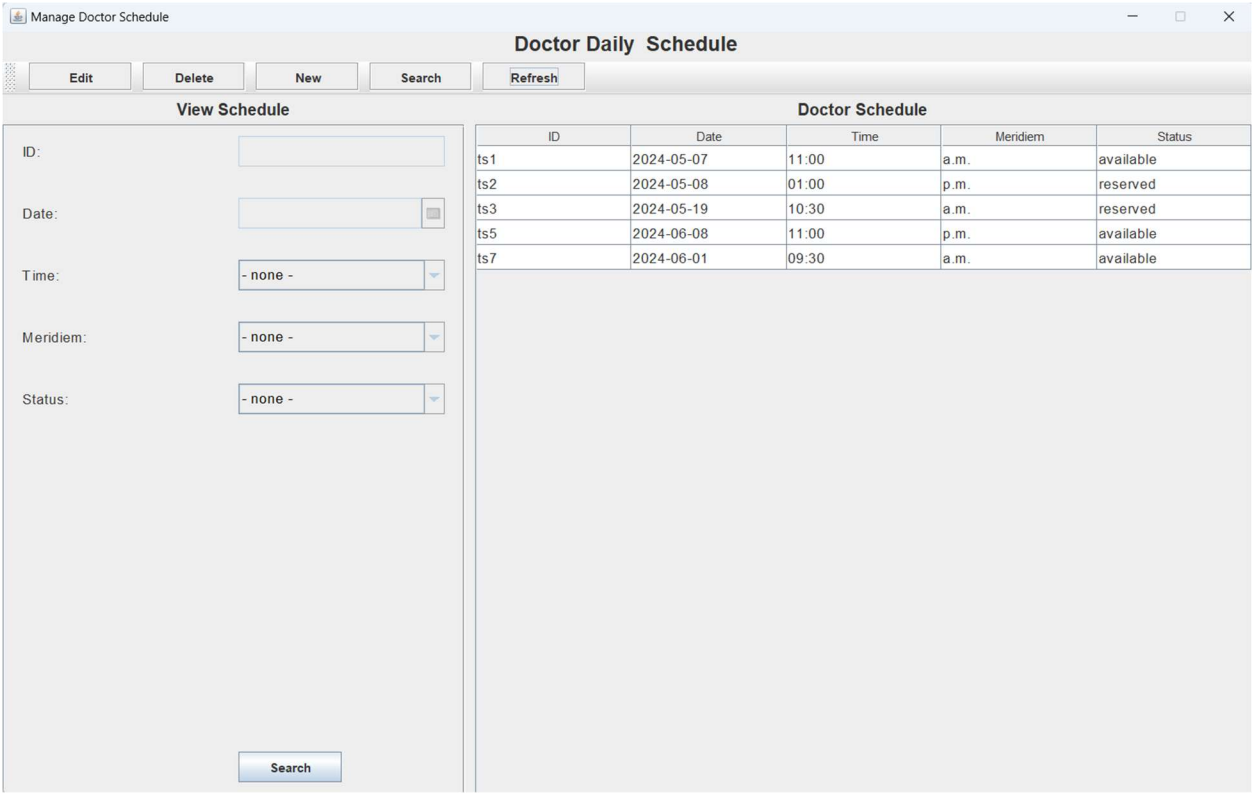


Figure 69:As shown above , after pressing the refresh button ,the data will show out

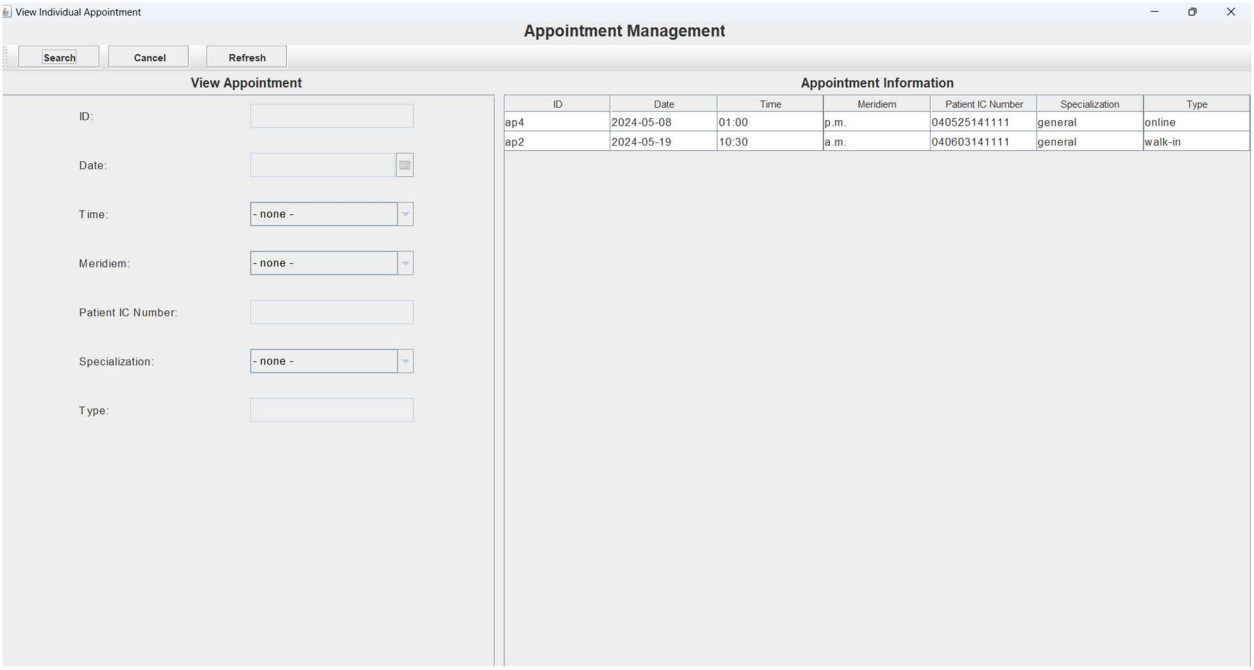


Figure 70: If the user want to search and appointment ,he or she can press the search

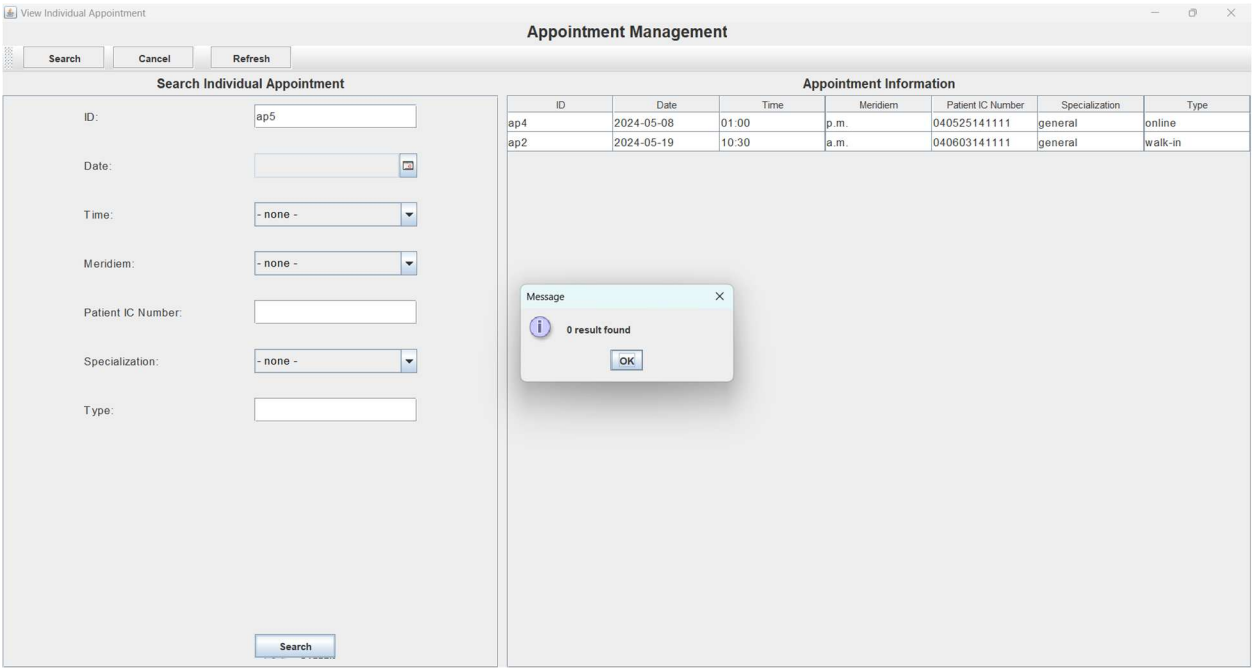


Figure 71: As shown above 0 results is found ,after the user enter the wrong ID

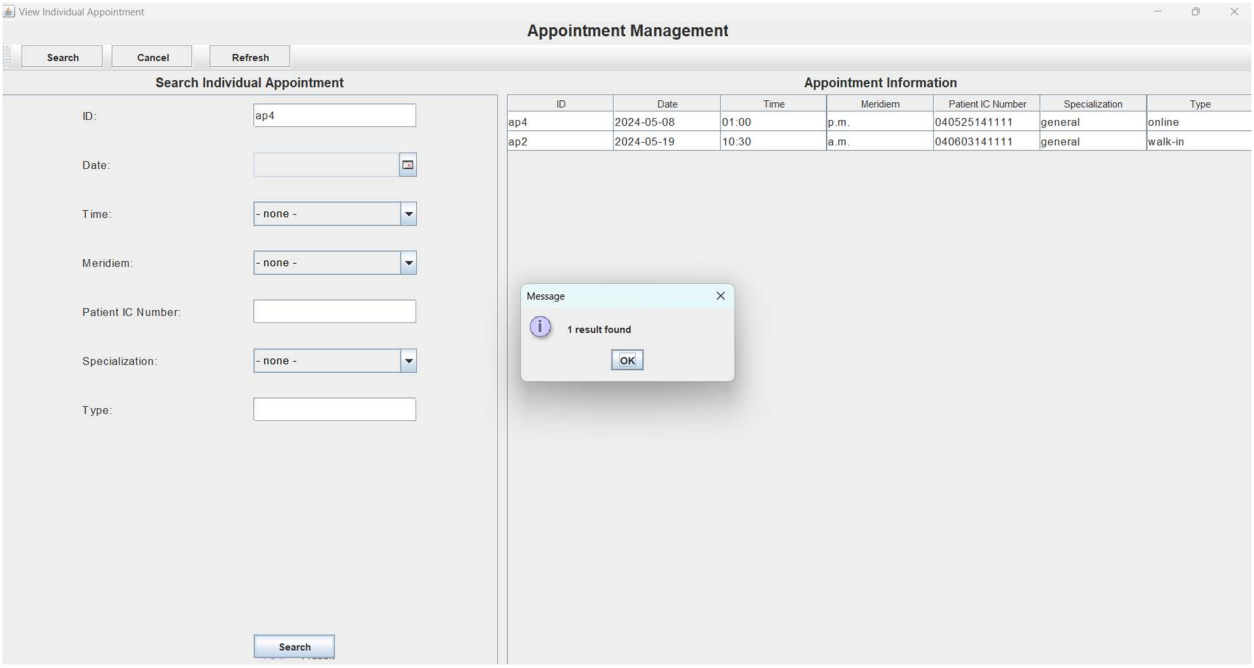


Figure 72: As shown above 1 results is found ,after the user enter correct ID

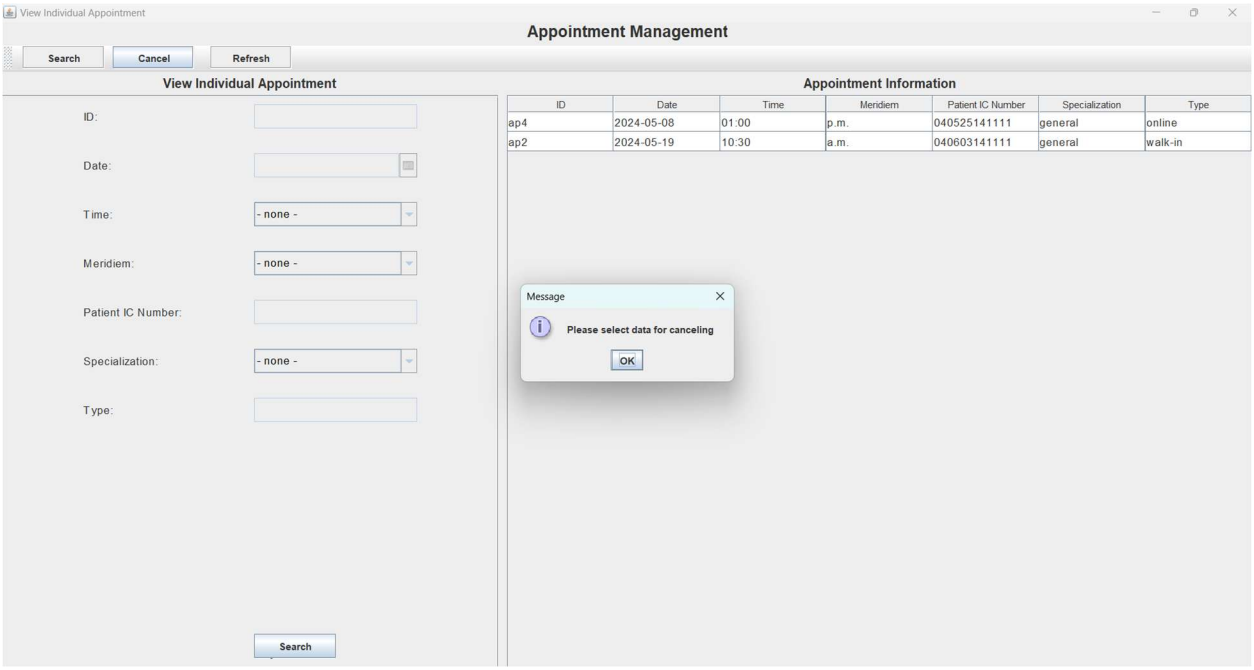


Figure 73 : User can choose to cancel appointment by pressing the cancel button. After pressing the cancel button, a pop up message box will ask the user to select an data for canceling

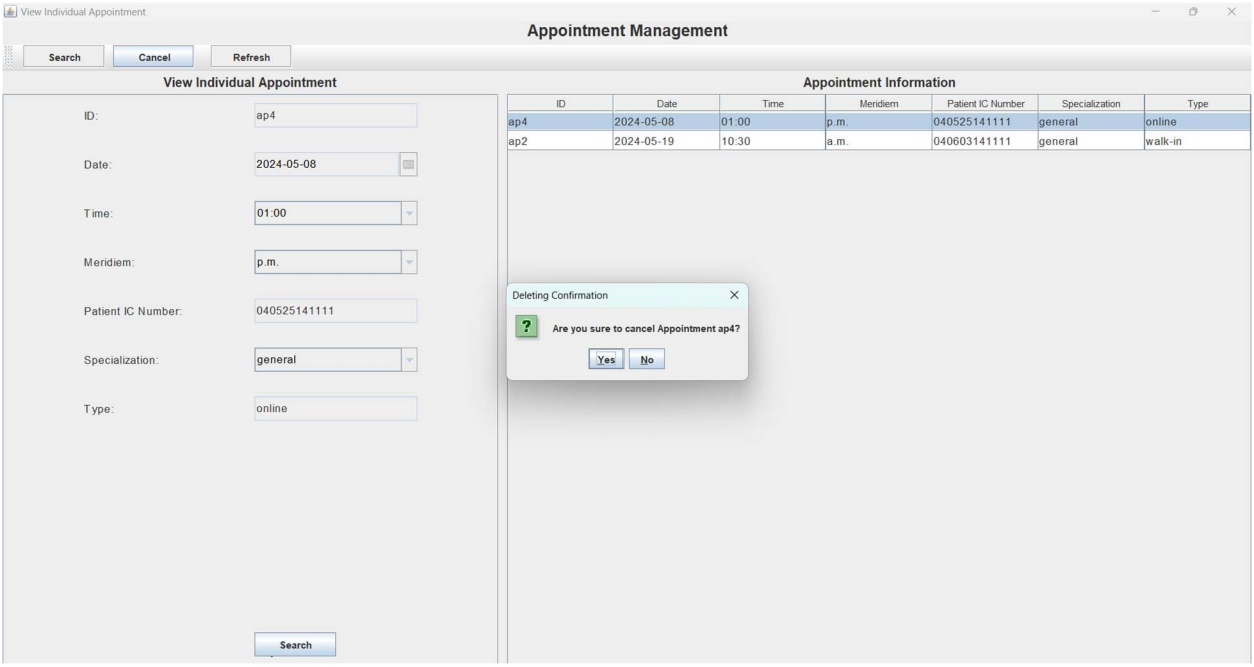


Figure 74 : A pop out message box will ask the user to ,if he or she confirm to cancel appointment

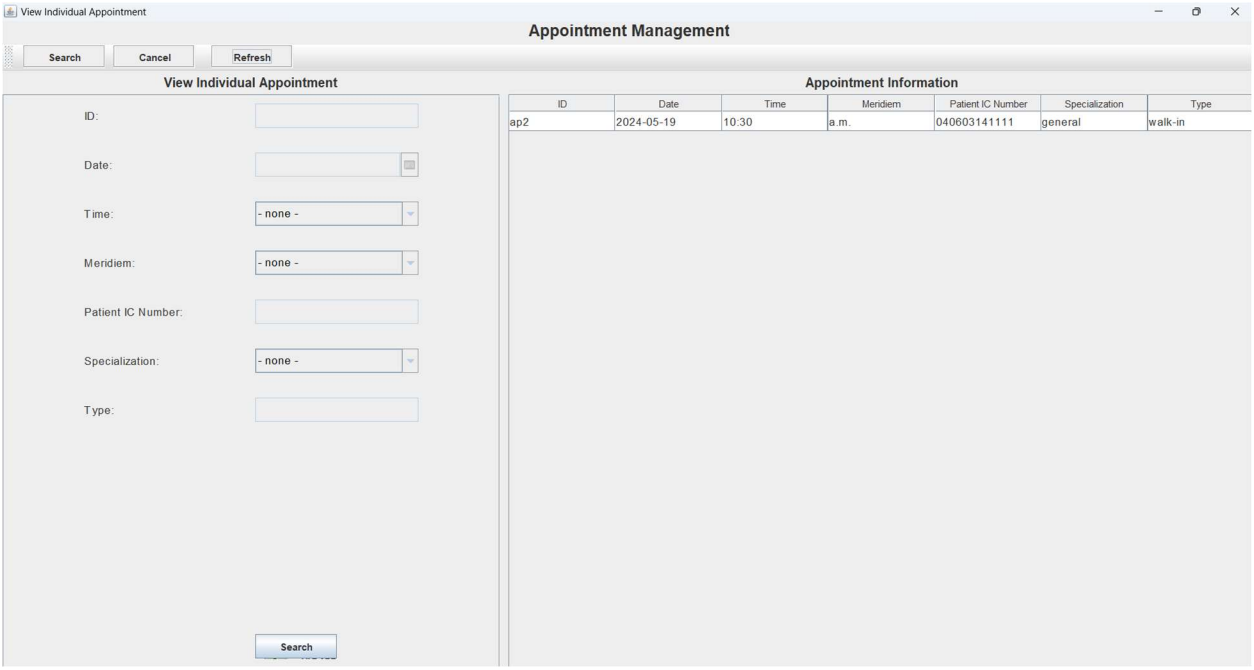


Figure 75 : After confirmation, the selected cancel appointment data is remove from the table

