
INTRODUCTION

The Blood Bank Management System is designed to manage blood donation, storage, and distribution efficiently. This system aims to ensure the availability of blood and related components while maintaining data integrity, security, and ease of access for donors, hospitals, and administrators.

Blood banks play a crucial role in the healthcare system by ensuring a reliable supply of blood for transfusions and emergencies. However, managing the collection, testing, storage, and distribution of blood requires meticulous organization. Traditional methods often face challenges such as delays in processing requests, insufficient data for decisionmaking and a lack of realtime inventory tracking.the proposed Blood Bank Management System seeks to overcome these challenges by leveraging modern technologies. It enables streamlined operations, minimizes human errors, and enhances communication between donors, recipients, and healthcare providers. With its focus on automation and transparency, this system is vital for meeting the increasing demand for blood and improving patient outcomes.

Additionally, the system aims to simplify the donor recruitment process, promote awareness of blood donation drives, and ensure the quality of stored blood. It integrates advanced features like notification alerts for blood shortages, AI-powered donor-recipient matching, and mobile accessibility to extend its reach and usability. By centralizing data and enhancing collaboration between stakeholders, the Blood Bank Management System contributes to building a sustainable and responsive healthcare infrastructure

SYSTEM SPECIFICATIONS

2.1 SYSTEM REQUIREMENTS:

2.1.1 HARDWARE REQUIREMENTS:

- Processor: Intel Core i5 or higher
- RAM: 8 GB minimum
- Hard Disk: 500 GB
- Monitor: 15-inch or larger
- Keyboard and Mouse
- Network Interface Card

2.1.2 SOFTWARE REQUIREMENTS:

- Operating System: Windows 10
- Database: MySQL 8.0
- Backend: Python
- Frontend: HTML, CSS
- Development Tools: VSCode
- Framework: Python Flask

LITERATURE SURVEY

1.AUTOMATION OF BLOOD BANK MANAGEMENT:

AUTHORS: T. Anandhi, M. S. Ananthi, and G. A. Vijayalakshmi Pai

TITLE: "Automated Blood Bank Management System Based on Cloud- Computing and Internet of Things (IoT)"

KEY FINDINGS:

The development of an advanced cloud-based blood bank system integrated with IoT devices to enable real-time tracking and monitoring of blood samples throughout the supply chain. The system incorporated key features such as donor registration, automated blood stock management, and notifications for critical shortages, ensuring efficient blood utilization and timely action. It emphasized system scalability, interoperability, and the importance of real-time data updates, making it particularly effective during emergency scenarios. The study also explored the use of IoT sensors to monitor environmental conditions such as temperature and humidity during storage and transport, ensuring the safety and quality of blood products. Additionally, the system facilitated centralized data management to improve coordination between hospitals, blood banks, and donors, reducing operational inefficiencies and response times. The research demonstrated how integrating IoT with cloud computing could revolutionize blood bank management, ensuring reliability, transparency, and enhanced service delivery.

2. MOBILE APPLICATIONS FOR BLOOD DONATION:

AUTHORS: S. Ali, S. S. Riaz, and M. A. Khan

TITLE: "Mobile Application for Blood Donation and Management"

KEY FINDINGS:

The authors developed a mobile application designed to directly connect blood donors and recipients, optimizing the blood donation process. The app integrates location-based services to help users easily find nearby blood banks or potential donors, ensuring quick and efficient access to blood when needed. Additionally, it features real-time notifications for users regarding urgent blood requirements, donation drives, or changes in blood availability, improving responsiveness. The application also allows users to track their donation history, ensuring transparency and encouraging repeat donations. The system aims to increase awareness and engagement among potential donors, ultimately addressing the challenges of blood shortages. By providing a simple and user-friendly platform, the app helps overcome barriers such as geographical distance and lack of information, making blood donation more accessible, especially in emergency situations.

3. CLOUD-BASED BLOOD BANK SYSTEMS:

AUTHORS: J. Singh and P. Kumar

TITLE: "Cloud Computing-Based Blood Bank Management System"

KEY FINDINGS:

The development of a cloud-based platform to manage donor information, blood availability, and requests across multiple locations. The system effectively addresses issues like data redundancy and retrieval delays by centralizing data storage and utilizing cloud technology for real-time updates. Additionally, the authors implemented role-based access control to ensure the security of sensitive information, making the system both reliable and secure. The cloud-based approach enhances the overall efficiency of blood bank management, improving coordination and resource accessibility across multiple regions.

4 . WEB-BASED SYSTEMS FOR BLOOD DONATION:

AUTHORS: A. Sharma, V. Gupta, and R. Kumar

TITLE:" Web-Based BloodBank System for Efficient Management"

KEY FINDINGS:

This platform allowed real-time management of blood inventory and facilitated direct interactions between hospitals, donors, and recipients. By integrating these stakeholders, the system ensured better coordination, more efficient resource allocation, and faster updates on blood availability, ultimately improving the responsiveness of the blood donation process. The web-based approach provided a user-friendly interface, making it easier for all parties involved to access information and manage their roles effectively.

5. SMS-BASED ALERT SYSTEMS :

AUTHORS: M. L. Mishra and S. Roy

TITLE: "SMS-Based Emergency Blood Bank Management System"

KEY FINDINGS:

This system aimed to address the challenges faced by remote areas with limited internet access, ensuring that critical blood shortages could be managed effectively. By leveraging SMS technology, the system improved response times and enhanced the ability to quickly mobilize donors during urgent situations, demonstrating its efficiency in facilitating rapid communication and saving lives in emergency blood bank scenarios.

4. SOFTWARE REQUIREMENT ANALYSIS:

Software requirements analysis is the process of identifying, documenting, and managing the needs and expectations of stakeholders for a software application. It is a critical phase in software development that ensures the final product meets user needs and functions as intended. During this phase, software engineers and analysts gather information through interviews, surveys, observations, and reviewing existing systems to define both functional and non-functional requirements. These requirements are then documented in detail, forming the basis for system design, development, testing, and deployment. Proper analysis helps avoid costly mistakes, scope creep, and misunderstandings throughout the software development lifecycle.

4.1 EXISTING SYSTEM :

Blood bank management systems are typically designed to manage the entire lifecycle of blood donation, storage, and distribution. These systems often include modules for donor registration, blood inventory management, compatibility testing, and tracking the storage conditions of blood products. They may use centralized databases to store information about donors, blood types, and availability, and allow hospitals and clinics to request specific blood types. Some systems also include features for monitoring blood expiration dates, generating reports, and ensuring compliance with regulatory standards. However, many of these systems are still reliant on manual processes or outdated technology, which can lead to inefficiencies, data discrepancies, and challenges in managing emergencies, blood shortages, and inventory control.

ADVANTAGES OF EXISTING SYSTEM :

Existing blood bank management systems offer several advantages, including streamlined processes for donor registration, blood inventory management, and compatibility testing. These systems help reduce human errors, improve record-keeping accuracy, and ensure that blood resources are efficiently tracked and utilized. Centralized databases facilitate quick access to critical information, enabling hospitals and clinics to respond promptly to blood requests. Additionally, automated features for monitoring expiration dates, generating reports, and ensuring regulatory compliance enhance operational efficiency and safety. Overall, these systems improve coordination among blood banks, hospitals, and donors, contributing to better management of blood supplies and improved patient care.

DISADVANTAGES OF EXISTING SYSTEM :

Blood bank management systems have several disadvantages, primarily stemming from reliance on outdated technologies and manual processes. Many systems suffer from limited scalability, making it difficult to accommodate growing demands or integrate with other healthcare systems. Data redundancy and delays in retrieval can lead to inefficiencies, particularly when managing large volumes of donor and blood stock information. Additionally, these systems may lack real-time updates, making it challenging to respond quickly to urgent situations or blood shortages.

