**Project Title:** Blood Bank Management System

1. **Synopsis**

The Blood Bank Management System is an innovative web-based application designed to automate and simplify the management of blood donor records. Traditionally, blood banks have relied on manual processes for storing and managing donor information, which can be inefficient and prone to errors. This system offers a digital solution by leveraging Flask for backend development, MySQL for database management, and HTML/CSS for frontend design.

The application allows blood bank administrators to efficiently add, update, delete, and view donor records. By offering a centralized and organized database, the system eliminates the need for paper-based records and ensures that donor information is easily accessible and securely stored. The responsive and user-friendly interface enhances the experience for both administrators and staff, making it a practical tool for managing blood donation operations.

1. **Objectives**

The objectives of this project are:

1. **Digitization of Blood Donor Management:** To replace manual processes with a digital system, improving efficiency and reducing human errors in managing donor information.
2. **Accurate and Centralized Record Keeping:** To maintain accurate and up-to-date records of blood donors, including personal details, donation history, and contact information.
3. **User-Friendly Interface:** To provide an intuitive and easy-to-navigate interface for administrators and users.
4. **Secure Data Handling:** To ensure the security and integrity of sensitive donor data through secure database management and proper user access controls.
5. **Efficient CRUD Operations:** To allow administrators to add, update, delete, and view donor records quickly and without hassle.
6. **Scalability:** To design a system capable of handling a growing number of donors and maintaining optimal performance as the blood bank expands.
7. **Enhanced Data Accessibility:** To ensure that all donor information is easily accessible and can be managed efficiently, contributing to smoother blood donation processes and better management of resources.
8. **Methodology/Flow Diagram/Block Diagram**

The SQL script defines the database schema for the Blood Bank Management System, creating a single table to store donor information:

Donors

1. Schema Breakdown:
2. **Database Initialization:**

CREATE DATABASE bloodbank;

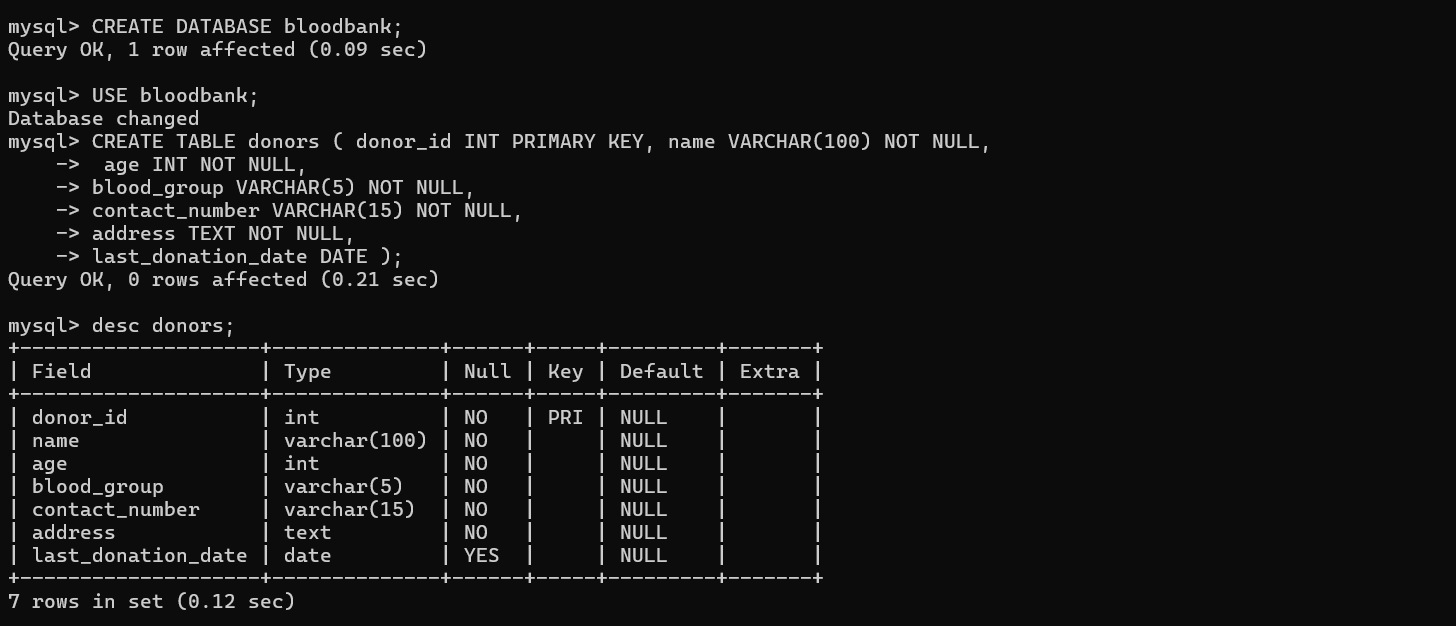
The database bloodbank is created and switched to using USE bloodbank;

1. **donors Table:**

Tracks donor information with fields such as donor\_id, name, age, blood\_group, contact\_number, address, and last\_donation\_date.