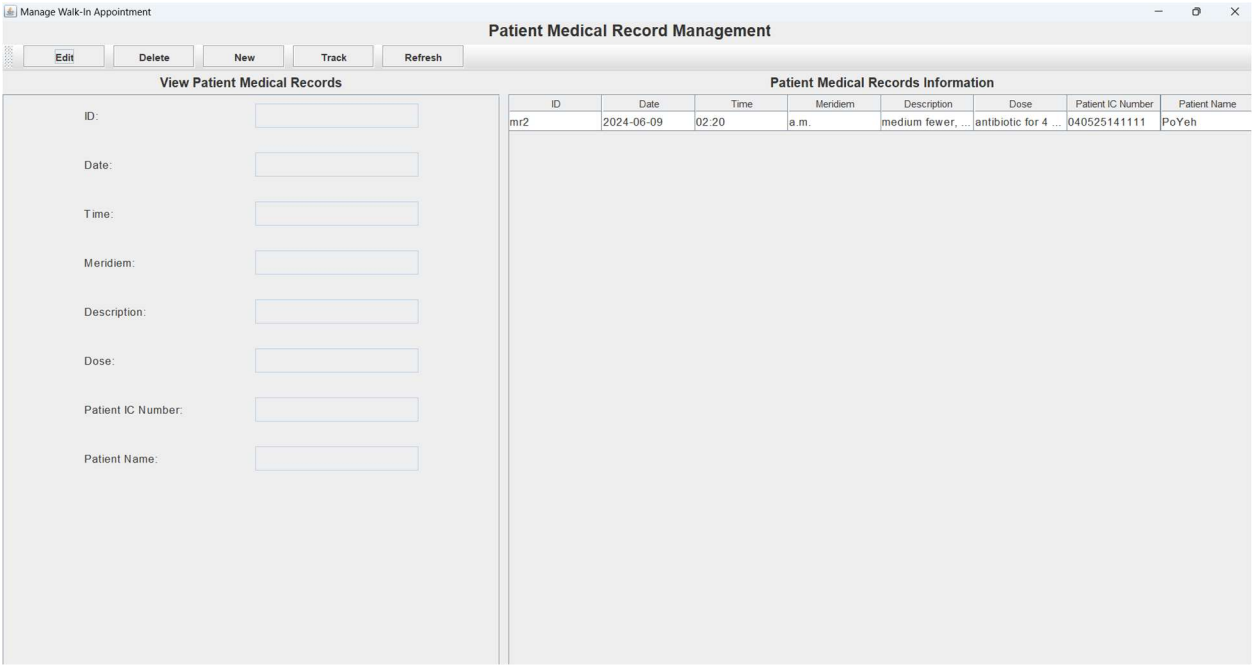


Figure 76 : As shown above, user can edit the patient medical records information. If the user want to edit , he or she can press the edit button

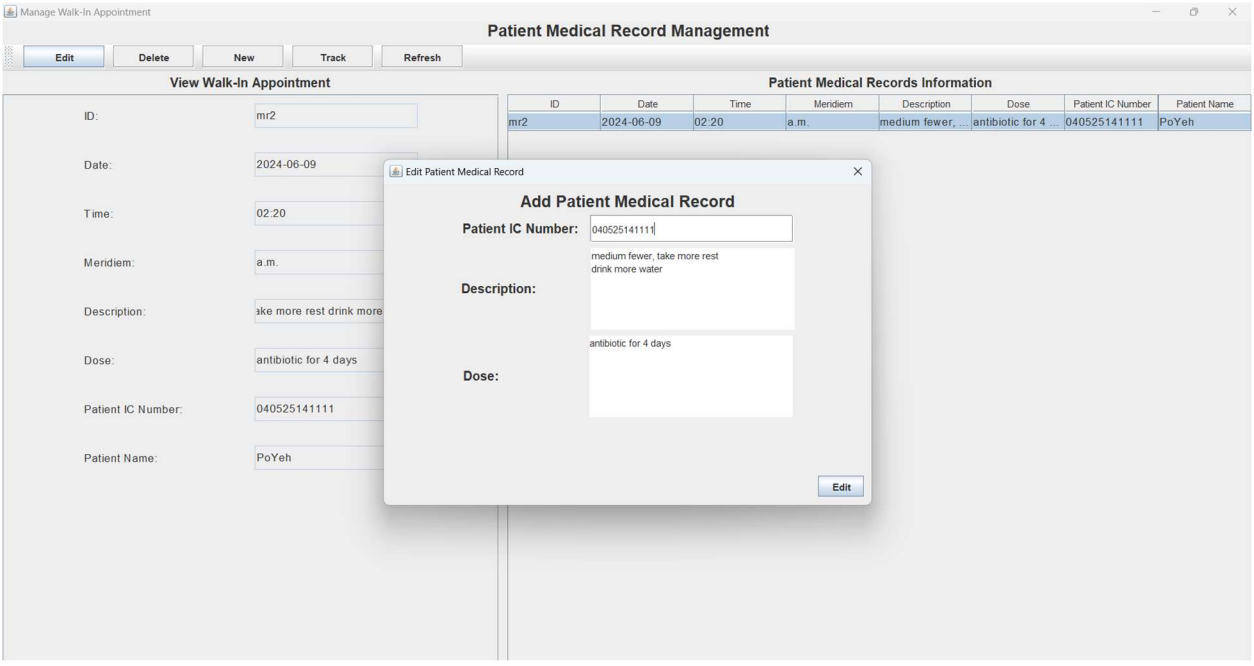


Figure 77 :After pressing the edit button, a pop up box where user can edit the Patient Medical Record .Editable information such as the Patient IC Number ,Patient Description , and Dose .

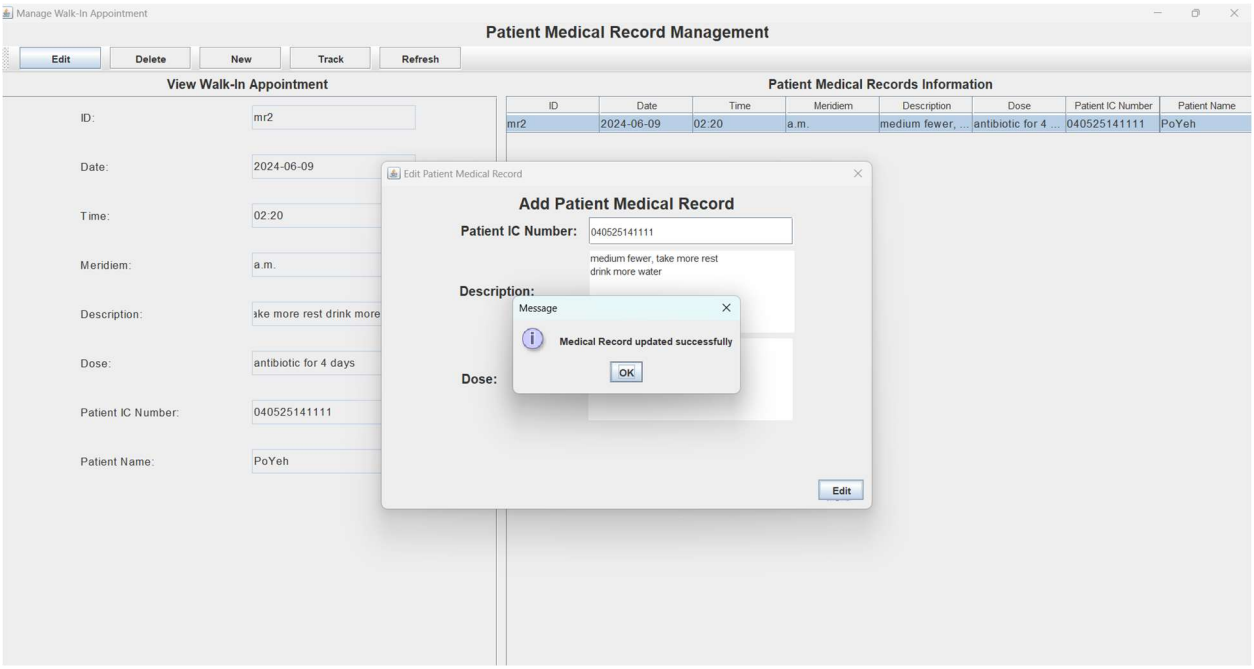


Figure 78 :After the user done editing the Patient information ,they can press the edit button again in the pop out message box a message box will show a message "Medical Record Updated Successfully".

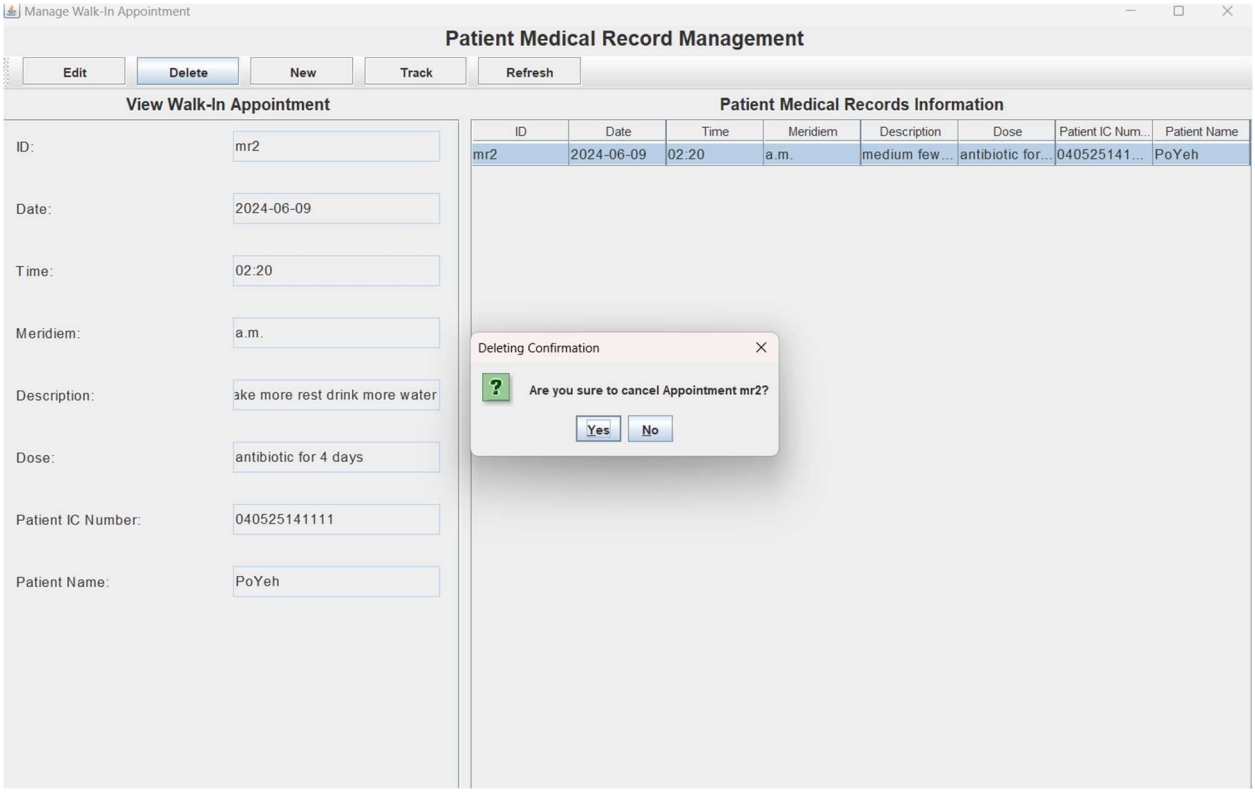


Figure 79 :If the user want to delete appointment he or she can press the delete button ,where the user must select the data. After selecting the data a pop out message box will appear asking for confirmation from the user.

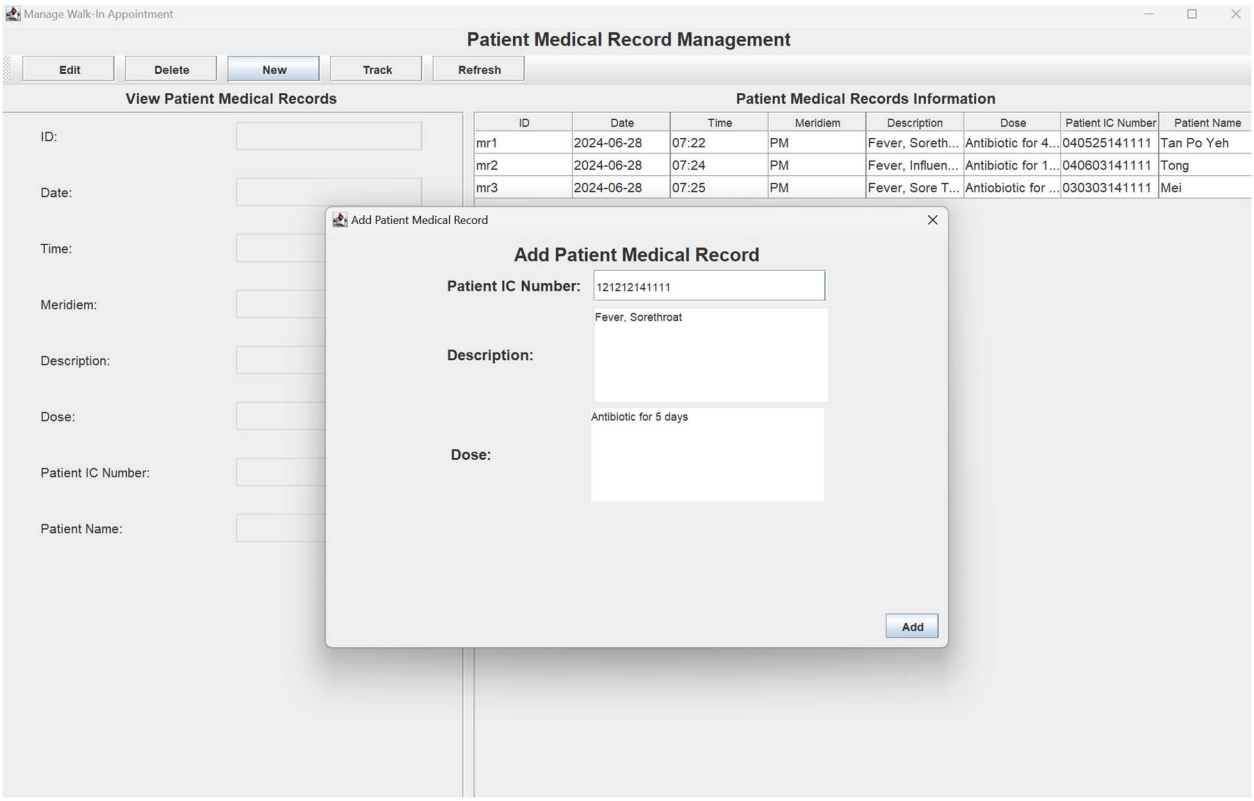


Figure 80 :If the user want add new Patient Medical Record ,he or she can press the new button where then the user will see a pop out message box .In the pop out message box user can insert new Patient IC Number ,Patient Description and Dose. After filling up the information the user can press the add button ,where the new data will be display at the Patient Medical Record Information Table.

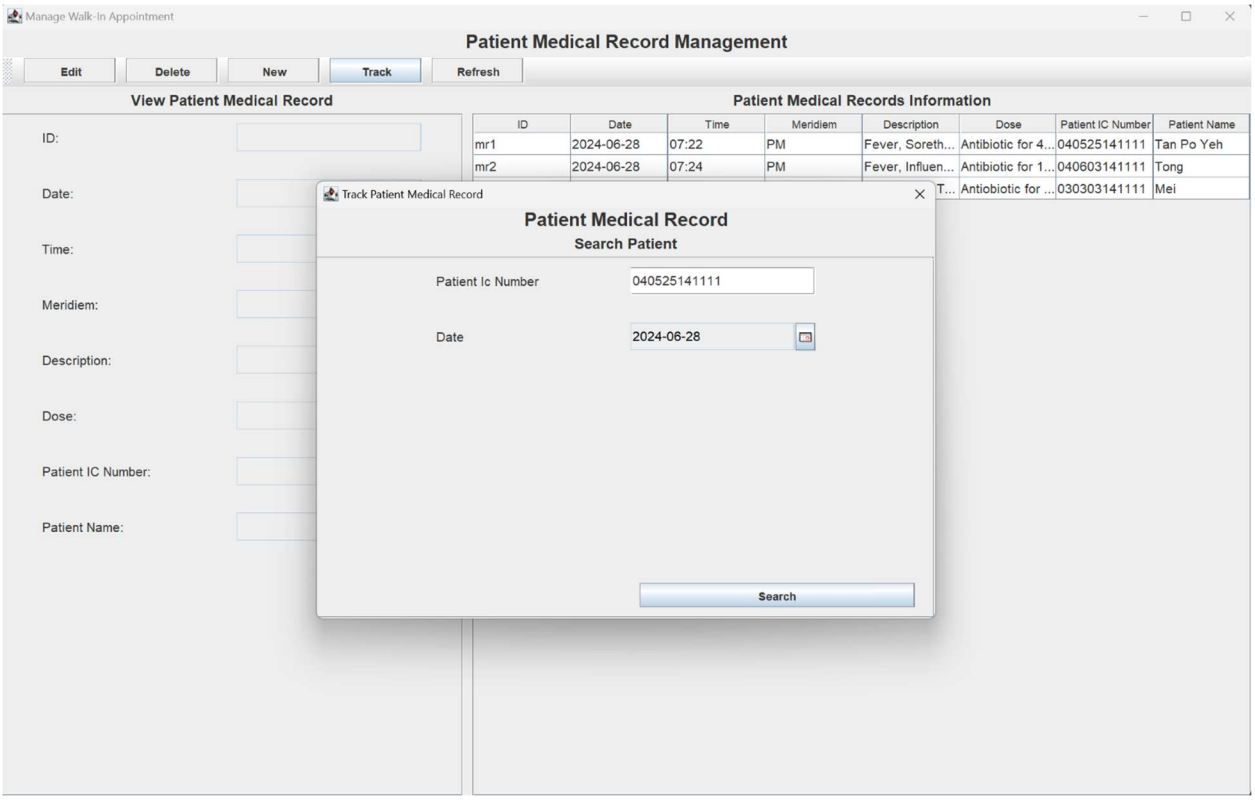


Figure 81 :If the user want to track Patient Medical Record ,he or she can press the track button .After pressing the track button a pop out message box will appear .