4.2 PROPOSED SYSTEM :

The proposed Blood Bank Management System is a centralized digital platform designed to streamline the processes of blood donation, storage, and distribution. It aims to enhance the efficiency, transparency, and accessibility of blood management by maintaining real-time records of blood stock, donor details, and recipient requests. The system will include features such as donor registration, blood group inventory management, automated notifications for low stock levels, and search functionalities for matching donors and recipients. Integrated with secure authentication protocols, the system ensures data privacy and prevents unauthorized access. This solution simplifies communication between donors, hospitals, and blood banks, reducing delays and ensuring timely availability of blood to save lives.

ADVANTAGES OF PROPOSED SYSTEM :

The proposed blood bank management systems, often incorporating modern technologies like cloud computing, IoT, and mobile applications, offer several key advantages. These systems enable real-time tracking of blood inventory, improving efficiency and reducing wastage. Cloud-based platforms provide scalability, allowing for better data storage and accessibility across multiple locations, ensuring seamless coordination between blood banks, hospitals, and donors. Automated notifications, whether for critical blood shortages or blood expiration, enhance responsiveness and decision-making. Additionally, the integration of mobile apps and IoT devices enables easier donor engagement, location-based services, and remote monitoring, ensuring timely blood donations and optimal blood supply management. Enhanced security protocols and data encryption further safeguard sensitive information, while these systems also allow for better forecasting and demand.

MODULES

5.MODULES:

5.1 Admin

5.2 Hospital

5.3 Donor

MODULES DESCRIPTION

5.1 ADMIN :

In a Blood Bank Management System, the **Admin** plays a pivotal role in managing the system's operations, ensuring efficiency, accuracy, and security. They oversee the entire workflow, from donor registration to blood inventory management and recipient matching. Admins are responsible for maintaining a comprehensive database of donors, recipients, and blood stock, including details like blood type, quantity, collection date, and expiration date. They handle user account management, assigning roles and permissions to staff, ensuring secure access to sensitive data.

The admin also coordinates donation drives by managing schedules, sending notifications to donors, and tracking attendance and contributions. They respond to urgent blood requests, ensuring compatibility and timely delivery to recipients or healthcare facilities. Additionally, admins monitor system performance, troubleshoot issues, and ensure regular backups and updates to maintain functionality and security. They generate detailed analytical reports on inventory status, blood demand trends, and donor activity to support decision-making and strategic planning. By enforcing compliance with healthcare regulations and maintaining high data accuracy, the admin ensures the reliability and integrity of the blood bank management system.

The admin tracks the expiration of blood units and ensures proper disposal of expired or contaminated blood to maintain safety standards. They also oversee integration with hospital systems to streamline blood requisitions and deliveries. Admins play a crucial role in maintaining communication between blood banks, hospitals, and donors, ensuring a seamless flow of information.

5.2 HOSPITAL:

A Hospital in the context of a Blood Bank Management System plays a critical role in the healthcare ecosystem by acting as a recipient of blood donations, ensuring that blood is available for patients in need. Hospitals work in collaboration with blood banks to manage blood requests based on patient requirements, including specific blood types, quantities, and emergency situations. Hospitals use the system to place orders, track the status of blood deliveries, and monitor the usage of blood units for procedures such as surgeries, trauma care, or treatments for conditions like anemia or cancer. They are also responsible for ensuring proper storage and handling of the blood once received to maintain its viability.

Hospitals maintain detailed records of blood transfusions, including recipient information, blood type compatibility, and any reactions during or after the transfusion. They also report back to the blood bank in cases of adverse reactions or wastage, helping to optimize future operations. By integrating with the Blood Bank Management System, hospitals ensure that their blood supply is well-managed, safe, and accessible when required. The hospital's role extends to managing relationships with donors, organizing donation drives, and educating patients and the public on the importance of blood donation for the well-being of the community.

5.3 DONOR:

The Donor side of a Blood Bank Management System focuses on individuals who contribute blood to the system. Donors are registered in the system, providing essential personal details, medical history, and blood type. The system allows donors to schedule appointments for blood donation, receive notifications about upcoming donation opportunities, and track their donation history, including the frequency and types of blood donated. Donors can also receive reminders

The system provides an easy-to-use interface for donors to access important information, such as donation guidelines, benefits, and any post-donation care instructions. Donors can view the impact of their contributions, such as how their blood has been used in medical treatments, fostering a sense of connection and contribution to public health. The donor side of the system also supports communication, sending thank-you messages

SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE :

The blood bank management system typically follows a layered, modular structure to ensure scalability, efficiency, and security. At the core, it includes a centralized database that stores critical information, such as donor details, blood types, inventory levels, and compatibility data. This is accessed and managed through user interfaces for different roles, including blood bank staff, medical professionals, and donors. The system often integrates with external components such as IoT devices for real-time monitoring of blood storage conditions and mobile applications for donor engagement. Cloud computing can be employed to ensure high availability, data redundancy, and accessibility across multiple locations. The architecture also incorporates security features, such as role-based access control, encryption, and regular backups, to protect sensitive information. Furthermore, the system may use automated notification services, such as SMS or email, to alert stakeholders about critical shortages or expiration dates, enhancing operational efficiency and responsiveness.

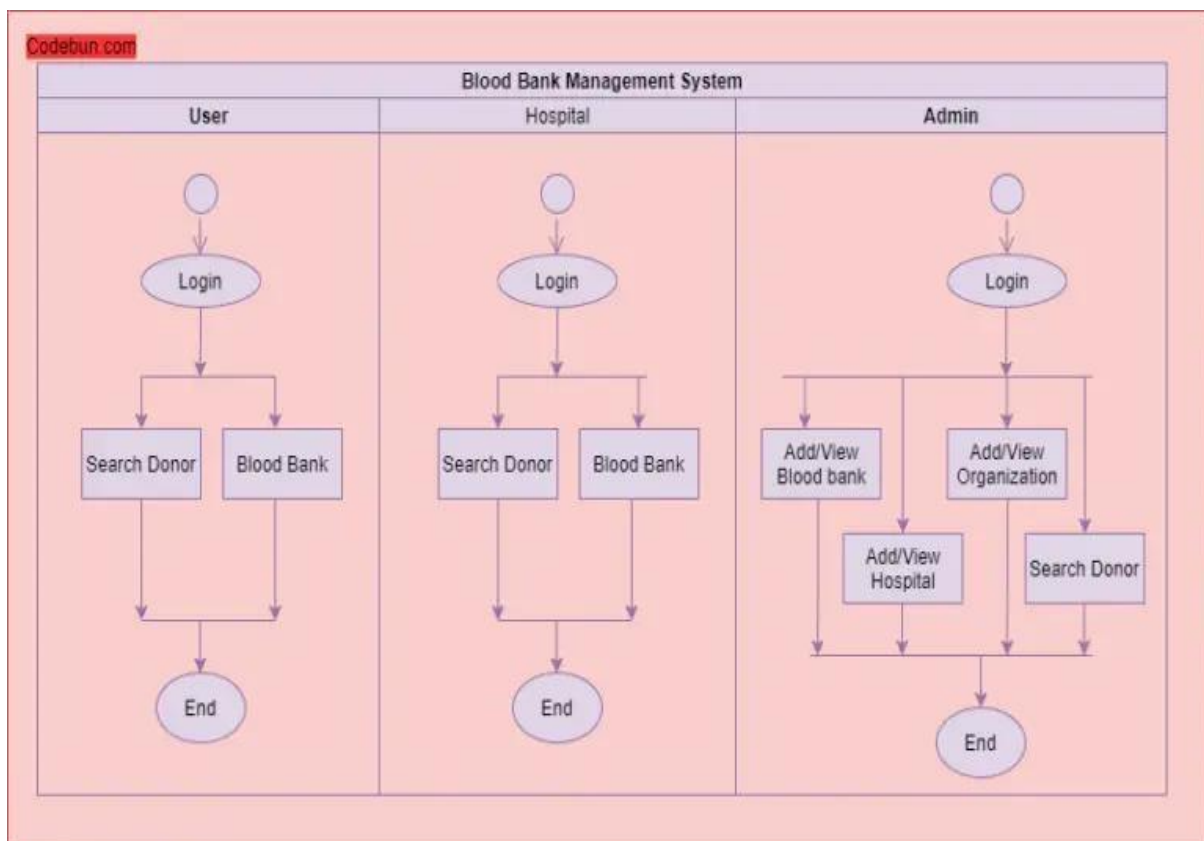


FIG 6.1: SYSTEM ARCHITECTURE

6.2 DATA FLOW DIAGRAM:

The Data Flow Diagram (DFD) of a Blood Bank Management System illustrates the flow of information between various components and entities within the system, ensuring smooth operations. The primary entities involved are the Admin, Donor, Hospital, and Blood Bank Database. The process begins with the Donor, who registers through the system, providing personal details, medical history, and blood type. The admin oversees this data, verifying donor eligibility and managing blood collection details such as date, location, and type of blood donated.

