

Blood Bank Information System

Problem Statement

Build a Blood Bank Information System which would give real-time

Problem Definition

At present, the public can only know about the blood donation events through conventional media means such as radio, newspaper or television advertisements. There is no information regarding the blood donation programs available on any of the portal. The current system that is using by the blood bank is manual system. With the manual system, there are problems in managing the donors' records. The records of the donor might not be kept safely and there might be missing of donor's records due to human error or disasters. Besides that, errors might occur when the staff keeps more than one record for the same donor. There is no centralized database of volunteer donors. So, it becomes really tedious for a person to search blood in case of emergency. The only option is to manually search and match donors and then make phone calls to every donor. There is also no centralized database used to keep the donors' records. Each bank is having their own records of donors. If a donor makes donation in different hospital, no previous records can be traced except if the donor brings along the donation certificate. Hence, the donor is considered to be a first-timer if they make blood donation in a new place. Without an automated management system, there are also problems in keeping track of the actual amount of each and every blood type in the blood bank. In addition, there is also no alert available when the blood quantity is below its par level or when the blood in the bank has expired.

Scope

There is no centralized blood bank information system under the Indian government. Such systems are operational in only few India states like Punjab etc. or under some private organizations. Although efforts have been made to make such a system operational all over India, it has not been possible to realize it in reality. Hence, there is a huge scope for centralized blood bank information system in India.

Goals and Objectives

The goals and objectives of the Blood Bank Management System are as follows:

1. To provide a means for the blood bank to publicize and advertise blood donation programs.
2. To allow the probable recipients to make search and match the volunteer donors, and make request for the blood.
3. To provide an efficient donor and blood stock management functions to the blood bank by recording the donor and blood details.
4. To improve the efficiency of blood stock management by alerting the blood bank staffs when the blood quantity is below its par level or when the blood stock has expired.
5. To provide synchronized and centralized donor and blood stock database.

6. To provide immediate storage and retrieval of data and information

THEORITICAL BACKGROUND

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform.

Java is:

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable:** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed:** Java is designed for the distributed environment of the internet.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Some basic elements of Java are discussed next.

Wrappers: Java provides Objects which wrap primitive types and supply methods.

Static: Static member function can access only static members. Static member function can be called without an instance.

Packages: Java code has hierarchical structure. The environment variable CLASSPATH contains the directory names of the roots. Every Object belongs to a package ('package' keyword). Object full name contains the name full name of the package containing it.

Access Control:

Public: Can be called/modified from outside.

Protected: Can be called/modified from derived classes

Private: Can be called/modified only from the current class

Default (if no access modifier stated): Can be accessed from the same package.

Garbage collector: In C++, you have to make sure that you destroy the objects when you are done with them. Otherwise, memory leak. In Java, garbage collector does it for you. It looks at all the objects created by new and figure out which ones are no longer being referenced. Then it releases the memory for those objects.

Java Inheritance: Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. It allows method overloading and code reusability.

Overriding and Hiding Methods in Java

An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

Polymorphism in Java

The dictionary definition of polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages. This principle can also be applied to object-oriented programming and languages like the Java language. Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.

Interfaces in Java

An interface is a collection of methods that have no implementation - they are just created, but have no functionality. An interface is a bit like a class, except you can only declare methods and variables in the interface. You cannot actually implement the methods.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

Super

The super is a keyword defined in the java programming language. Keywords are basically reserved words which have specific meaning relevant to a compiler in java programming language likewise the super keyword indicates the following:

- The super keyword in java programming language refers to the superclass of the class where the super keyword is currently being used.
- The super keyword as a standalone statement is used to call the constructor of the superclass in the base class.
- If your method overrides one of its superclass's methods, you can invoke the overridden method through the use of the keyword super. You can also use super to refer to a hidden field (although hiding fields is discouraged).

Exception Handling and Checked and Unchecked Exception

A Java Exception is an abnormal condition that arises during the execution of a program and also called a run-time error. An exception in Java signals the unusual or catastrophic situations that can arise.

Examples of Java Exceptions are:

1. Incorrect user input such as division by zero
2. File that needs to be opened not found
3. Network connection problem

Multithreading in Java

Multithreading is a process of executing multiple threads simultaneously. A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). Multithreaded programs contain two or more threads that can run concurrently. This means that a single program can perform two or more tasks simultaneously. For example, one thread is writing content on a file at the same time another thread is performing spelling check.

Java Swing

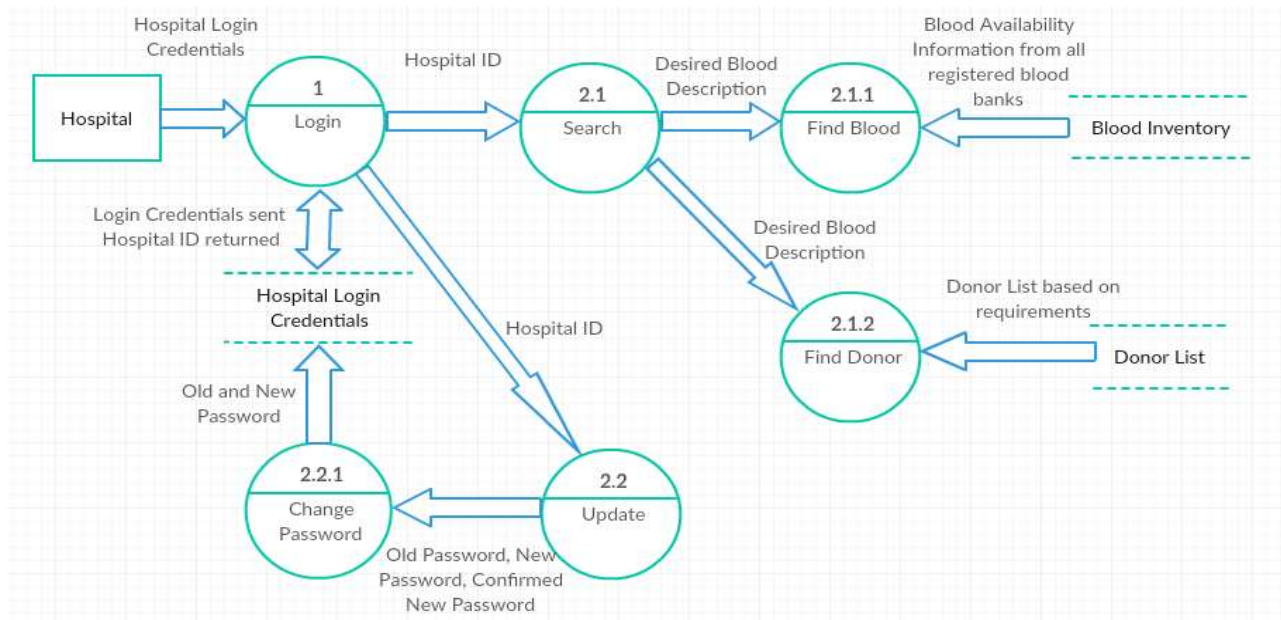
Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