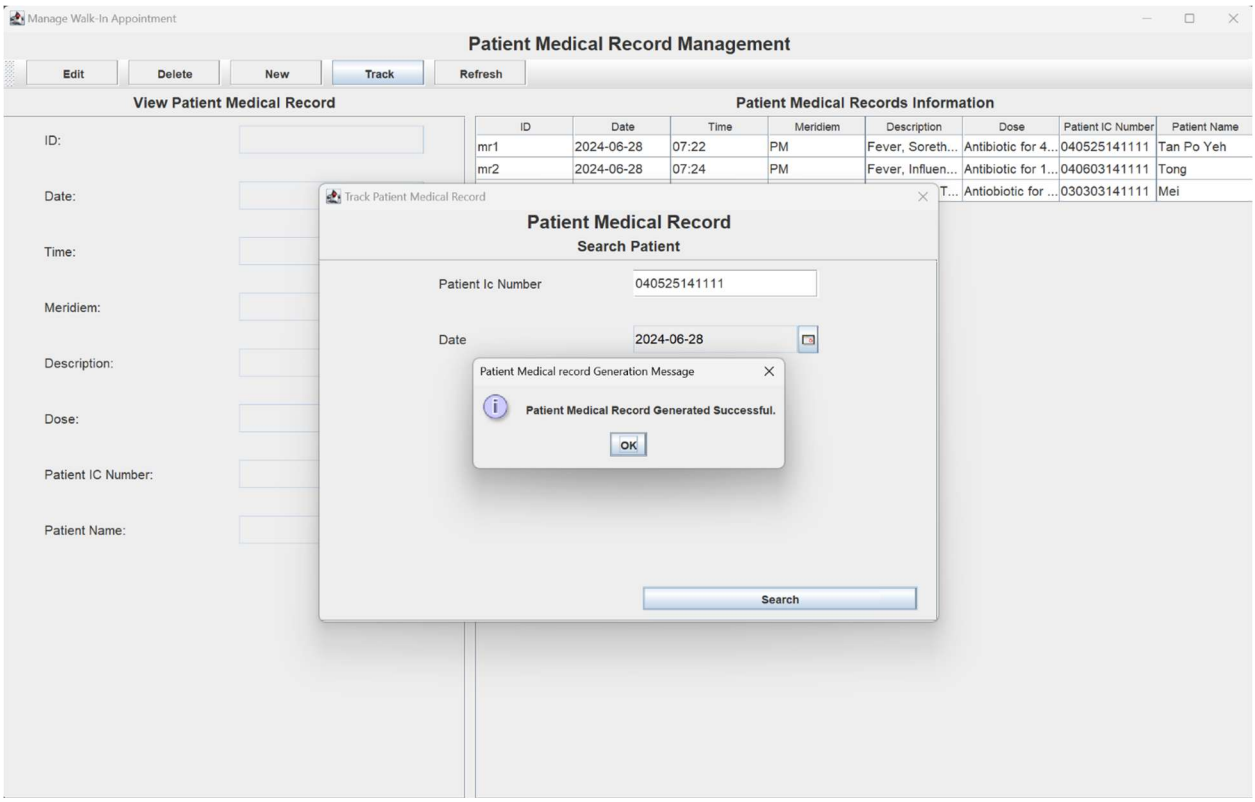


Figure 82 :After pressing the search button , a pop message box will appear with a Pateint Medical Record Generated Successful

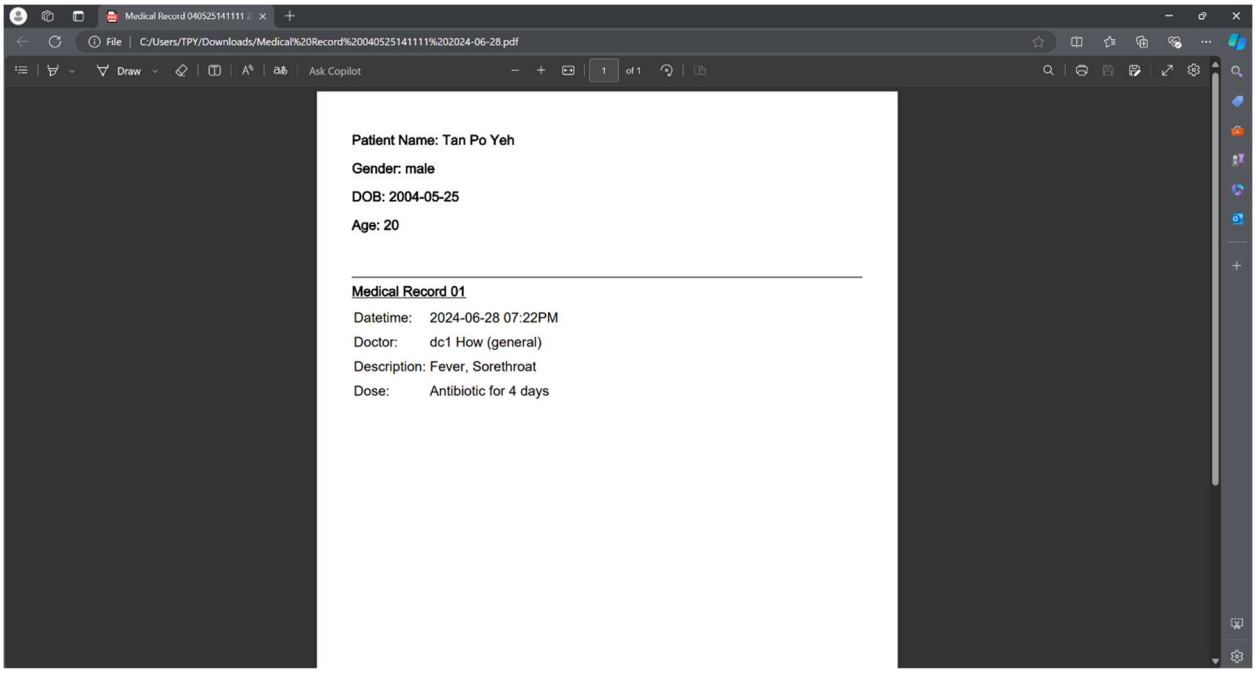


Figure 83 :After pressing the ok button , the program will bring the user to a pdf file where all the Pateint inromation will be store

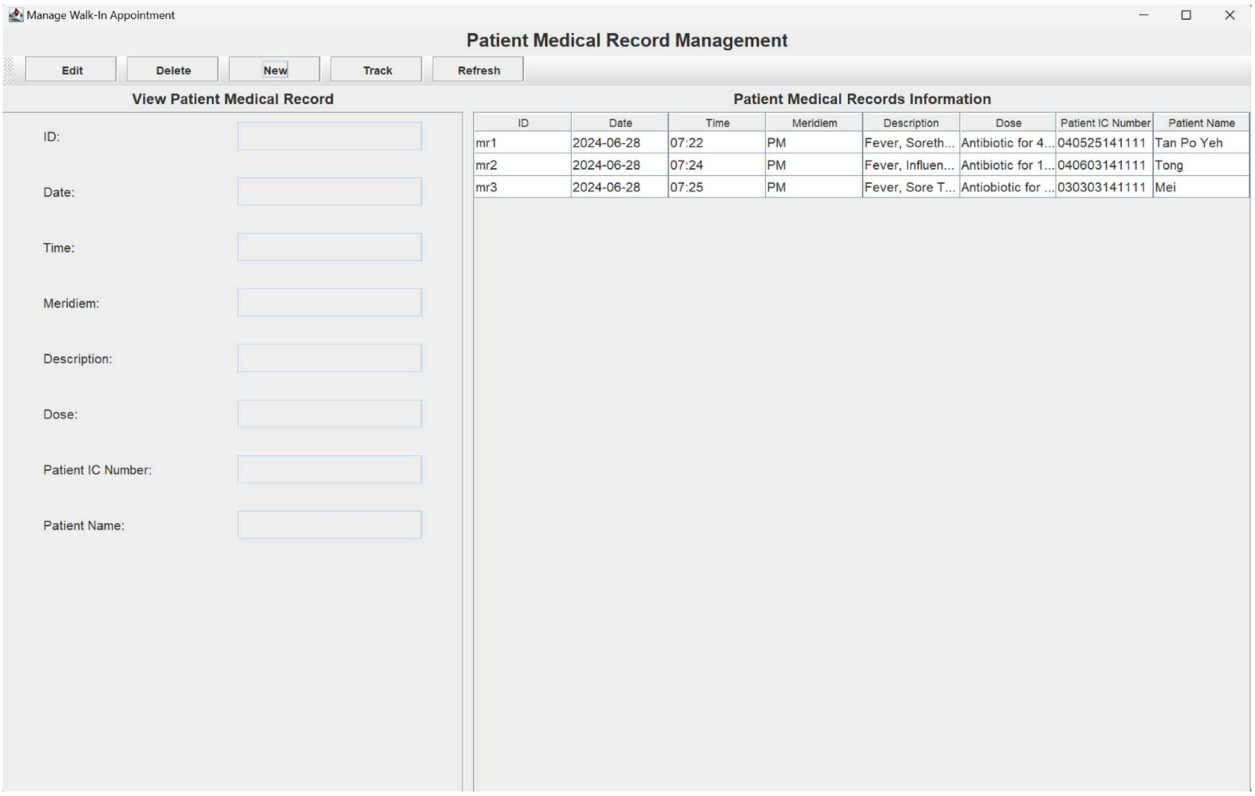
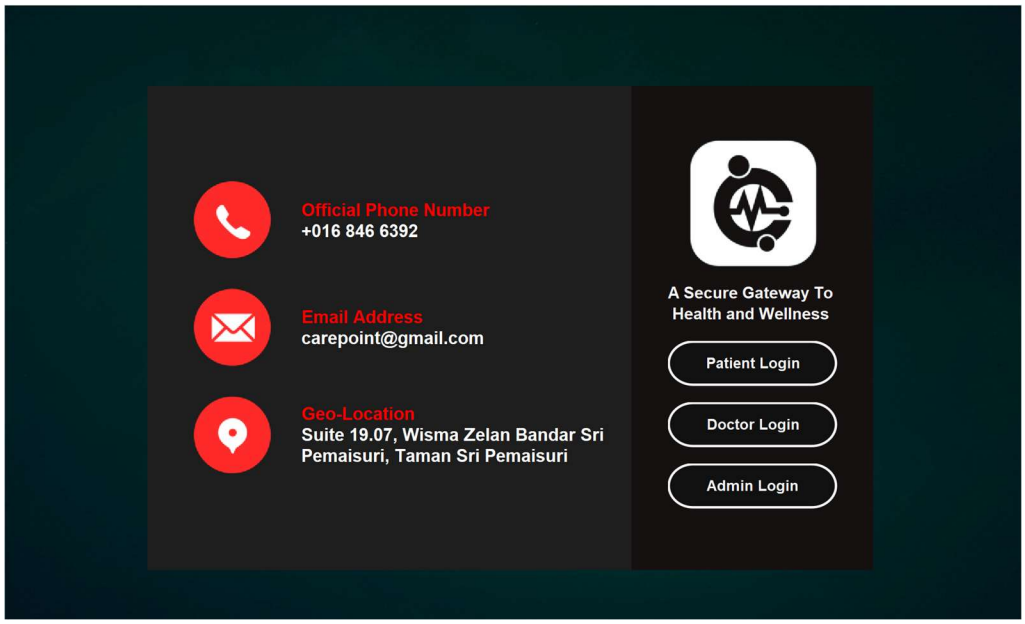


Figure 84 :if the user want to refresh the page ,he or she can press the refesh button where all the data will be refesh incldiung the new data that been added to the table .

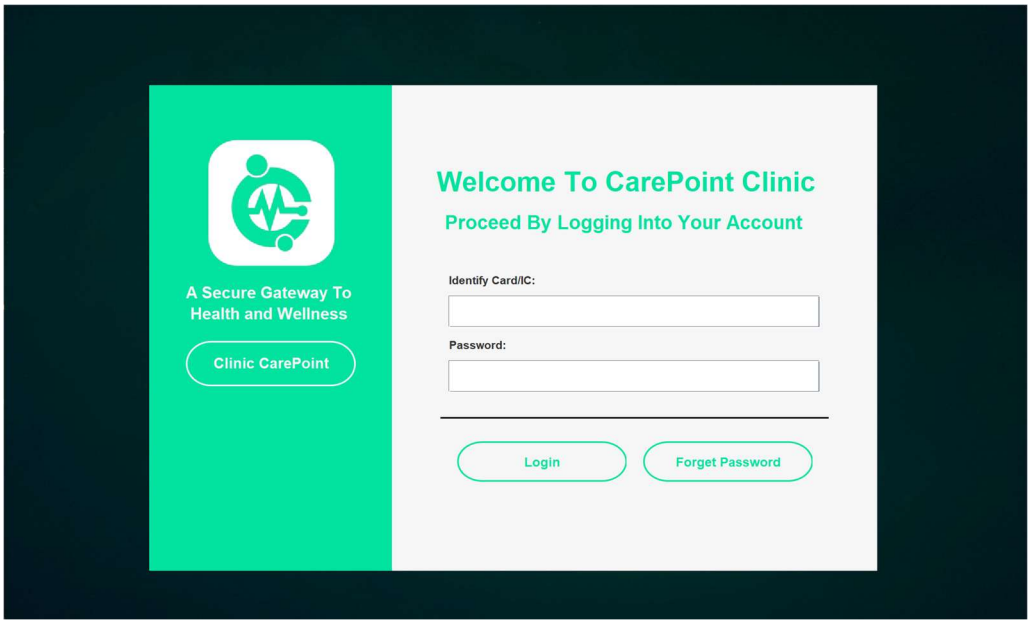
2.3 Patient Features

2.3.1 Patient Authentication

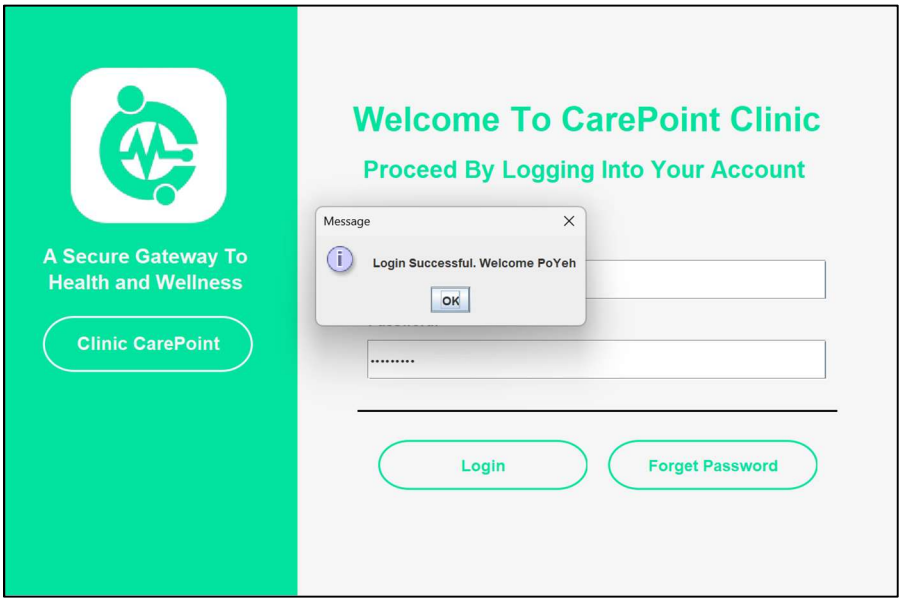
In order to log in as a patient, the user will need to press the "Patient Login" button. Once the button is clicked, the system will display the patient login page.



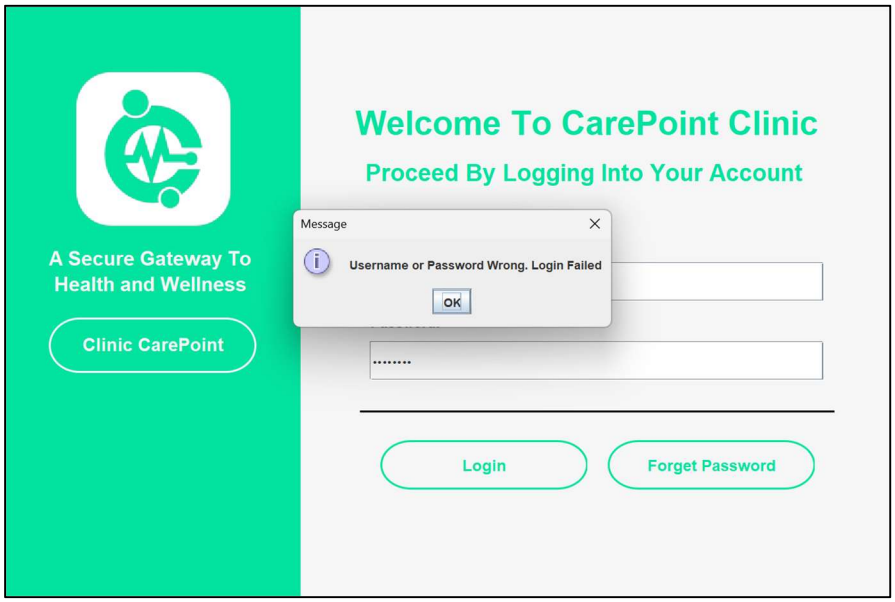
Below are the login page for the patient, user will need to enter the correct credentials to access the system.



In the case below, we enter the correct user information as the figure shown below. The system will display a success message to the user and redirect the user to the main page of the appointment system page.

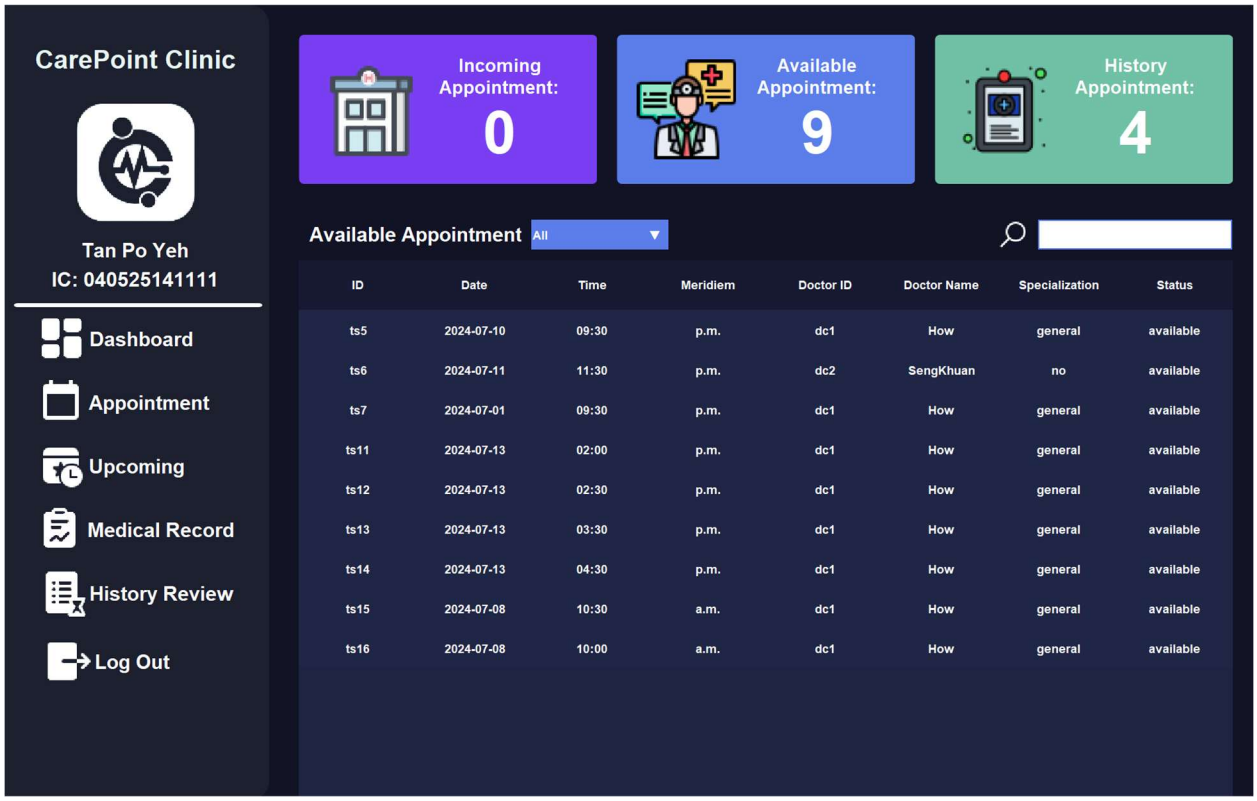


In contrast, if the user enters an invalid IC number or incorrect user password, the system will display an error message to the user. If the user is unable to log in to their account, they will need to either use the forget password feature to reset their password or contact the admin to retrieve back their information.