Working of all the attributes:

1. **donor\_id** is the primary key to uniquely identify each donor.
2. **name** stores the donor's full name.
3. **age** stores the donor's age, which must be a positive integer.
4. **blood\_group** stores the blood group of the donor.
5. **contact\_number** stores the contact information of the donor, formatted as a string.
6. **address** stores the address details of the donor.
7. **last\_donation\_date** stores the date of the donor’s last blood donation, allowing tracking of the donation cycle.



1. Relationships and Constraints:

* The table uses appropriate data types (VARCHAR, INT, DATE) for each field.
* The donor\_id field is auto-incremented to ensure each donor has a unique identifier.
* Ensures all fields are properly indexed for efficient search and retrieval operations.
* Validation can be added in the application code (Flask routes) to ensure data integrity, such as ensuring age is positive, and contact numbers follow a valid format.

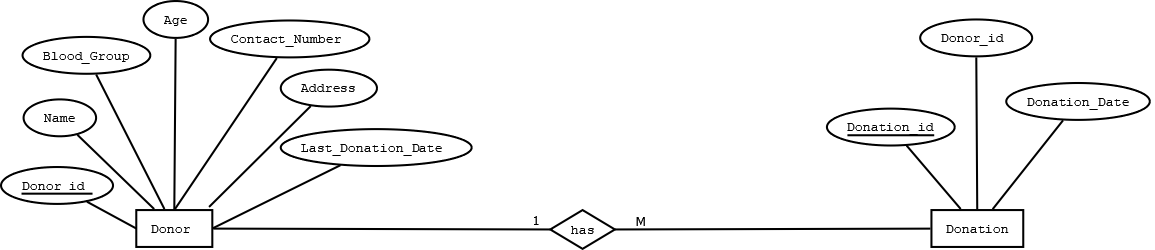
1. Flow of Data:

* The application interacts with the database via SQL queries to insert, update, delete, and fetch donor records.
* The backend (Flask) processes the user inputs (from forms) and interacts with the database to store and retrieve data.
* The frontend (HTML and CSS) provides a user-friendly interface to display, add, edit, and remove donor information.

1. Technology Stack

* **Backend:** Flask (Python)
* **Database:** MySQL
* **Frontend:** HTML, CSS, JavaScript

1. ER Diagram



**4. Implementation and Result Analysis**

**Implementation Process**

The implementation of the Blood Bank Management System consisted of the following steps:

**1. Database Creation:**

* Designed the bloodbank database using MySQL.
* Created a donors table with fields such as donor\_id, name, age, blood\_group, contact\_number, address, and last\_donation\_date.
* Ensured data types and relationships between fields to store donor information effectively.

The donors table stores the information about blood donors, including their personal details, contact information, and donation history. The donor\_id field is set as the primary key, with AUTO\_INCREMENT to ensure that each donor record is uniquely identified.

**2. Backend Development:**

The backend of the application was developed using the Flask framework and Python's pymysql library for MySQL database interaction. The following routes were created to manage donor information:

1. **Home Route (/)**:

* Displays the home page of the application.
* Uses render\_template('home.html') to load the home page template.

1. **Add Donor Route (/add)**:

* Handles both GET and POST requests.
* If the request method is POST, it processes the form data (donor's name, age, blood group, contact number, address, and last donation date) and inserts the data into the donors table.
* Redirects to the view donors page after a successful insertion.

1. **View Donors Route (/view)**:

* Fetches all donor records from the donors table using a SQL SELECT query.
* Passes the fetched donor data to the view\_donors.html template for display.

1. **Update Donor Route (/update/<int:donor\_id>)**:

* Fetches a specific donor's data based on donor\_id for editing.
* If the request method is POST, it updates the donor's information in the database.
* Redirects to the view donors page after updating.

1. **Delete Donor Route (/delete/<int:donor\_id>)**:

* Deletes a donor record from the database based on donor\_id.
* Redirects to the view donors page after deletion.

1. **Frontend Development:**

HTML templates were designed for each functionality (home, add donor, view donors, and update donor).

