Criterion C

Programming techniques implemented into the programme:

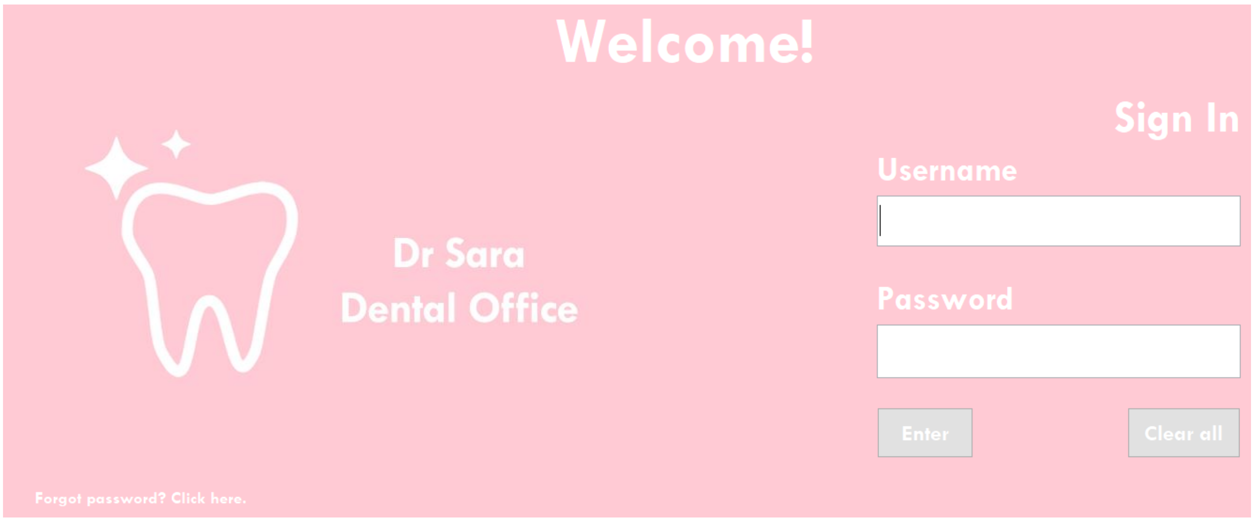
- Java (Object Oriented Programming)
    - Functions
    - Classes
    - Useful Libraries (jCalendar, rs2xml (ResultSet Management))
    - Threading
- MySQL Database (MySQL Workbench)
- GUI (Graphic User Interface)

This programme is a Database Management System implemented in a GUI environment.

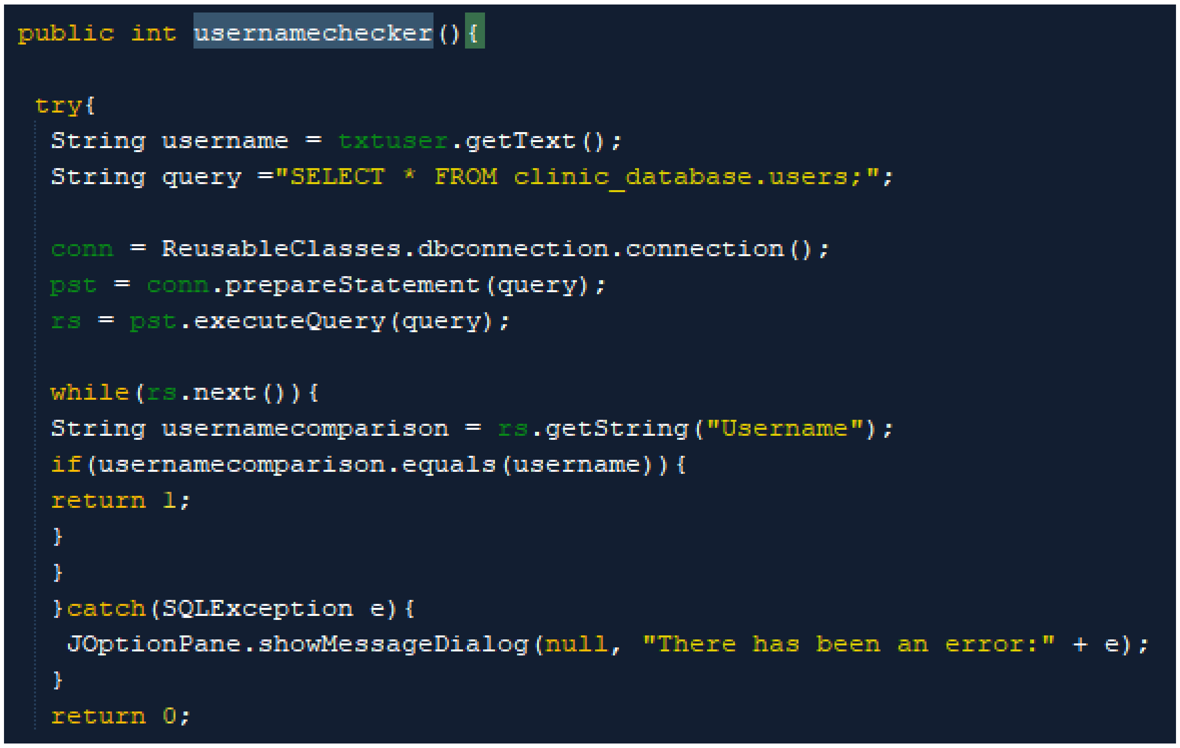
Functions are extensively used for Database manipulation.

1. ***Complex functions:***

(Figure 1.) Sign_In class GUI

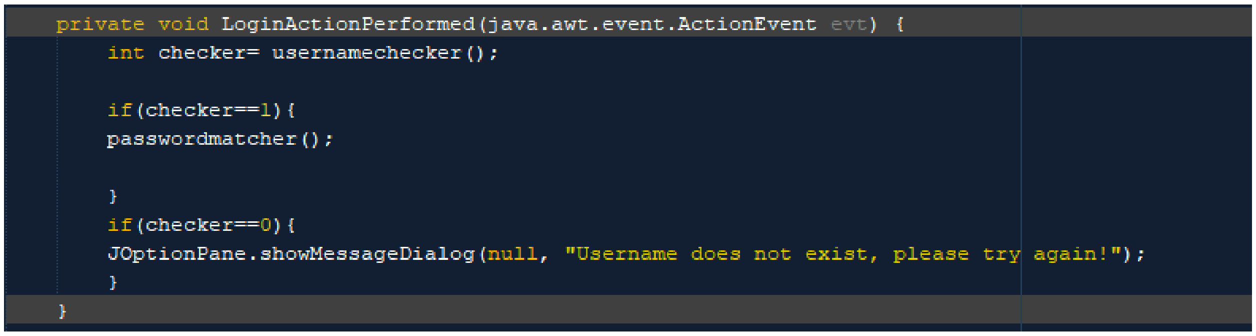


This is the starting point of the application. There is a Username and Password text field used to password protect the programme.



