Bangladesh University of Business & Technology

Project Report

Project Name: *Blood Bank Management System*

SUBMITTED BY:

Utsho Roy(22235103261)

Md Rahad(22235103285)

Faiza Hasan(22235103280)

Afrin Akter Achol(22235103358)

Nur Mohammed Simanto(22235103656)

SUPERVISED BY:

Sumona Yeasmin

Lecturer, Dept. of CSE

Bangladesh University of Business & Technology

Submission Date: 01-01-2025

Abstract

The Blood Bank Management System is a desktop application designed to streamline the management of blood donations, donor details, and blood stock. Built using Python, Tkinter for the graphical interface, and MySQL for backend database management, the system automates crucial operations such as blood requests, donor registration, and blood stock tracking. It serves three main user roles: donors, administrators, and requesters. The system provides an intuitive interface for managing user accounts, updating blood stock, tracking donation records, and processing blood requests. By centralizing the data management and automating workflows, this system enhances efficiency, reduces errors, and improves the response time for blood requests. The project aims to provide an efficient solution for blood banks, ensuring faster and reliable blood supply.

Background

Blood banks play a critical role in the healthcare system by providing essential blood supplies to patients in need. However, traditional methods of managing blood stocks and donor records often result in inefficiencies, delays in response times, and difficulties in tracking donor history. In many cases, blood banks still rely on manual record-keeping or outdated software systems that lack integration with real-time data updates.

This project aims to address these challenges by automating and centralizing the process through a Blood Bank Management System. The system is designed to manage the entire lifecycle of blood donations, from donor registration and blood donation tracking to blood request processing. By incorporating Object-Oriented Programming (OOP) principles, Python, Tkinter for the user interface, and MySQL for database management, the system ensures data accuracy, security, and easy access to vital information. This system will facilitate faster and more accurate blood requests and donations, leading to improved blood bank operations and better healthcare outcomes.

Table of Contents

Chapter 1
Introduction

1.1 Introduction

The Blood Bank Management System is a desktop-based application aimed at automating and streamlining blood donation processes. Built using Python and incorporating object-oriented programming (OOP) concepts, the system allows users (donors, administrators, and requesters) to interact with a centralized database for managing blood stock, donor details, blood donation records, and blood requests.

This system aims to simplify the process of requesting, donating, and managing blood stock efficiently in blood banks. It will be developed using Python and Tkinter for the GUI, with MySQL as the backend database.

1.2 Objectives

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

- To enable users to create accounts, request blood, and view their request statuses.
- To provide administrators with features for managing donor records, tracking blood donations, and updating the blood stock.
- To facilitate efficient handling of blood requests, donor suggestions in case of unavailability, and stock updates when requests are approved.
- To provide a comprehensive database management system for storing user, donor, and blood stock data, ensuring reliability and scalability.

1.3 System Objectives

- **User Management**: Provide login and registration for two roles: *Admin* and *User*. Users can request blood, and admins can manage requests and donations.
- **Donor Management**: Admins can add, update, and view donors.

- **Blood Stock Management**: Track available blood units by blood group and adjust stock when donations are made or blood is provided.
- **Blood Requests**: Users can request blood, and the system will check for available stock. If unavailable, the system will suggest nearby donors, and if available, the request is marked as “Processing.”

- **Request Approval/Rejection**: Admins can approve or reject requests, and the stock is updated accordingly.

1.4 Methodology

The methodology for this project follows the **Waterfall Model** due to its structured and sequential phases. The phases are:

- **Requirement Analysis**: Collecting requirements for features such as blood requests, donor management, and admin control.
- **System Design**: Designing the application architecture and database.
- **Implementation**: Writing code in Python, using OOP concepts for modularity and maintainability.
- **Database Design**: Designing the schema using MySQL for data storage and retrieval.
- **Testing**: Testing the system for bugs, data consistency, and ensuring that all features work as expected.
- **Deployment**: Releasing the desktop application.

1.5 Techniques

- **Object-Oriented Programming (OOP)**: The system is structured using classes and objects to encapsulate features such as user roles, blood requests, and donor information.
- **GUI Development**: The Tkinter library is used to create a graphical interface for both users and admins.
- **Database Manipulation**: The system uses MySQL to store and manage data. SQL queries are used for CRUD (Create, Read, Update, Delete) operations.
- **Validation and Error Handling**: Input validation (e.g., ensuring that units entered are numeric) and error handling for database operations will be incorporated.

Chapter 2
Analysis

2.1 Existing System Analysis

Most blood banks rely on traditional methods for managing data, such as manual record-keeping or isolated software systems that lack integration. This leads to inefficiencies, including:

- Delayed response times for blood requests.
- Inconsistent data management, with no central repository.
- Lack of real-time updates in blood stock and donor availability.
- Difficulty in tracking donor history or sending timely reminders for future donations.

Our proposed system addresses these challenges by automating and centralizing the entire process, ensuring real-time availability and accessibility of blood bank data.

2.2 Feasibility Study

1. **Technical Feasibility**
    - The system uses robust technologies such as Python and MySQL.
    - Compatibility with desktop platforms ensures ease of deployment.
2. **Operational Feasibility**
    - Intuitive GUI design enables smooth navigation for users and admins.
    - Predefined roles (Admin, User) simplify task allocation and permissions.
3. **Economic Feasibility**
    - Open-source tools reduce development costs.
    - Minimal hardware requirements ensure affordability for small blood banks.

Chapter 3
Used Software and Programming Language

3.1 IntelliJ IDEA

- A versatile IDE that supports Python development with features like debugging, version control, and intelligent code completion.
- Simplifies integration with MySQL and GUI libraries.

3.2 Python Programming Language

- Python is the core language for development due to its simplicity and vast libraries.
- Key Libraries Used:
  - **Tkinter:** For GUI development.
  - **MySQL Connector:** For database operations.
  - **Datetime:** For handling date and time operations.

3.3 MySQL

- Serves as the backend database to store structured information.
- Features like indexing and foreign keys ensure data integrity and faster querying.

3.4 Additional Tools

- **XAMPP:** To host the MySQL database locally.

Chapter 4
Code Structure and Purpose

4.1 Code Structure

The application is modularized into the following components:

- **User Management Module:** Handles login, signup, and role-based access.
- **Donor Management Module:** Enables adding, viewing, and filtering donor details.
- **Blood Stock Module:** Displays and updates the stock of blood units based on donations and requests.
- **Donation Module:** Records donation details and updates stock.
- **Blood Request Module:** Processes user requests and manages approval or rejection.
- **Utilities Module:** Includes auto-suggestion for nearest doner, error handling, and input validation.

4.2 Purpose

- **Modular Code Design:** To enhance maintainability and scalability.
- **Efficient Data Handling:** Ensures accuracy and reduces redundancy in database operations.
- **Seamless Integration:** Allows smooth communication between the GUI and the database.
- **Error-Free Processing:** Incorporates comprehensive validation and exception handling mechanisms.

Chapter 5
Database Design

5.1 User Table

- **Id:** Primary key for user identification.
- **Username:** Unique username for login.
- **Password:** Password for authentication.