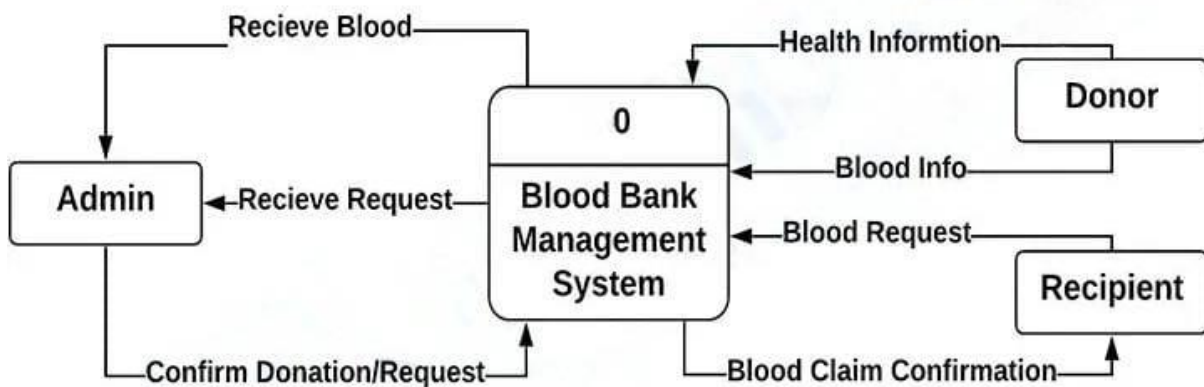


FIG 6.2: DATA FLOW DIAGRAM

6.3 UML DIAGRAMS:

The Blood Bank Management System typically include key components such as **Use Case Diagrams**, **Class Diagrams**, **Sequence Diagrams**, and **Activity Diagrams**. The **Use Case Diagram** outlines interactions between actors (Donor, Recipient, and Admin) and the system, covering activities like registering donors, managing blood donations, processing blood requests, and tracking inventory. The **Class Diagram** defines entities such as Donor, Recipient, Admin, and BloodBank, with attributes (e.g., blood type, donor details) and methods (e.g., validate request, update inventory). The **Sequence Diagram** captures the flow of events for processes like blood donation or request fulfillment, showing interactions between actors and the system over time. The **Activity Diagram** represents workflows such as donor registration, blood request handling, and blood inventory management, highlighting decision points and actions. Together, these diagrams provide a comprehensive view of system design and functionality.

GOALS

The goal of UML diagrams in a system like a Blood Bank Management System is to provide a clear, structured, and standardized representation of the system's architecture, processes, and interactions. UML diagrams help in visualizing the functional requirements, defining relationships between components, and ensuring a shared understanding among stakeholders, such as developers, project managers, and end-users. They facilitate communication by breaking down complex systems into manageable elements, support efficient design by identifying redundancies and inconsistencies early, and aid in maintaining and scaling the system. Overall, UML diagrams aim to enhance system development, documentation, and collaboration throughout the project lifecycle.

- i. **Visual Representation:** Provide a clear and structured visual representation of the system's architecture, components, and processes.
- ii. **Define Functional Requirements:** Highlight key functionalities, workflows, and interactions between system components.
- iii. **Simplify Communication:** Serve as a common language for developers, designers, and stakeholders to understand the system.
- iv. **Improve Design:** Identify redundancies, gaps, or inconsistencies in the design to improve efficiency and structure.
- v. **Enhance Documentation:** Act as a reference for system documentation and future maintenance.
- vi. **Support Scalability:** Aid in planning system upgrades or scalability by clearly showing component relationships.

6.3.1 USE CASE DIAGRAM:

Blood Bank Management System shows the structures of information or data that will be handled in the system. These data or information will be represented by classes. Each of the classes will have their attributes in accord to the methods they will use. So the UML Class diagram was illustrated by a box with 3 partitions and the upper part was the name of the class, the middles are the attributes and the bottom is for the methods. The arrows on them represents their relationships in each other.

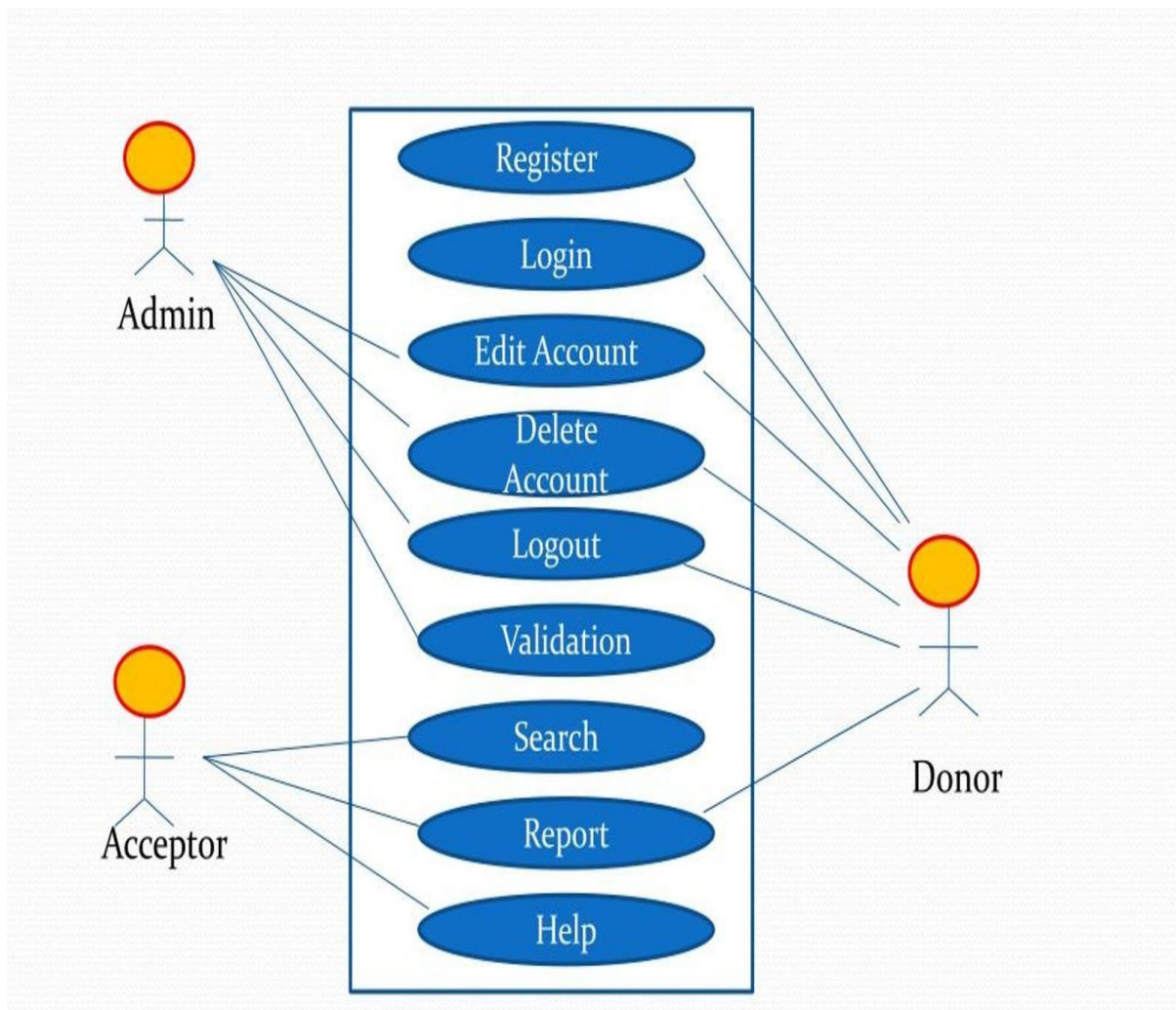


FIG 6.4: USE CASE DIAGRAM

6.3.2 CLASS DIAGRAM:

Blood Bank Management System is a visual representation of the system's structure, showing its key components (classes) and their relationships. It defines the system's static view, focusing on the attributes and methods of each class as well as the interactions between them. Below are the main classes typically found in a Blood Bank Management System and their descriptions