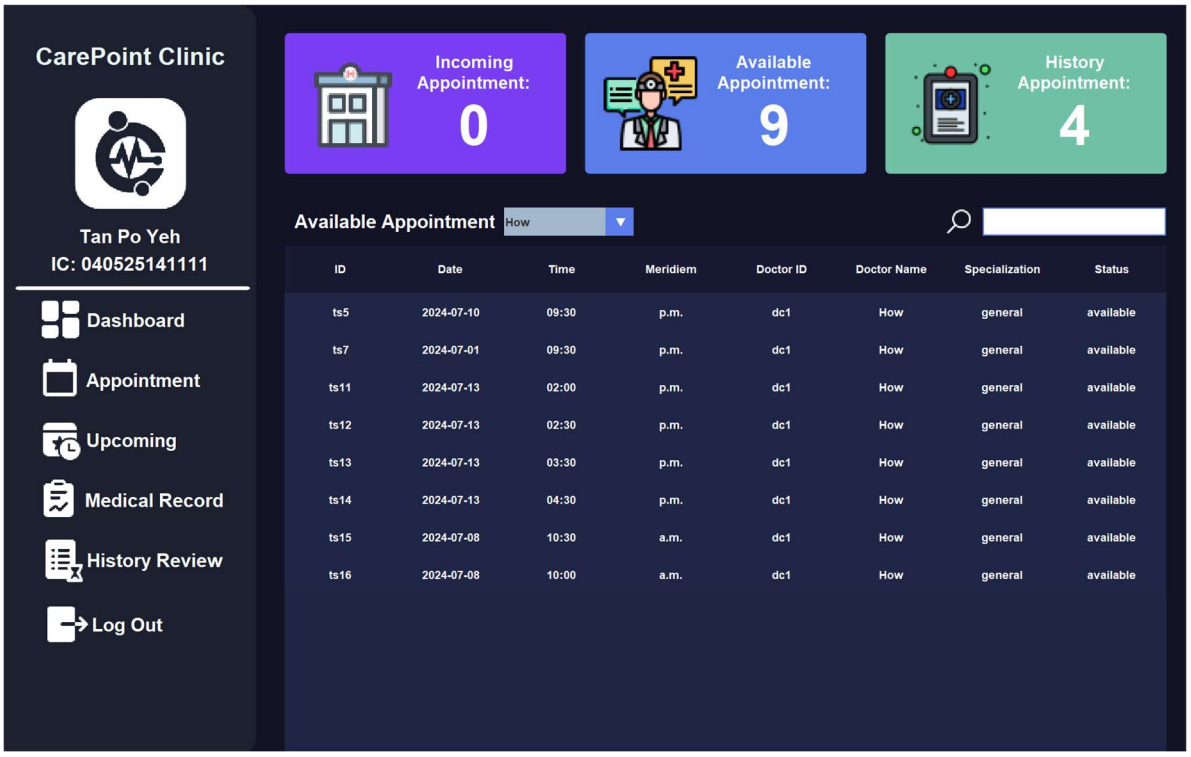
2.3.2 Dashboard - View Available Timeslot

Once the user login successfully, the system will direct the user to the patient dashboard page as the figure shown below. On this page, patients are able to view their own incoming appointment numbers, the available appointments and the total history appointments they made.

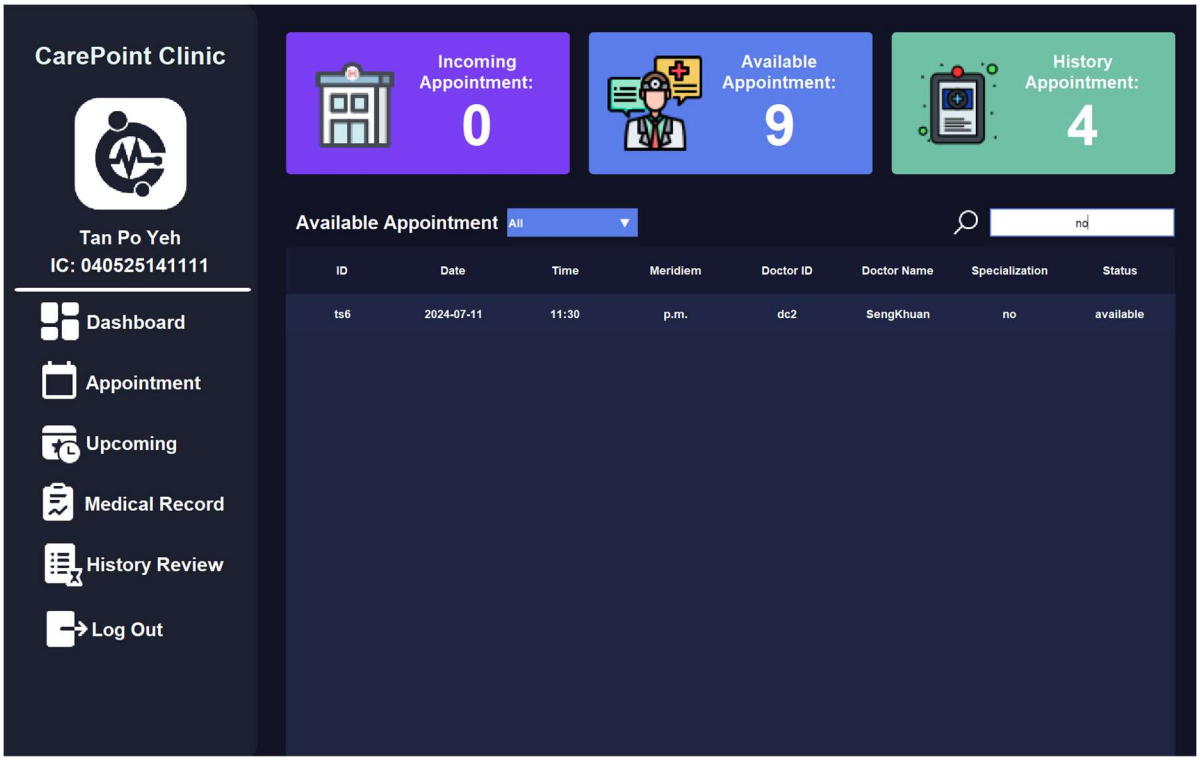


The number of indicators on the page above will change according to the patient's action. In this scenario, there are a total of 9 appointments available in the clinic and the patient "Tan Po Yeh" booked 4 appointments before.

On the dashboard page, patients can also search for specific appointments by using the search bar and the multiple-select bar. In the figure below, the patient searches for the Doctor "How" to find the specific doctor's appointments. The system will change the table list and only display Doctor "How" appointments on the page.

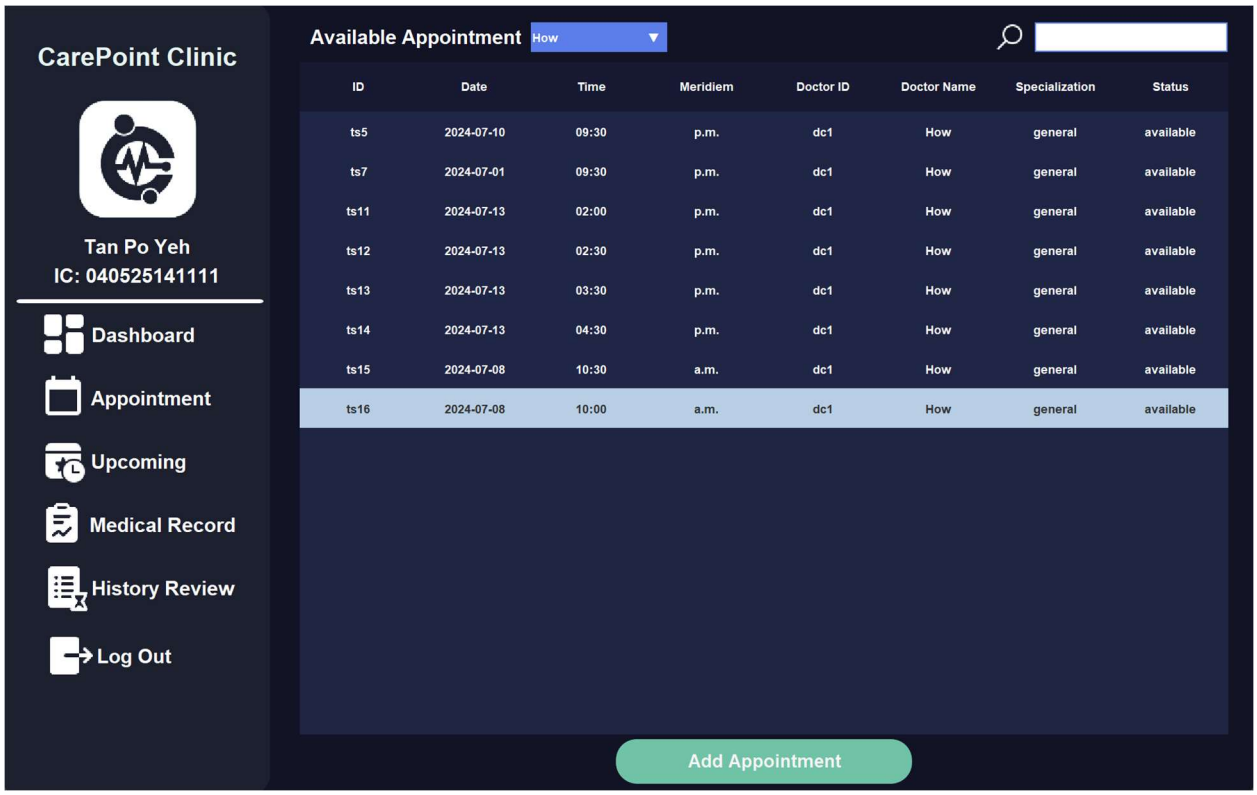


If the patient wants to find an appointment based on the specialization, the patient can just use the search bar to filter the results. The figure below shows the result of searching the specialization "No" on the page.

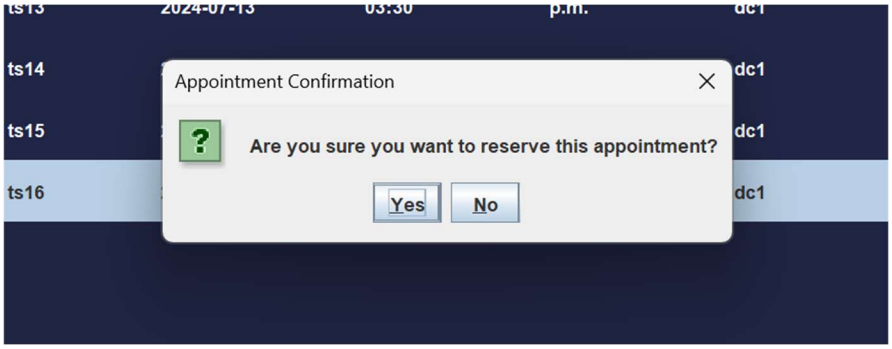


2.3.3 Appointment - Make Appointment

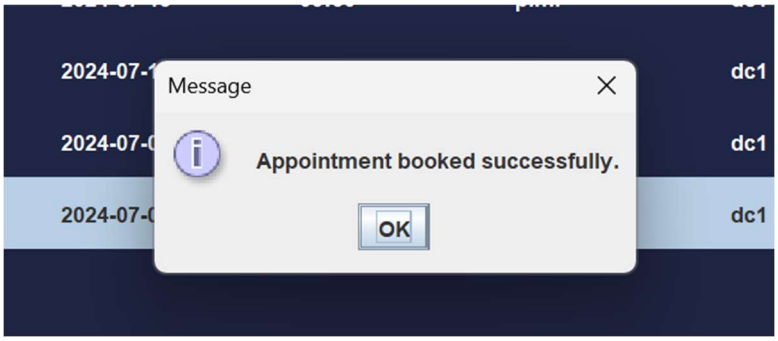
The second feature is located on the appointment page, patients can access the page by pressing the "Appointment" button on the left panel. Below are the patient appointment page.



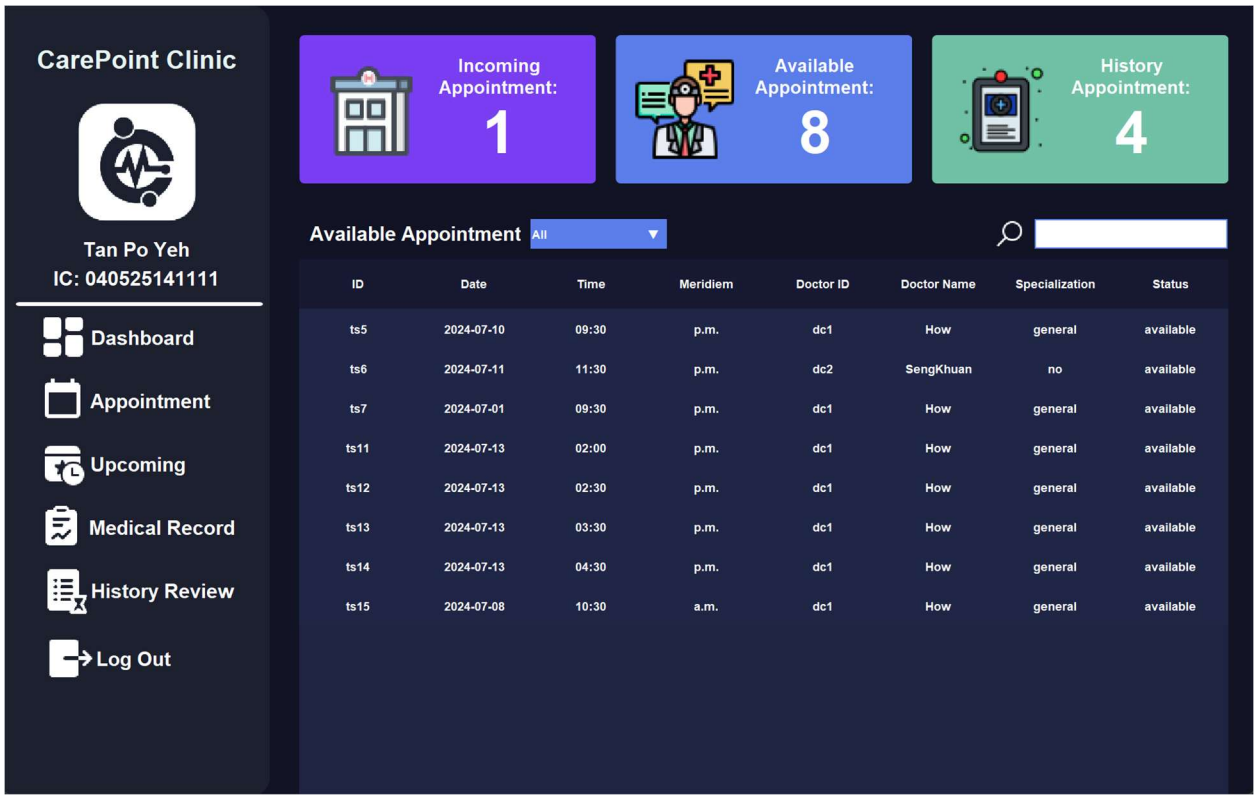
On this page, the patients are able to make an appointment by selecting the desired time slot in the table list. Then, patients simply click the "Add Appointment" button to add the appointment to the page. The system will display a confirmation panel for the patient to double-confirm the appointment reservation.



Patients can proceed to appointment booking by clicking the "Yes" button. If there is no error, the system will display a success message to the patients and redirect the user to the dashboard page.

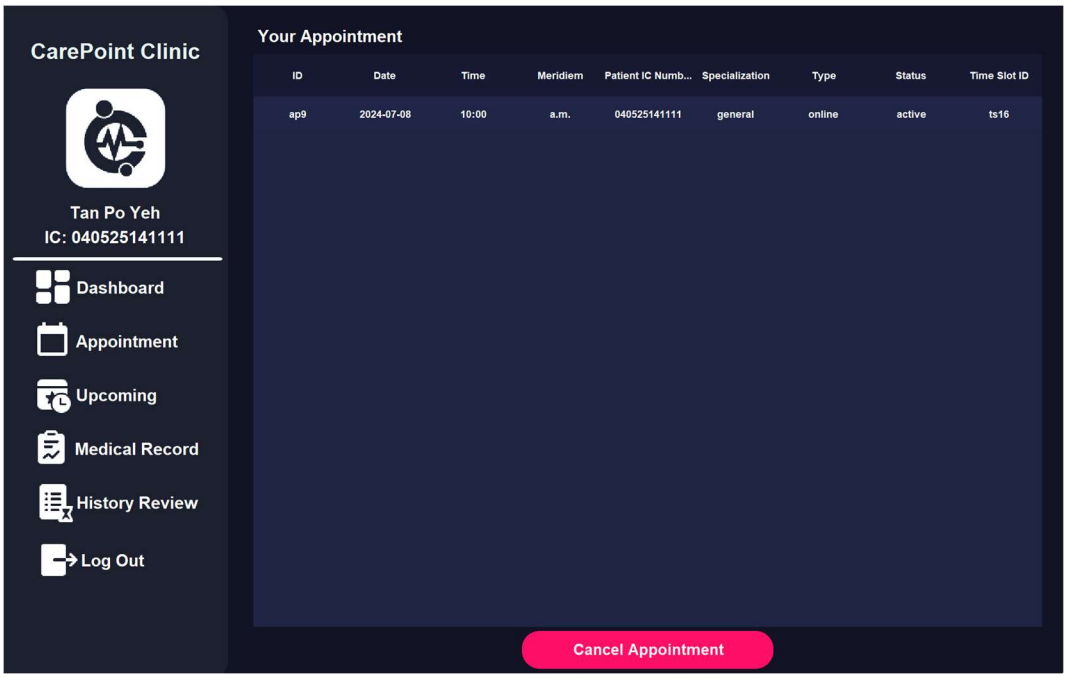


On the dashboard page, if the appointment is successfully booked. The dashboard will display the total number of appointments currently booked by the patient. Here, the number of incoming appointments changes from 0 to 1 as the patients just make an appointment and the total available appointments decrease from 9 to 8.