SQL:

```
CREATE TABLE User (
    Id INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(100) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL
 );
```

5.2 Admin Table

- **Id**: Primary key for user identification.
- **Username**: Unique username for login.
- **Password**: Password for authentication

SQL:

```
CREATE TABLE Admin (
    Id INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(100) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL
 );
```

5.3 Doner Table

- **Id**: Primary key for identifying each donor.
- **Name**: Donor's name.
- **Age**: Donor’s age.
- **BloodGroup**: Blood group of the donor.
- **City**: City where the donor resides.
- **Number**: Contact number of the donor.

SQL:

```
CREATE TABLE Doner (
     Id INT AUTO_INCREMENT PRIMARY KEY,
     Name VARCHAR(100) NOT NULL,
     Age INT NOT NULL,
     BloodGroup VARCHAR(10) NOT NULL,
     City VARCHAR(100) NOT NULL,
     Number VARCHAR(15) NOT NULL
);
```

5.4 Blood Stock Table

- **BloodGroup**: Blood group type.
- **Unit**: Available units of blood in stock.

SQL:

```
CREATE TABLE BloodStock (
  BloodGroup VARCHAR(10) PRIMARY KEY,
  Unit INT NOT NULL
);
```

5.5 Donation Table

- **Serial**: Unique identifier for the donation.
- **DonerID**: Foreign key referencing the Doner table.
- **Date**: Date of donation.
- **Unit**: Number of blood units donated.

SQL:

```
    CREATE TABLE Donation (
      Serial INT AUTO_INCREMENT PRIMARY KEY,
      DonerID INT NOT NULL,
      Date DATE NOT NULL,
      Unit INT NOT NULL,
      FOREIGN KEY (DonerID) REFERENCES Doner(Id) ON DELETE
CASCADE
    );
```

5.6 Request_Blood Table

- **Id**: Unique identifier for each blood request.
- **Name**: Name of the requester.
- **BloodGroup**: Requested blood group.
- **Unit**: Number of blood units requested.
- **Reason**: Reason for the request.
- **City**: Requester's city.
- **Number**: Contact number of the requester.
- **Date**: Date of the request.
- **Status**: Status of the request (Processing, Accepted, Rejected).

SQL:

```
    CREATE TABLE request_blood (
      Id INT AUTO_INCREMENT PRIMARY KEY,
      Name VARCHAR(100) NOT NULL,
      BloodGroup VARCHAR(10) NOT NULL,
      Unit INT NOT NULL,
      Reason VARCHAR(255),
     City VARCHAR(100) NOT NULL,
     Number VARCHAR(15) NOT NULL,
     Date DATE NOT NULL,
     Status ENUM('Processing', 'Accepted', 'Rejected') NOT NULL
DEFAULT 'Processing'
  );
```

Chapter 6
Necessary Diagram

6.1 Feature Diagram

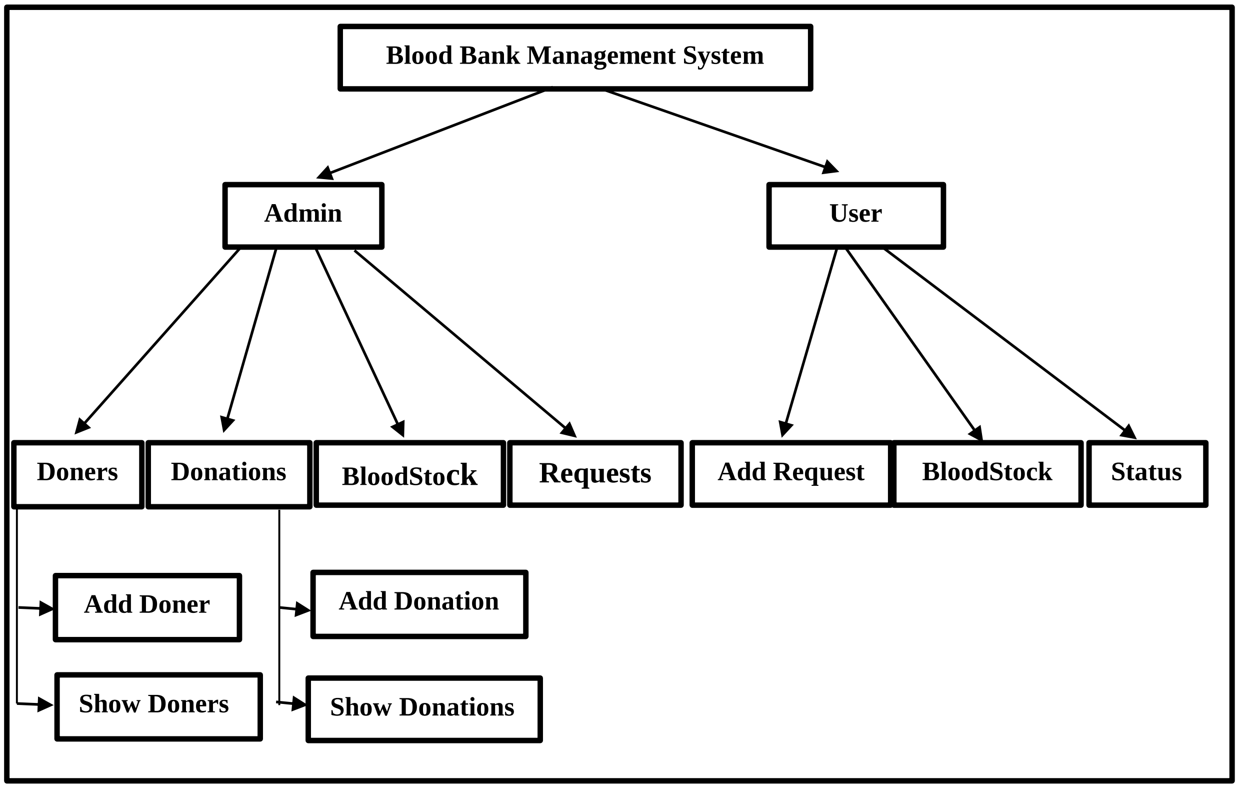


Figure 6.1: Feature Diagram of Blood Bank Management System

This diagram represents the hierarchical structure of the Blood Bank Management System, detailing the features accessible to both the Admin and User roles.

- **Admin Role:**
  - **Donors**:
    - *Add Donor*: Allows the admin to register new donors.
    - *Show Donors*: Displays the list of registered donors.
  - **Donations**:
    - *Add Donation*: Facilitates adding new blood donations.
    - *Show Donations*: Displays the history of blood donations.
  - **BloodStock**: Displays the available blood stock for all blood groups.
  - **Requests**: Manages patient blood requests, including approvals or denials.
- **User Role:**
  - **Add Request**: Enables users to request blood based on their requirements.
  - **BloodStock**: Allows users to view available blood stock for each blood group.
  - **Status**: Displays the status of blood requests (e.g., pending, approved, denied).