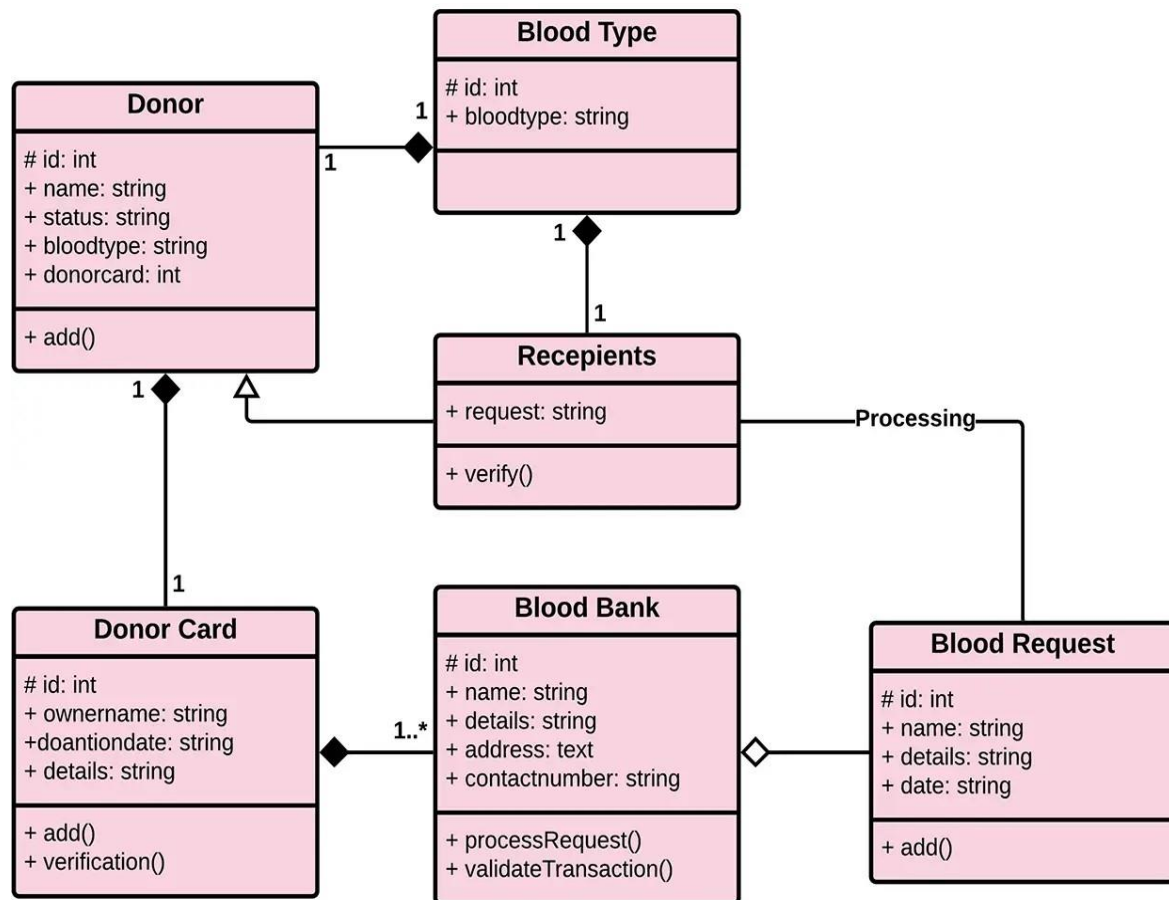


FIG 6.5: CLASS DIAGRAM

6.3.3 SEQUENCE DIAGRAM :

Blood Bank Management System is a dynamic model that shows the interaction between objects in a time-ordered sequence. It captures how processes flow in the system and how different components (actors and objects) interact with each other to perform specific tasks. Below is a general description of a sequence diagram for a Blood Bank Management System

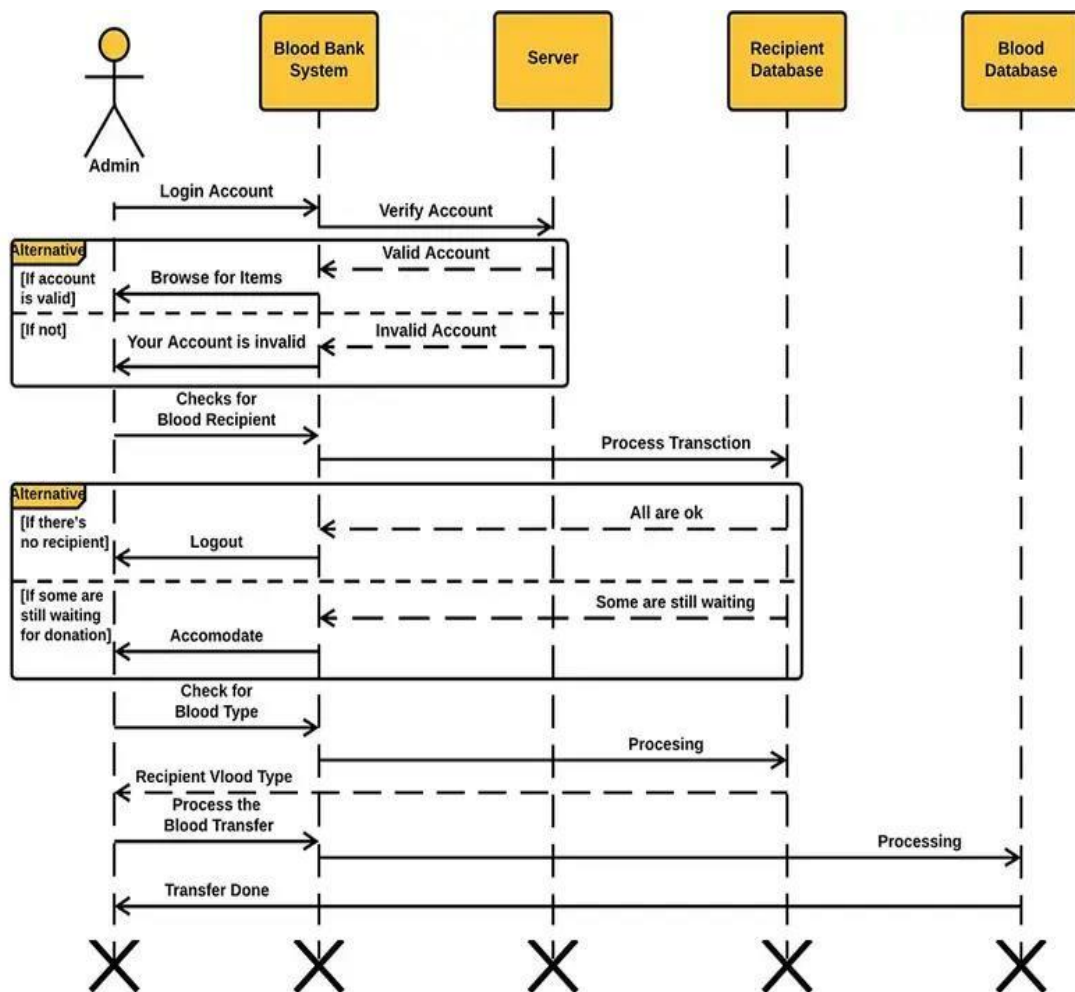


FIG 6.6: SEQUENCE DIAGRAM

6.3.4 ACTIVITY DIAGRAM:

The Blood Bank Management System represents the step-by-step workflows and processes involved in managing blood donations, requests, and inventory. It begins with donor registration, followed by health screening and donation approval. If approved, the blood is collected, stored, and updated in the inventory. For recipients, the process starts with submitting a blood request, which is verified for availability. If the requested blood type is available, the system processes the claim and allocates the blood to the recipient. The admin oversees these workflows, including monitoring inventory levels and handling exceptional cases like low stock or invalid requests. The diagram includes decision points (e.g., donor eligibility, blood availability) and actions (e.g., updating records, notifying stakeholders), ensuring a streamlined and efficient flow of activities across the system.