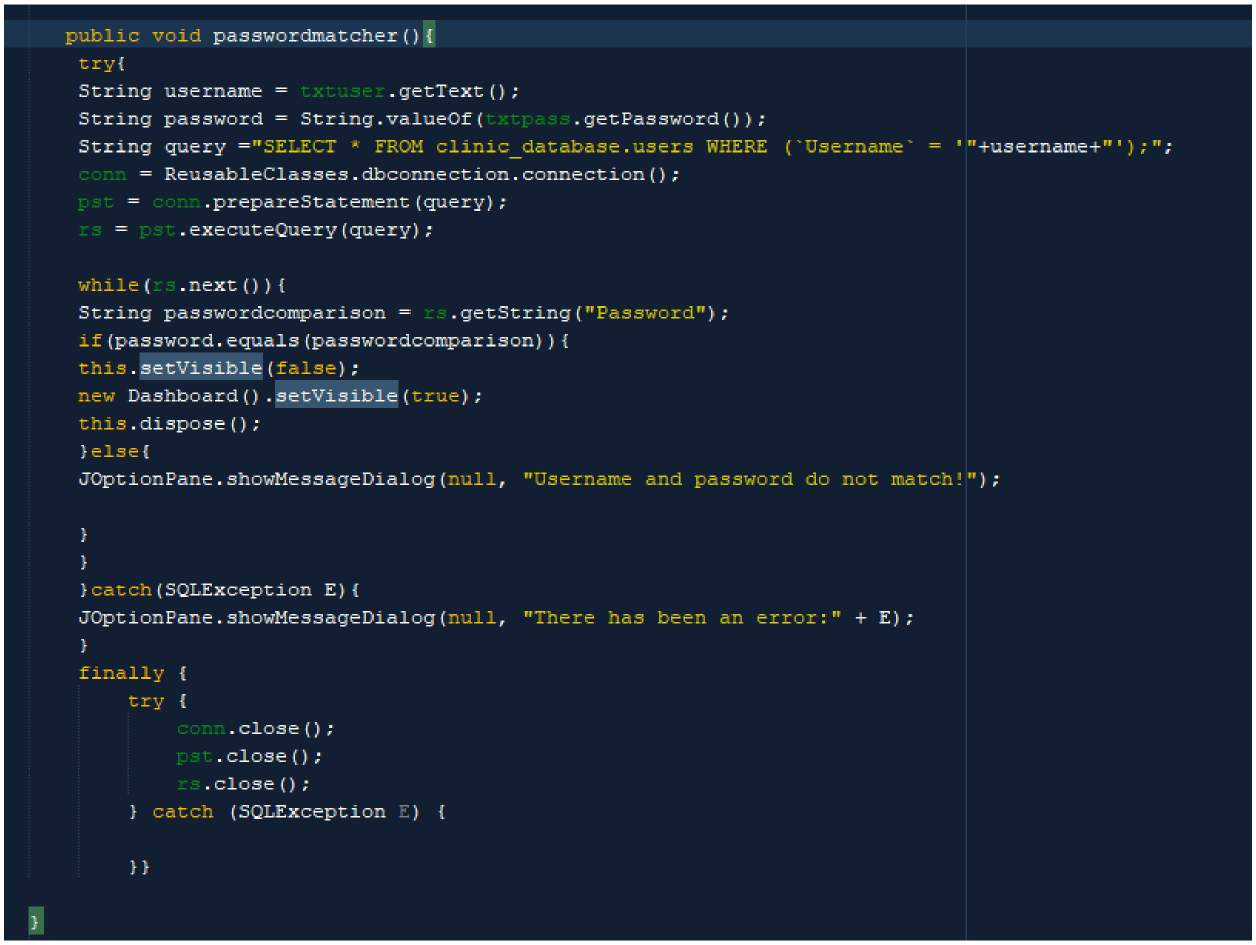
```
public int usernamechecker(){

 try{
  String username = txtuser.getText();
  String query ="SELECT * FROM clinic_database.users;";

  conn = ReusableClasses.dbconnection.connection();
  pst = conn.prepareStatement(query);
  rs = pst.executeQuery(query);

  while(rs.next()){
  String usernamecomparison = rs.getString("Username");
  if(usernamecomparison.equals(username)){
  return 1;
  }
  }
  }catch(SQLException e){
   JOptionPane.showMessageDialog(null, "There has been an error:" + e);
  }
  return 0;
```

The usernamechecker() function is used to check whether the user-entered username matches the one available in the database, and if found, returns 1.



```
private void LoginActionPerformed(java.awt.event.ActionEvent evt) {
    int checker= usernamechecker();

    if(checker==1){
    passwordmatcher();

    }
    if(checker==0){
    JOptionPane.showMessageDialog(null, "Username does not exist, please try again!");
    }
}
```

Once the user presses Enter jButton, the code checks the return value of the function usernamechecker(). If it returns a 1, the code then invokes the passwordmatcher() function. Else, it displays the message above.

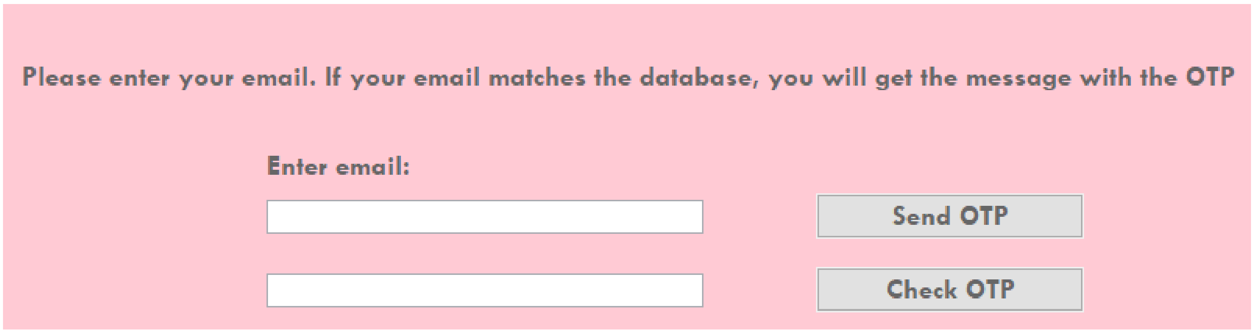


```
    public void passwordmatcher(){
     try{
     String username = txtuser.getText();
     String password = String.valueOf(txtpass.getPassword());
     String query ="SELECT * FROM clinic_database.users WHERE (`Username` = '"+username+"');";
     conn = ReusableClasses.dbconnection.connection();
     pst = conn.prepareStatement(query);
     rs = pst.executeQuery(query);

     while(rs.next()){
     String passwordcomparison = rs.getString("Password");
     if(password.equals(passwordcomparison)){
     this.setVisible(false);
     new Dashboard().setVisible(true);
     this.dispose();
     }else{
     JOptionPane.showMessageDialog(null, "Username and password do not match!");

     }
     }
     }catch(SQLException E){
     JOptionPane.showMessageDialog(null, "There has been an error:" + E);
     }
     finally {
         try {
             conn.close();
             pst.close();
             rs.close();
         } catch (SQLException E) {

         }}

  }
```

Here we see that the login is finalized if the ResultSet String Password matches the Username from the Database.

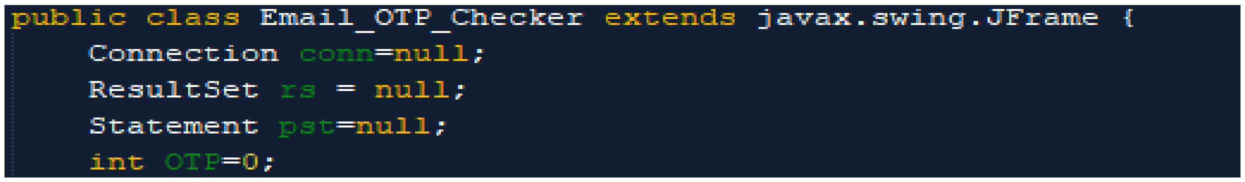
Furthermore, in order to further extend the versatility of the project, one of the functionalities that was implemented was an OTP password reset system.

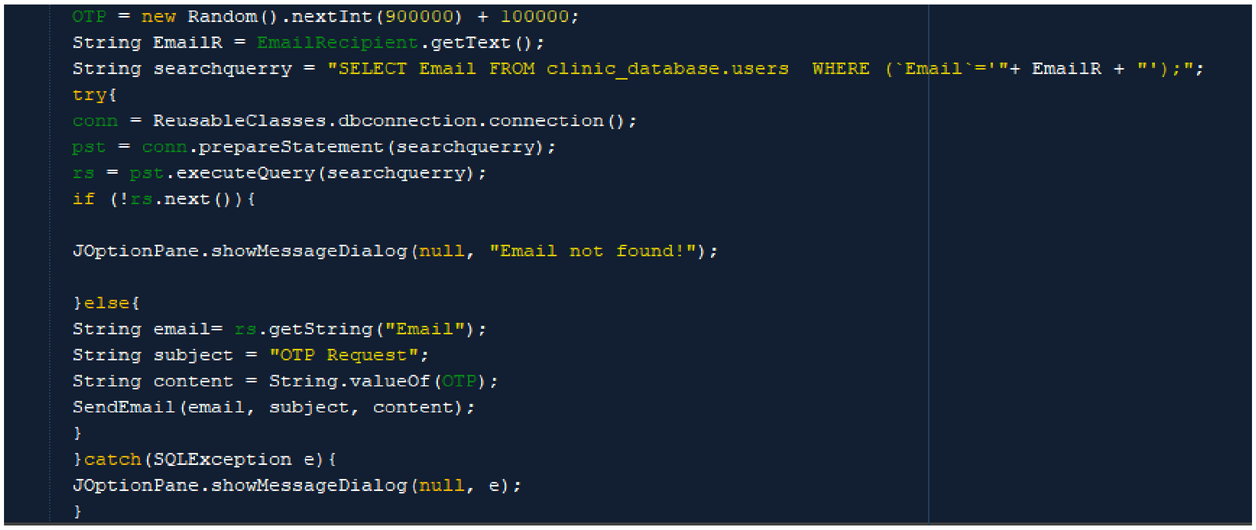
(Figure 2.) Email_OTP_Checker GUI


Please enter your email. If your email matches the database, you will get the message with the OTP

Enter email:

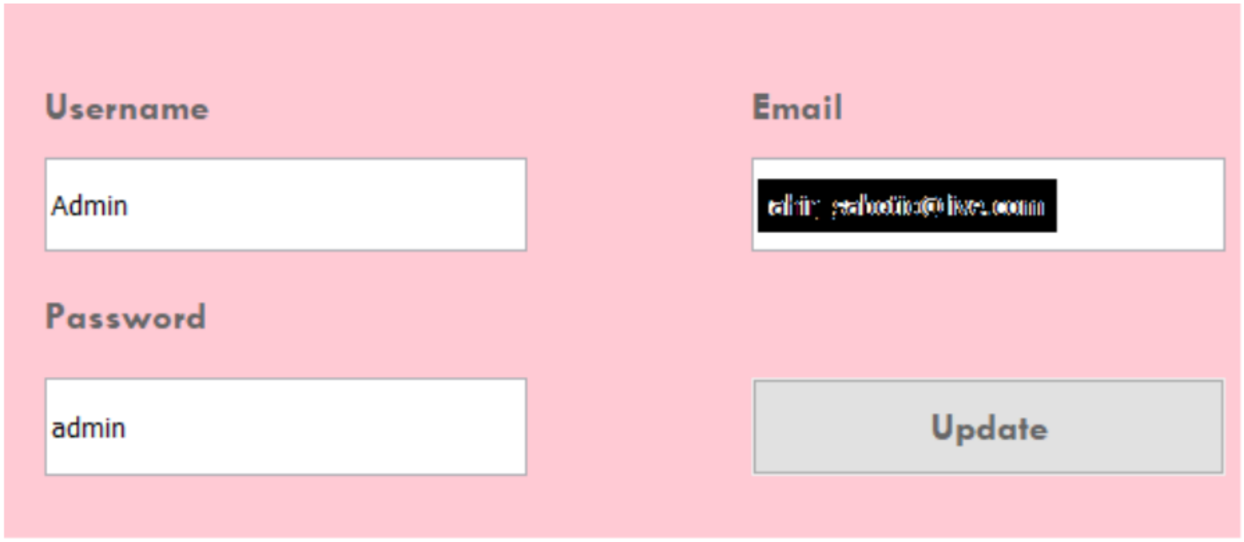
Firstly, a global OTP variable is declared.


```
public class Email_OTP_Checker extends javax.swing.JFrame {
    Connection conn=null;
    ResultSet rs = null;
    Statement pst=null;
    int OTP=0;
```

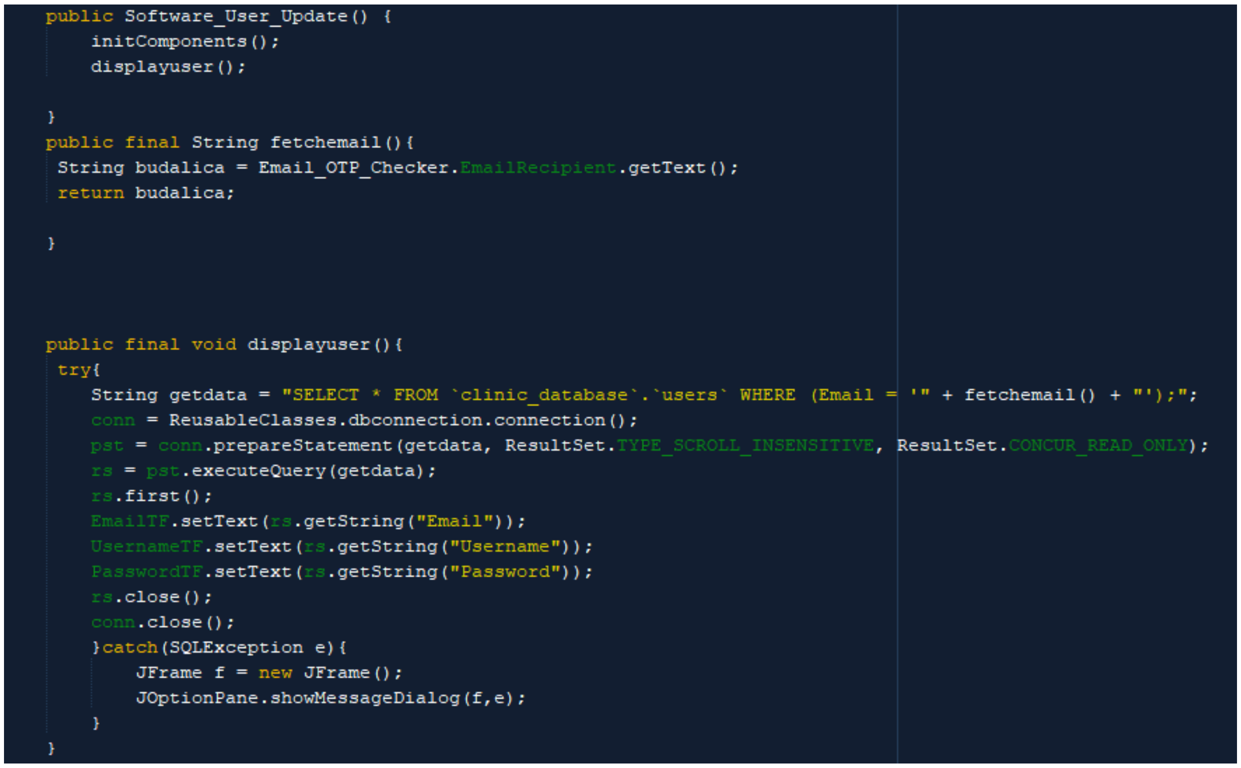

```
OTP = new Random().nextInt(900000) + 100000;
String EmailR = EmailRecipient.getText();
String searchquerry = "SELECT Email FROM clinic_database.users  WHERE (`Email`='"+ EmailR + "');";
try{
conn = ReusableClasses.dbconnection.connection();
pst = conn.prepareStatement(searchquerry);
rs = pst.executeQuery(searchquerry);
if (!rs.next()){

JOptionPane.showMessageDialog(null, "Email not found!");

}else{
String email= rs.getString("Email");
String subject = "OTP Request";
String content = String.valueOf(OTP);
SendEmail(email, subject, content);
}
}catch(SQLException e){
JOptionPane.showMessageDialog(null, e);
}
```

Then, once jButton is pressed, a random 6-digit number is created and assigned to the OTP variable. The user then enters the email address of the account. In order to validate the email address, a connection to the clinic_database.users is established. In the case that the email address is not found, i.e. the Result Set is empty, the programme outputs “Email not found”. Else, in the case of the matching email address, SendEmail() function is invoked, prompting the execution of the function where the content of the email is the OTP.