6.2 Database ER Diagram

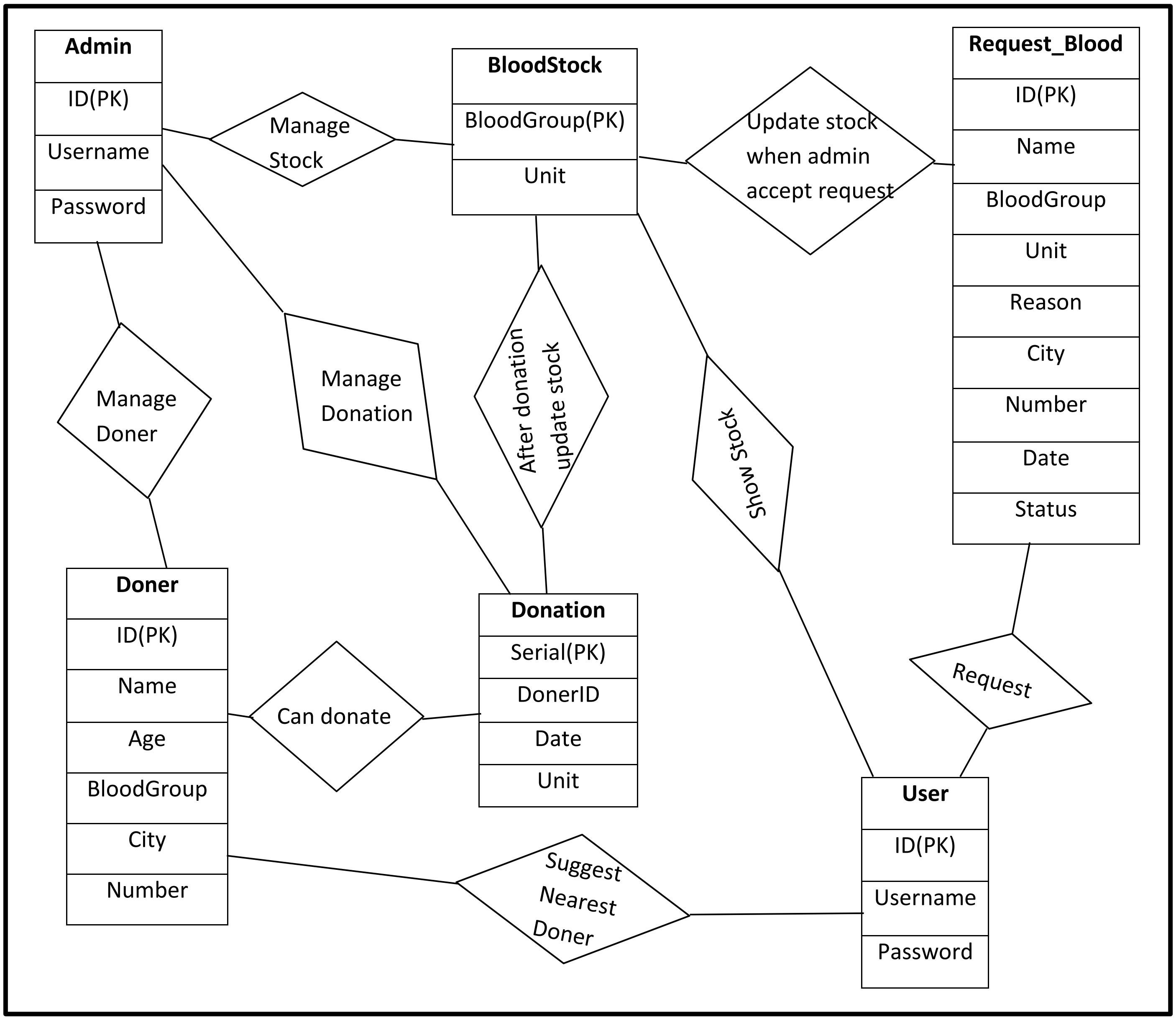


Figure 6.2: Database ER Diagram for Blood Bank Management System

This Entity-Relationship (ER) diagram outlines the relationships and structure of the database entities for the Blood Bank Management System, highlighting the roles of Admin and User, and their interactions with core system components.

Chapter 7
Snapshots of The Project

7.1 LogIn and SignUp Page

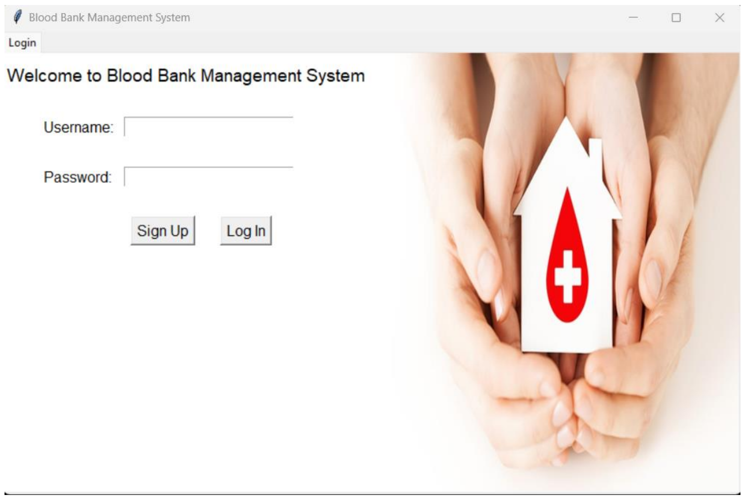

Blood Bank Management System
Login
Welcome to Blood Bank Management System
Username:
Password:
Sign Up
Log In

Figure 7.1: Admin and User Sign-Up and Log-In Page
This snapshot illustrates the form used by administrators to create an account. Fields include username, password, and confirmation then Login.

7.1.1 Sign Up

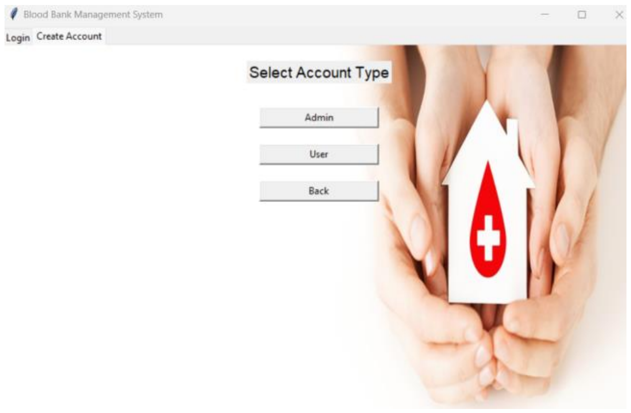
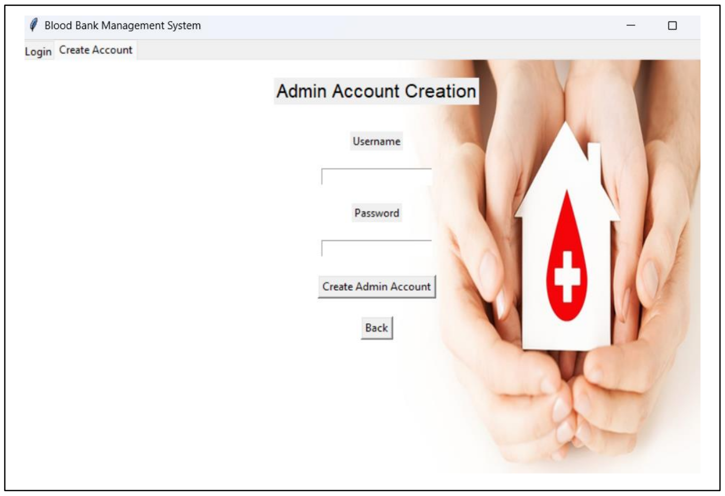
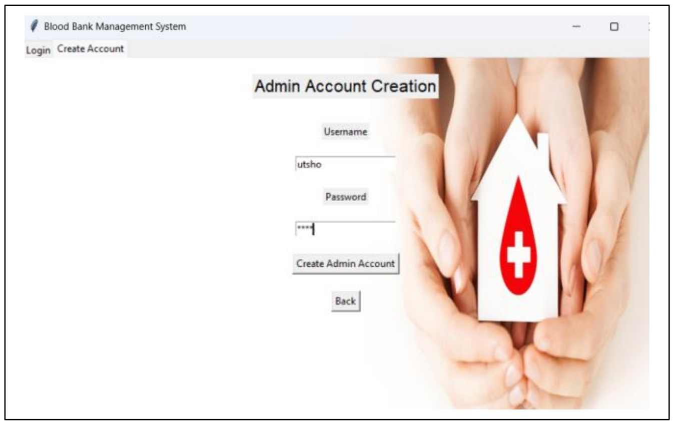


Figure 7.1.1: Sign-Up Page
This snapshot show select account type like Admin and User.