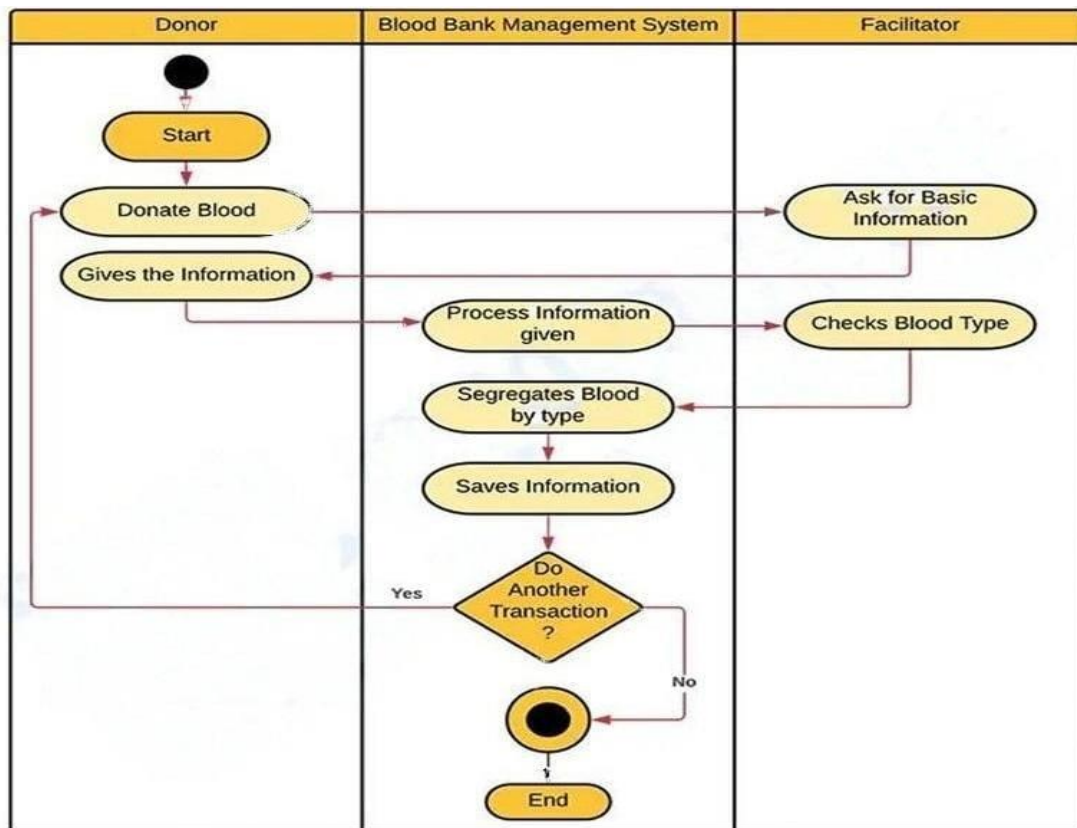


FIG 6.7: ACTIVITY DIAGRAM

7.SYSTEM STUDY

7.1 FEASIBILITY STUDY:

- 7.1.1 Technical Feasibility
- 7.1.2 Economic Feasibility
- 7.1.3 Operational Feasibility
- 7.1.4 Legal and Ethical Feasibility
- 7.1.5 Social Feasibility

7.1.1 TECHNICAL FEASIBILITY :

The development of a Blood Bank Management System is technically feasible as it can be built using modern programming languages, frameworks, and database systems. Tools such as MySQL for database management, and technologies like Python, Java, or PHP for backend development, ensure that the system is robust and scalable. Additionally, integration with cloud platforms can enhance accessibility and reliability. Most of the required technologies are readily available and widely supported by IT professionals.

7.1.2 ECONOMIC FEASIBILITY:

The cost of developing and maintaining the system is reasonable when compared to the benefits it provides. Initial investment includes hardware, software, and development costs, while operational costs involve periodic updates and maintenance. The system helps reduce wastage of blood, optimize storage, and save lives, which outweighs the initial cost. Non-profit organizations and government grants can further support implementation and operation.

7.1.3 OPERATIONAL FEASIBILITY:

The system is highly feasible in terms of operation as it simplifies processes like donor registration, blood inventory tracking, and request management. Automation reduces errors and increases efficiency, making it easier for medical staff to operate. With minimal training, healthcare workers can effectively use the system to serve the community.

7.1.4 LEGAL AND ETHICAL FEASIBILITY:

The system complies with legal and ethical guidelines, such as ensuring donor confidentiality and adhering to blood safety standards. It can incorporate features to ensure proper documentation and secure sensitive data to meet regulatory requirements.

7.1.5 SOCIAL FEASIBILITY:

The system has strong social acceptance as it addresses critical healthcare needs. By ensuring timely availability of blood, it improves trust between hospitals, blood banks, and the community. Educating the public about the system's benefits will further encourage voluntary blood donations.

SOURCE CODE

8.1 ADMIN.PY:

```
from flask import Flask, render_template, request, redirect, flash, url_for, session
from database import connect_db

app = Flask(__name__)
app.secret_key = "your_secret_key" # Use a strong secret key for production

@app.route('/')
def home():
    if 'username' in session: # Check if user is logged in
        return render_template('index.html')
    return redirect(url_for('login')) # Redirect to login if not logged in

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # For simplicity, we're using hardcoded values. Replace this with your database validation
        # logic.
        if username == 'admin' and password == 'password': # Replace with proper validation
            session['username'] = username
            flash("Login successful!", "success")
            return redirect(url_for('home'))
        else:
            flash("Invalid credentials. Please try again.", "danger")
            return render_template('login.html', username=username) # Pass the username back

    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('username', None) # Remove the user from session
    flash("You have been logged out.", "info")
    return redirect(url_for('login'))

@app.route('/stock')
def view_stock():
    if 'username' not in session: # Ensure user is logged in
        return redirect(url_for('login'))

    conn = connect_db()
    cursor = conn.cursor()
```

```

    cursor.execute("SELECT * FROM blood_stock")
    stock = cursor.fetchall()
    conn.close()
    return render_template('stock.html', stock=stock)

@app.route('/donor/admin', methods=['GET', 'POST'])
def add_donor():
    if 'username' not in session: # Ensure user is logged in
        return redirect(url_for('login'))
    if 'username' not in session: # Ensure user is logged in
        return redirect(url_for('login'))

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM donors")
    donors = cursor.fetchall()

    cursor.execute("SELECT * FROM recipients")
    recipients = cursor.fetchall()

    conn.close()
    return render_template('admin.html', donors=donors, recipients=recipients)
@app.route('/update_status/<int:request_id>', methods=['POST'])
def update_status(request_id):
    status = request.form['status']

    conn = connect_db()
    cur = conn.cursor()
    cur.execute("UPDATE donors SET status=%s WHERE id=%s", (status, request_id))
    conn.commit()
    cur.close()
    flash('Request status updated successfully!', 'success')
    return redirect(url_for('add_donor')) # Make sure this is the correct URL to redirect to

@app.route('/recipient/request', methods=['GET', 'POST'])
def request_blood():
    if 'username' not in session: # Ensure user is logged in
        return redirect(url_for('login'))

    if request.method == 'POST':
        name = request.form['name']
        blood_type = request.form['blood_type']
        contact = request.form['contact']

        conn = connect_db()