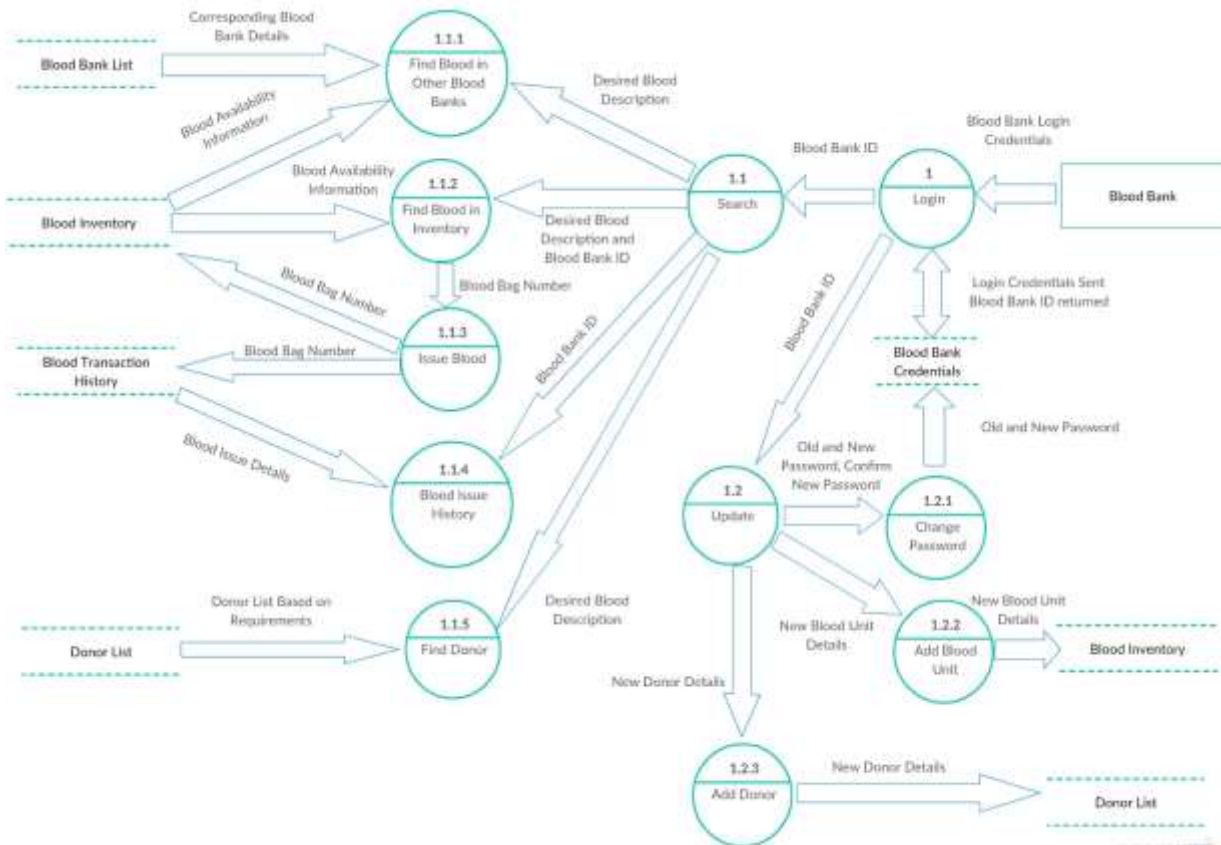
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Data Flow Diagram

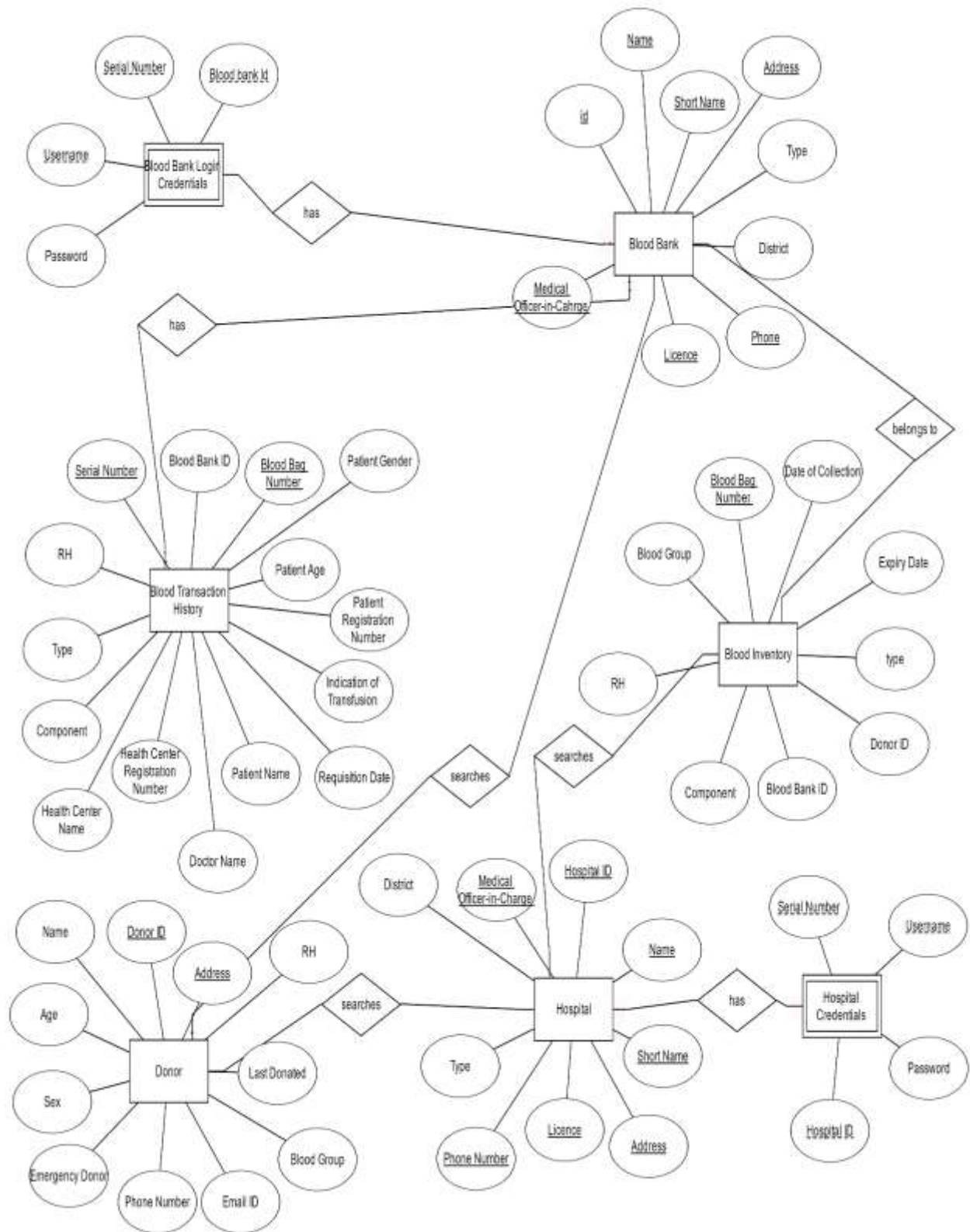
- Data Flow Diagram (Level 1) for Hospital