7.1.2 Create Admin account

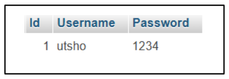




utsho

****

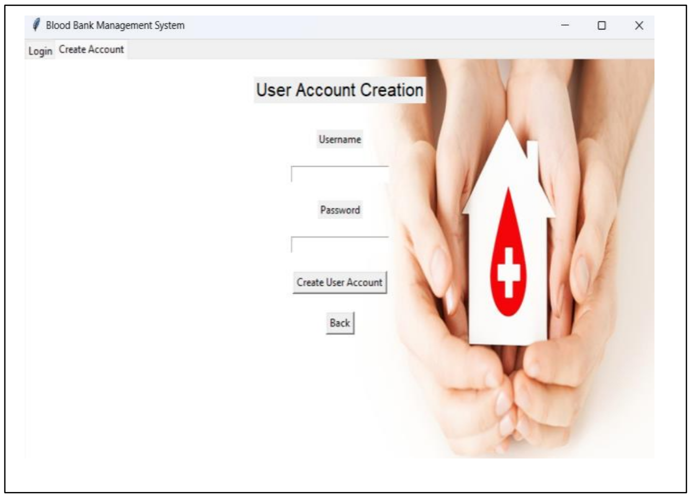
Database View

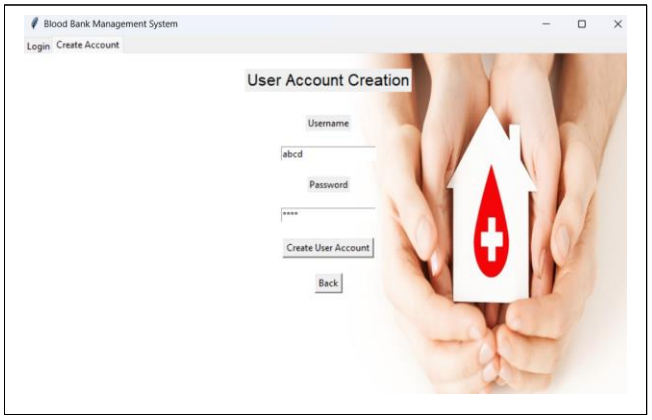


Id	Username	Password
1	utsho	1234

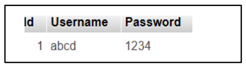
Figure 7.1.2: Admin Sign-Up Page
This snapshot illustrates the process of creating a admin account. It includes fields for entering a username and password, allowing new users to register. Once the account is created, users can log in to access the system.

7.1.3 Create User Account





Database View



Id	Username	Password
1	abcd	1234

Figure 7.1.3: User Sign-Up Page

This snapshot illustrates the process of creating a user account. It includes fields for entering a username and password, allowing new users to register. Once the account is created, users can log in to access the system.

7.2 LogIn Admin Account

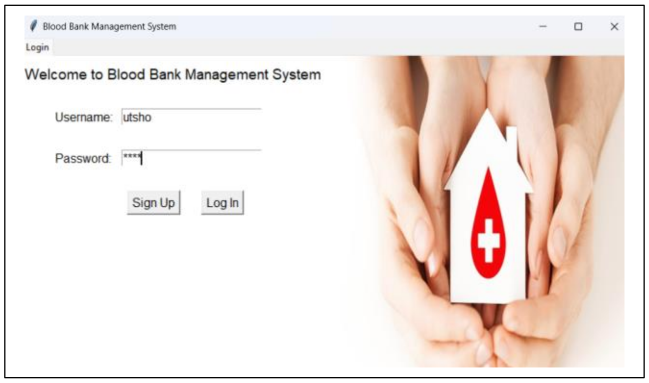
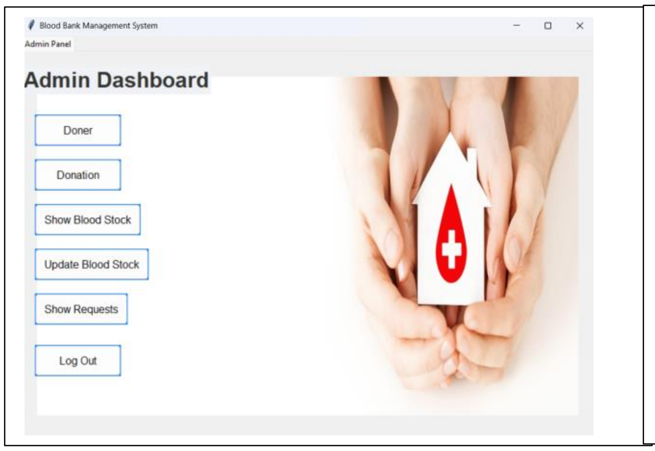

Blood Bank Management System
Login
Welcome to Blood Bank Management System
Username: utsho
Password: ****
Sign Up
Log In

Figure 7.2: LogIn Admin Account
This snapshot displays the login interface for both admins and users, enabling them to access their respective panels by entering valid credentials.

7.2.1 Admin Panel

When click Doner and Donation Button:



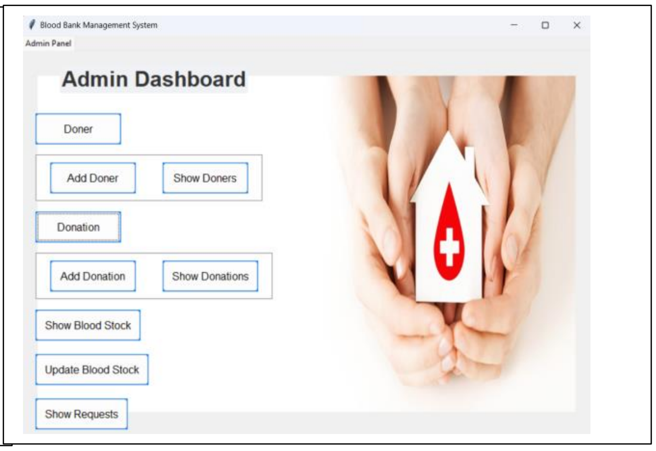
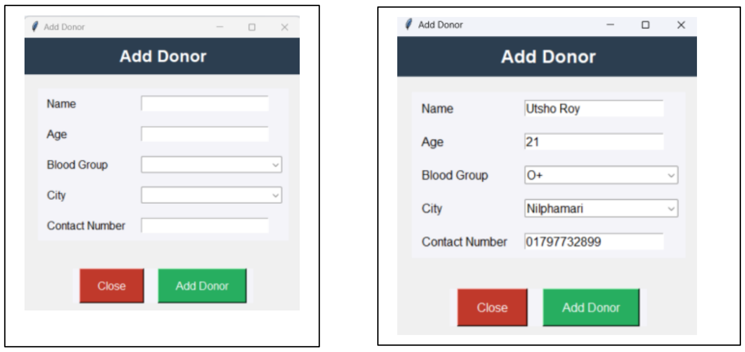