(Figure 3.) Software_User_Update



Username	Email
Admin	[redacted]
Password	
admin	

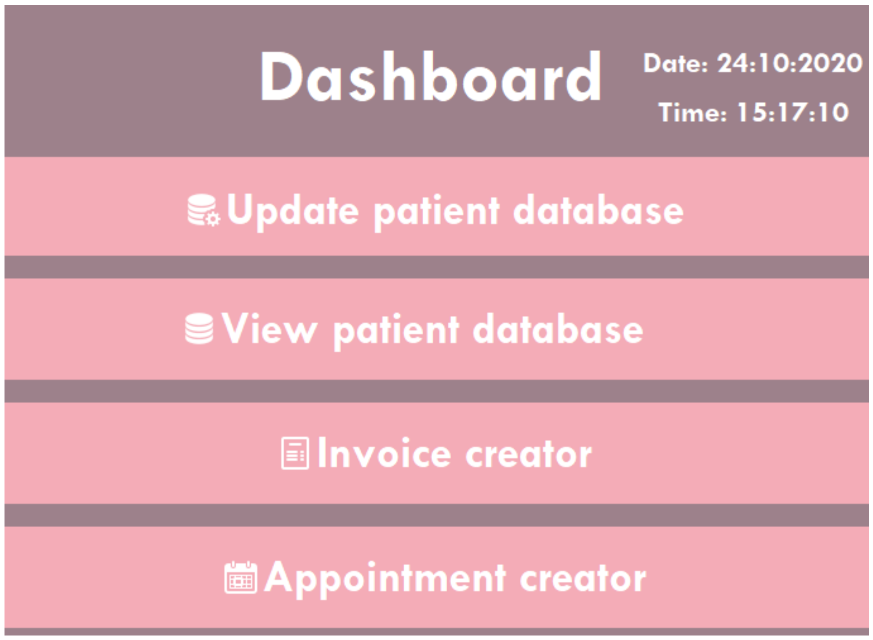


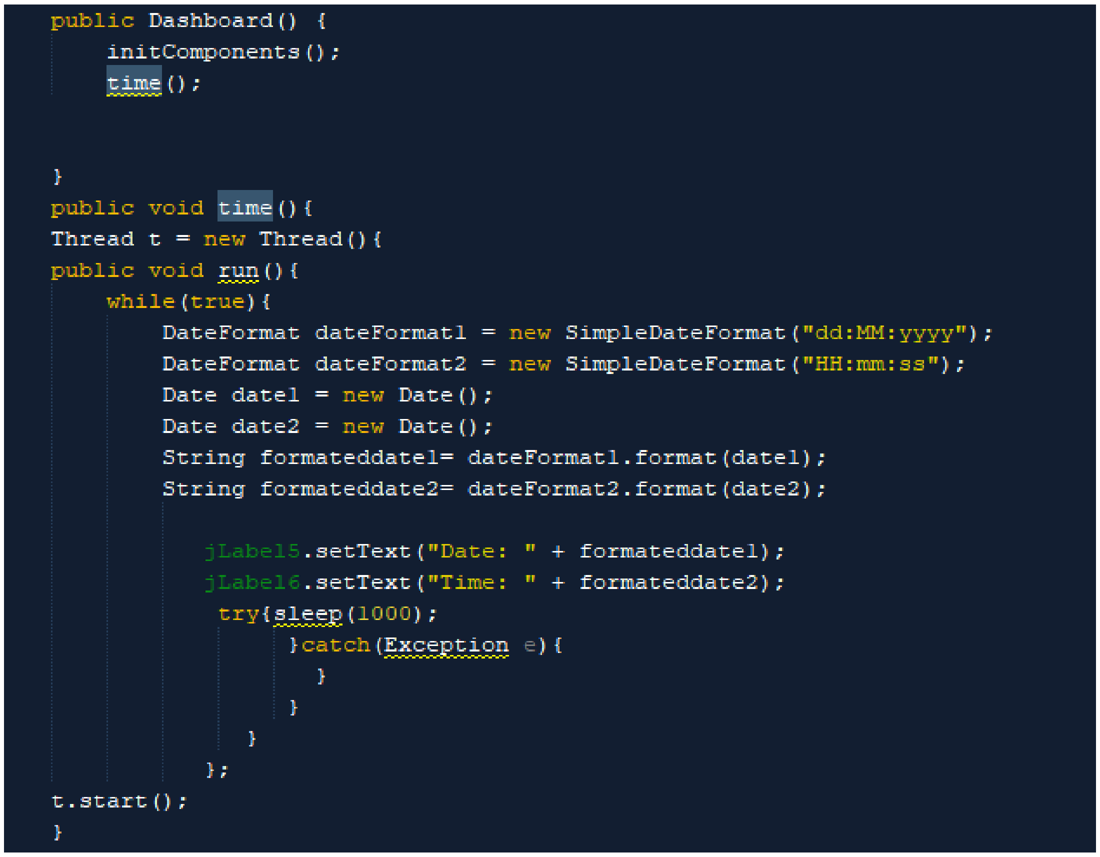
```
public Software_User_Update() {
    initComponents();
    displayuser();

}
public final String fetchemail(){
 String budalica = Email_OTP_Checker.EmailRecipient.getText();
 return budalica;

}



public final void displayuser(){
 try{
    String getdata = "SELECT * FROM `clinic_database`.`users` WHERE (Email = '" + fetchemail() + "');";
    conn = ReusableClasses.dbconnection.connection();
    pst = conn.prepareStatement(getdata, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    rs = pst.executeQuery(getdata);
    rs.first();
    EmailTF.setText(rs.getString("Email"));
    UsernameTF.setText(rs.getString("Username"));
    PasswordTF.setText(rs.getString("Password"));
    rs.close();
    conn.close();
    }catch(SQLException e){
        JFrame f = new JFrame();
        JOptionPane.showMessageDialog(f,e);
    }
}
```

Now that we have validated the OTP, the function fetchemail() fetches the validated email address from the Email_OTP_Checker. This is used in order to access the clinic_database.users and display the user data (Username, Password, Email), which the user can modify (functions showing the way the databases are modified are shown later in the document). displayuser() function is called in the constructor so that the user can automatically see all the contents of the clinic_database.user database (since the database contains only one row, i.e. the admin).

Another technique that was used is Java Threading. Threads allow the programme to execute different lines of code concurrently. In this example, I have created a clock that works in real time.

(Figure 4.) Dashboard class GUI

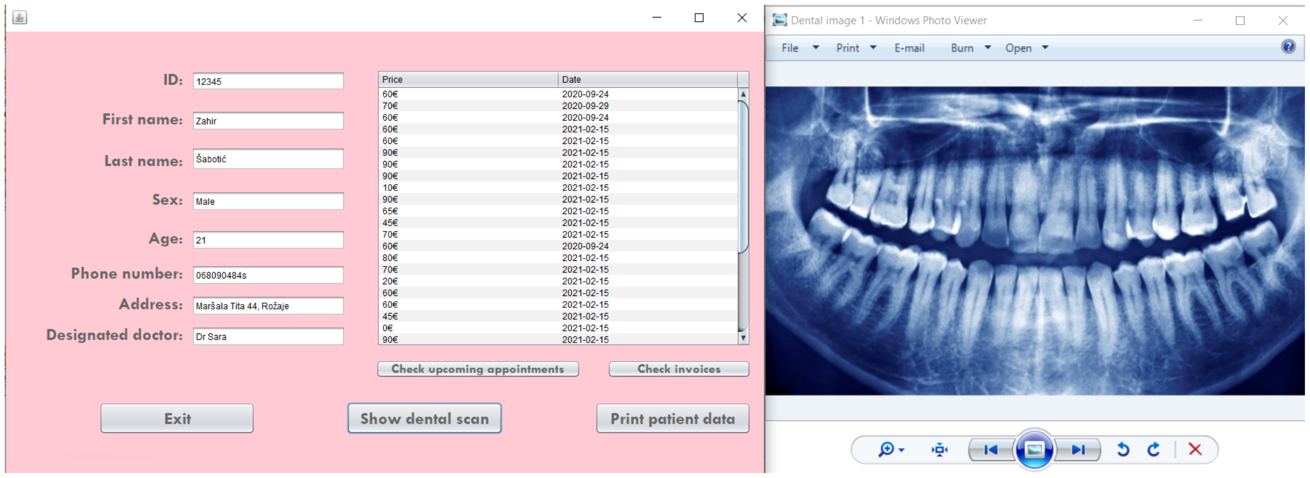




```java
public Dashboard() {
    initComponents();
    time();


}
public void time(){
Thread t = new Thread(){
public void run(){
    while(true){
        DateFormat dateFormat1 = new SimpleDateFormat("dd:MM:yyyy");
        DateFormat dateFormat2 = new SimpleDateFormat("HH:mm:ss");
        Date date1 = new Date();
        Date date2 = new Date();
        String formateddate1= dateFormat1.format(date1);
        String formateddate2= dateFormat2.format(date2);

           jLabel5.setText("Date: " + formateddate1);
           jLabel6.setText("Time: " + formateddate2);
            try{sleep(1000);
                 }catch(Exception e){
                   }
                 }
              }
           };
t.start();
}
```

We can see that the method time() is called in the constructor of the Dashboard Class, running concurrently to the other code.

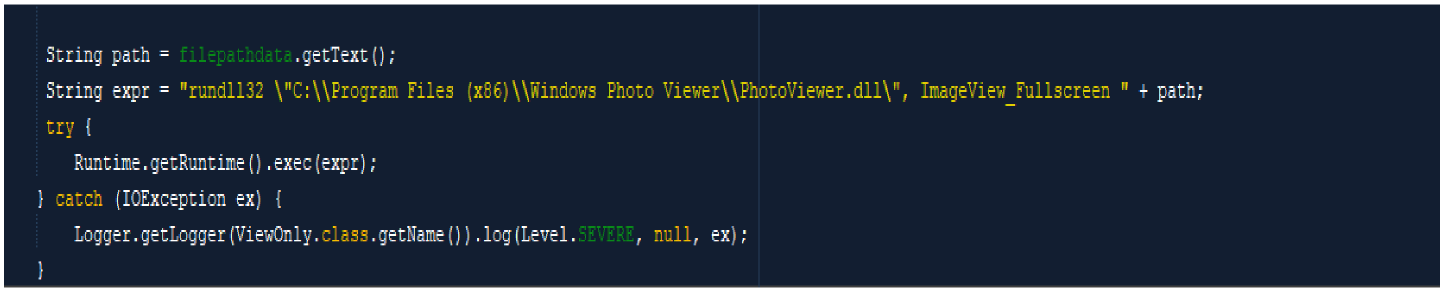
(Figure 5.) Patient_Data class GUI and sample of the dental scan



ID:	12345
First name:	Zahir
Last name:	Šabotić
Sex:	Male
Age:	21
Phone number:	068090484s
Address:	Maršala Tita 44, Rožaje
Designated doctor:	Dr Sara