- Data Flow Diagram (Level 1) for Blood Bank



ERD Diagram



Requirement Analysis:

System Requirements:

- 1) **OS:** Window Vista, Window 7 and above, Windows Server 2008 and above.
- a) **Browsers:** Internet Explorer 9 and above, Firefox, Google Chrome, Safari

Hardware Requirements:

- 1) **RAM:** 128 MB
- 2) **Disk space:** 124 MB for JRE; 2 MB for Java Update
- 3) **Processor:** Minimum Pentium 2 266 MHz processor

Other Requirements

- **Active Internet connection:** The computer must have an active internet connection as the data is directly fetched and stored in the server. Without an active internet connection, the application would not work.
- All the data related to blood inventory must be updated continuously through app, else it will lead to faulty data.

Coding

Most of the GUI for the application has been developed using the Netbeans GUI Editor. Screenshots of the GUI has been provided at the later sections of the project.

The app consists of two main packages:

- 1) ApplicationHandler
- 2) DatabaseHandler

These packages represent the two fundamental layers of the application. They are:

- 1) Application Layer: 'ApplicationHandler' package represents the application layer. It has the following functions:
 - a) It generates GUI for the application.
 - b) It interacts with the users.
 - c) It filters and processes the data.
 - d) It communicates with the data layer to update or retrieve data from the database.
- 2) Database Layer: 'DatabaseHandler' package represents the application layer. It has the following functions:
 - a) It establishes connection with the remote database.
 - b) It receives queries from the application layer and updates and retrieves data from the database based on the queries.
 - c) It sends the retrieved data to the application layer.

Following are some of the major code snippets which are responsible for controlling the logic and data flow of the application:

1) DatabaseConnection.java

Package: DatabaseHandler

Function: Establishes connection with the SQL database and returns the connection to DatabaseHandler class.

Code:

```
package DatabaseHandler;

import java.sql.*;

public class DatabaseConnection {

    private Connection con;

    public DatabaseConnection() throws SQLException {

        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/bbms",
            "root", "");

    }

}
```



```

        public Connection getConnection() {

            return con;

        }

    }

```

2) DatabaseHandler.java

Package: DatabaseHandler

Function: It gets the connection from DatabaseConnection class. It receives queries from the application layer and returns the result set. It also updates the database based on the received query.

Code:

```

package DatabaseHandler;
import java.sql.*;
public class DatabaseHandler {
    Connection con;
    Thread t;
    public DatabaseHandler () {
        try {
            con = new DatabaseConnection().getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public ResultSet read(String str) throws SQLException {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(str);
        return rs;
    }

    public void update(String str) throws SQLException {
        Statement st = con.createStatement();
        st.executeUpdate(str);
        st.close();
    }
}

```

3) MonitorLock.java

Package: DatabaseHandler

Function: It maintains synchronization of thread during updating of database.

Code:

```

package DatabaseHandler;

```

```

public class MonitorLock implements Runnable {
    public Thread t;
    DatabaseHandler dh;
    String sqlStatement, task;
    public MonitorLock(String sqlStatement, String task, DatabaseHandler dh) {
        this.sqlStatement = sqlStatement;
        this.task = task;
        this.dh = dh;
        t = new Thread(this);
        t.start();
    }
    @Override
    public void run() {
        try {
            synchronized(dh) {
                if(task.equals("update")) {
                    dh.update(sqlStatement);
                }
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

4) AddBloodUnit.java

Package: ApplicationHandler

Function: It generates a form to take the details of new blood unit and sends the details to the DatabaseHandler layer for updating the blood_inventory table.

Code:

```
package ApplicationHandler;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```

import java.sql.SQLException;
import java.util.Properties;
import javax.swing.*;
import org.jdatepicker.impl.*;
import DatabaseHandler.DatabaseHandler;
import DatabaseHandler.MonitorLock;

public class AddBloodUnit extends JDialog implements ActionListener {
[Generated Code]
    public AddBloodUnit(java.awt.Frame parent, boolean modal, DatabaseHandler dh) {
        super(parent, modal);
        this.dh = dh;
        initComponents();
    }
    @Override
        public void actionPerformed(ActionEvent e) {
            String s = e.getActionCommand();
            if(s.equals("Submit")) {
                String query = "insert into blood_inventory values('"+
                    jTextField1.getText() + "','"+ jComboBox1.getSelectedItem().toString()
                    + "','"+ jComboBox2.getSelectedItem().toString() + "','"+
                    datePicker1.getModel().getYear() + "-" +
                    datePicker1.getModel().getMonth() + "-" +
                    datePicker1.getModel().getDay() + "','"+
                    datePicker.getModel().getYear() + "-" +
                    datePicker.getModel().getMonth() + "-" +
                    datePicker.getModel().getDay() + "','"+
                    jComboBox3.getSelectedItem().toString() + "','"+
                    jComboBox4.getSelectedItem().toString() + "','"+ jTextField6.getText()
                    + "','"+ jTextField7.getText() + "')";
                MonitorLock ml = new MonitorLock(query, "update", dh);
                try {
                    ml.t.join();
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
                JOptionPane.showMessageDialog(this,"Update successful!");
                dispose();
            }
        }
    }
}

```

5) AddDonor.java

Package: ApplicationHandler

Function: It generates a form to take the details of new donor and sends the details to the DatabaseHandler layer for updating the donor_list table.

Code:

```
package ApplicationHandler;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.Properties;
import javax.swing.*;
import org.jdatepicker.impl.*;
import DatabaseHandler.DatabaseHandler;

public class AddDonor extends JDialog implements ActionListener {
[Generated Code]
    public AddDonor(java.awt.Frame parent, boolean modal, DatabaseHandler dh) {
        super(parent, modal);
        this.dh = dh;
        initComponents();
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        String s = e.getActionCommand();
        if(s.equals("Submit")) {
            System.out.println("Name: " + jTextField1.getText());
            System.out.println("Address: " + jTextArea1.getText());
            System.out.println("Age: " + jTextField2.getText());
            System.out.println("Sex: " + jComboBox1.getSelectedItem().toString());
            System.out.println("Phone: " + jTextField3.getText());
            System.out.println("Email: " + jTextField4.getText());
            System.out.println("Blood Group: " +
                jComboBox3.getSelectedItem().toString());
            System.out.println("Emergency Donor: " +
                bg.getSelection().getActionCommand());
            System.out.println("Last Donated: " +
                datePicker.getModel().getValue());
            int ed = (bg.getSelection().getActionCommand().equals("Yes"))? 1 : 0;
            String donorID = null;
            try {
                ResultSet rs = dh.read("select count(*) from donor_list");
                if(rs.next()) {
                    donorID = new
                        DonorCodeGenerator().generateCode(rs.getInt(1));
                }
            }
        }
    }
}
```

```

    } catch (SQLException e2) {
        e2.printStackTrace();
    }
    StringBuilder strb = new StringBuilder();
    strb.append("insert into donor_list values(");
    strb.append(""" + donorID + "," + "");
    strb.append(""" + jTextField1.getText() + "," + "");
    strb.append(""" + jTextArea1.getText() + "," + "");
    strb.append(""" + jTextField2.getText() + "," + "");
    strb.append(""" + jComboBox1.getSelectedItem().toString() + "," + "");
    strb.append(""" + jTextField3.getText() + "," + "");
    strb.append(""" + jTextField4.getText() + "," + "");
    strb.append(""" + jComboBox3.getSelectedItem().toString() + "," + "");
    strb.append(""" + jComboBox4.getSelectedItem().toString() + "," + "");
    strb.append(""" + ed + "," + "");
    strb.append(""" + datePicker.getModel().getYear() + "-" +
        datePicker.getModel().getMonth() + "-" +
        datePicker.getModel().getDay() + """);
    strb.append(")");
    try {
        dh.update(strb.toString());
    } catch (SQLException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    dispose();
}
}
}
}
}

```

6) Login.java

Package: ApplicationHandler

Function: Takes login credentials and validates them with the data in the database. If login is unsuccessful, it displays an error message. If login credentials match, then it logs the user into the main application.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import DatabaseHandler.DatabaseHandler;

public class Login extends javax.swing.JDialog implements ActionListener {
    [Generated Code]

```

```

@Override
public void actionPerformed(ActionEvent e) {
    String s = e.getActionCommand();
    if(s.equals("Submit")) {
        try {
            String type =
(jComboBox1.getSelectedItem().toString().equals("Hospital"))? "hospital" : "blood_bank";
            ApplicationData.type = type;
            String str = "select * from " + type + " where username = '" +
jTextField1.getText() + "' and password = '" + jPasswordField1.getText().toString() + "'";
            ResultSet rs = dh.read(str);
            if(rs.next()) {
                ApplicationData.bbID = rs.getString(4);
                ResultSet rs1 = dh.read("select short_name from " +
ApplicationData.type + "_list where id = " + ApplicationData.bbID);
                rs1.next();
                ApplicationData.setData(rs1.getString(1));
                JOptionPane.showMessageDialog(this,"Login successful!");
                dispose();
                new MainFrame(dh);
            }
            else {
                JOptionPane.showMessageDialog(this,
                    "Login Error. Try Again!",
                    "Inane error",
                    JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
}
}

```

7) MainFrame.java

Package: ApplicationHandler

Function: It defines the main frame of the application. Either the blood bank view or the hospital view is rendered into this frame.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import DatabaseHandler.DatabaseHandler;

```

```

public class MainFrame {
    DatabaseHandler dh;
    JFrame jfrm;
    JPanel jp;
    CardLayout cd;
    public MainFrame(DatabaseHandler dh) {
        this.dh = dh;
        initComponents();
    }
    public void initComponents() {
        jfrm = new JFrame();
        jfrm.setTitle("Blood Bank Information System");
        jfrm.setSize(1280, 720);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setVisible(true);
        if(ApplicationData.type.equals("blood_bank")) {
            BBHomePage jpn = new BBHomePage(dh);
            jfrm.setJMenuBar(jpn.getMenuBar());
            jfrm.add(jpn, BorderLayout.CENTER);
        }
        else {
            HHomePage hpn = new HHomePage(dh);
            jfrm.setJMenuBar(hpn.getMenuBar());
            jfrm.add(hpn, BorderLayout.CENTER);
        }
    }
}

```

8) BBHomePage.java

Package: ApplicationHandler

Function: It defines the view of the blood bank. Various components are rendered into this view including the search panel, task list panel, etc.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import DatabaseHandler.DatabaseHandler;
public class BBHomePage extends JPanel implements ActionListener {
    [Generated]
    JMenuBar jmb;
    JMenu jmFile, jmNew, jmOptions;
    JMenuItem jmiBloodBag, jmiDonor, jmiExit, jmiChangePassword;
    DatabaseHandler dh;
    QueryPanel qp;

```

```

TaskListPanel tlp;
public BBHomePage(DatabaseHandler dh) {
    this.dh = dh;
    jPanel5 = new javax.swing.JPanel();
    jPanel6 = new javax.swing.JPanel();
    specialInit();
    initComponents();
}
public void specialInit() {
    qp = new QueryPanel(jPanel5, dh);
    tlp = new TaskListPanel(jPanel6, qp);
}
public void initComponents() {
    createMenuBar();
    [Generated Code]
}
public void createMenuBar() {
    jmb = new JMenuBar();
    jmFile = new JMenu("File");
    jmNew = new JMenu("New");
    jmiBloodBag = new JMenuItem("Blood Unit");
    jmiBloodBag.addActionListener(this);
    jmiDonor = new JMenuItem("Donor"); jmiDonor.addActionListener(this);
    jmNew.add(jmiBloodBag);
    jmNew.add(jmiDonor);
    jmiExit = new JMenuItem("Exit"); jmiExit.addActionListener(this);
    jmFile.add(jmNew);
    jmFile.add(jmiExit);
    jmb.add(jmFile);
    jmOptions = new JMenu("Options");
    jmiChangePassword = new JMenuItem("Change Password");
    jmiChangePassword.addActionListener(this);
    jmOptions.add(jmiChangePassword);
    jmb.add(jmOptions);
}
public JMenuBar getMenuBar() { return jmb; }
@Override
public void actionPerformed(ActionEvent e) {
    String s = e.getActionCommand();
    if(s.equals("Blood Unit")) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                AddBloodUnit dialog = new AddBloodUnit(new javax.swing.JFrame(), true, dh);
                dialog.setTitle("Add Blood Unit");
            }
        });
    }
}