Figure 7.2.1: Admin Panel
This snapshot illustrates the admin panel, where administrators can manage donors, blood donations, blood stock, and blood requests. The panel provides tabs for easy navigation and efficient system control

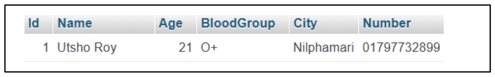
7.2.1.1 Add Doner



Name	
Age	
Blood Group	
City	
Contact Number	

Name	Utsho Roy
Age	21
Blood Group	O+
City	Nilphamari
Contact Number	01797732899

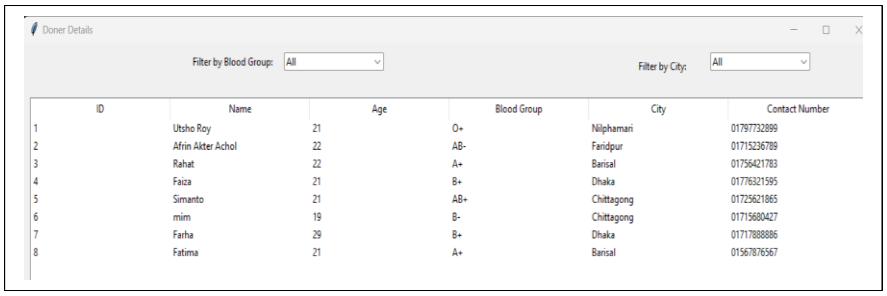
Database View



Id	Name	Age	BloodGroup	City	Number
1	Utsho Roy	21	O+	Nilphamari	01797732899

Figure 7.2.1.1: Add Doner
This snapshot illustrates the admin panel, where administrators can add donors.

7.2.1.2 Show Doners

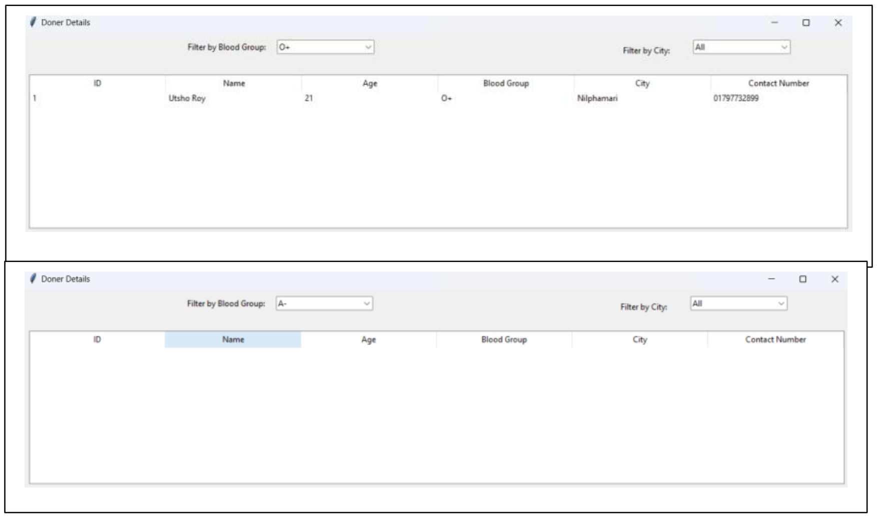


Filter by Blood Group: All
Filter by City: All

ID	Name	Age	Blood Group	City	Contact Number
1	Utsho Roy	21	O+	Nilphamari	01797732899
2	Afrin Akter Achol	22	AB-	Faridpur	01715236789
3	Rahat	22	A+	Barisal	01756421783
4	Faiza	21	B+	Dhaka	01776321595
5	Simanto	21	AB+	Chittagong	01725621865
6	mim	19	B-	Chittagong	01715680427
7	Farha	29	B+	Dhaka	01717888886
8	Fatima	21	A+	Barisal	01567876567

Figure 7.2.1.2: Show Doners
This snapshot illustrates the admin panel, where administrators can show donors.

7.2.1.2.1 Filter By Blood Group



Filter by Blood Group: O+
Filter by City: All

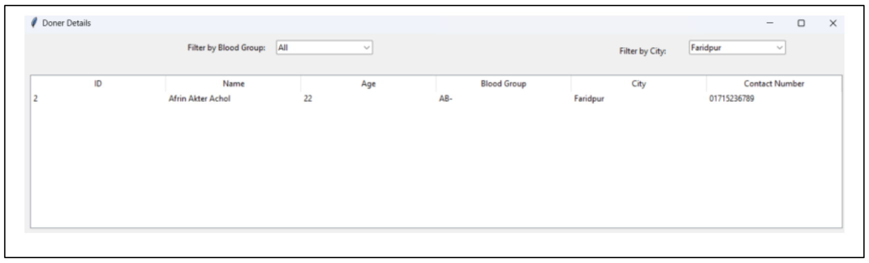
ID	Name	Age	Blood Group	City	Contact Number
1	Utsho Roy	21	O+	Nilphamari	01797732899

Filter by Blood Group: A-
Filter by City: All

ID	Name	Age	Blood Group	City	Contact Number

Figure 7.2.1.2.1 Filter By Blood Group
This snapshot shows the donor listing interface with a filter to view donors based on their blood group.

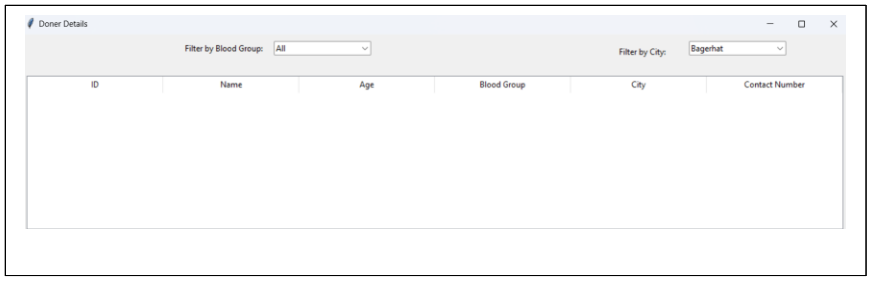
7.2.1.2.2 Filter By City


Doner Details

Filter by Blood Group: All

Filter by City: Faridpur

ID	Name	Age	Blood Group	City	Contact Number
2	Afrin Akter Achol	22	AB-	Faridpur	01715236789


Doner Details

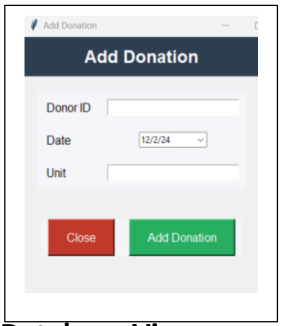
Filter by Blood Group: All

Filter by City: Bagerhat

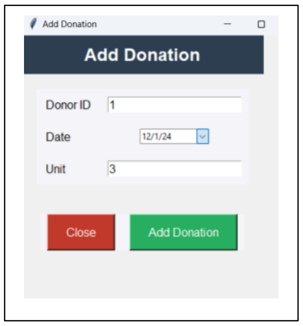
ID	Name	Age	Blood Group	City	Contact Number

Figure 7.2.1.2.2 Filter By City
This figure demonstrates the donor listing interface where admins can filter donors by city.