```

```

        cursor = conn.cursor()

        # Check if blood is available
        cursor.execute("SELECT units_available FROM blood_stock WHERE blood_type=%s",
            (blood_type,))
        stock = cursor.fetchone()

        if stock and stock[0] > 0:
            # Deduct blood stock and record the recipient
            cursor.execute("UPDATE blood_stock SET units_available = units_available - 1
WHERE blood_type=%s", (blood_type,))
            cursor.execute(
                "INSERT INTO recipients (name, blood_type, contact) VALUES (%s, %s, %s)",
                (name, blood_type, contact)
            )
            conn.commit()
            flash("Blood request processed successfully!", "success")
        else:
            flash("Requested blood type is out of stock.", "danger")

        conn.close()
        return redirect(url_for('home'))

    return render_template('request_blood.html')
'''

@app.route('/admin')
def admin():
    return render_template('login.html')
    if 'username' not in session: # Ensure user is logged in
        return redirect(url_for('login'))

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM donors")
    donors = cursor.fetchall()

    cursor.execute("SELECT * FROM recipients")
    recipients = cursor.fetchall()

    conn.close()
    return render_template('admin.html', donors=donors, recipients=recipients)

'''

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=8000, debug=True)

```

8.2 LOGIN.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Donor Login</title>
</head>
<body>
  <div class="login-container">
    <h2>Login to VSB Blood Bank</h2>
    <form action="/login" method="POST">
      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
      <input type="submit" value="Login">
    </form>
    <p>Don't have an account? <a href="/register">Register here</a></p>
    <p>click here <a href="/donate">guest Login</a></p>
  </div>
</body>
</html>
```

8.3 INDEX.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blood Bank Management</title>
  <!-- Link to external CSS file -->
  <link rel="stylesheet" href="static/css/styles.css">
</head>
<body>
  <div class="container">
    <div class="card">
      <h1>Welcome to the Blood Bank Management System</h1>
      <a class="card" href="/stock">View Blood Stock</a> <br><br>
      <a class="card" href="/donor/admin">Admin</a> <br><br>

      <br><br>
      <!-- Logout Button -->
      <a class="logout-btn" href="/logout">Logout</a>
    </div>
  </div>
</body>
</html>
```

8.4 ADMIN.HTML:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Panel</title>
  <link rel="stylesheet" href="static/css/styles.css">
</head>
<body>
  <h1>Admin Dashboard</h1>

  <h2>Requests</h2>
  <table>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Blood Type</th>
      <th>Units</th>
      <th>Contact</th>
      <th>status</th>
      <th>send status</th>
    </tr>
    {% for donor in donors %}
    <tr>
      <td>{{ donor[0] }}</td>
      <td>{{ donor[1] }}</td>
      <td>{{ donor[2] }}</td>
      <td>{{ donor[3] }}</td>
      <td>{{ donor[4] }}</td>
      <td>{{ donor[5] }}</td>

      <td>
        <form method="POST" action="/update_status/{{ donor[0] }}">
          <select name="status">
            <option value="Pending" {% if request[5] == 'pending' %}>selected{% endif %}>Pending</option>
            <option value="Completed" {% if request[5] == 'Accept' %}>selected{% endif %}>Accept</option>
            <option value="Rejected" {% if request[5] == 'Reject' %}>selected{% endif %}>Rejected</option>
          </select>
      </td>
    </tr>
    {% endfor %}
  </table>

```

```
        <button type="submit">Update</button>
    </form>
</td>
</tr>
{% endfor %}
</table>

<h2>Donars</h2>
<table>
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
        <th>Blood Type</th>
        <th>Contact</th>
    </tr>
    {% for recipient in recipients %}
    <tr>
        <td>{{ recipient[0] }}</td>
        <td>{{ recipient[1] }}</td>
        <td>{{ recipient[2] }}</td>
        <td>{{ recipient[3] }}</td>
        <td>{{ recipient[4] }}</td>
    </tr>
    {% endfor %}
</table>

<a href="/">Back to Home</a>
</body>
</html>
```

8.5 ADD_DONOR.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Donor</title>
  <link rel="stylesheet" href="static/css/styles.css">
</head>
<body>
  <h1>Add a New Donor</h1>
  <form method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>

    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required>

    <label for="blood_type">Blood Type:</label>
    <input type="text" id="blood_type" name="blood_type" required>

    <label for="contact">Contact:</label>
    <input type="text" id="contact" name="contact" required>

    <button type="submit">Add Donor</button>
  </form>
  <a href="/">Back to Home</a>
</body>
</html>
```

9. TESTINGS

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

9.1 TYPES OF TESTS:

9.1.1 UNIT TESTING :

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

9.1.2 INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent

9.1.3 FUNCTIONAL TESTING:

Functional testing is a type of software testing that focuses on verifying that the software system performs its intended functions as specified by requirements. It involves testing the application's features and functionality, such as user inputs, business processes, and outputs, to ensure they work correctly.

9.1.4 SYSTEM TESTING:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

9.1.5 WHITE BOX TESTING :

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

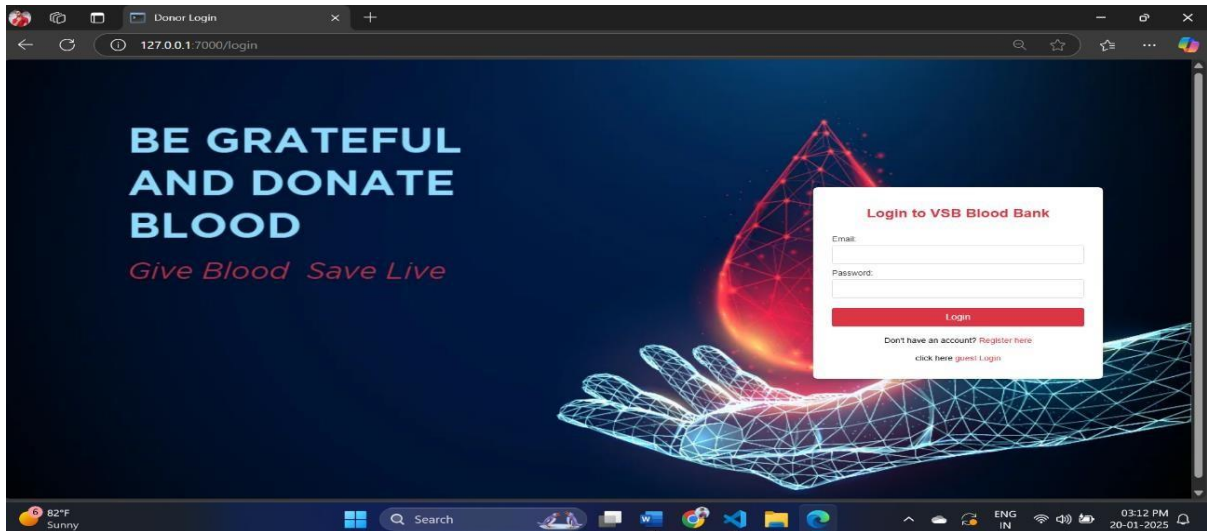
9.1.6 BLACK BOX TESTING :