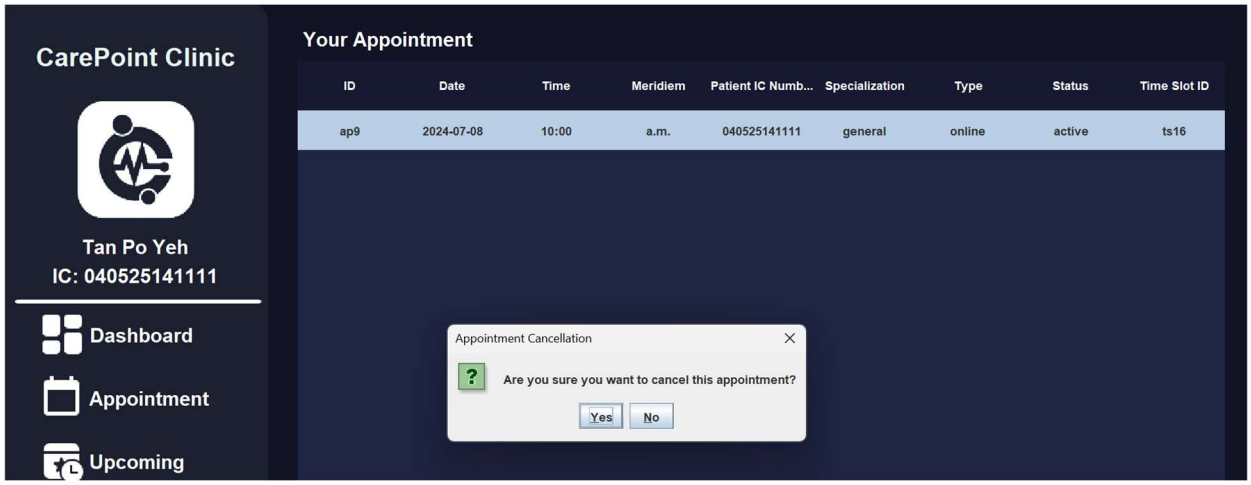


2.3.3 Upcoming - Cancel Appointment

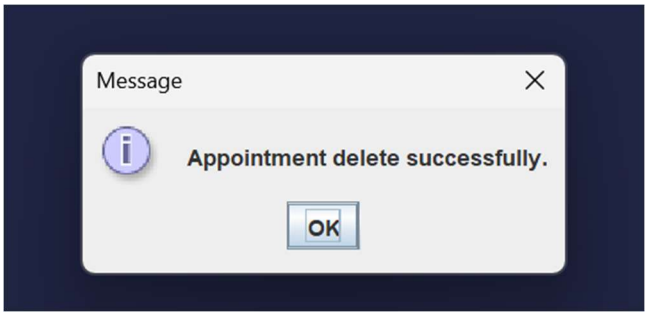
To cancel or view the reserved appointments, patients can click the "Upcoming" button on the left panel. On this page, patients are able to cancel the reserved appointment.



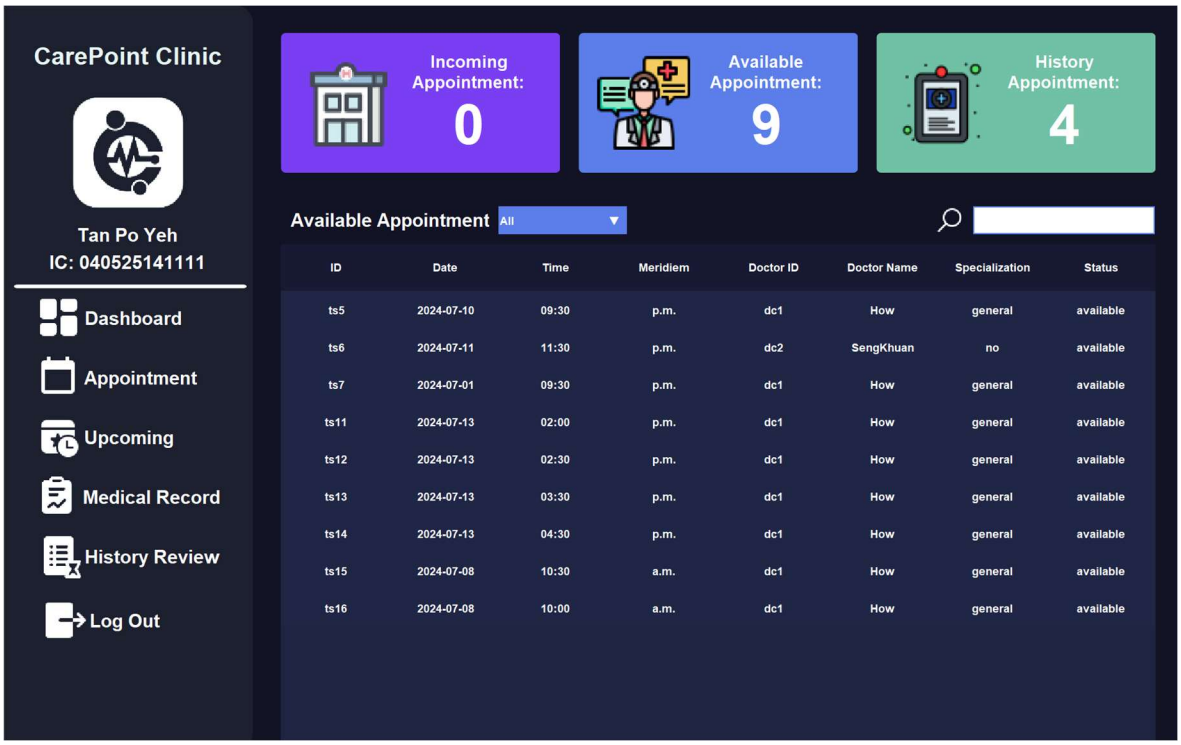
By selecting the appointment want to cancel, patients can just click the "Cancel Appointment" button to cancel an appointment. The system will display a confirmation panel for patients to double-confirm



The patients can process the deletion by clicking "Yes" on the pop-up message. If the appointment is deleted successfully, the system will display a success message to the patients.

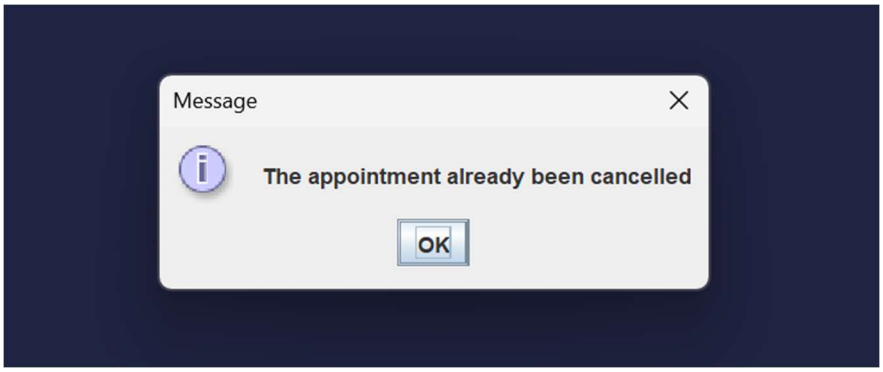


The system will direct the patients to the dashboard page and the three indicators on the above will change.



The canceled appointment will remain on the upcoming page for viewing purposes. The canceled record will not able to be canceled on this page.

The system will display an error message whenever the patients try to delete an invalid appointment (cancelled appointment).


Message
The appointment already been cancelled
OK

2.3.4 Medical Record - Track Personal Medical Record

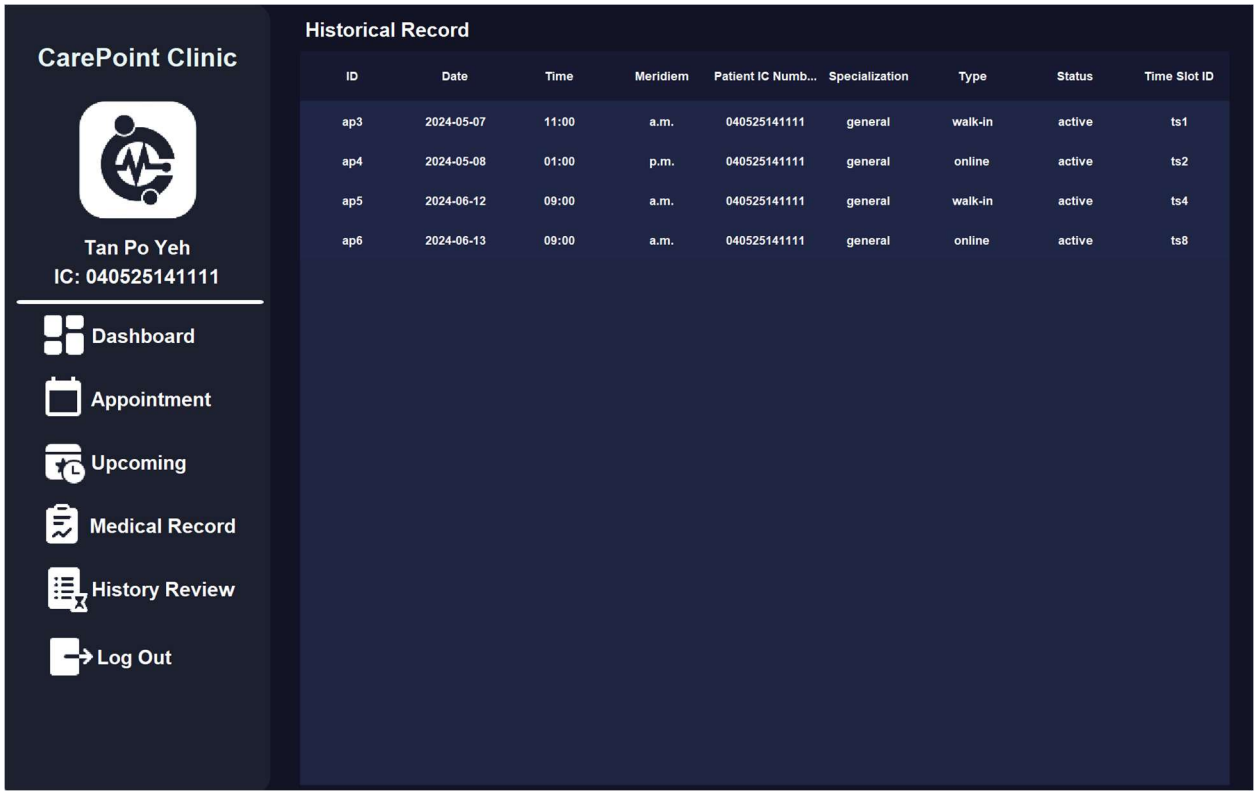
The fourth feature for the patients is the personal medical record, this features allow the patients to review their own personal medical records provided by the doctor. The patient can access the medical record page by clicking the "Medical Record" button on the left page.

Once clicked, patients are able to review the medical records belong to them. The medical record contains the specific time the doctor gave this record and the doctor's information as well as the patient's information. Below are the medical record page in our system.

2.3.5 History Review - Track Personal Historical Appointment

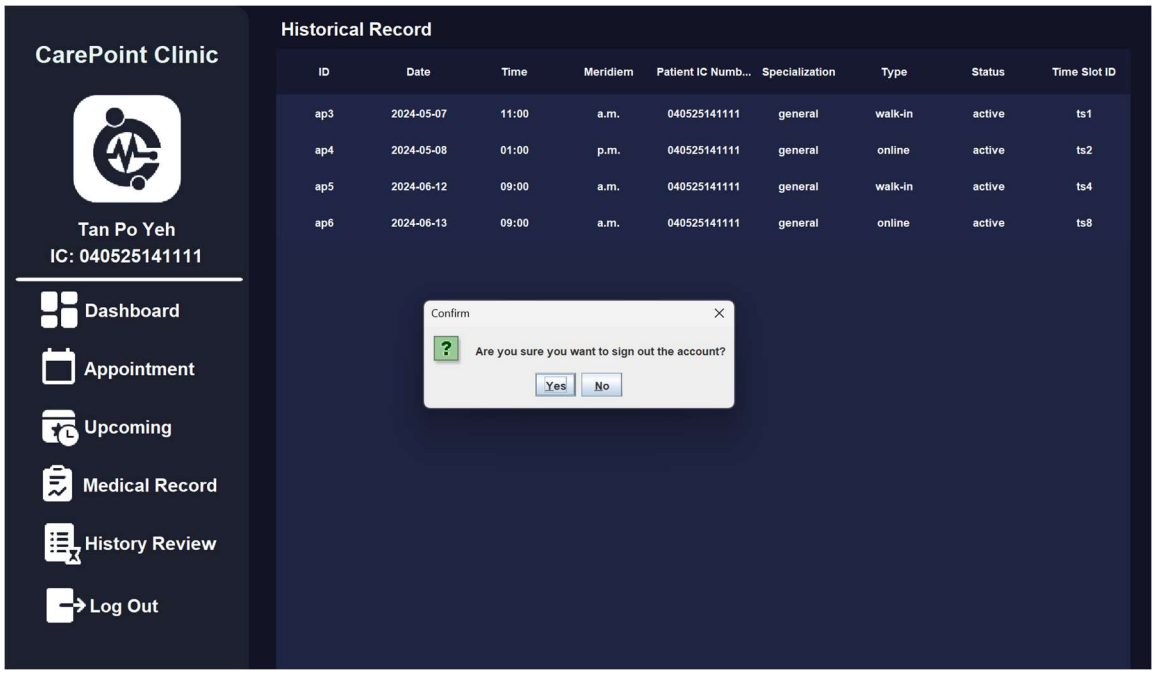
The fifth feature for the patients is the personal historical appointment review system, these features allow the patients to review their own historical appointment. The patient can access the record page by clicking the "History Review" button on the left section.

Once clicked, patients are able to review the historical appointment records belong to them. The historical appointment record contains the appointment date, time, type and patient information. Below is the historical appointment page in our system.

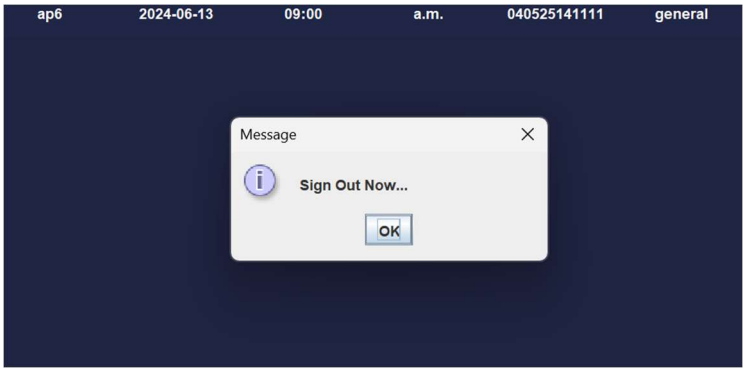


2.3.6 Log Out

The patient is able to log out their account by clicking the "Log Out" button on the left section on the page. Once the "Log Out" button is clicked, the system will display out the confirmation panel to enable the patient double-confirm the decision.



The system will then display a success message to the patient like the picture shown below. By clicking "Ok", patients will be redirect to the login page of the system.



3.0 Object Oriented Concepts

Java is one of the most popular Object-Oriented Programming (OOP) languages in the world. According to Kyle (2023), OOP organized a set of data attributes along with methods into object. (Kyle Herrity, 2023) It acts as a blueprint for creating objects that contain similar attributes and behavior. Each class in Java can instantiate multiple objects (instances), and each instance operates independently of the others.

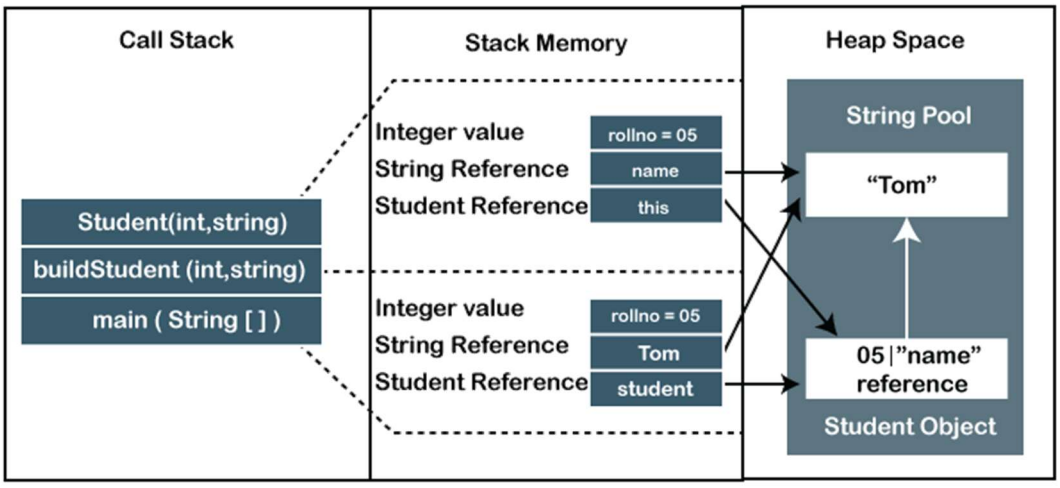
Object-Oriented Programming in Java is built around four fundamental principles that provide the foundation for creating robust and maintainable software.

3.1 Class and Object

Class and Object are the main concepts in the realm of Object-Oriented Programming. A class is essentially a blueprint or template for creating objects. It defines a set of attributes and methods that the objects created from the class will possess. Classes can be imagined as the design for a house, specifying what features the house will have and what facilities can be used. This system has various User Interface pages to show information to the user. For example, “DoctorAccUI” class defines the structure and properties of the doctor account management page. It declares 5 buttons, 7 panel boxes, 3 textbox and 1 table field inside the page. Additionally, it includes variables such as “header” and “data” to store the doctor account data for display in the table.

```
public class DoctorAccUI extends JFrame {  4 usages  wilson
    Doctors doctors = new Doctors();  11 usages
    Object[] header;  7 usages
    Object[][] data;  13 usages
    int selectedRow = -1;  9 usages

    private JPanel MainPanel;  2 usages
    private JLabel Title;  1 usage
    private JToolBar TB1;  1 usage
    private JToolBar.Separator TB1TBS1;  1 usage
    private JButton EDITButton1;  2 usages
    private JToolBar.Separator TB1TBS3;  1 usage
    private JButton DELETEButton;  2 usages
    private JToolBar.Separator TB1TBS4;  1 usage
    private JButton NEWButton;  2 usages
    private JButton searchButton;  2 usages
    private JButton refreshButton;  2 usages
    private JPanel P2;  2 usages
    private JLabel P2L1;  1 usage
    private JScrollPane P2SP1;  1 usage
    private JTable Table;  10 usages
    private JPanel P1;  2 usages
    private JLabel dataInputTitle;  7 usages
    private JSeparator P1S1;  1 usage
    private JScrollPane P1ScrollPanel;  1 usage
    private JPanel dataPanel;  1 usage
    private JPanel dataInputPanel;  27 usages
    private JPanel dataDescriptionPanel;  3 usages
    private JPanel controlPanel;  5 usages
```

An object is an instance of a class. When a class is instantiated, memory is allocated on the heap for the object, and its attributes are initialized. In Java, the lifecycle and resource management of objects are handled by the Java Virtual Machine (JVM). The JVM is responsible for allocating memory for objects, managing their lifetime, and releasing resources through a process called garbage collection. Each object has its own unique memory address and state, even if they created from same class. The diagram below shows the memory allocation for creating objects. The object will be placed into Heap Space after initializing. It will be removed from the space once the object is no longer needed or referenced.



For instance, an admin can click on the “doctorAccountButton” to create a new instance of the “DoctorAccUI” class. This action involves the “new” command, which directs the system to allocate memory for the new object. Consequently, a new doctor account management page appears. If there is an existing reference in “doctorAccUI” variable, this reference will be reset and refer to a new “DoctorAccUI” object. The old object is released from memory because the new “DoctorAccUI” object is distinct from the previous one, even though they are instantiated from the same class.

```
82          doctorAccountButton.addActionListener(new ActionListener() {  wilson
83              @Override  wilson
84              public void actionPerformed(ActionEvent e) {
85                  if (doctorAccUI != null) {
86                      doctorAccUI.dispose();
87                  }
88                  doctorAccUI = new DoctorAccUI();
89              }
90          });
91
```

3.2 Encapsulation

Encapsulation involves bundling data (variables) and methods that operate on the data into a single unit, known as a class. (Thorben, 2024) This concept ensures that the internal state of an object is hidden from the outside and can only be accessed or modified through a controlled interface. Access Modifiers are key to implementing encapsulation in Java. Java provides four types of access modifier that control the visibility and accessibility of class members such as private, protected, public and default.