7.2.1.3 Add Donation

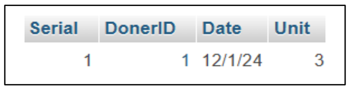

Add Donation

Donor ID	
Date	12/2/24
Unit	


Add Donation

Donor ID	1
Date	12/1/24
Unit	3

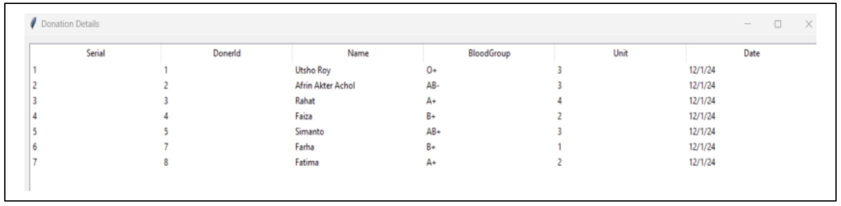
Database View



Serial	DonerID	Date	Unit
1	1	12/1/24	3

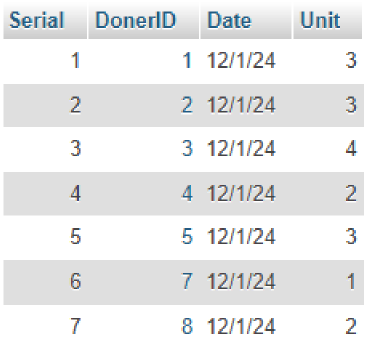
Figure 7.2.1.3: Add Donation Page
This snapshot illustrates the form used by admins to record blood donations, including details like donor ID, date, and quantity.

7.2.1.4 Show Donations



Serial	DonerId	Name	BloodGroup	Unit	Date
1	1	Utsho Roy	O+	3	12/1/24
2	2	Afrin Akter Achol	AB-	3	12/1/24
3	3	Rahat	A+	4	12/1/24
4	4	Faiza	B+	2	12/1/24
5	5	Simanto	AB+	3	12/1/24
6	7	Farha	B+	1	12/1/24
7	8	Fatima	A+	2	12/1/24

Database View

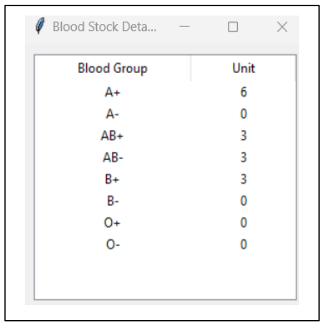


Serial	DonerID	Date	Unit
1	1	12/1/24	3
2	2	12/1/24	3
3	3	12/1/24	4
4	4	12/1/24	2
5	5	12/1/24	3
6	7	12/1/24	1
7	8	12/1/24	2

Figure 7.2.1.4 : Show Donations Page

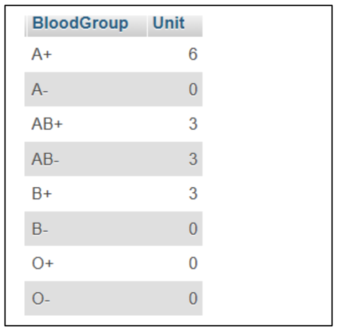
The donations listing interface displays all donation records, including donor details and the quantity of blood donated.

7.2.1.5 Show Blood Stock



Blood Group	Unit
A+	6
A-	0
AB+	3
AB-	3
B+	3
B-	0
O+	0
O-	0

Database View



BloodGroup	Unit
A+	6
A-	0
AB+	3
AB-	3
B+	3
B-	0
O+	0
O-	0

Figure 7.2.1.5: Show Blood Stock Page

This snapshot displays the blood stock page, where admins can monitor the available units for each blood group.

7.2.1.6 Update Stock

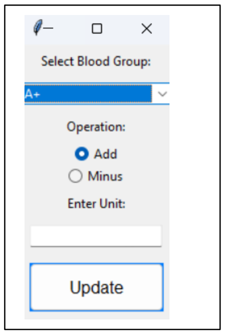
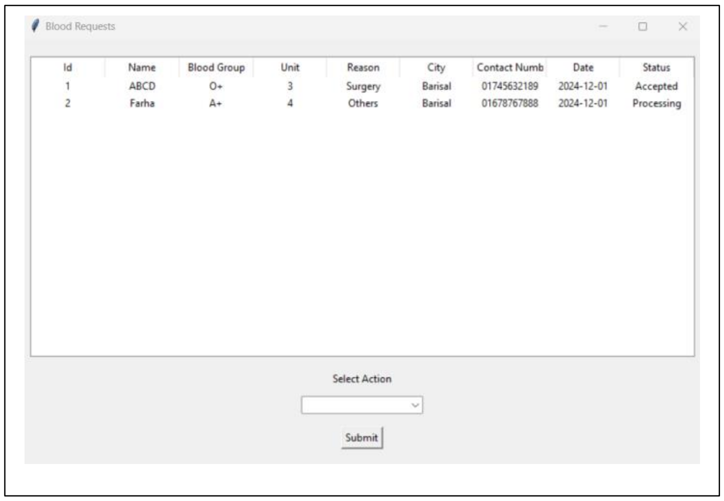


Figure 7.2.1.6 Update Stock

This snapshot displays the update stock page, where admins can update the available units.

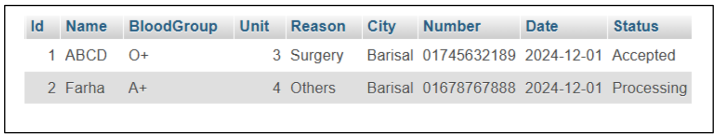
7.2.1.7 Show Requests



Blood Requests

Id	Name	Blood Group	Unit	Reason	City	Contact Numb	Date	Status
1	ABCD	O+	3	Surgery	Barisal	01745632189	2024-12-01	Accepted
2	Farha	A+	4	Others	Barisal	01678767888	2024-12-01	Processing

Database View



Id	Name	BloodGroup	Unit	Reason	City	Number	Date	Status
1	ABCD	O+	3	Surgery	Barisal	01745632189	2024-12-01	Accepted
2	Farha	A+	4	Others	Barisal	01678767888	2024-12-01	Processing

Figure 7.2.1.7: Show Requests Page

This figure shows the blood requests page where admins can review, approve, or reject blood requests submitted by users.