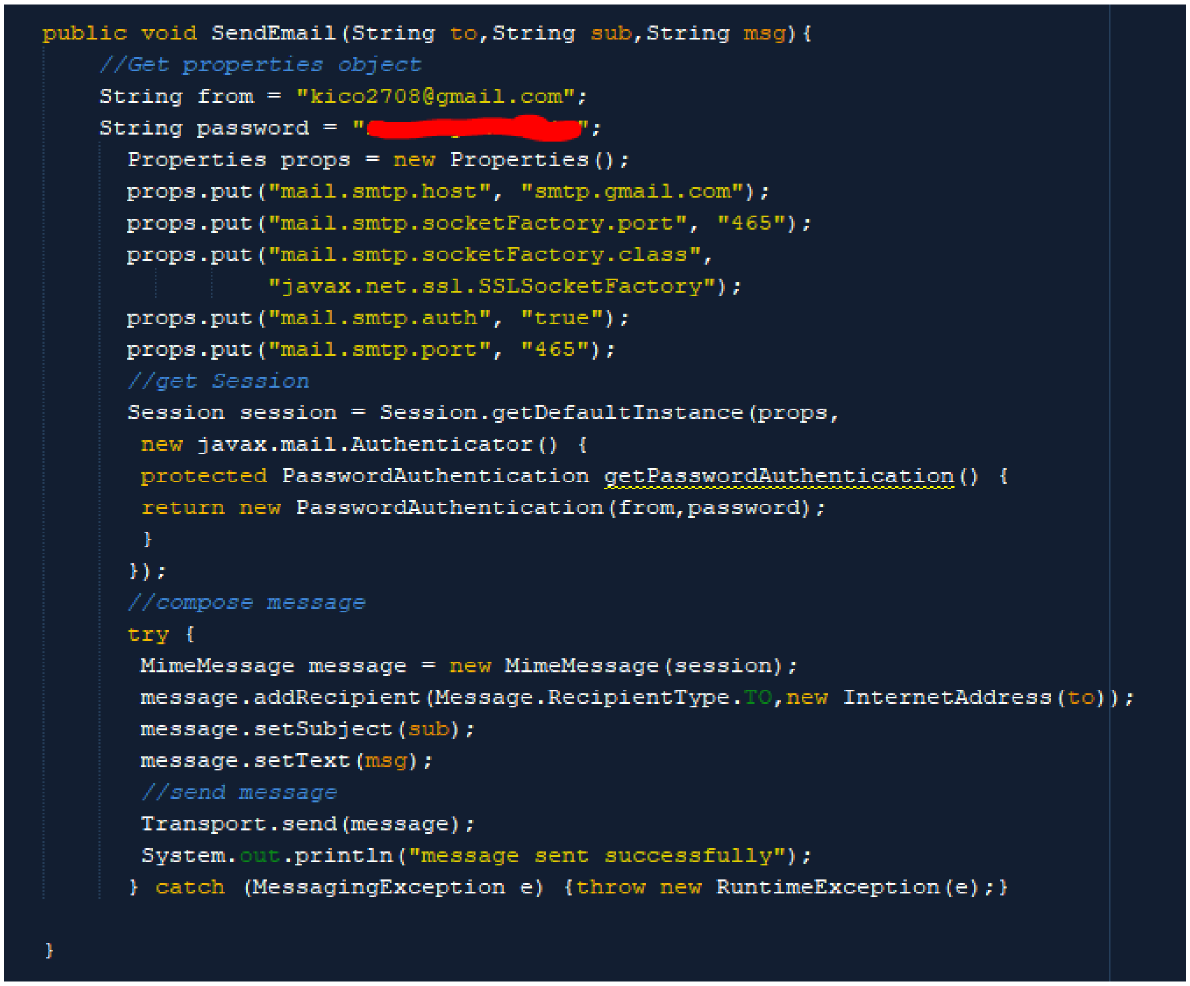
Price	Date
60€	2020-09-24
70€	2020-09-29
60€	2020-09-24
60€	2021-02-15
60€	2021-02-15
90€	2021-02-15
90€	2021-02-15
90€	2021-02-15
10€	2021-02-15
90€	2021-02-15
65€	2021-02-15
45€	2021-02-15
70€	2021-02-15
60€	2020-09-24
80€	2021-02-15
70€	2021-02-15
20€	2021-02-15
60€	2021-02-15
60€	2021-02-15
45€	2021-02-15
0€	2021-02-15
90€	2021-02-15

In order to store images in the MySQL database, rather than directly storing the picture within the database (it was discovered that it might slow down the efficiency of the database), it was decided to use the JFileChooser object to create a reference to the file as a path in the database. The path is stored as Varchar value in the database.



```
String path = filepathdata.getText();
String expr = "rundll32 \"C:\\Program Files (x86)\\Windows Photo Viewer\\PhotoViewer.dll\", ImageView_Fullscreen " + path;
try {
    Runtime.getRuntime().exec(expr);
} catch (IOException ex) {
    Logger.getLogger(ViewOnly.class.getName()).log(Level.SEVERE, null, ex);
}
```

Filepathdata jTextfield was previously filled with file path extracted from the database. Here the function runs the file path and opens the image in Windows Photo viewer.

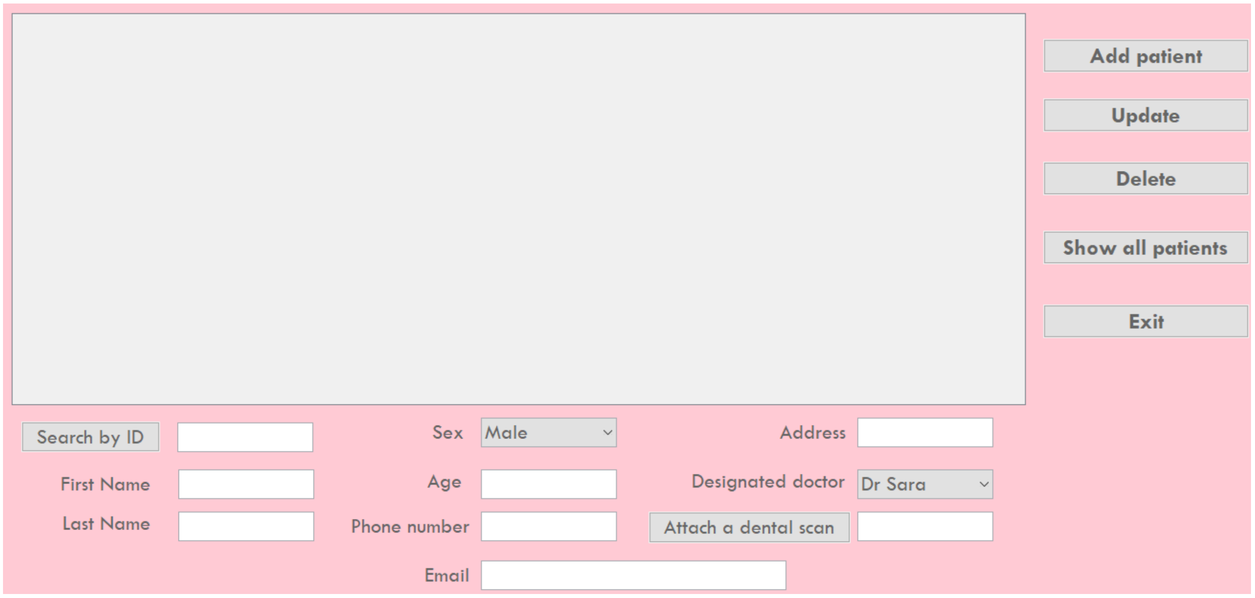


```
public void SendEmail(String to,String sub,String msg){
    //Get properties object
    String from = "kico2708@gmail.com";
    String password = "[redacted]";
      Properties props = new Properties();
      props.put("mail.smtp.host", "smtp.gmail.com");
      props.put("mail.smtp.socketFactory.port", "465");
      props.put("mail.smtp.socketFactory.class",
                "javax.net.ssl.SSLSocketFactory");
      props.put("mail.smtp.auth", "true");
      props.put("mail.smtp.port", "465");
      //get Session
      Session session = Session.getDefaultInstance(props,
       new javax.mail.Authenticator() {
       protected PasswordAuthentication getPasswordAuthentication() {
       return new PasswordAuthentication(from,password);
       }
      });
      //compose message
      try {
       MimeMessage message = new MimeMessage(session);
       message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
       message.setSubject(sub);
       message.setText(msg);
       //send message
       Transport.send(message);
       System.out.println("message sent successfully");
      } catch (MessagingException e) {throw new RuntimeException(e);}

}
```

Function SendEmail() takes advantage of the Java Email API in order to allow the user to directly send emails from the application. This function has been implemented across the programme, from the OTP system to sending users reminders of the scheduled appointment.