```



```

        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        dialog.setVisible(true);
    }
});
}
else if(s.equals("Donor")) {
    java.awt.EventQueue.invokeLater(new Runnable() {
public void run() {
    AddDonor dialog = new AddDonor(new javax.swing.JFrame(), true, dh);
    dialog.setTitle("Add Donor");
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    dialog.setVisible(true);
}
});
}
else if(s.equals("Change Password")) {
    java.awt.EventQueue.invokeLater(new Runnable() {
public void run() {
    ChangePassword dialog = new ChangePassword(new javax.swing.JFrame(),
true, dh);
    dialog.setTitle("Change Password");
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    dialog.setVisible(true);
}
});
}
else if(s.equals("Exit")) {
    System.exit(0);
}
}
}
}

```

9) **BloodIssueHistoryPanel.java**

Package: ApplicationHandler

Function: It shows the blood issue history for a particular blood bank.

Code:

```

package ApplicationHandler;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JPanel;
import javax.swing.table.DefaultTableModel;
import DatabaseHandler.DatabaseHandler;

public class BloodIssueHistoryPanel extends JPanel implements Runnable{
[Generated]

```

```

DatabaseHandler dh;
ResultSet rs;
DefaultTableModel tableModel;
Thread th;
public BloodIssueHistoryPanel(DatabaseHandler dh) {
    this.dh = dh;
    initComponents();
    th = new Thread(this);
    th.start();
}
[Generated Code]
private void populateTable() {
    String sqlStatement;
    StringBuilder query = new StringBuilder();
    query.append("SELECT `bag_no`, `blood_group`, `rh`, `type`, `component`,
`health_center`, `doctor_name`, `patient_name`, `p_age`, `p_gender`,
`p_req_date`, `indication_of_transfusion` FROM `blood_transaction` where ");
    query.append("bbid = '" + ApplicationData.bbID + "'");
    query.append(" ORDER BY blood_group ASC");
    sqlStatement = query.toString();
    while (tableModel.getRowCount() > 0) {
        tableModel.removeRow(0);
    }
    try {
        rs = dh.read(sqlStatement);
        while(rs.next()) {
            Object[] obj = new Object[12];
            obj[0] = rs.getString(1); // bag number
            obj[1] = rs.getString(2); // blood group
            obj[2] = rs.getString(3); // rh
            obj[3] = rs.getString(4); // type
            obj[4] = rs.getString(5); // component
            obj[5] = rs.getString(6); // health center
            obj[6] = rs.getString(7); // doctor
            obj[7] = rs.getString(8); // patient
            obj[8] = rs.getInt(9); // age
            obj[9] = rs.getString(10); // gender
            obj[10] = rs.getDate(11).toString(); // requisition date
            obj[11] = rs.getString(12); // indication of transfusion
            tableModel.addRow(obj);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

    }
    @Override
    public void run() {
        populateTable();
        try {
            Thread.sleep(10000); // Updates view after every 10 seconds
            run();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

10) FindBlood.java

Package: ApplicationHandler

Function: Finds and lists desired blood from all blood banks.

Code:

```

package ApplicationHandler;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.sql.SQLException;
import javax.swing.JDialog;
import javax.swing.table.DefaultTableModel;
import DatabaseHandler.DatabaseHandler;

public class FindBlood extends SearchInventoryPanel {
    public FindBlood(DatabaseHandler dh) {
        super(dh);
        jLabel1.setText("Search for Blood");
    }
    [Generated Code]
    private void findBlood() {
        String sqlStatement;
        StringBuilder query = new StringBuilder();
        query.append("SELECT count(*), blood_bank_list.name,
        blood_bank_list.district, blood_bank_list.type, blood_bank_list.phone,
        blood_bank_list.address FROM `blood_inventory` join blood_bank_list on
        blood_inventory.bbid = blood_bank_list.id WHERE ");
        query.append("abo = '" + jComboBox1.getSelectedItem().toString() + "'");
        query.append(" and ");
        query.append("rh = '" + jComboBox2.getSelectedItem().toString() + "'");
        query.append(" and ");
    }
}

```

```

        query.append("blood_inventory.type = '" +
jComboBox3.getSelectedItem().toString() + "'");
if(jComboBox3.getSelectedItem().toString().equals("Component")) {
        query.append(" and ");
        query.append("Component = '" +
jComboBox4.getSelectedItem().toString() + "'");
}
query.append(" group by bbid order by blood_bank_list.district ASC");
sqlStatement = query.toString();
while (tableModel.getRowCount() > 0) {
        tableModel.removeRow(0);
}
try {
        rs = dh.read(sqlStatement);
        while(rs.next()) {
                Object[] obj = new Object[6];
                obj[0] = rs.getInt(1);
                obj[1] = rs.getString(2);
                obj[2] = rs.getString(3);
                obj[3] = rs.getString(4);
                obj[4] = rs.getString(5);
                obj[5] = rs.getString(6);
                tableModel.addRow(obj);
        }
        } catch (SQLException e) {
                e.printStackTrace();
        }
}

```

11) TaskListPanel.java

Package: ApplicationHandler

Function: Lists the various tasks which can be performed by the blood bank. When an option is clicked, it loads the corresponding view in the central panel.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TaskListPanel {
        JPanel jp;
        JLabel li1, li2, li3, li4;
        QueryPanel qp;
        public TaskListPanel (JPanel jp, QueryPanel qp) {
                this.jp = jp;

```

```

        this.qp = qp;
        jp.setLayout(new BorderLayout(jp, BorderLayout.Y_AXIS));
        jp.setBackground(Color.WHITE);
        initComponents();
    }

    public void initComponents() {
        li1 = new JLabel("Search Inventory"); li1.setCursor(new
        Cursor(Cursor.HAND_CURSOR));
        li4 = new JLabel("General Blood Search"); li4.setCursor(new
        Cursor(Cursor.HAND_CURSOR));
        li2 = new JLabel("Search Donor"); li2.setCursor(new
        Cursor(Cursor.HAND_CURSOR));
        li3 = new JLabel("View Blood Issue History"); li3.setCursor(new
        Cursor(Cursor.HAND_CURSOR));
        li1.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                qp.changePanel("Search Inventory");
            }
        });

        li2.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                qp.changePanel("Search Donor");
            }
        });

        li3.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                qp.changePanel("Blood Issue History");
            }
        });

        li4.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                qp.changePanel("General Blood Search");
            }
        });

        Font font = new Font("Times New Roman", 1, 20);
        jp.setFont(font);        jp.add(li1);
        jp.add(Box.createRigidArea(new Dimension(0, 10)));
    }