7.3 LogIn User Account

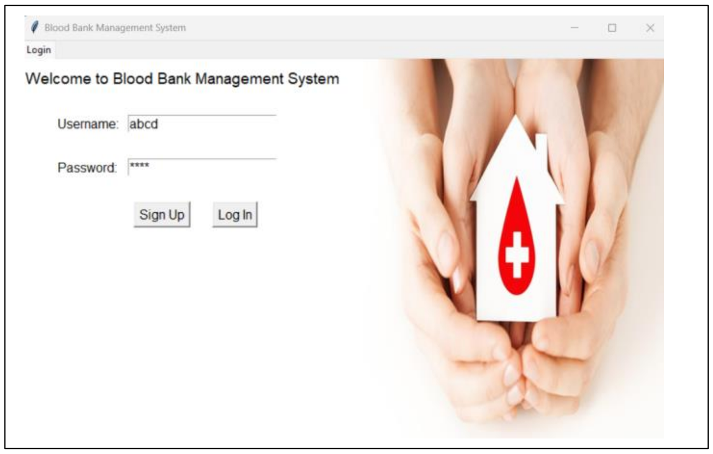

Blood Bank Management System
Login
Welcome to Blood Bank Management System
Username: abcd
Password: ****
Sign Up
Log In

Figure 7.3: LogIn User Account page
This snapshot displays the login interface for both admins and users, enabling them to access their respective panels by entering valid credentials.

7.3.1 User Panel

When click Request Blood Button:

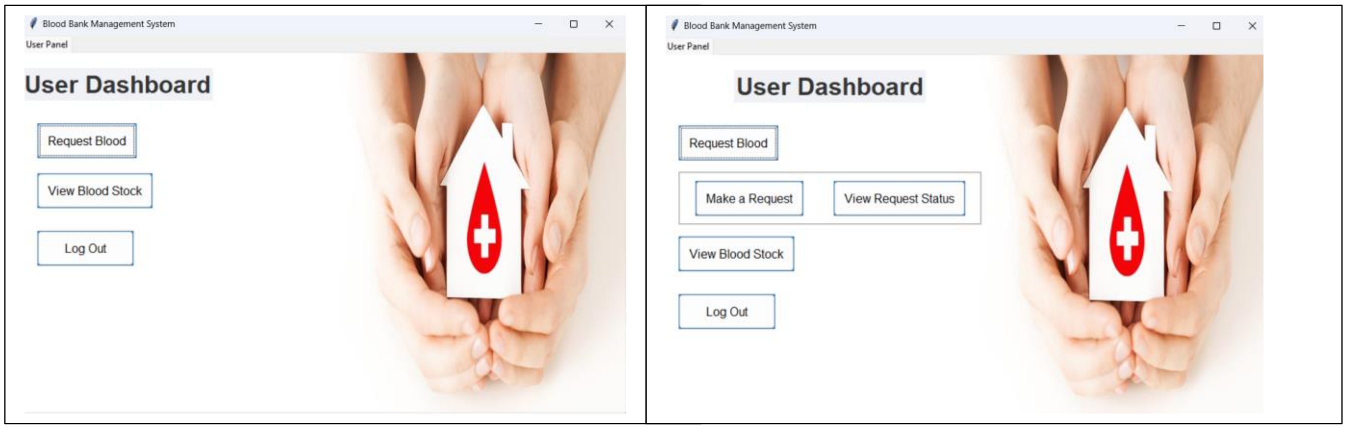
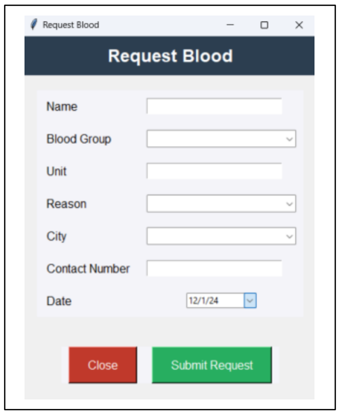
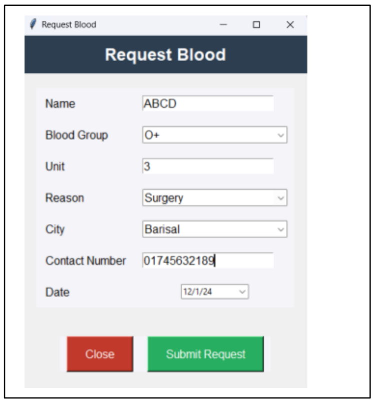


Figure 7.3.1: User Panel
This snapshot displays User Dashboard where user request for blood, show blood stock, status.

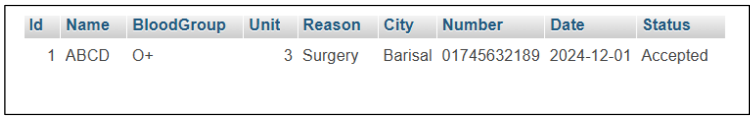
7.3.1.1 Request Blood





Name	ABCD
Blood Group	O+
Unit	3
Reason	Surgery
City	Barisal
Contact Number	01745632189
Date	12/1/24

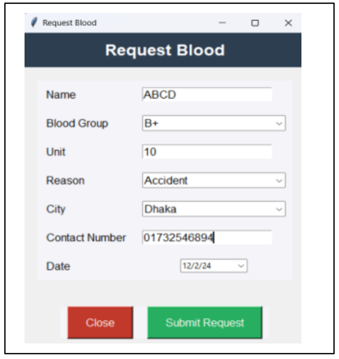
Database View



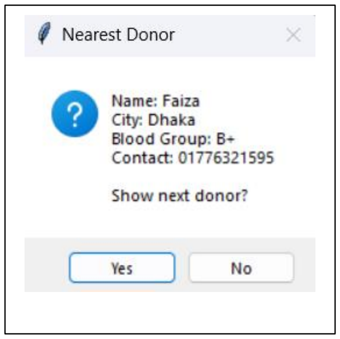
Id	Name	BloodGroup	Unit	Reason	City	Number	Date	Status
1	ABCD	O+	3	Surgery	Barisal	01745632189	2024-12-01	Accepted

Figure 7.3.1.1: Request Blood page
This snapshot shows the form available in the user panel, allowing users to request blood by specifying the blood group, quantity, reason, and contact details.

7.3.1.2 Suggest Nearest Doners

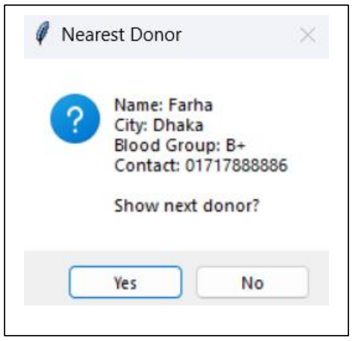


Name	ABCD
Blood Group	B+
Unit	10
Reason	Accident
City	Dhaka
Contact Number	01732546894
Date	12/2/24



Name: Faiza
City: Dhaka
Blood Group: B+
Contact: 01776321595

Show next donor?

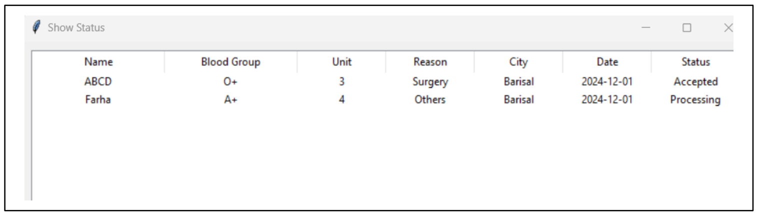


Name: Farha
City: Dhaka
Blood Group: B+
Contact: 01717888886

Show next donor?