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works. Unit Testing

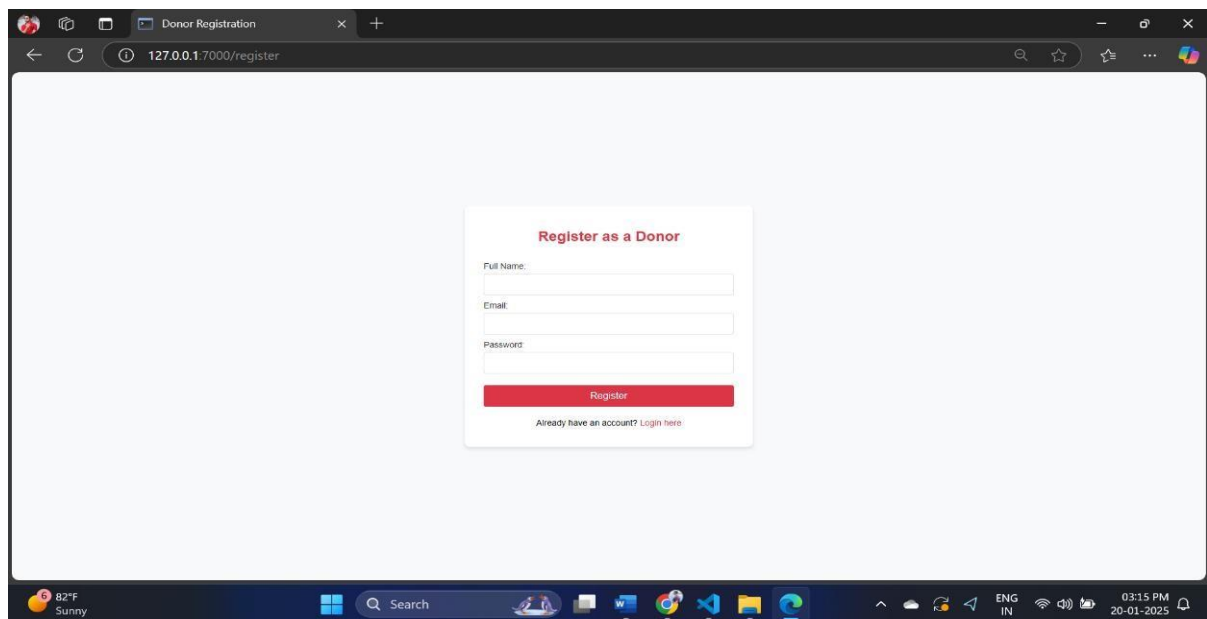
9.2 TESTCASES

S.no	Test Case	Excepted Result	Result	Remarks(IF Fails)
1	User Register	If User registration successfully.	Pass	If already user email exist then it fails.
2	User Login	If Username and password is correct then it will getting valid page.	Pass	Un Register Users will not logged in.
3	Random forest and svm	The request will be accepted by the random forest and svm	Pass	The request will be accepted by the random forest and svm otherwise its failed
4	Decision Tree and multilayer perceptron	The request will be accepted by the Decision Tree and multilayer perceptron	Pass	The request will be accepted by the Decision Tree and multilayer perceptron otherwise its failed
5	Naive Bayes and k-nearest neighbour	The request will be accepted by the Naive Bayes and k-nearest neighbour	Pass	The request will be accepted by the Naive Bayes and k-nearest neighbour otherwise its failed
6	View dataset by user	Data set will be displayed by the user	Pass	Results not true failed
7	User classification	Display reviews with true results	Pass	Results not true failed