```

```

                jp.add(li4);                jp.add(Box.createRigidArea(new Dimension(0,10)));
                jp.add(li2);                jp.add(Box.createRigidArea(new Dimension(0,10)));
                jp.add(li3);
            }
        }
    }
}

```

12) QueryPanel.java

Package: ApplicationHandler

Function: It initiates the different views which will be rendered in the central panel of BBHomePage.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import DatabaseHandler.DatabaseHandler;

public class QueryPanel {
    JPanel jp;
    CardLayout cd;
    DatabaseHandler dh;

    public QueryPanel(JPanel jp, DatabaseHandler dh) {
        this.dh = dh;
        this.jp = jp;
        initComponents();
    }

    public void initComponents() {
        cd = new CardLayout();
        jp.setLayout(cd);
        jp.setBackground(new Color(240, 230, 230));
        SearchInventoryPanel si = new SearchInventoryPanel(dh);
        jp.add("Search Inventory", si);
        FindBlood fb = new FindBlood(dh);
        jp.add("General Blood Search", fb);
        SearchDonorPanel sd = new SearchDonorPanel(dh);
        jp.add("Search Donor", sd);
        BloodIssueHistoryPanel bihp = new BloodIssueHistoryPanel(dh);
        jp.add("Blood Issue History", bihp);
        cd.show(jp, "Search Inventory");
    }

    void changePanel(String panelName) {
        cd.show(jp, panelName);
    }
}

```

13) SearchInventoryPanel.java

Package: ApplicationHandler

Function: Searches and lists the desired blood units available in the inventory of the corresponding blood bank.

Code:

```
package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import DatabaseHandler.DatabaseHandler;

public class SearchInventoryPanel extends JPanel implements Runnable {
[Generated Code]
    DatabaseHandler dh;
    ResultSet rs;
    Thread th;

    public SearchInventoryPanel(DatabaseHandler dh) {
        this.dh = dh;
        initComponents();
        th = new Thread(this);
        th.start();
    }

    public void searchInventory() {
        String sqlStatement;
        StringBuilder query = new StringBuilder();
        query.append("select blood_bag_number, abo, rh, type, Component, donorid,
date_of_collection, expiry_date from blood_inventory where ");
        query.append("bbid = '" + ApplicationData.bbID + "'");
        query.append(" and ");
        query.append("abo = '" + jComboBox1.getSelectedItem().toString() + "'");
        query.append(" and ");
        query.append("rh = '" + jComboBox2.getSelectedItem().toString() + "'");
        query.append(" and ");
        query.append("type = '" + jComboBox3.getSelectedItem().toString() + "'");
        if(jComboBox3.getSelectedItem().toString().equals("Component")) {
            query.append(" and ");
            query.append("Component = '" +
jComboBox4.getSelectedItem().toString() + "'");
        }
    }
}
```

```

        sqlStatement = query.toString();
        while (tableModel.getRowCount() > 0) {
            tableModel.removeRow(0);
        }
        try {
            rs = dh.read(sqlStatement);
            while(rs.next()) {
                Object[] obj = new Object[9];
                obj[0] = rs.getString(1);
                obj[1] = rs.getString(2);
                obj[2] = rs.getString(3);
                obj[3] = rs.getString(4);
                obj[4] = rs.getString(5);
                obj[5] = rs.getString(6);
                obj[6] = rs.getDate(7).toString();
                obj[7] = rs.getDate(8).toString();
                obj[8] = "Issue";
                tableModel.addRow(obj);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void run() {
        searchInventory();
        try {
            Thread.sleep(10000);
            run();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

14) SearchDonorPanel.java

Package: ApplicationHandler

Function: Searches donor from the donor_list table with the desired blood group.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;

```



```

import javax.swing.*.*;
import javax.swing.table.DefaultTableModel;
import DatabaseHandler.DatabaseHandler;

public class SearchDonorPanel extends JPanel {
    [Generated Code]
    DatabaseHandler dh;
    ResultSet rs;
    public SearchDonorPanel(DatabaseHandler dh) {
        this.dh = dh;
        initComponents();
    }
    [Generated Code]
    void searchDonor() {
        String sqlStatement;
        StringBuilder query = new StringBuilder();
        query.append("select name, age, sex, phone, email from donor_list where ");
        query.append("blood_group = '" + jComboBox1.getSelectedItem().toString() +
            "'");
        query.append(" and ");
        query.append("rh = '" + jComboBox2.getSelectedItem().toString() + "'");
        sqlStatement = query.toString();
        while (tableModel.getRowCount() > 0) {
            tableModel.removeRow(0);
        }
        try {
            rs = dh.read(sqlStatement);
            while(rs.next()) {
                Object[] obj = new Object[5];
                obj[0] = rs.getString(1);
                obj[1] = rs.getInt(2);
                obj[2] = rs.getString(3);
                obj[3] = rs.getString(4);
                obj[4] = rs.getString(5);
                tableModel.addRow(obj);
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

15) IssueBlood.java

Package: ApplicationHandler

Function: It issues the desired blood unit, updates it in the blood issue history table and removes the record from the blood inventory list.

Code:

```
package ApplicationHandler;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import org.jdatepicker.impl.JDatePanelImpl;
import org.jdatepicker.impl.JDatePickerImpl;
import org.jdatepicker.impl.UtilDateModel;
import DatabaseHandler.DatabaseHandler;
import DatabaseHandler.MonitorLock;

public class IssueBlood extends JDialog {
    UtilDateModel model;
    JDatePanelImpl datePanel;
    JDatePickerImpl datePicker;
    DatabaseHandler dh;

    public IssueBlood(java.awt.Frame parent, boolean modal, DatabaseHandler dh, String group,
String rh, String type, String component, String bag_no) {
        super(parent, modal);
        this.dh = dh;
        initComponents();
        jLabel15.setText(group);
        jLabel16.setText(rh);
        jLabel19.setText(type);
        if(!component.equals(null)) {
            jLabel21.setText(component);
        }
        jLabel17.setText(bag_no);
    }

    private void initComponents() {
        Properties p = new Properties();
        p.put("text.today", "Today");
        p.put("text.month", "Month");
        p.put("text.year", "Year");
    }
}
```

```

        model = new UtilDateModel();
        datePanel = new JDatePanelImpl(model, p);
        datePicker = new JDatePickerImpl(datePanel, new DateLabelFormatter());
[Generated Code]
    }