Members declared as private are accessible only within the class in which they are defined. This level of access restricts external classes from directly interacting with the private members, ensuring data protection. The attributes inside “AppointmentData” class have the least accessibility, they only can be modified by the method inside the class. This encapsulation ensures the credential and integrity of these attributes.

```
6   public class AppointmentData extends Data {  wilson
7       private String date;  3 usages
8       private String time;  3 usages
9       private String meridiem;  3 usages
10      private String patientIcNumber;  3 usages
11      private String specialization;  3 usages
12      private String type;  3 usages
13      private String status;  3 usages
14      private String timeSlotId;  3 usages
```

Members marked as protected are accessible within their own package and by subclasses even if they are in different packages. This allows for controlled access, especially accessible in inheritance hierarchies. For example, the id with string type in “Data” class is marked as protected as this attribute can be reassigned value and read in subclasses. However, classes which do not inherit Data class cannot access the attribute.

```
package Data;

public class Data {  10 inheritors  wilson
    protected String id;

    public String getId() {  wilson
        return id;
    }
}
```

Members labeled as public are accessible from any other class. This provides the widest level of access, allowing methods and fields to be universally available. Public access is commonly used when certain functionality or variable wants to be used freely by other part of program. The “UserData” class contains two public methods, getter and setter for the “name” attribute. The getter method allows outside classes to retrieve the value of the “name” attribute, while the setter method enables them to modify it.

```java
public class UserData extends Data {  11 usages  3 inheritors  wilson
    protected String name;
    protected String icNumber;
    protected String pass;  6 usages
    protected String contact;

    public String getName () {  wilson
        return name;
    }

    public void setName(String name) {  wilson
        this.name = name;
    }
```

3.3 Inheritance

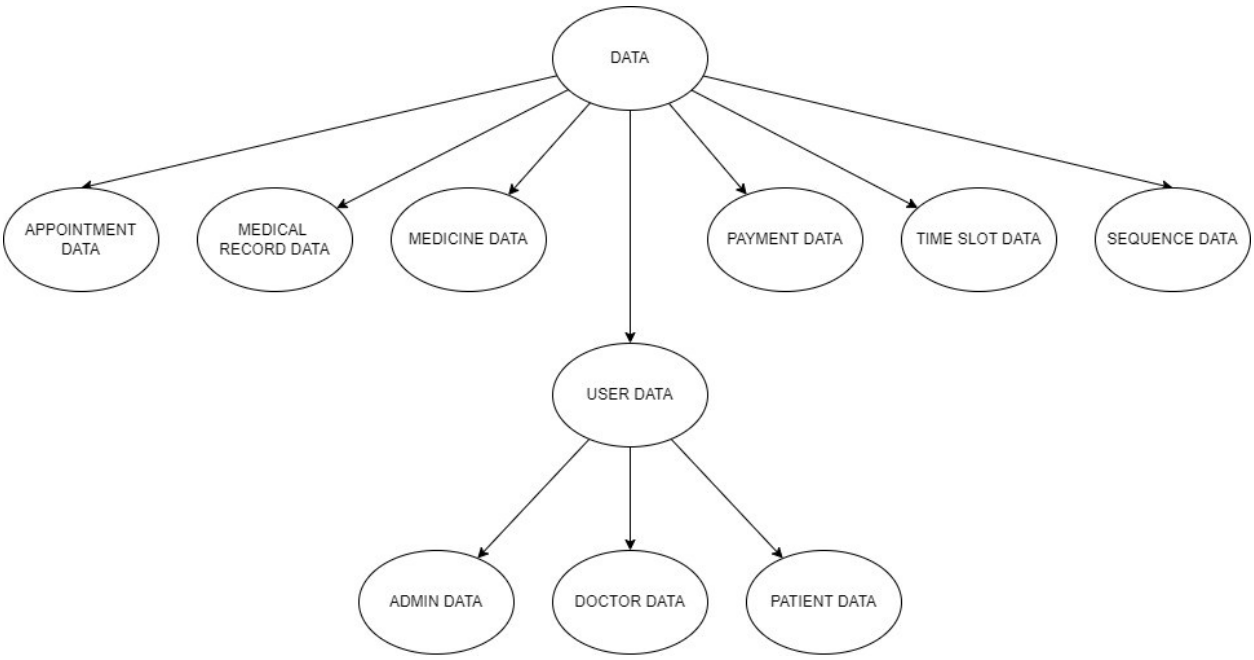
Inheritance is a fundamental concept in OOP that allows a child class (subclass) to inherit the attributes and methods from a parent class (superclass). This feature promotes code reuse and helps establish a hierarchical relationship between classes. In essence, common functionality defined in a superclass can be extended by its subclasses, which can then extend or override their own attributes and methods to perform additional functions.

```java
public class UserData extends Data {  11 usages  3 inheritors  wilson
    protected String name;
    protected String icNumber;
    protected String pass;  6 usages
    protected String contact;

    public String getName () { return name; }

    public void setName(String name) { this.name = name; }

    public String getIcNumber () { return icNumber; }

    public void setIcNumber(String icNumber) { this.icNumber = icNumber; }

    public String getPass () { return pass; }

    public String setPass(String newPass, String confirmPass) {...}

    public String getContact () { return contact; }

    public void setContact(String contact) { this.contact = contact; }

    // for login, to check the username and password is same or not
    public boolean login(String username, String password) {...}

    // changed wh
    public String checkIcNumber () {...}
    // changed wh
}
```

```java
public class DoctorData extends UserData {  29 usages  wilson
    private String gender;  3 usages
    private String specialization;  3 usages
    private String education;  3 usages
    private String experienceYear;  3 usages

    public DoctorData (String id, String name, String icNumber, String pass, String gender, String contact, String specialization, String education, String experienceYear)

    public String getGender () { return gender; }

    public String getSpecialization () { return specialization; }

    public String getEducation () { return education; }

    public String getExperienceYear () { return experienceYear; }

    public void setGender(String gender) { this.gender = gender; }

    public void setSpecialization(String specialization) { this.specialization = specialization; }

    public void setEducation(String education) { this.education = education; }

    public void setExperienceYear(String experienceYear) { this.experienceYear = experienceYear; }
    // changed wh
}
```

The “DoctorData” class extends the “UserData” class. This means “DoctorData” inherits the attributes such as “name”, “icNumber”, “pass” and “contact” from “UserData”, allowing these attributes to be reused without redeclaration. Additionally, “DoctorData” adds its own specific attributes such as “gender”, “specialization”, “education” and “experienceYear”, and provides methods to manage these attributes. This is because doctor has its own account information that other’s role might do not have.

The diagram below shows the inheritance relationship of our system.



3.4 Polymorphism

Polymorphism allows objects of different classes to implement one or more methods defined in a superclass. This is particularly useful when we want to execute a common action in response to user events, such as button clicks. In our Java development, the “ActionListener” interface is commonly used to handle these events. Specifically, the “LoginUI” class needs an interface to enable user interaction with our system. By implementing the “ActionListener” interface, the “LoginUI” class gains access to the “actionPerformed” method, which handles button events.

```
45  public interface ActionListener extends EventListener {
46
        Invoked when an action occurs.
        Params: e – the event to be processed
51      public void actionPerformed(ActionEvent e);
52
53  }
```

```
14  public class LoginUI extends JFrame implements ActionListener {  7 usages  wilson
15      String role;  5 usages
16
17      private final JLayeredPane layeredPane = new JLayeredPane();  11 usages
18      private final JPanel sidePanel = new JPanel();  10 usages
19      private final JPanel whitePanel = new JPanel();  11 usages
20      private final JLabel loginTitle = new JLabel( text: "Welcome To CarePoint Clinic");  5 usages
21      private final JLabel loginSubtitle = new JLabel( text: "Proceed By Logging Into Your Account");  5 usages
22      private final JPanel divisionLine = new JPanel();  7 usages
23      private final JTextField userIc;  3 usages
24      private final JPasswordField userPass;  3 usages
25      private final JPanel buttonsPanel = new JPanel();  10 usages
26      private final JPanel blackPanel = new JPanel();  9 usages
27      private final JButton loginPatientButton = RoundedButton.createRoundedButton( text: "Patient Login",  radius: 60,  thickness: 3, new Color( r: 246,  g: 246,  b: 246));  7 usage
28      private final JButton loginDoctorButton = RoundedButton.createRoundedButton( text: "Doctor Login",  radius: 60,  thickness: 3, new Color( r: 246,  g: 246,  b: 246));  7 usages
29      private final JButton loginAdminButton = RoundedButton.createRoundedButton( text: "Admin Login",  radius: 60,  thickness: 3, new Color( r: 246,  g: 246,  b: 246));  6 usages
30      private final JButton clinicButton = RoundedButton.createRoundedButton( text: "Clinic CarePoint",  radius: 60,  thickness: 2, new Color( r: 246,  g: 246,  b: 246));  7 usages
31      private JButton loginButton = RoundedButton.createRoundedButton( text: "Login",  radius: 60,  thickness: 2, new Color( r: 255,  g: 15,  b: 100));  9 usages
32      private JButton forgetPassButton = RoundedButton.createRoundedButton( text: "Show Password",  radius: 60,  thickness: 2, new Color( r: 255,  g: 15,  b: 100));  9 usages
```

The “LoginUI” class provides its own implementation of the “actionPerformed” method defined in the “ActionListener” interface. Through method overrding, the “LoginUI” class can exhibit different behavior such as submitting login input, applying forget password feature and animation reset.

```
    @Override  wilson
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == loginPatientButton || e.getSource() == loginDoctorButton || e.getSource() == loginAdminButton) {
            animateComponents();
            updateUIForLoginType(e.getSource());
        } else if (e.getSource() == clinicButton) {
            resetAnimateComponents();
            updateUIForClinicButton();
        } else if (e.getSource() == loginButton) {
            String username = userIc.getText();
            String password = String.valueOf(userPass.getPassword());
            switch (role) {
                case "patient":
                    Patients patients = new Patients();
                    patients.PatientLogin(username, password,  frame: this);
                    break;
                case "doctor":
                    Doctors doctors = new Doctors();
                    doctors.DoctorLogin(username, password,  frame: this);
                    break;
                case "admin":
                    Admins admins = new Admins();
                    admins.AdminLogin(username, password,  frame: this);
                    break;
            }
        } else if (e.getSource() == forgetPassButton) {
            new ForgetPassUI( parent: this, role);
        }
    }
```

4.0 Java Features

1.Exception Handling:

```
while (true) {
    try {
        // Show the input dialog
        String input = JOptionPane.showInputDialog(dialog, message, title: "Confirm Payment", messageType);
        if (input == null) {
            // User clicked Cancel or closed the dialog
            return;
        }

        // Parse the input value
        payAmount = Float.parseFloat(input);
        if (payAmount < sumOfAmount) {
            message = "Paid amount is less than the required amount";
            messageType = JOptionPane.ERROR_MESSAGE;
        }
        else {
            break;
        }
    } catch (NumberFormatException err) {
        // Input is not a valid currency value
        message = "Invalid input! Please enter a valid currency amount:";
        messageType = JOptionPane.ERROR_MESSAGE;
    }
}
```

The code as shown above manages patient invoicing with the need for user to input a specific payment amount using the JOptionPane.showInputDialog .The user input is processed as a float; if this is unsuccessful, a NumberFormatException occurs, and the user gets prompted again with an error message. If the amount is insufficient, the user gets notified and the program repeats. This ensures, that the user must enter a specific payment amount.

2. IF, Else :

```
Component comp;
for (int j = 0; j < header.length; j++) {
    if (header[j].equals("Price")) {
        NumberFormat format = NumberFormat.getCurrencyInstance(new Locale( language: "ms",  country: "MY"));
        format.setMinimumFractionDigits(2);
        format.setMaximumFractionDigits(2);
        NumberFormatter formatter = new NumberFormatter(format) {  wilson
            @Override  11 usages  wilson
            public Object stringToValue(String text) throws ParseException {
                if (text == null || text.trim().isEmpty()) {
                    return null;
                }
                return super.stringToValue(text);
            }
        };
        formatter.setMinimum(0.0);
        formatter.setAllowsInvalid(false);
        JFormattedTextField priceField = new JFormattedTextField(formatter);
        priceField.setColumns(10);
        priceField.setEditable(false);
        comp = priceField;
    }
    else {
        JTextField inputField = new JTextField();
        inputField.setEditable(false);
        comp = inputField;
    }
```

This program starts by looping through the header array. IF the header is “Price”, the program creates “JFormattedTextField” for user input. Furthermore, A “NumberFormatter” is used to make sure correct currency input and the value is not negative. The “JFormattedTextField” is created with a formatter, set to a specific width (setColumns(10)) and to make sure is non-editable. The “Else” is used to identify if the header if is not “Price” then the programs will create a “JTextField” and this field is set to non-editable .

3. Try - with -Resources :

```
public class File {  wilson

    public static void readFile(Sequences sequences) {  1 usage  wilson

        try (BufferedReader reader = new BufferedReader(new FileReader( fileName: "Resource/sequence.txt"))) {
            sequences.addSequenceHeader(reader.readLine());
            for (String data = reader.readLine(); data != null; data = reader.readLine()) {
                String[] dataArray = data.split( regex: ",");
                SequenceData sequence = new SequenceData(dataArray[0], dataArray[1], dataArray[2], dataArray[3]);
                sequences.addSequenceData(sequence);
            }
        } catch (IOException err) {
            throw new RuntimeException(err);
        }
    }
```

The readFile method in Java illustrates quick file processing and strong error management by utilizing the java features such as try-with-resources and exception handling. In the procedure, the try block creates a BufferedReader called reader to read the file "Resource/sequence.txt". This usage of try-with-resources makes sure that the BufferedReader is immediately closed when the block exits, such and throws an IOException. For example, if an IOException occur, like when a file is missing or unreadable, the catch block quickly solves the issue. After that it throws a RuntimeException, which encapsulates the original IOException. In addition to that, this step ensures that the BufferedReader resource is correctly run and terminate properly.

4. File.IO :

```
    public static void readFile(Medicines medicines) {  1 usage  wilson
        try (BufferedReader reader = new BufferedReader(new FileReader( fileName: "Resource/medicine.txt"))) {
            medicines.addMedicineHeader(reader.readLine());
            for (String data = reader.readLine(); data != null; data = reader.readLine()) {
                String[] medicineInfo = data.split( regex: ",");
                MedicineData medicine = new MedicineData(medicineInfo[0], medicineInfo[1], medicineInfo[2], medicineInfo[3]);
                medicines.addMedicineList(medicine);
            }
        } catch (IOException err) {
            throw new RuntimeException(err);
        }
    }
```

Java's BufferedReader class is crucial for effective file reading, particularly when working with text files. By doing this so, buffering input and reducing the amount of read operations on the underlying I/O flow.The readLine() function, whiach reads a line of text one at a time, is one of its primary characteristics. This makes it easier to handle text files results in cleaner and easy to read code. ReadLine() provides a whole line as a string rather than reading individual characters and looking for line terminators, which is particularly helpful for structured text files. Moreover, BufferedReader also manages resources efficiently. When used in a try-with-resources statement, it ensures the file stream is properly closed after operations, even if exceptions occur, preventing resource leaks and related issues. In summary, BufferedReader and readLine() method improves performance and simplifies file reading in Java.

5.While Loop: Creates an infinite loop with condition

```
while (true) {
    String input = JOptionPane.showInputDialog(frame, message, title: "Deleting Confirmation", messageType);
    if (input == null) {
        return;
    }
    if (Authentication.matchPassword(input, Cookie.getAdminCookie().getPass())) {
        break;
    }
    message = "Wrong password, please try again";
    messageType = JOptionPane.ERROR_MESSAGE;
}
```

The while (true) loop generates an endless loop that is can be broken out of under certain conditions. In the loop, the JOptionPane.showInputDialog function displays a dialog box that requests input from the user. In context to that, if the user enters a password, the code uses Authentication.matchPassword to see whether it matches the admin password that is saved in a file. The program continues if the passwords match, it will break out of the loop using a break statement as shown above. The user will be prompted to try again if the passwords do not match and if the user terminates the dialog or quits the program, the input is null and the process returns, ending the loop and its execution. In summary, this to make sure that the user must provide correct input before proceeding.

6.Switch Case:

```
else if (header[j].equals("Time") || header[j].equals("Meridiem") || header[j].equals("Specialization")) {
    String combo = (String) header[j];
    String[] list = switch (combo) {
        case "Time" -> generateTimeSlots();
        case "Meridiem" -> new String[]{"- none -", "a.m.", "p.m."};
        case "Specialization" -> specialization = doctorSpecialization.toArray(specialization);
        default -> throw new IllegalStateException("Unexpected value: " + combo);
    };
    JComboBox<String> comboBox = new JComboBox<>(list);
    comboBox.setEditable(false);
    comboBox.setEnabled(false);
    comboBox.setRenderer((DefaultListCellRenderer) paint(g) -> {
            setForeground(Color.BLACK);
            super.paint(g);
    });
    comp = comboBox;
}
```

The switch statement is an effective control structure that improves code readability. The code first checks if the current header value matches "Time", "Meridiem", or "Specialization". If it does, it assigns the header value to the variable combo. The switch statement then evaluates the combo and executes the corresponding case block as shown below.

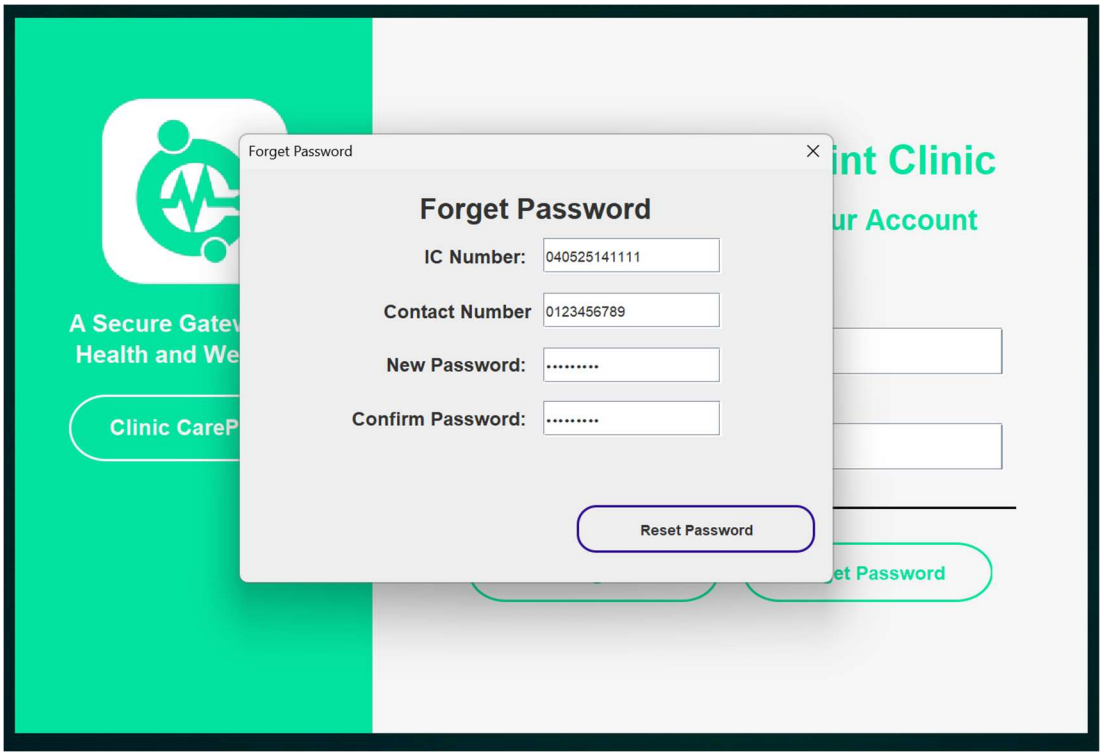
- **Case "Time"**: If the header value is "Time", it calls the method generateTimeSlots(), which presumably returns an array of time slots. These slots are then used to populate the JComboBox.
- **Case "Meridiem"**: If the header value is "Meridiem", it assigns an array containing "- none -", "a.m.", and "p.m." to the JComboBox.
- **Case "Specialization"**: If the header value is "Specialization", it converts a list of specializations to an array and assigns it to the JComboBox.