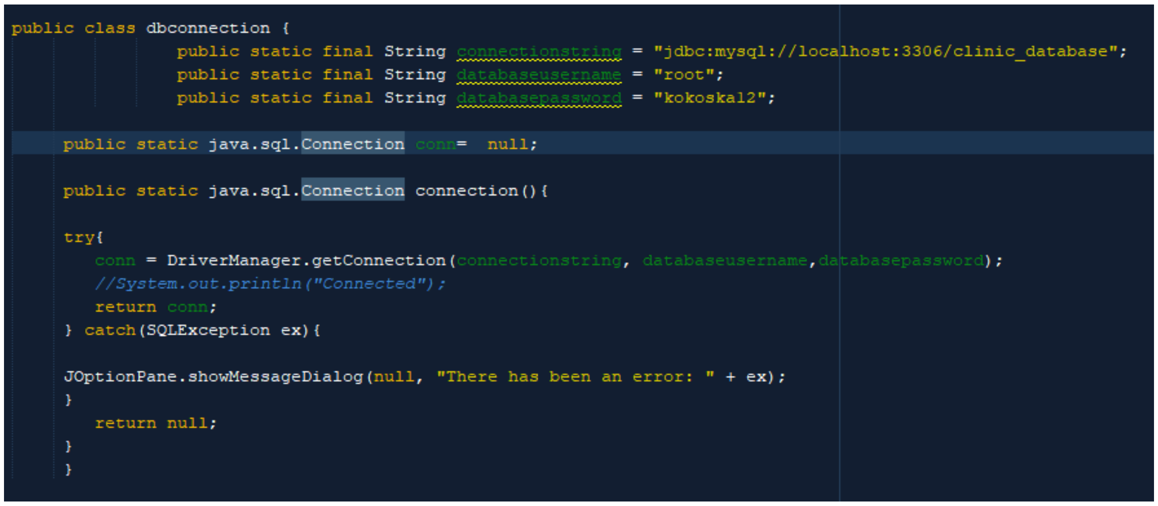
2. **SQL Queries:**

(Figure 5.) GUI of the Modify_Patient_Database class



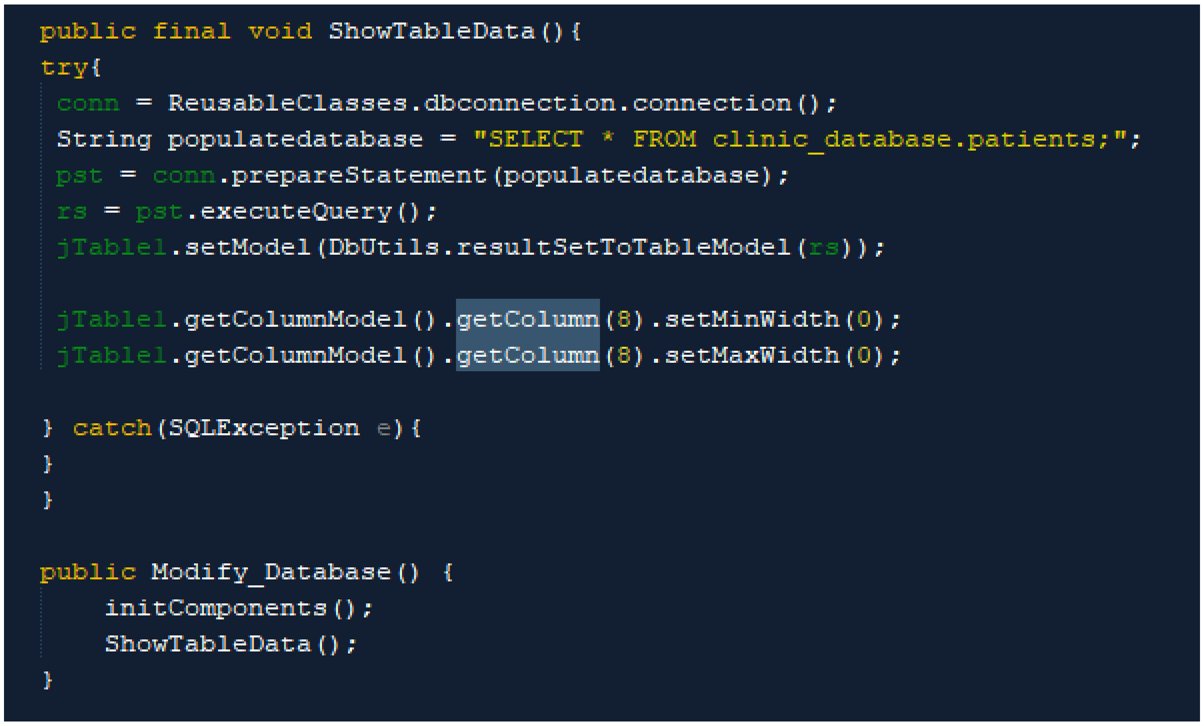
The Modify_Patient_Database class encompasess all the SQL queries implemented in the programme. Other classes also use some of the below mentioned functions.

SQL manipulation has been through creating a dbconnection() Class. This class reduces the amount of code connection() method can be called in any other needed class.

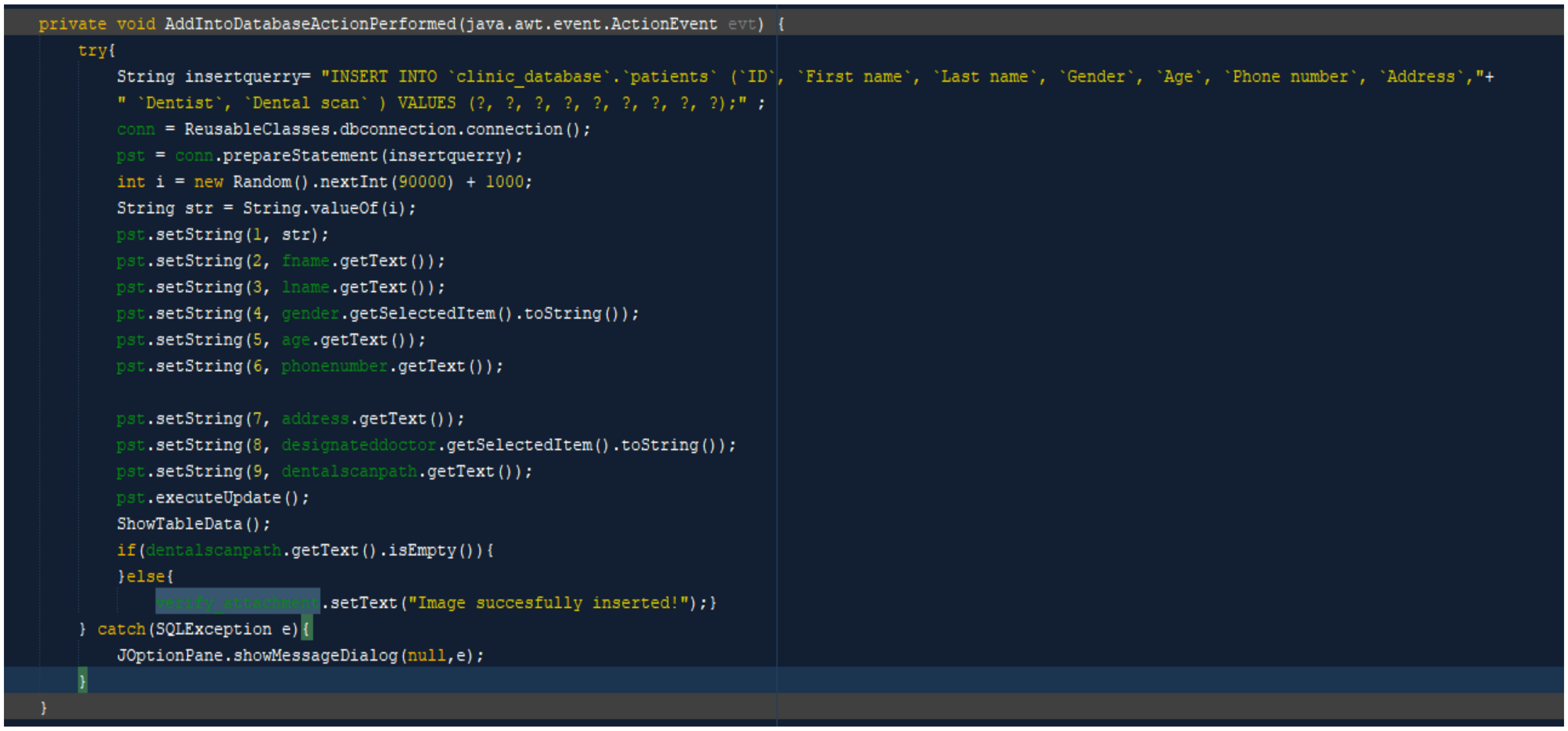


```
public class dbconnection {
                public static final String connectionstring = "jdbc:mysql://localhost:3306/clinic_database";
                public static final String databaseusername = "root";
                public static final String databasepassword = "kokoska12";

    public static java.sql.Connection conn=  null;

    public static java.sql.Connection connection(){

    try{
       conn = DriverManager.getConnection(connectionstring, databaseusername,databasepassword);
       //System.out.println("Connected");
       return conn;
    } catch(SQLException ex){

    JOptionPane.showMessageDialog(null, "There has been an error: " + ex);
    }
       return null;
    }
    }
```

In the following picture we see the dbconnection called in order to display the contents of the database.