```

```

void issueBlood() throws SQLException {
    String sqlStatement;
    ResultSet rs = dh.read("select count(*) from blood_transaction");
    rs.next();
    StringBuilder query = new StringBuilder();
    query.append("insert into blood_transaction(`serial`, `bbid`, `bag_no`,
`blood_group`, `rh`, `type`, `component`, `health_center`, `hc_reg_no`, `doctor_name`,
`patient_name`, `p_reg_no`, `p_age`, `p_gender`, `p_req_date`, `indication_of_transfusion`)
values(");

    query.append(rs.getInt(1));
    query.append(",");
    query.append("''" + ApplicationData.bbID + "''");
    query.append(",");
    query.append("''" + jLabel17.getText() + "''");
    query.append(",");
    query.append("''" + jLabel15.getText() + "''");
    query.append(",");
    query.append("''" + jLabel16.getText() + "''");
    query.append(",");
    query.append("''" + jLabel19.getText() + "''");
    query.append(",");
    query.append("''" + jLabel21.getText() + "''");
    query.append(",");
    query.append("''" + jTextField3.getText() + "''");
    query.append(",");
    query.append("''" + jTextField4.getText() + "''");
    query.append(",");
    query.append("''" + jTextField5.getText() + "''");
    query.append(",");
    query.append("''" + jTextField2.getText() + "''");
    query.append(",");
    query.append("''" + jTextField7.getText() + "''");
    query.append(",");
    query.append("''" + jTextField6.getText() + "''");
    query.append(",");
    query.append("''" + jComboBox3.getSelectedItem().toString() + "''");
    query.append(",");

```

```

        query.append("'" + datePicker.getModel().getYear() + "-" +
datePicker.getModel().getMonth() + "-" + datePicker.getModel().getDay() + "'");
        query.append(",");
        query.append("'" + jComboBox4.getSelectedItem().toString() + "'");
        query.append(")");
        sqlStatement = query.toString();
        MonitorLock ml = new MonitorLock(sqlStatement, "update", dh);
        try {
            ml.t.join();
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }
        sqlStatement = "delete from blood_inventory where blood_bag_number = '" +
jLabel17.getText() + "' and bbid = '" + ApplicationData.bbID + "'";
        System.out.println(sqlStatement);
        MonitorLock m2 = new MonitorLock(sqlStatement, "update", dh);
        try {
            m2.t.join();
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }
        JOptionPane.showMessageDialog(this, "Update successful!");
        dispose();
    }
}

```

16) ChangePassword.java

Platform: ApplicationHandler

Function: Generates a form which allows the user to enter a new password which replaces the old password present in the database.

Code:

```

package ApplicationHandler;
import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.regex.Pattern;
import javax.swing.*;
import DatabaseHandler.DatabaseHandler;

public class ChangePassword extends JDialog implements ActionListener {
    [Generated Code]
    DatabaseHandler dh;
    public ChangePassword(java.awt.Frame parent, boolean modal, DatabaseHandler dh) {
        super(parent, modal);
    }
}

```

```

        this.dh = dh;
        initComponents();
    }

```

[Generated code]

```

private void jTextField3KeyReleased(java.awt.event.KeyEvent evt) {
    String s1 = jPasswordField2.getText().toString();
    String s2 = jPasswordField3.getText().toString();
    if(s1.equals(s2)) {
        jLabel5.setText("Password is matching!");
        jLabel5.setForeground(new Color(30, 120, 0));
    }
    else {
        jLabel5.setText("Passwords do not match!");
        jLabel5.setForeground(Color.RED);
    }
}

private void jTextField2KeyReleased(java.awt.event.KeyEvent evt) {
    String s1 = jPasswordField2.getText().toString();
    if(s1.matches("(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\S+$).{8,}")) {
        jLabel6.setForeground(new Color(30, 120, 0));
        jLabel6.setText("Strong");
    }
    else {
        jLabel6.setForeground(java.awt.Color.red);
        jLabel6.setText("Weak");
    }
}

@Override
public void actionPerformed(ActionEvent e) {
}
}

```

17) DonorCodeGenerator.java

Package: ApplicationHandler

Function: It automatically generates an id when a new donor is registered.

Code:

```
package ApplicationHandler;
```

```

public class DonorCodeGenerator {
    String id;
    static String ida;
}

```

```

static String idb;
static {
    ida = "D";
    idb = "0000000000";
}
public DonorCodeGenerator() {
    System.out.println("Generating code ...");
}
public String generateCode(int donorCount) {
    idb = idb + donorCount;
    return ( ida + idb );
}
}

```

18) StartApp.java

Package: default package

Function: Initializes the application by setting the theme and creating the login dialog box.

Code:

```

import javax.swing.JDialog;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;
import ApplicationHandler.Login;

public class StartApp {
    public static void main(String args[]) {
        try {
            for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {
            // If Nimbus is not available, you can set the GUI to another look and feel.
        }

        /* Create and display the dialog */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                Login dialog = new Login(new javax.swing.JFrame(), true);
                dialog.setTitle("Login");
                dialog.setSize(500, 280);
                dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
                dialog.setVisible(true);
            }
        });
    }
}

```

```
        }  
    }  
};
```

Note: There are few other java files like HHomePage.java, HQueryPanel.java and HTaskListPanel which are analogous to BBHomePage.java, QueryPanel.java and TaskListPanel, under blood bank view, respectively with minor modifications.

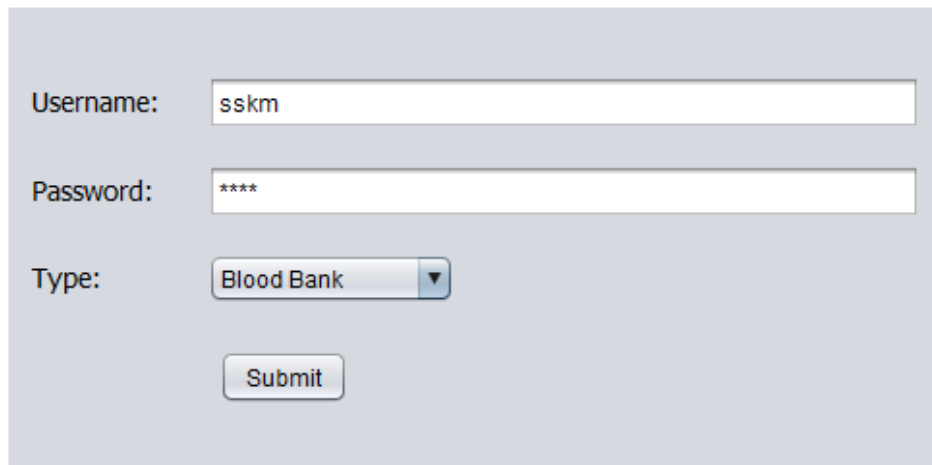
Limitation

Since the application is still in the initial stages, there are a few limitations of the application:

- 1) Internet Dependency: The application must be connected to the internet for proper functioning.
- 2) Human Factor: If there is any delay or error in updating the blood inventory by the respective blood banks, then it would result in faulty search results.
- 3) Donor Tracking: There is no infrastructure to track where the donors are at different points of time. Also, donors cannot update their information especially the last donated field. A donor can donate only after 56 days from their previous donation date.
- 4) Public Access: The general mass or family of a patient do not have access to the database. They cannot search for blood themselves.