However, if the cases don't match the default case, it throws an IllegalStateException with a message indicating an unexpected value. In a nutshell, the switch statement in this Java code manages the content of a JComboBox based on different header values.

5.0 Additional features

5.1 Forget Password

The forget password features are able to be accessed by all the users in the system, including patients, doctors and admins. This feature allows the user to reset their password by entering their IC number and phone number registered in the system/clinic. Users are able to access this feature by clicking the "Forget Password" button.



Then, users will be able to see a pop-up panel for resetting the password. To reset the password, users just need to enter the correct IC Number and Contact Number. Next, the user will need to enter a new password in the length of between 8 to 16 characters with a combination of one digit, one uppercase letter and one lowercase letter. Otherwise, the system will display an error message to the user.

The setPass() method is responsible for checking and validating the entered password length and character so the password fits the condition of between 8 to 16 characters with a combination of one digit, one uppercase letter and one lowercase letter.

```
38 @    public String setPass(String newPass, String confirmPass) {  1 usage  wilson
39          boolean numberFlag = false;
40          boolean capitalFlag = false;
41          boolean lowercaseFlag = false;
42          if (newPass.length() < 8 || newPass.length() > 16) {
43              return "Your password length must be between 8 and 16 characters";
44          }
45          for (int i = 0; i < newPass.length(); i++) {
46              if (Character.isDigit(newPass.charAt(i))) {
47                  numberFlag = true;
48              }
49              if (Character.isUpperCase(newPass.charAt(i))) {
50                  capitalFlag = true;
51              }
52              if (Character.isLowerCase(newPass.charAt(i))) {
53                  lowercaseFlag = true;
54              }
55              if (numberFlag && capitalFlag && lowercaseFlag) {
56                  if (newPass.equals(confirmPass)) {
57                      this.pass = Authentication.hashPassword(newPass);
58                      return null;
59                  }
60                  return "Confirm password does not match";
61              }
62          }
63          return "Your password must contain at least one digit, one uppercase letter, and one lowercase";
64      }
```

Lastly, click the "Reset Password" button to proceed with the process. Once everything is done, the system will display a success message to the user.


```
15  public class ForgetPassUI extends JDialog {  3 usages  wilson
37      private void setButton(String role) {  1 usage  wilson
42          resetPassBtn.addActionListener(new ActionListener() {  wilson
43              @Override  wilson
44              public void actionPerformed(ActionEvent e) {
45                  String icNum = icField.getText().trim();
46                  String contactNum = contactField.getText().trim();
47                  String newPassword = new String(newPasswordField.getPassword()).trim();
48                  String confirmPassword = new String(confirmPasswordField.getPassword()).trim();
49
50                  Object users = null;
51                  Object[] checkICStatus = switch (role) {
52                      case "patient" -> {
53                          Patients patients = new Patients();
54                          users = patients;
55                          yield patients.checkPatientIc(icNum);
56                      }
57                      case "doctor" -> {
58                          Doctors doctors = new Doctors();
59                          users = doctors;
60                          yield doctors.checkDoctorIc(icNum);
61                      }
62                      case "admin" -> {
63                          Admins admins = new Admins();
64                          users = admins;
65                          yield admins.checkAdminIc(icNum);
66                      }
67                      default -> new Object[2];
68                  };
69                  if (checkICStatus[1] == null) {
70                      JOptionPane.showMessageDialog( parentComponent: null, checkICStatus[0]);
71                      return;
72                  }
```

```java
public Object[] checkPatientIc(String icNumber) {  2 usages  wilson
    Object[] statusMessage = new Object[2];
    if (icNumber.length() != 12) {
        statusMessage[0] = "Invalid IC number.";
    }
    for (int i=0; i < icNumber.length(); i++) {
        if (!Character.isDigit(icNumber.charAt(i))) {
            statusMessage[0] = "Invalid IC number.\n";
            break;
        }
    }

    if (statusMessage[0] != null) {
        return statusMessage;
    }

    List<PatientData> queriedPatient = queryPatient( queryId: null,  queryName: null, icNumber,  queryGender: null,  queryDob: null,  queryContact: null);
    if (queriedPatient.isEmpty()) {
        statusMessage[0] = "Patient IC Number does not exist in the system.";
    }
    else {
        statusMessage[1] = queriedPatient.getFirst();
    }
    return statusMessage;
}
```

```java
public class ForgetPassUI extends JDialog {  3 usages  wilson
    private void setButton(String role) {  1 usage  wilson
        resetPassBtn.addActionListener(new ActionListener() {  wilson
            public void actionPerformed(ActionEvent e) {
                    }
                    default -> new Object[2];
                };
                if (checkICStatus[1] == null) {
                    JOptionPane.showMessageDialog( parentComponent: null, checkICStatus[0]);
                    return;
                }
                UserData queriedUser = (UserData) checkICStatus[1];
                if (!queriedUser.getContact().equals(contactNum)) {
                    JOptionPane.showMessageDialog( parentComponent: null,  message: "Contact number does not match");
                    return;
                }
                String changePassMessage = queriedUser.setPass(newPassword, confirmPassword);
                if (changePassMessage != null) {
                    JOptionPane.showMessageDialog( parentComponent: null, changePassMessage);
                    return;
                }
                if (users instanceof Patients patients) {
                    patients.updatePatient();
                }
                else if (users instanceof Doctors doctors) {
                    doctors.updateDoctor();
                }
                else if (users instanceof Admins admins) {
                    admins.updateAdmin();
                }
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Password changed successfully");
                dialog.dispose();
            }
        });
    }
}
```

```
148    public void updatePatient() {  3 usages  wilson
149        StringBuilder stringBuilder = new StringBuilder();
150        stringBuilder.append(String.join( delimiter: ",", patientHeaders)).append("\n");
151        for (PatientData patient : patientDataList) {
152            stringBuilder.append(patient.getId()).append(",")
153                    .append(patient.getName()).append(",")
154                    .append(patient.getIcNumber()).append(",")
155                    .append(patient.getPass()).append(",")
156                    .append(patient.getGender()).append(",")
157                    .append(patient.getDob()).append(",")
158                    .append(patient.getContact())
159                    .append("\n");
160        }
161        File.writeFile( fileName: "patient.txt", stringBuilder.toString());
162    }
```

5.2 Hashing Password

Hashing passwords is another feature that is vital in our system. When a user registers or updates their password, the system will hash the password using a secure hashing. This feature protects the user's password from being exposed in its plain text form, even if the database is compromised.

```
9   public class Authentication {  wilson
10      public static Boolean matchPassword (String inputPassword, String accountPassword) {  2 usages
11          String hashedPassword = hashPassword(inputPassword);
12          return hashedPassword.equals(accountPassword);
13      }
14
15      public static String hashPassword(String password) {  4 usages  wilson
16          try {
17              // called with algorithm SHA-512
18              MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-512");
19              // digest the password into byte
20              byte[] passDigest = md.digest(password.getBytes(StandardCharsets.UTF_8));
21              BigInteger bi = new BigInteger( signum: 1, passDigest);
22              // convert password into hex value
23              return bi.toString( radix: 16);
24
25          } catch (NoSuchAlgorithmException e) {
26              throw new RuntimeException(e);
27          }
28      }
29  }
```

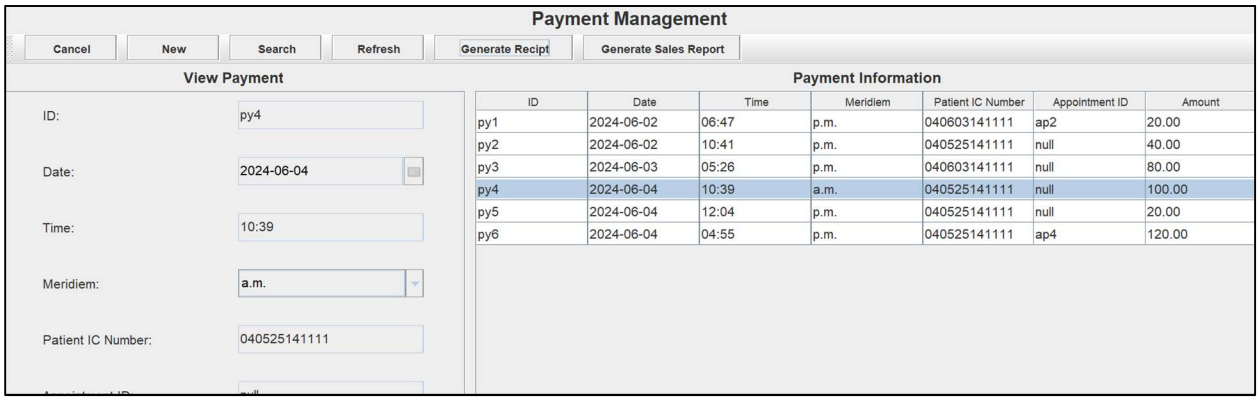
The input password is hashed using the SHA-512 algorithm. This is done in the hashPassword() method. Below is the hashed password sample from the test user.

```
Password,Gender,Date Of Birth,Contact Number                                                    8
1111,1b80cbd7e32c58935b8cf9c7312ba5f357368aa6442d1eeb9f6ce6a9f8b0b6536aebe5c69b516347d03a773d33c983c1526e22c7292b093383fc64f68fc9b5f5,male
111,3627909a29c31381a071ec27f7c9ca97726182aed29a7ddd2e54353322cfb30abb9e3a6df2ac2c20fe23436311d678564d0c8d305930575f60e2d3d048184d79,femal
11,c7b3a4823957edfd947280fe488c6c31808f5a46bfee48d7aa6bf6b261f84c0945254602bfcac22f05b1088c95d5d3159751bfbd88faf2b6ef1e020f276a061e,male,2
41111,c965e59d9fae45ccde91d9f92711a40b5d7645d4408c2c94f3cffd7d3207a6005a7b65f0edc1ee36da1d0df017a6baab75a5e3501cd7e25c9f5653d178ce1256,fem
```

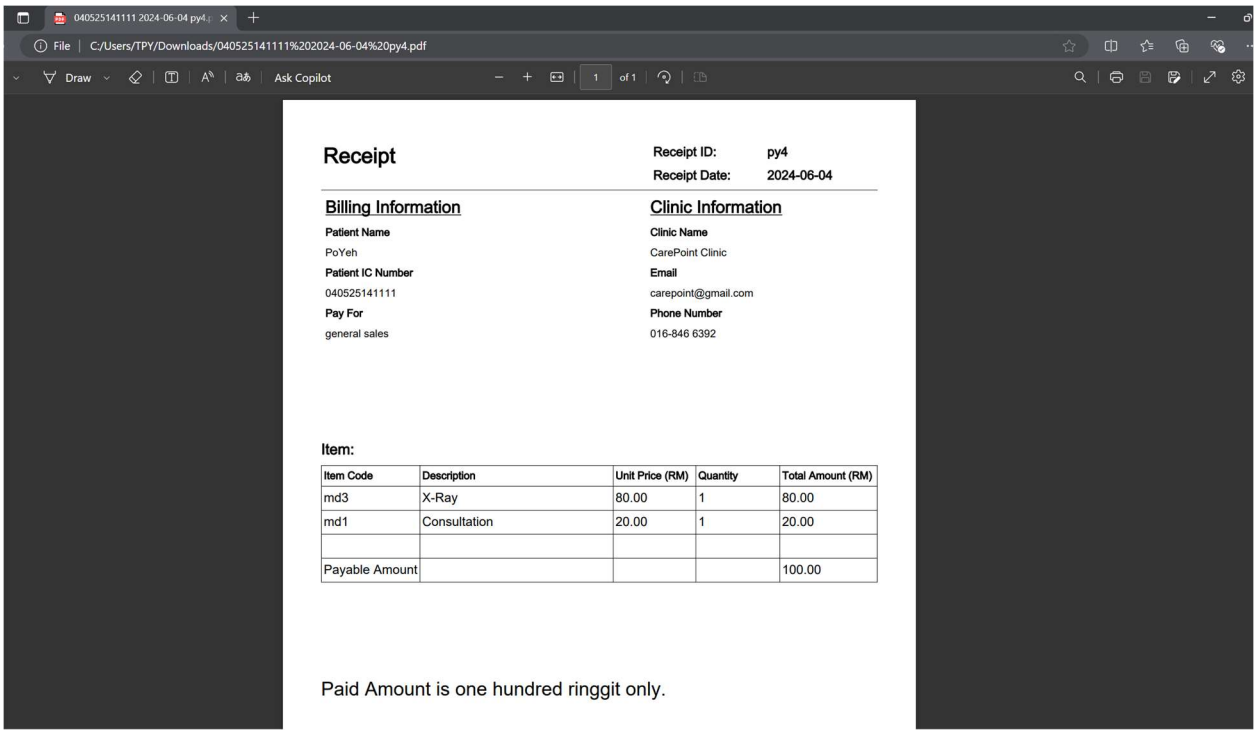
When a user attempts to log in, the system hashes the input password and compares it with the stored hashed password using the matchPassword() method. If the hashed passwords match, authentication is successful; otherwise, it fails.

5.3 Generate Payment Receipt

For the admin, there is an extra feature called Generate Payment Receipt. With this feature, admins are able to generate a payment receipt in the format of PDF. Admins are able to select specific receipts to generate, by simply selecting the target record in the table.



Once successful, the system will generate a PDF file to the admin machine. Below is a sample of a payment receipt belonging to one of the test users.



The generateReceipt() method is responsible for handling the receipt generation. The code first collects the selected row record data and sets them inside the PDF content. Below are the code processing the pdf.

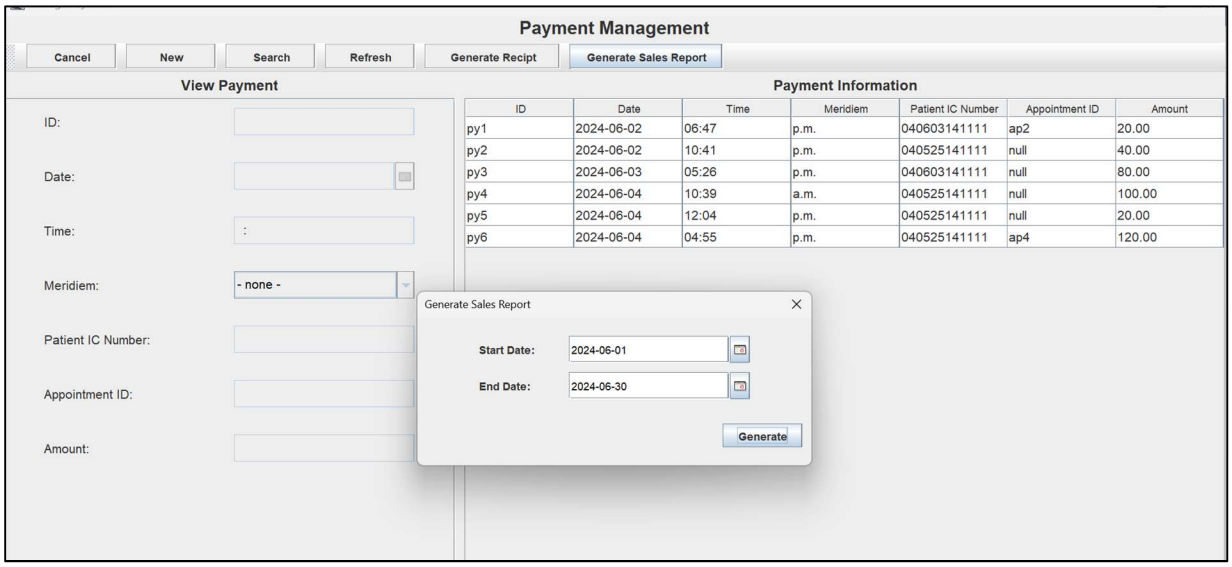
```
private void generateReceipt() {  1 usage  wilson *
    Medicines medicines = new Medicines();
    // get all data that may use for generating receipt
    String patientName = patientNameList[selectedRow];
    String[] medicineItem = paymentItemList[selectedRow];
    String[] quantityItem = paymentItemQuantityList[selectedRow];
    String paymentId = data[selectedRow][0].toString();
    String date = data[selectedRow][1].toString();
    String patientIcNum = data[selectedRow][4].toString();
    String appointmentId = data[selectedRow][5].toString();
    String payFor;
    if (appointmentId.equals("null")) {
        payFor = "general sales";
    }
    else {
        AppointmentData appointment = new Appointments().queryAppointment(appointmentId, queryDate: null,
                queryTime: null, queryMeridiem: null, queryPatientIcNumber: null, querySpecialization: null, queryType: null,
                queryStatus: null, queryTimeSlotId: null).getFirst();
        payFor = appointmentId + " (" + appointment.getDate() + " " + appointment.getTime() + appointment.getMeridiem() + ")";
    }
    String amount = data[selectedRow][6].toString();

    // set the downloaded path
    String os = System.getProperty("os.name").toLowerCase();
    String downloadDir = null;

    if (os.contains("win")) {
        // Windows OS
        downloadDir = System.getenv( name: "USERPROFILE") + "\\Downloads";
    } else if (os.contains("mac")) {
        // macOS
        downloadDir = System.getProperty("user.home") + "/Downloads";
    } else if (os.contains("nix") || os.contains("nux") || os.contains("aix")) {
        // Unix/Linux OS
```

The code below aims to construct the PDF structure using an external library, specifically the iText library. This library provides robust tools for creating and manipulating PDF documents in Java.