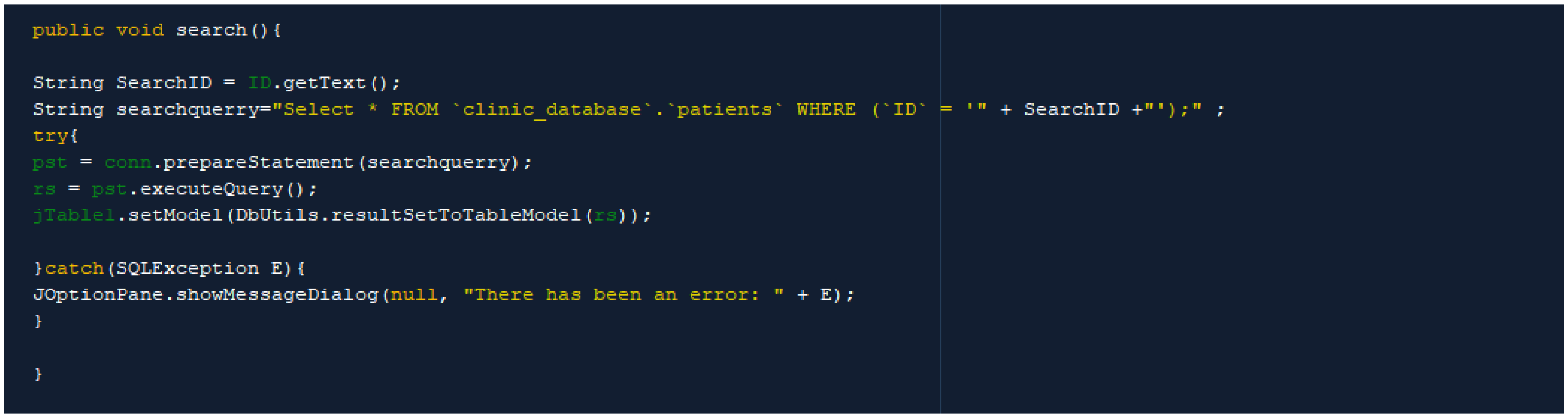
```
public final void ShowTableData(){
try{
 conn = ReusableClasses.dbconnection.connection();
 String populatedatabase = "SELECT * FROM clinic_database.patients;";
 pst = conn.prepareStatement(populatedatabase);
 rs = pst.executeQuery();
 jTable1.setModel(DbUtils.resultSetToTableModel(rs));

 jTable1.getColumnModel().getColumn(8).setMinWidth(0);
 jTable1.getColumnModel().getColumn(8).setMaxWidth(0);

} catch(SQLException e){
}
}

public Modify_Database() {
    initComponents();
    ShowTableData();
}
```



```java
private void AddIntoDatabaseActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String insertquerry= "INSERT INTO `clinic_database`.`patients` (`ID`, `First name`, `Last name`, `Gender`, `Age`, `Phone number`, `Address`,"+
        " `Dentist`, `Dental scan` ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);" ;
        conn = ReusableClasses.dbconnection.connection();
        pst = conn.prepareStatement(insertquerry);
        int i = new Random().nextInt(90000) + 1000;
        String str = String.valueOf(i);
        pst.setString(1, str);
        pst.setString(2, fname.getText());
        pst.setString(3, lname.getText());
        pst.setString(4, gender.getSelectedItem().toString());
        pst.setString(5, age.getText());
        pst.setString(6, phonenumber.getText());

        pst.setString(7, address.getText());
        pst.setString(8, designateddoctor.getSelectedItem().toString());
        pst.setString(9, dentalscanpath.getText());
        pst.executeUpdate();
        ShowTableData();
        if(dentalscanpath.getText().isEmpty()){
        }else{
            verify_attachment.setText("Image succesfully inserted!");}
    } catch(SQLException e){
        JOptionPane.showMessageDialog(null,e);
    }
}
```

ShowTableData() method has been used extensively in the programme. It uses the aforementioned RS library to automatically fill the jTable from the Result Set.

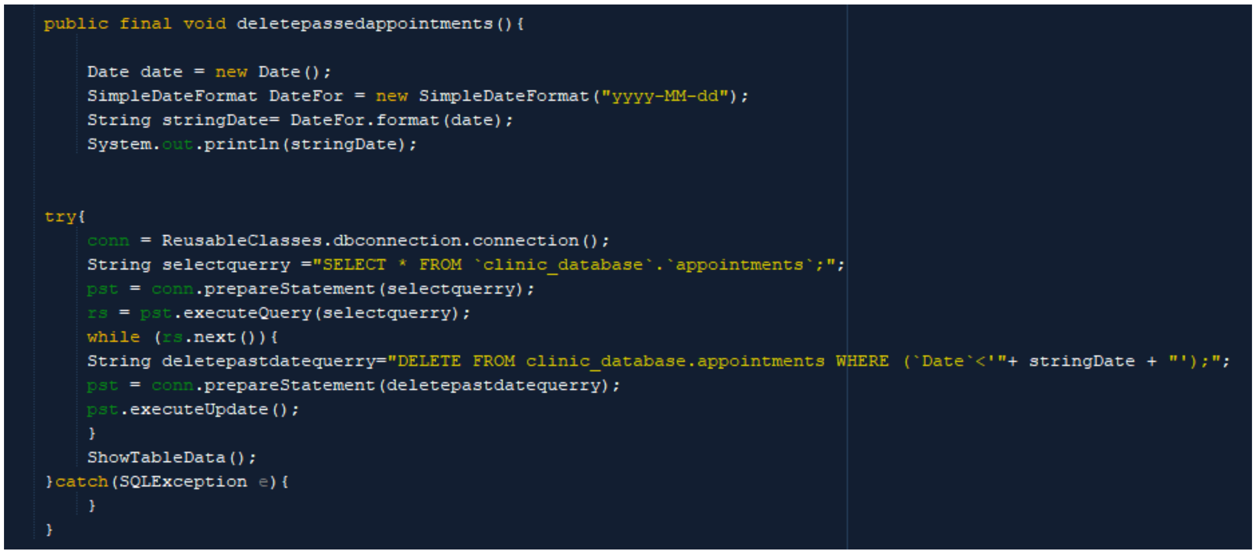
As we see, the conn variable (which has been created beforehand in the Class in question) is used to establish a connection with the database. Prepared statements are used to insert data into the database to ensure safety and prevent SQL Injections.



```java
public void search(){

String SearchID = ID.getText();
String searchquerry="Select * FROM `clinic_database`.`patients` WHERE (`ID` = '" + SearchID +"');" ;
try{
pst = conn.prepareStatement(searchquerry);
rs = pst.executeQuery();
jTable1.setModel(DbUtils.resultSetToTableModel(rs));

}catch(SQLException E){
JOptionPane.showMessageDialog(null, "There has been an error: " + E);
}

}
```

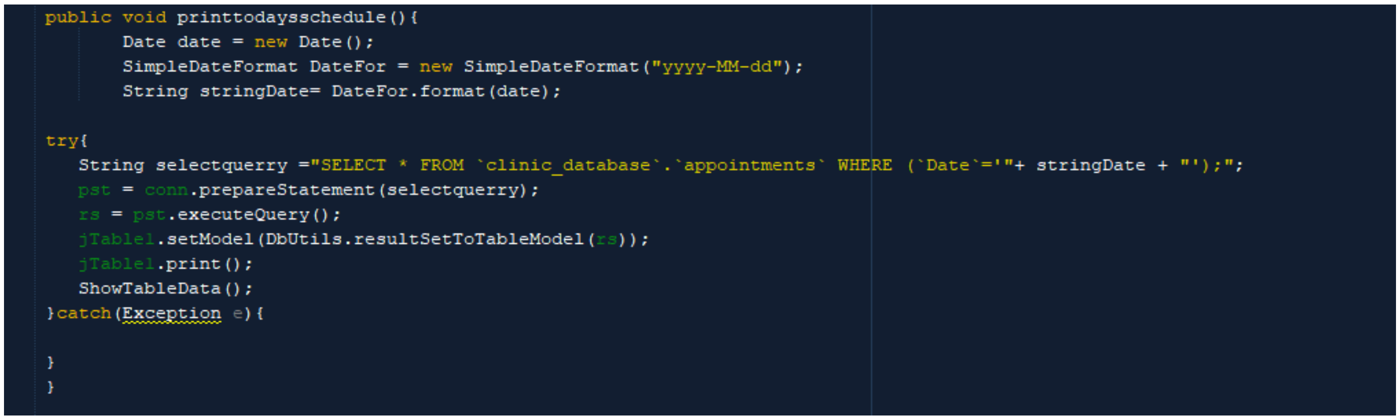
Here we see a simple search function used across the program for searched rows (ID was used as a primary key to identify each row). All these functions have been implemented across different database tables, and below we see the GUI of the primary table, Patients, and different functionalities through the use of jButton.

All the above functions have been implemented across the app, all aiding the user in manipulating the data entries of the database.