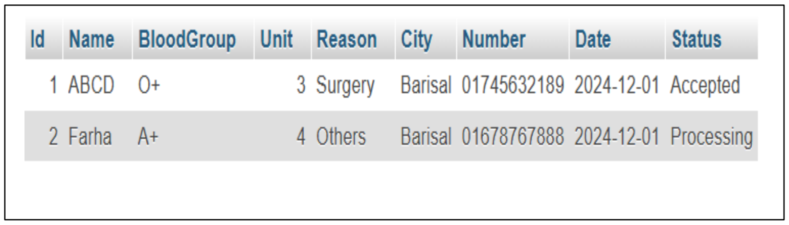
Figure 7.3.1.2: Nearest Donor Suggestions
This figure illustrates the system's response when the requested blood is unavailable, suggesting the nearest donors based on the selected blood group.

7.3.1.3 Show Status



Name	Blood Group	Unit	Reason	City	Date	Status
ABCD	O+	3	Surgery	Barisal	2024-12-01	Accepted
Farha	A+	4	Others	Barisal	2024-12-01	Processing

Database View



Id	Name	BloodGroup	Unit	Reason	City	Number	Date	Status
1	ABCD	O+	3	Surgery	Barisal	01745632189	2024-12-01	Accepted
2	Farha	A+	4	Others	Barisal	01678767888	2024-12-01	Processing

Figure 7.16: Show Request Status
This snapshot displays the status of blood requests submitted by the user, indicating whether the request is being processed, accepted, or rejected.

Chapter 8
Limitations, Future Work & Conclusion

8.1 Limitations

1. **Geographic Restriction**
   - The system currently supports only predefined districts in Bangladesh.
   - No provision for international usage.
2. **Limited Scalability**
   - Designed for single-location use, making multi-branch integration complex.
3. **Lack of Mobile Application**
   - No mobile version for easier user accessibility.

8.2 Future Work

1. **Mobile Application Development:**
   - Create a cross-platform mobile app for better user engagement.
2. **Cloud Integration:**
   - Transition to cloud-based databases for better scalability and real-time synchronization across branches.
3. **Advanced Analytics:**
   - Integrate AI for predicting blood demand based on historical data.
4. **Multilingual Support:**
   - Add language options for diverse user bases.
5. **Geo-Mapping for Donor Search:**
   - Include interactive maps to locate donors using geospatial data.

8.3 Conclusion

The Blood Bank Management System has been meticulously designed and implemented to address the inefficiencies of traditional blood donation processes. Through the use of Python, MySQL, and Tkinter, the system integrates modern programming techniques with an intuitive graphical user interface to provide a robust platform for managing donors, blood stock, and blood requests.

This project successfully achieves its objectives by offering features such as donor management, blood stock tracking, donation records, and efficient handling of blood requests. Admins can effectively manage blood bank operations, including updating stock levels and addressing user requests, while users benefit from a seamless process for blood requests and donor recommendations. Furthermore, the integration of geographical data for suggesting nearest donors ensures that even in situations of unavailability, alternative options are readily accessible, significantly enhancing response times.

The system not only simplifies operations for blood banks but also ensures data accuracy and scalability, making it adaptable for future enhancements. By automating key processes, it minimizes human errors, reduces delays, and promotes better coordination between blood banks and the community.

In its current state, the Blood Bank Management System demonstrates how technology can be leveraged to address critical public health needs. Future expansions, such as the integration of mobile platforms, enhanced donor prioritization algorithms, and advanced analytics for predicting blood demand, can further elevate its utility and impact.

In conclusion, this project serves as a valuable tool for modernizing blood bank operations, fostering a culture of efficiency, reliability, and community support. It exemplifies the potential of technology to bridge gaps in healthcare services and ensure the availability of life-saving resources when and where they are needed most.

Appendix

1. Database Connection

```
class Database:
    @classmethod
    def connect_to_database(cls):
        return mysql.connector.connect(host="localhost", user="root", password="",
database="bloodbank")
```

2. Donor Management

Add Donor:

```
class Doner:
    def add_doner(self, notebook):
        # Form to add donor details
        doner_window = tk.Toplevel()
        doner_window.title("Add Donor")
        doner_window.geometry("400x400")
        # Fields include Name, Age, Blood Group, City, and Contact Number
        add_button = tk.Button(
            doner_window, text="Add Donor",
            command=lambda: self.validate_doner_data(
                name_entry.get(), age_entry.get(), blood_group_combobox.get(),
                city_combobox.get(), number_entry.get(), doner_window
            )
        )
```

Validate and Save Donor:

```
def validate_doner_data(self, name, age, bloodgroup, city, number, doner_window):
    if not name or not age or not bloodgroup or not city or not number:
        messagebox.showerror("Error", "All fields are required.")
    else:
        self.add_doner_to_db(name, age, bloodgroup, city, number)
        self.close_doner_window(doner_window)
```

3. **Blood Donation Management**

Add Donation:

```
class Donation:
   def add_donation_to_db(self, doner_id, date, unit, donation_window):
      cursor.execute("SELECT BloodGroup FROM Doner WHERE Id = %s",
(doner_id,))
      blood_group = cursor.fetchone()
      if blood_group:
         cursor.execute("SELECT Unit FROM BloodStock WHERE BloodGroup = %s",
(blood_group[0],))
         existing_unit = cursor.fetchone()[0]
         new_unit = existing_unit + int(unit)
         cursor.execute("UPDATE BloodStock SET Unit = %s WHERE BloodGroup =
%s", (new_unit, blood_group[0]))
      db.commit()
```

4. **Blood Stock Management**

Show Blood Stock:

```
class BloodStock:
   def show_blood_stock(self):
      cursor.execute("SELECT * FROM BloodStock")
      rows = cursor.fetchall()
      tree = ttk.Treeview(stock_window, columns=("BloodGroup", "Unit"),
show="headings")
      tree.heading("BloodGroup", text="Blood Group")
      tree.heading("Unit", text="Unit")
      for row in rows:
         tree.insert("", tk.END, values=row)
```

5. **Blood Request Handling**

Submit Request:

```
class RequestBlood:
   def add_blood_request(self, name, bloodgroup, unit, reason, city, number, date,
request_window):
      cursor.execute("SELECT Unit FROM BloodStock WHERE BloodGroup = %s",
(bloodgroup,))
      result = cursor.fetchone()
      if result and result[0] >= int(unit):
         cursor.execute("INSERT INTO request_blood (Name, BloodGroup, Unit, ...)
VALUES (%s, %s, ...)")
      else:
         self.suggest_nearest_donor(bloodgroup)
```

Suggest Nearest Donor:

```
def suggest_nearest_donor(self, bloodgroup):
    cursor.execute("SELECT Name, City, Contact FROM Doner WHERE BloodGroup = 
%s", (bloodgroup,))
    donors = cursor.fetchall()
    if donors:
        for donor in donors:
            messagebox.showinfo("Nearest Donor", f"Name: {donor[0]}, City: {donor[1]}, 
Contact: {donor[2]}")
```

6. Authentication System

```
class Authenticator:
    def authenticate_user(self, username, password):
        cursor.execute("SELECT 'Admin' FROM Admin WHERE Username = %s AND 
Password = %s", (username, password))
        admin = cursor.fetchone()
        if admin:
            return 'Admin'
        cursor.execute("SELECT 'User' FROM User WHERE Username = %s AND 
Password = %s", (username, password))
        user = cursor.fetchone()
        return 'User' if user else None
```

7. Main Application Structure

```
if __name__ == "__main__":
    root = tk.Tk()
    root.geometry("800x500")
    notebook = ttk.Notebook(root)
    main_menu(root, notebook)
    root.mainloop()
```