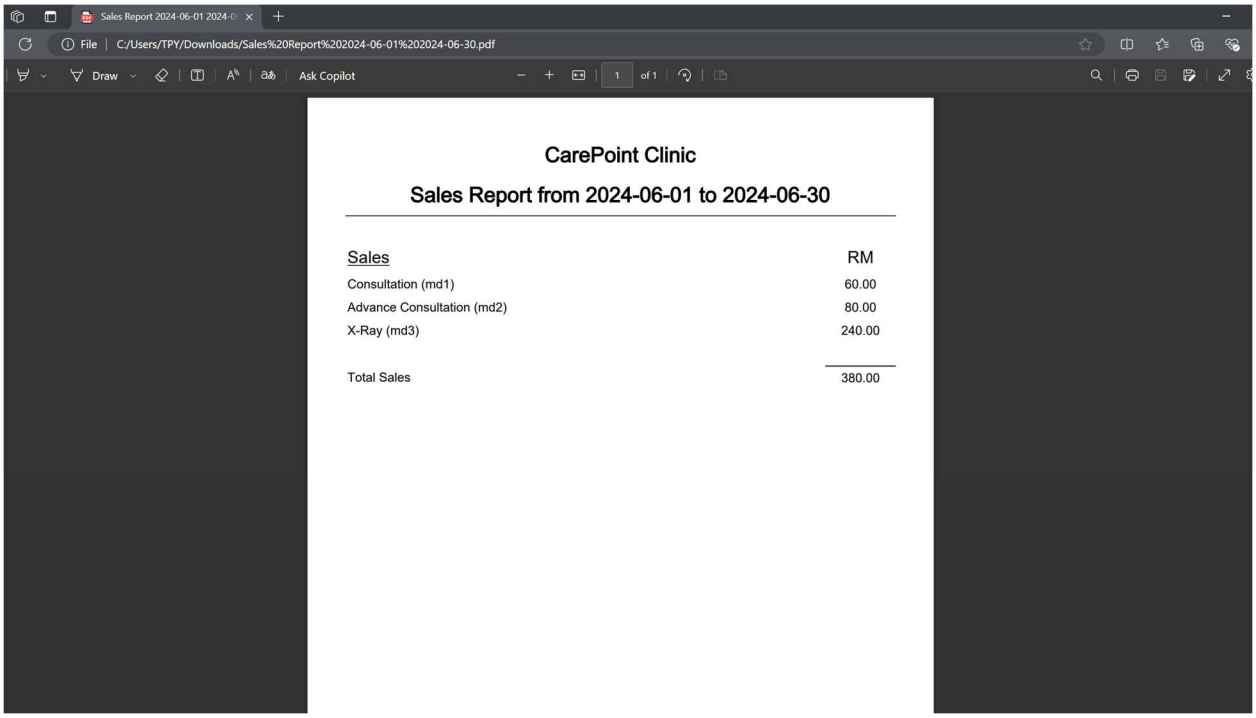
```
        // Unknown OS
        downloadDir = System.getProperty("user.home") + "/Downloads";
    }
    String fileName = patientIcNum + " " + date + " " + paymentId + ".pdf";
    String path = Paths.get(downloadDir, ...more: "/", fileName).toString();
    // generate pdf receipt
    try {
        PdfWriter pdfWriter = new PdfWriter(path);
        PdfDocument pdfDocument = new PdfDocument(pdfWriter);
        pdfDocument.setDefaultPageSize(PageSize.A4);
        Document document = new Document(pdfDocument);

        float headerSecondColumn = 285f;
        float headerFirstColumn = headerSecondColumn + 150f;
        float[] headerColumns = {headerFirstColumn, headerSecondColumn};

        // set receipt header
        Table headerTable = new Table(headerColumns);
        headerTable.addCell(new Cell().add(new Paragraph( text: "Receipt")).setFontSize(20f).setBorder(Border.NO_BORDER).setBold());
        Table nestedHeaderTable = new Table(new float[]{headerSecondColumn/2, headerSecondColumn/2});
        nestedHeaderTable.addCell(new Cell().add(new Paragraph( text: "Receipt ID: ")).setBold().setBorder(Border.NO_BORDER));
        nestedHeaderTable.addCell((new Cell().add(new Paragraph(paymentId))).setBold().setBorder(Border.NO_BORDER));
        nestedHeaderTable.addCell(new Cell().add(new Paragraph( text: "Receipt Date: ")).setBold().setBold().setBorder(Border.NO_BORDER));
        nestedHeaderTable.addCell(new Cell().add(new Paragraph(date)).setBold().setBorder(Border.NO_BORDER));
        headerTable.addCell(new Cell().add(nestedHeaderTable).setBorder(Border.NO_BORDER));
        document.add(headerTable);
```

5.4 Generate Sales Report

Other than the payment receipt, admins are also able to generate a sales report based on the date selected. By clicking the "Generate Sales Report" button, admins can specify the date range, and the system will generate a summary sales report for the admins. Below is the pop-up window of the date selection after clicking the button.



Once successful, the system will generate a sales report pdf as the picture below shows. In this case, the PDF will list out all the sales information followed by the corresponding cash flow.



The generateBtn (Generate Sales Report Button) has an action listener for reading the range of data and getting the values to the generateSalesReport() method through the parameter.

```
64        generateBtn.addActionListener(new ActionListener() {  wilson
65            @Override  wilson
66            public void actionPerformed(ActionEvent e) {
67                SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd");
68                if (startDateChooser.getDate() == null || endDateChooser.getDate() == null) {
69                    JOptionPane.showMessageDialog( parentComponent: null,  message: "Please select date range");
70                    return;
71                }
72
73                String startDate = formatter.format(startDateChooser.getDate());
74                String endDate = formatter.format(endDateChooser.getDate());
75                generateSalesReport(startDate, endDate);
76            }
77        });
```

The generateSalesReport() method is used for handling the sales report using iText Library. This report is generated based on payment data recorded in the system. If no sales are found within the specified date range, an error message is displayed to inform the user.

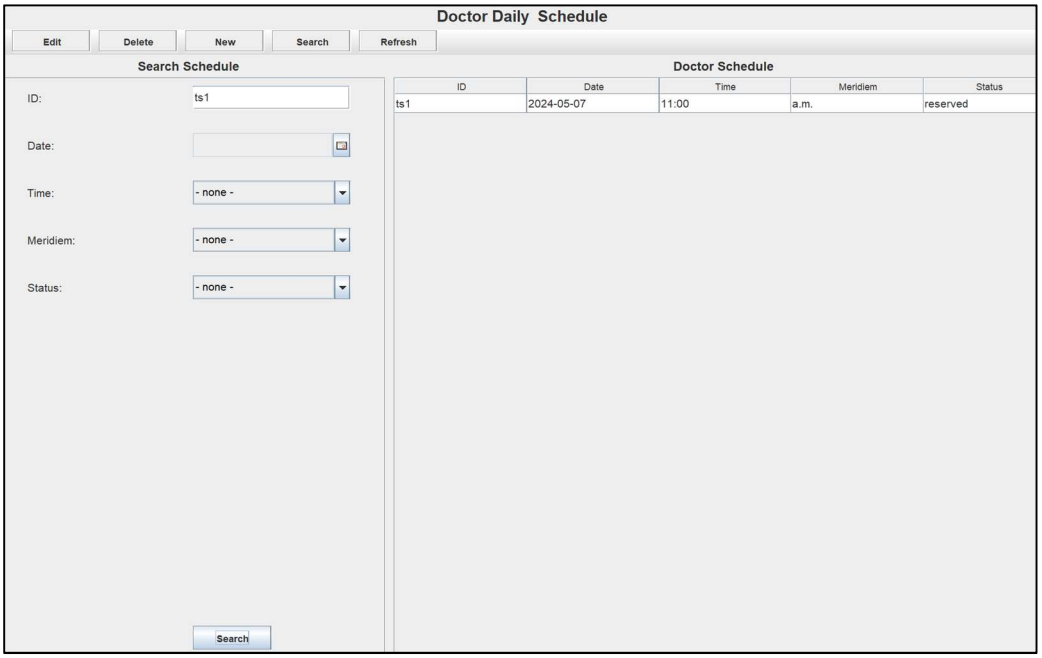
```
84    private void generateSalesReport(String startDate, String endDate) {  1 usage  wilson *
85        DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
86        LocalDate startLocalDate = LocalDate.parse(startDate, dateFormatter);
87        LocalDate endLocalDate = LocalDate.parse(endDate, dateFormatter);
88
89        Medicines medicines = new Medicines();
90        Payments payments = new Payments();
91        // retrieve payment that is between start date and end date
92        List<PaymentData> requiredPayment = payments.getPaymentList().stream()
93                .filter(payment -> startLocalDate.isBefore(LocalDate.parse(payment.getDate(), dateFormatter))
94                        && endLocalDate.isAfter(LocalDate.parse(payment.getDate(), dateFormatter)))
95                .toList();
96
97        if (requiredPayment.isEmpty()) {
98            JOptionPane.showMessageDialog( parentComponent: this,  message: "No Sales Between This Date Range");
99            return;
100       }
101
102       List<Object[]> medicineList = new ArrayList<>();
103       for (PaymentData payment : requiredPayment) {
104           String[] items = payment.getItems();
105           String[] quantities = payment.getQuantities();
106           for (int i=0; i< items.length; i++) {
107               itemExistInMedicineList(medicineList, items[i], quantities[i]);
108           }
109       }
110
111       String os = System.getProperty("os.name").toLowerCase();
112       String downloadDir = null;
```

Once successful, the system will generate a sales report PDF based on the date selection. Below are the codes that construct the PDF content.

```
114       if (os.contains("win")) {
115           // Windows OS
116           downloadDir = System.getenv( name: "USERPROFILE") + "\\Downloads";
117       } else if (os.contains("mac")) {
118           // macOS
119           downloadDir = System.getProperty("user.home") + "/Downloads";
120       } else if (os.contains("nix") || os.contains("nux") || os.contains("aix")) {
121           // Unix/Linux OS
122           downloadDir = System.getProperty("user.home") + "/Downloads";
123       } else {
124           // Unknown OS
125           downloadDir = System.getProperty("user.home") + "/Downloads";
126       }
127       String fileName = "Sales Report " + startDate + " " + endDate + ".pdf";
128       String path = Paths.get(downloadDir,  ...more: "/", fileName).toString();
129       DecimalFormat df = new DecimalFormat( pattern: "#0.00");
130       // generate pdf receipt
131       try {
132           PdfWriter pdfWriter = new PdfWriter(path);
133           PdfDocument pdfDocument = new PdfDocument(pdfWriter);
134           pdfDocument.setDefaultPageSize(PageSize.A4);
135           Document document = new Document(pdfDocument);
136
137           document.add(new Paragraph( text: "CarePoint Clinic").setTextAlignment(TextAlignment.CENTER).setFontSize(20f).setBold());
138           document.add(new Paragraph( text: "Sales Report from " + startDate + " to " + endDate).setTextAlignment(TextAlignment.CENTER).setFontSize(20f)
139           // set split border
140           Border blackBorder = new SolidBorder(ColorConstants.BLACK,  width: 1f/4f);
141           Table divider = new Table(new float[]{190f*3});
142           divider.setBorder(blackBorder);
143           document.add(divider);
```

5.5 Searching Table Data

In our Java system, search features are heavily implemented across the patient, doctor, and admin pages. The doctor and admin search bars share the same logic, while the patient search bar operates with its own distinct logic. Below is the sample of the doctor and admin search structure.



We will be taking the doctor search section as our sample in this case. The code below, DoctorUpdateScheduleUI() method handles different types like JComboBox for gender and JDateChooser for dates using conditional checks. The search parameters gathered are then used to query a list of TimeSlotData objects using the timeSlots object. Finally, it displays a JOptionPane message with the number of results found and updates a table (updateQueryTable) to show the queried results.

```
public class DoctorUpdateScheduleUI extends JFrame{  wilson *
    public void setOperation() {  wilson *
        searchButton.addActionListener(new ActionListener() {  wilson *
            public void actionPerformed(ActionEvent e) {
                controlButton.addActionListener(new ActionListener() {  wilson *
                    @Override  wilson *
                    public void actionPerformed(ActionEvent e) {
                        selectedRow = -1;
                        String[] valueArray = new String[dataInputPanel.getComponents().length];
                        String value;
                        for (int i = 0; i < dataInputPanel.getComponents().length; i++) {
                            if (dataInputPanel.getComponent(i) instanceof JComboBox<?> genderComboBox) {
                                value = (String) genderComboBox.getSelectedItem();
                                valueArray[i] = (value.equals("- none -")) ? null : value;
                                continue;
                            }
                            else if (dataInputPanel.getComponent(i) instanceof JDateChooser dateChooser) {
                                SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd");
                                valueArray[i] = (dateChooser.getDate() == null)? null: formatter.format(dateChooser.getDate());
                                continue;
                            }
                            JTextField inputField = (JTextField) dataInputPanel.getComponents()[i];
                            value = inputField.getText();
                            valueArray[i] = (value.isBlank())? null: value;
                        }
                        List<TimeSlotData> queriedTimeSlotlist = timeSlots.queryTimeSlot(valueArray[0], valueArray[1], valueArray[2],valueArray[3],
                                Cookie.getDoctorCookie().getId(),  queryDoctorName: null,  queryDoctorSpecialization: null, valueArray[4]);
                        JOptionPane.showMessageDialog(frame, String.format("%d result found", queriedTimeSlotlist.size()));
                        updateQueryTable(queriedTimeSlotlist);
                    }
                });
            }
        });
```

From the other view, the search structure for the patient contains two types of search methods, the multiple selects choice and a search bar. Below is the picture shows the search bar and the multiple select choice panel for patients (Rao, 2015).

ID	Date	Time	Meridiem	Doctor ID	Doctor Name	Specialization	Status
ts5	2024-07-10	09:30	p.m.	dc1	How	general	available
ts6	2024-07-11	11:30	p.m.	dc2	SengKhuan	no	available
ts7	2024-07-01	09:30	p.m.	dc1	How	general	available
ts11	2024-07-13	02:00	p.m.	dc1	How	general	available
ts12	2024-07-13	02:30	p.m.	dc1	How	general	available
ts13	2024-07-13	03:30	p.m.	dc1	How	general	available
ts14	2024-07-13	04:30	p.m.	dc1	How	general	available
ts15	2024-07-08	10:30	a.m.	dc1	How	general	available
ts16	2024-07-08	10:00	a.m.	dc1	How	general	available

The combo box is to search for the doctor's name appointments while the search bar is for filtering results by specialization. The code below, setAllTimeSlotTable() method iterates through the filtered time slot data, checking for matches based on doctor name and specialization, while

ensuring the time slot date is after a defined "currentDate". Matching rows are added to "matchingRows", and if no matches are found ("matchFound" is false), all time slot data entries are added to "matchingRows".

```
291     public void setAllTimeSlotTable(String doctorName, String specialize, JTable table) {  6 usages  wilson *
292         List<String> timeSlotHeaders = timeSlots.getTimeSlotHeaders().stream().toList();
293
294         Object[] header = timeSlotHeaders.toArray();
295
296         List<TimeSlotData> timeSlotList = timeSlots.queryTimeSlot( queryId: null,  queryDate: null,  queryTime: null,
297                 queryMeridiem: null,  queryDoctorId: null,  queryDoctorName: null,  queryDoctorSpecialization: null,  queryStatus: "available");
298         List<Object[]> matchingRows = new ArrayList<>();
299         boolean matchFound = false;
300
301         for (TimeSlotData timeSlotData : timeSlotList) {
302             if (timeSlotData.getDoctorName().equals(doctorName) || timeSlotData.getDoctorSpecialization().equals(specialize)) {
303                 LocalDate timeSlotDate = LocalDate.parse(timeSlotData.getDate());
304                 if(timeSlotDate.isAfter(currentDate)) {
305                     matchingRows.add(new Object[]{
306                             timeSlotData.getId(),
307                             timeSlotData.getDate(),
308                             timeSlotData.getTime(),
309                             timeSlotData.getMeridiem(),
310                             timeSlotData.getDoctorId(),
311                             timeSlotData.getDoctorName(),
312                             timeSlotData.getDoctorSpecialization(),
313                             timeSlotData.getStatus()
314                     });
315                     matchFound = true;
```

Finally, a DefaultTableModel is created using "matchingRows"and header, and this model is set to the provided JTable (table) to display the filtered time slot data.

```
320         if (!matchFound) {
321             for (TimeSlotData timeSlotData : timeSlotList) {
322                 matchingRows.add(new Object[]{
323                         timeSlotData.getId(),
324                         timeSlotData.getDate(),
325                         timeSlotData.getTime(),
326                         timeSlotData.getMeridiem(),
327                         timeSlotData.getDoctorId(),
328                         timeSlotData.getDoctorName(),
329                         timeSlotData.getDoctorSpecialization(),
330                         timeSlotData.getStatus()
331                 });
332             }
333         }
334
335         TableModel tableModel = new DefaultTableModel(matchingRows.toArray(new Object[0][0]), header) {
336             @Override  wilson
337             public boolean isCellEditable(int row, int column) { return false; }
340         };
341
342         table.setModel(tableModel);
```

6.0 Conclusion

In a nutshell, our team created a simple robust Clinic Management System by utilizing object-oriented programming concepts and Java's GUI user interface features. This system was developed to address requirements of different roles for users in the clinic management system. User Roles such as administrators, doctors, and patients. This is done by including features such user registration, appointment administration, medical record data, and payment data. Through ensuring that all data is kept in text files, our teams have ensured data stability and integrity.

Limitations

- **Inventory Management:** Our existing Clinic Management System does not allow inventory management system due to the complexities of the coding required. Implementing an inventory management system involves a wide range of functionality and features to make sure the program run properly. Furthermore, merging inventory management with the user payment function is another challenge of complexity in the code.
- **Appointment Booking:** Our system currently does not limit the number of appointments that a customer may book in a day. This may lead to a situation where the customer may intentionally set many appointments, preventing other customers from making bookings. This limitation raises ethical problems in our program system, since it may result in unethical use of the system. To ensure that all customers have equal access to booking appointment times, we should limit customers to only booking a maximum of three appointments per day.

In future, we will integrate an inventory management system into our clinic management system to provide more service and efficiently coordinate the inflow and outflow of inventory. This improvement can help the administrators to understand the balance amount of each medicine. Besides that, we will add a limitation booking features inside the system. This feature ensures

that each patient can only make a maximum of 5 booking slots for a specific day. Consequently, the system can prevent unethical use, for example, a patient reserves all time slots for a day.

7.0 References

Introduction To Java Swing. (2024). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/introduction-to-java-swing/

Kyle Herrity. (2023, February 4). *What Is Object-Oriented Programming (OOP)? A Complete Guid*. Retrieved from indeed: https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming

NeelumAyub. (2019). *Java SWING JFrame Layouts Example*. Retrieved from Java Tutorial Network: https://javatutorial.net/java-swing-jframe-layouts/

Rao, S. N. (2015). *JList Multiple Selection Example Java Swing*. Retrieved from Way2Java: https://way2java.com/swing/jlist-with-multiple-selection/

Thompson, A. (2012). *Border with rounded corners and transparency*. Retrieved from StackOverflow: https://stackoverflow.com/questions/15025092/border-with-rounded-corners-transparency

Thorben. (2024, February 28). *OOP Concept for Beginners: What is Encapsulation*. Retrieved from Stackify: https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/

Wu, J. (2021). *Java Swing Layouts Example*. Retrieved from Java Code Geeks: https://examples.javacodegeeks.com/java-swing-layouts-example/

8.0 Workload Matrix

Task	LEW WAI HOW	TAN PO YEH	TONG JIA CHUEN
Documentation			
Introduction (2%)	0%	0%	100%
Explanation Sample Output (15%)	33%	33%	33%
Object Oriented Concept (10%)	100%	0%	0%
Java Features (10%)	0%	0%	100%
Explanation of additional features (10%)	0%	100%	0%
Conclusion (2%)	0%	0%	100%
Reference (1%)	100%	0%	0%
Implementation			
User Authentication (5%)	10%	90%	0%
Admin functions (15%)	100%	0%	0%
Doctor functions (15%)	20%	0%	80%
Patient functions (15%)	0%	100%	0%
Total percentage	34.5	34.5	31