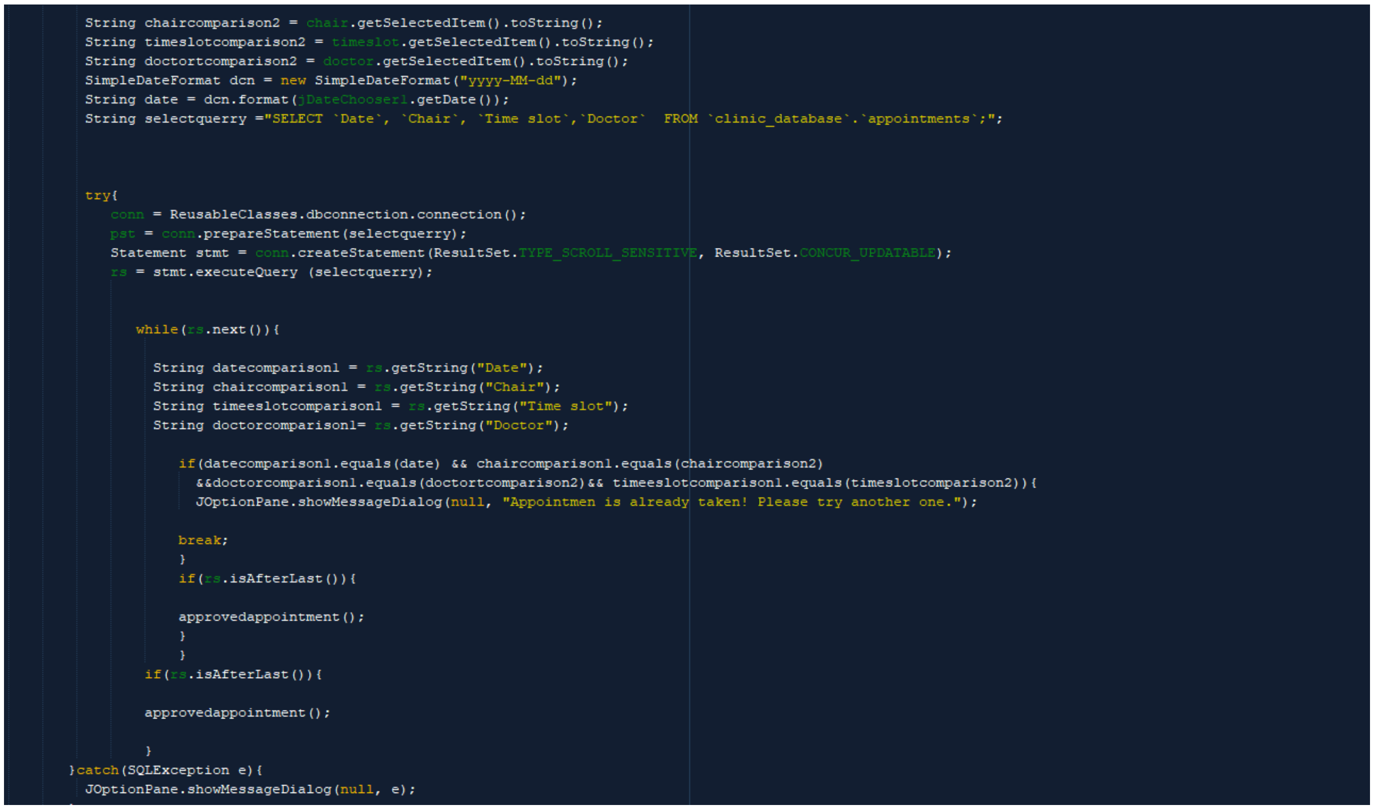
```
public final void deletepassedappointments(){

    Date date = new Date();
    SimpleDateFormat DateFor = new SimpleDateFormat("yyyy-MM-dd");
    String stringDate= DateFor.format(date);
    System.out.println(stringDate);


try{
    conn = ReusableClasses.dbconnection.connection();
    String selectquerry ="SELECT * FROM `clinic_database`.`appointments`;";
    pst = conn.prepareStatement(selectquerry);
    rs = pst.executeQuery(selectquerry);
    while (rs.next()){
    String deletepastdatequerry="DELETE FROM clinic_database.appointments WHERE (`Date`<'"+ stringDate + "');";
    pst = conn.prepareStatement(deletepastdatequerry);
    pst.executeUpdate();
    }
    ShowTableData();
}catch(SQLException e){
    }
}
```

In order to maintain appointment database functionality, a function that updates the Appointments table by deleting passed date appointments. This function is called in the constructor class of the Appointment_Database class.



```
public void printtodaysschedule(){
        Date date = new Date();
        SimpleDateFormat DateFor = new SimpleDateFormat("yyyy-MM-dd");
        String stringDate= DateFor.format(date);

try{
   String selectquerry ="SELECT * FROM `clinic_database`.`appointments` WHERE (`Date`='"+ stringDate + "');";
   pst = conn.prepareStatement(selectquerry);
   rs = pst.executeQuery();
   jTable1.setModel(DbUtils.resultSetToTableModel(rs));
   jTable1.print();
   ShowTableData();
}catch(Exception e){

}
}
```

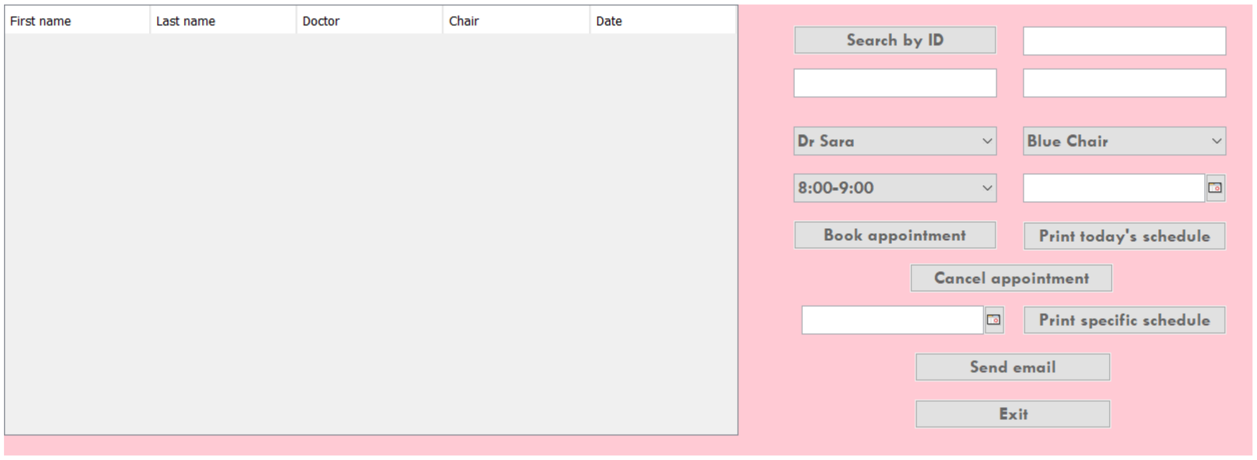
Additionally, in order to increase efficiency, the user has been given to directly print today's schedule, as seen above. This function registers the Result set which matches the Date parameter (i.e.) current date and trims the jTable before printing it, incurring also ShowTableData() function to reset the jTable to normal.



```
    String chaircomparison2 = chair.getSelectedItem().toString();
    String timeslotcomparison2 = timeslot.getSelectedItem().toString();
    String doctortcomparison2 = doctor.getSelectedItem().toString();
    SimpleDateFormat dcn = new SimpleDateFormat("yyyy-MM-dd");
    String date = dcn.format(jDateChooser1.getDate());
    String selectquerry ="SELECT `Date`, `Chair`, `Time slot`,`Doctor`  FROM `clinic_database`.`appointments`;";



    try{
       conn = ReusableClasses.dbconnection.connection();
       pst = conn.prepareStatement(selectquerry);
       Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
       rs = stmt.executeQuery (selectquerry);


          while(rs.next()){

            String datecomparison1 = rs.getString("Date");
            String chaircomparison1 = rs.getString("Chair");
            String timeeslotcomparison1 = rs.getString("Time slot");
            String doctorcomparison1= rs.getString("Doctor");

               if(datecomparison1.equals(date)  && chaircomparison1.equals(chaircomparison2)
                 &&doctorcomparison1.equals(doctortcomparison2)&& timeeslotcomparison1.equals(timeslotcomparison2)){
                 JOptionPane.showMessageDialog(null, "Appointmen is already taken! Please try another one.");

               break;
               }
               if(rs.isAfterLast()){

               approvedappointment();
               }
               }
           if(rs.isAfterLast()){

           approvedappointment();

           }
  }catch(SQLException e){
    JOptionPane.showMessageDialog(null, e);
```

Here we take advantage of the Result Set functions in order to make an efficient scheduler. The rs.next() sorts through the database searching for clashing appointments and gives the user feedback. If no clashing appointments are to be found, the appointment is added to the database.

Below you can see the GUI of the Appointment_Java class.



PDF’s of all print functions can be found in the Appendix 3 - Sample PDF’s

Word count: 987

Resources:

- Syntax referencing: StackOverflow. Retrieved from: https://stackoverflow.com

- MySQL documentation for syntax and error handling. Retrieved from: https://docs.oracle.com/cd/E17952_01/index.html

- Java documentation and manual: The Java™ Tutorials (oracle.com)

- Dimitriou, Kostas, and Markos Hatzitaskos. Advanced Computer Science: for the IB Diploma Program (International Baccalaureate) High Level Computer Science. Express Publishing, 2018.