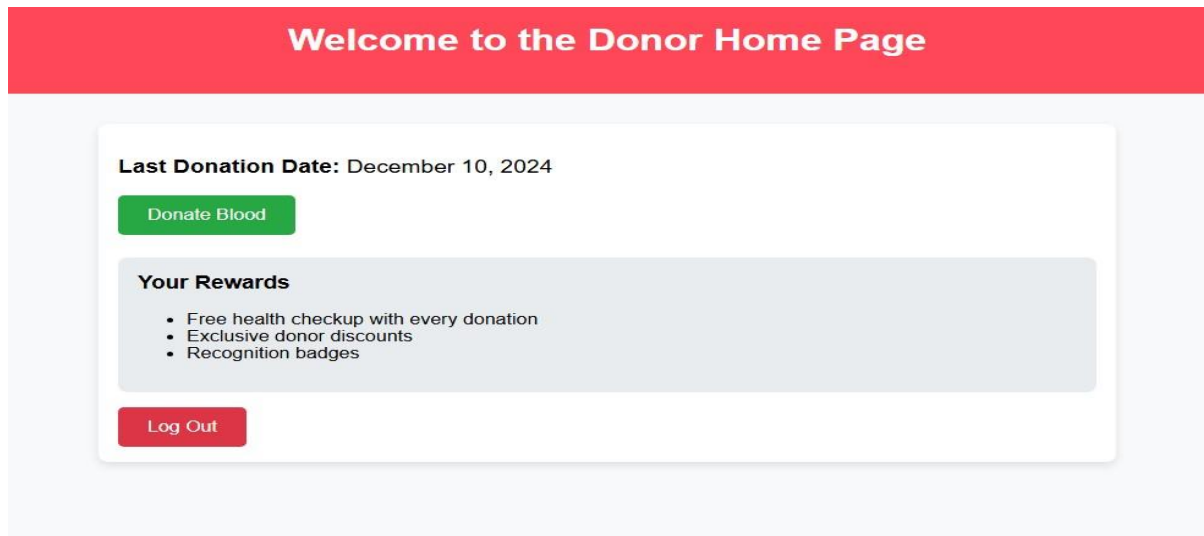
OUTPUT SCREENS



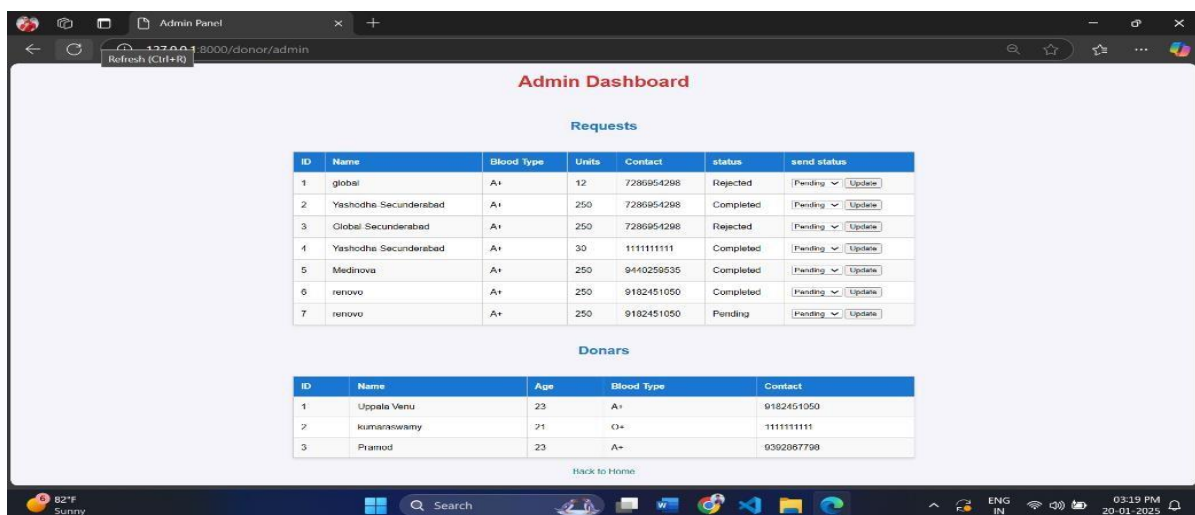
10.1: LOGIN PAGE OUTPUT SCREEN



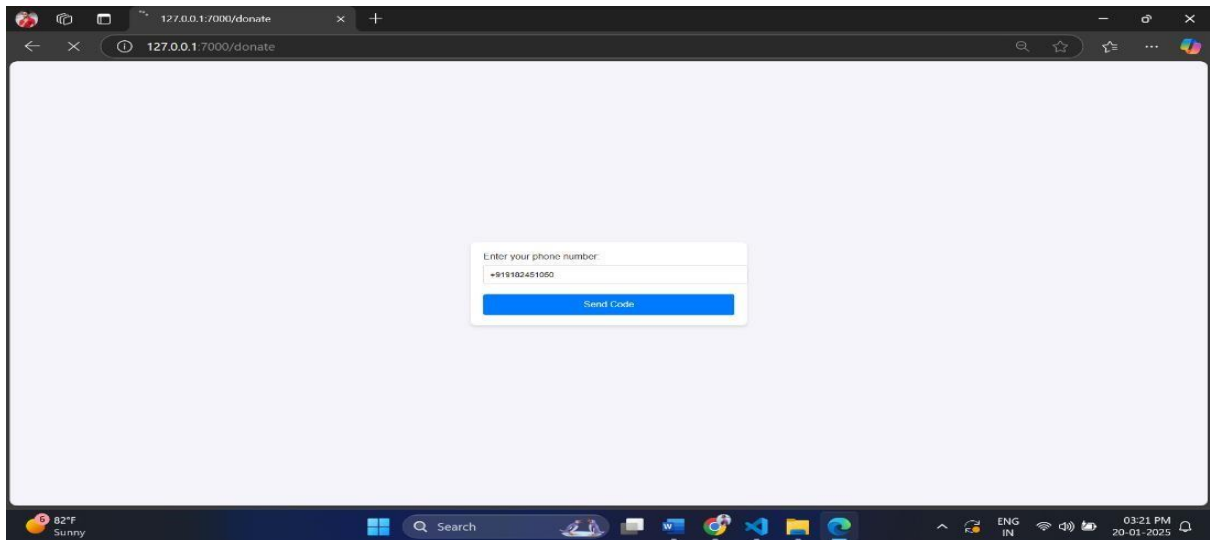
10.2: REGISTER FORM OUTPUT SCREEN



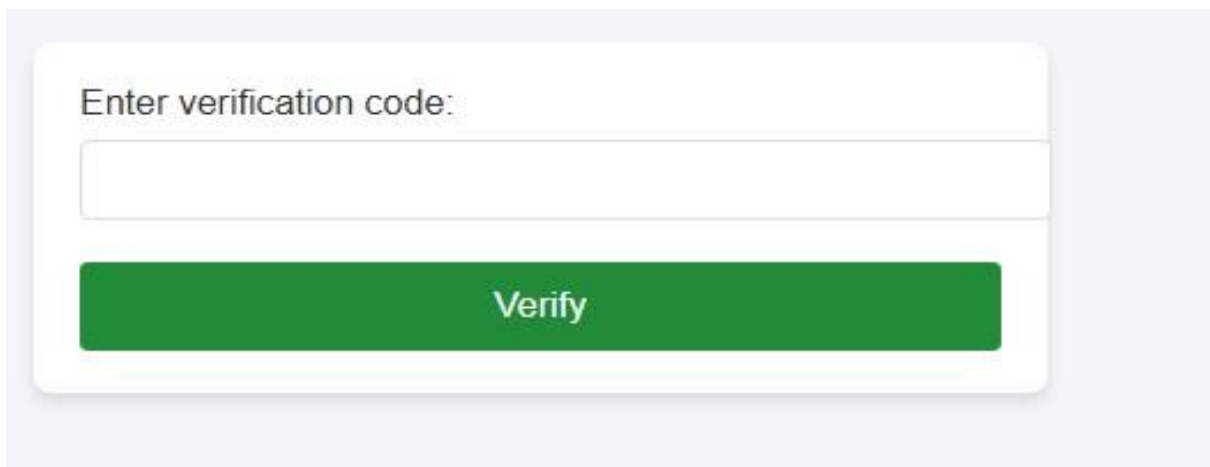
10.3: INDEX PAGE OUTPUT SCREEN



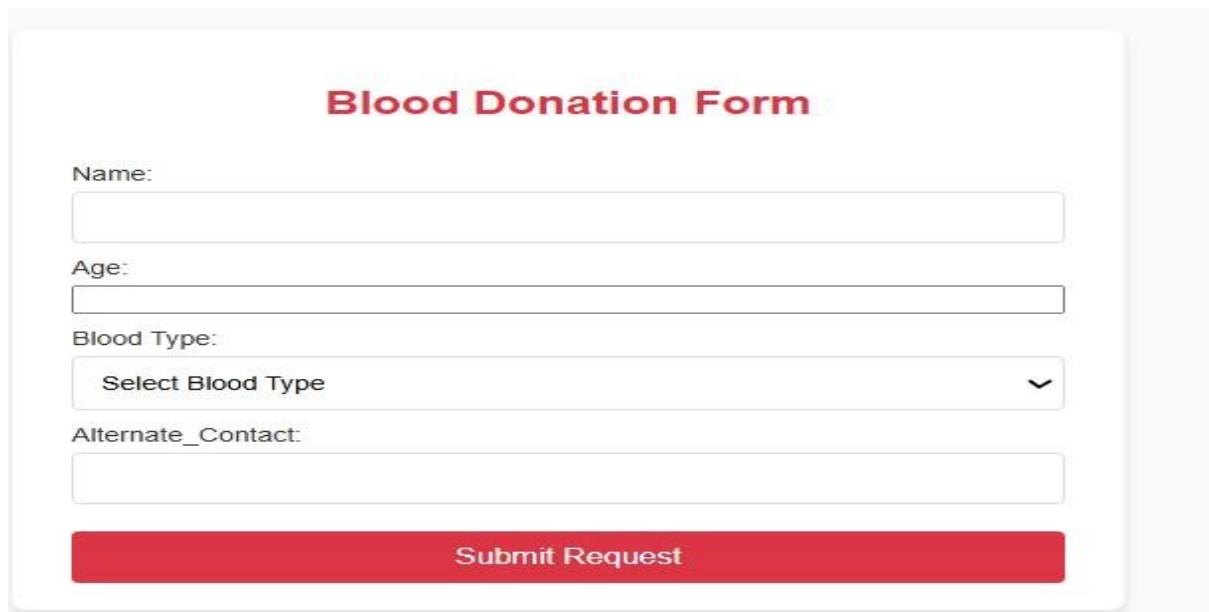
10.4: ADMIN DASHBOARD OUTPUT SCREEN



10.5: OTP AUTHENTICATION OUTPUT SCREEN



10.6: VERIFICATION OUTPUT SCREEN



Blood Donation Form

Name:

Age:

Blood Type:

Alternate_Contact:

Submit Request

10.7: DONATION OUTPUT SCREEN



Request Blood

Hospital_Name-Branch:

Blood Group:

Units:

patient-ref-Contact:

Submit Request

10.8: REQUEST FORM OUTPUT SCREEN

CONCLUSION

The Blood Bank Management System is a transformative solution designed to address the critical challenges in blood donation and transfusion services. It centralizes and automates processes, enabling accurate tracking of blood units, efficient donor management, and seamless coordination between blood banks, hospitals, and recipients. The system ensures compliance with safety standards through meticulous screening and reduces human errors through automated processes. With features such as real-time inventory monitoring, expiry tracking, and donor eligibility verification, the system minimizes wastage and ensures timely availability of blood. It fosters transparency by providing detailed records of donations, distributions, and usage, building trust among stakeholders.

The scalability of the system allows integration with advanced technologies like AI for demand prediction, machine learning for donor behavior analysis, and blockchain for secure, tamper-proof records. Mobile app integration empowers donors with notifications for blood drives, eligibility updates, and the ability to schedule donations conveniently.

In conclusion, the Blood Bank Management System not only optimizes operations but also enhances the reliability and accessibility of life-saving resources, paving the way for a more efficient and connected healthcare ecosystem. It has the potential to save countless lives and create a robust, sustainable framework for blood donation and distribution worldwide.

REFERENCES

- PATEL, P., & PATIL, A. (2019). "A REVIEW ON BLOOD BANK MANAGEMENT SYSTEM AND ITS APPLICATIONS." *INTERNATIONAL JOURNAL OF ENGINEERING AND TECHNOLOGY (IJET)*.
- GUPTA, R., & SINGH, D. (2021). "AUTOMATED BLOOD BANK MANAGEMENT SYSTEM: A SURVEY OF TECHNIQUES AND TECHNOLOGIES." *JOURNAL OF MEDICAL SYSTEMS*.
- NATARAJAN, R., & SATHYADEVI, G. (2020). "BLOCKCHAIN-BASED BLOOD BANK MANAGEMENT SYSTEM." *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS (IJACSA)*.
- PRAKASH, A. (2020). *INFORMATION SYSTEMS FOR HEALTHCARE AND BLOOD MANAGEMENT*. ELSEVIER.
- GUPTA, M., & AGARWAL, A. (2018). *HEALTHCARE INFORMATICS: A PRACTICAL APPROACH FOR STUDENTS AND PRACTITIONERS*. SPRINGER.