Screenshots

- Login Screen



The screenshot displays a login interface with a light gray background. It features three input fields: a text box for 'Username' containing 'sskm', a text box for 'Password' containing five asterisks '*****', and a dropdown menu for 'Type' with 'Blood Bank' selected. Below these fields is a 'Submit' button.

- Search Inventory Interface and Home Screen

The screenshot shows the 'Blood Bank Information System' home screen. The header is red with the text 'Blood Bank Information System' and 'Welcome, SSKM Blood Bank!'. A 'Sign Out' button is in the top right. On the left, a 'Tasks' sidebar lists 'Search Inventory', 'General Blood Search', 'Search Donor', and 'View Blood Issue History'. The main area is titled 'Search Inventory' and contains a search form with dropdowns for 'Group' (A+), 'RH' (Positive), 'Type' (Whole Blood), and 'Component' (Not Applicable), followed by a 'Search' button. Below the form is a table with the following data:

Bag Number	Blood Group	RH	Type	Component	Donor ID	Date of Collect	Expiry Date	
BB1	A+	Positive	Whole Blood	Not Applicable	D00000000001	2017-02-24	2018-02-16	Issue
BB2	A+	Positive	Whole Blood	Not Applicable	D0000000001	2017-03-15	2018-04-11	Issue
BB7	A+	Positive	Whole Blood	Not Applicable	D0000000002	2017-02-10	2017-09-20	Issue

- Search Blood Banks for desired blood

The screenshot shows the 'Search for Blood' interface in the 'Blood Bank Information System'. The header is red with the text 'Blood Bank Information System' and 'Welcome, SSKM Blood Bank!'. A 'Sign Out' button is in the top right. On the left, a 'Tasks' sidebar lists 'Search Inventory', 'General Blood Search', 'Search Donor', and 'View Blood Issue History'. The main area is titled 'Search for Blood' and contains a search form with dropdowns for 'Group' (A+), 'RH' (Positive), 'Type' (Whole Blood), and 'Component' (Not Applicable), followed by a 'Search' button. Below the form is a table with the following data:

Number of Units	Blood Bank	District	Type	Phone	Address
3	SSKM Hospital Blood Bank	Kolkata	Government	22234174,9338207842	AJC Bose Road, Kolkata - 20
2	Institute of Blood Transfusio...	Kolkata	Government	23510519,9433892164	205, Vivekananda Road, Ma...

- Search Donor Interface

File Options

Blood Bank Information System Welcome, SSKM Blood Bank! [Sign Out](#)

Tasks

- [Search Inventory](#)
- [General Blood Search](#)
- [Search Donor](#)
- [View Blood Issue History](#)

Search Donor

Group: RH:

Name	Age	Gender	Phone	Email
Debdyuti Hajra	21	Male	7892842321	debdyuti@rockers.hu

- View Blood Issue History Interface

File Options

Blood Bank Information System Welcome, SSKM Blood Bank! [Sign Out](#)

Tasks

- [Search Inventory](#)
- [General Blood Search](#)
- [Search Donor](#)
- [View Blood Issue History](#)

Blood Issue History

Bag Number	Blood Group	RH	Type	Component	Health Center	Doctor	Patient	Age	Gender	Requestion...	Indication of...
BB1	A+	Positive	Whole Blood	Not Applicab...	Calcutta Me...	Farukh Shah	Mostafa Sweth	42	Male	2017-02-14	Acute Haem...
BB2	A+	Positive	Whole Blood	Not Applicab...	SSKM	Debasmita G.	Debdyuti Haj	52	Male	2017-02-13	Routine Sur...

- Add Blood Unit Dialog

Bag Number:	<input type="text"/>
ABO:	<input type="text" value="A+"/>
RH:	<input type="text" value="Positive"/>
Type:	<input type="text" value="Whole Blood"/>
Compnents:	<input type="text" value="Not Applicable"/>
Blood Bank ID:	<input type="text" value="2"/>
Donor ID:	<input type="text"/>
Date of Collection:	<input type="text"/> ...
Expiry Date:	<input type="text"/> ...
<input type="button" value="Submit"/>	

- Add Donor Dialog

Name:	<input type="text"/>
Address:	<input type="text"/>
Age:	<input type="text"/>
Sex:	<input type="text" value="Male"/>
Phone:	<input type="text"/>
Email:	<input type="text"/>
Blood Group:	<input type="text" value="A+"/>
RH:	<input type="text" value="Positive"/>
Emergency Donor:	<input checked="" type="radio"/> Yes <input type="radio"/> No
Last Donated:	<input type="text"/> ...
<input type="button" value="Submit"/>	

- Change Password Dialog

Old Password:

New Password:

Password Strength: **Weak**

Re-enter new password:

Passwords do not match!

- Hospital Interface

File Options

Blood Bank Information System Welcome, R G Kar Hospital!

Tasks

[Search for Blood](#)
[Search Donor](#)

Search for Blood

Group: Rh: Type: Component:

Number of Units	Blood Bank	District	Type	Phone	Address
3	SGKM Hospital Blood Bank	Kolkata	Government	22234174, 9339207942	AJC Bose Road, Kolkata - 20
2	Institute of Blood Transfusion	Kolkata	Government	23510819, 9433892164	205, Vivekananda Road, Ma...

Conclusion

In this project, we have successfully created a centralized blood bank information system to be used by blood banks and hospitals. It can be used in real life. If this system is adopted by all the blood banks and hospitals, then it will help to bring transparency, give access to blood inventory information of various blood banks to patients and their families and hence save many lives.

Bibliography

- Java Reference: 'Java: The Complete Reference' by Herbert Schildt
- Blood Bank Information: www.wbhealth.gov.in
- About Blood: www.hematology.org
- About Blood Bank Information System: www.sciencedirect.com
- DFD and ERD drawing tool: creately.com