* **Home Template (home.html)**: Contains a basic landing page with links to other functionalities like viewing, adding, or editing donor details.
* **Add Donor Template (add\_donor.html)**: Provides a form to add a new donor with fields like name, age, blood group, contact number, address, and last donation date.
* **View Donors Template (view\_donors.html)**: Displays a list of all donors retrieved from the database.
* **Update Donor Template (update\_donor.html)**: Displays the details of a donor and allows updating their information.

CSS was used to style the forms and make them responsive, ensuring the application is visually appealing and user-friendly.

**4. Testing and Validation:**

* The application was tested with various types of input to ensure that the database interactions worked correctly, including adding, updating, and deleting donor records.
* Form validation was implemented to ensure required fields were not left empty.
* Error handling was implemented to catch database connection issues, missing fields, or invalid data entries.

**Technologies Used:**

* **Frontend**: HTML5 for structure and layout of pages.
* CSS3 for styling the application, ensuring a modern, responsive design.
* Optional JavaScript could be added for further interactivity, such as form validation or dynamic content.
* **Backend**: Python with Flask for web server, routing, and managing logic.
* pymysql for connecting the application to the MySQL database and performing CRUD operations.
* **Database**: MySQL for storing donor data, including personal details and donation history.
* **Database Connection**: PyMySQL for connecting Flask to MySQL.
* **Session Management**: Flask sessions can be implemented to manage user roles (admin or customer) for restricted operations such as updating or deleting records.
* **Security**: Flask-Login for user authentication to manage admin access securely.

**1. Home Page (/)**

* **Purpose**: Display the home page to the user.
* **Steps**:
  1. Receive an HTTP GET request to the / route.
  2. Render and return the home.html template.

**2. Add Donor (/add)**

* **Purpose**: Add a new donor to the database.
* **Steps**:
  1. If the request method is GET, render the add\_donor.html template to show the form for adding a new donor.
  2. If the request method is POST:
     + Extract the donor details (name, age, blood\_group, contact\_number, address, last\_donation\_date) from the form data.
     + Create a SQL INSERT query to add the new donor into the donors table.
     + Execute the query using a database cursor.
     + Commit the transaction to save the changes.
     + Redirect the user to the /view route to display the updated donor list.

**3. View Donors (/view)**

* **Purpose**: Retrieve and display a list of all donors.
* **Steps**:
  1. Receive an HTTP GET request to the /view route.
  2. Create a SQL SELECT query to fetch all records from the donors table.
  3. Execute the query using a database cursor.
  4. Fetch all results and store them in a variable (donors).
  5. Render the view\_donors.html template, passing the list of donors as context.

**4. Update Donor (/update/<int:donor\_id>)**

* **Purpose**: Update an existing donor's details in the database.
* **Steps**:
  1. Receive an HTTP request to the /update/<donor\_id> route.
  2. If the request method is GET:
     + Create a SQL SELECT query to fetch the donor's details by donor\_id.
     + Execute the query and retrieve the donor record.
     + Render the update\_donor.html template, passing the donor's details as context.
  3. If the request method is POST:
     + Extract the updated details (name, age, blood\_group, contact\_number, address, last\_donation\_date) from the form data.
     + Create a SQL UPDATE query to update the donor's record in the donors table.
     + Execute the query using a database cursor.
     + Commit the transaction to save the changes.
     + Redirect the user to the /view route to display the updated donor list.

**5. Delete Donor (/delete/<int:donor\_id>)**

* **Purpose**: Delete a donor from the database.
* **Steps**:
  1. Receive an HTTP GET request to the /delete/<donor\_id> route.
  2. Create a SQL DELETE query to remove the donor's record from the donors table using the donor\_id.
  3. Execute the query using a database cursor.
  4. Commit the transaction to save the changes.
  5. Redirect the user to the /view route to display the updated donor list.

**6. Run the Application**

* **Purpose**: Start the Flask application in debug mode.
* **Steps**:
  1. Use the app.run() method with the debug=True flag to start the development server.
  2. The application listens for incoming requests and handles routes as described above.

WORKING OF CODE AND RESULT :

1. **Importing Required Libraries**

This Flask application manages a blood donation system. It provides functionalities to add, view, update, and delete blood donors stored in a MySQL database. This application allows users to perform CRUD (Create, Read, Update, Delete) operations on blood donor information.

* **flask**: Used to create the web application and handle routes, rendering HTML templates, and handling HTTP requests.
* **pymysql**: Used to interact with a MySQL database.

**2. Flask App Initialization**

* **app = Flask(\_\_name\_\_)**: Initializes the Flask application.

**3. Database Configuration**

* **db = pymysql.connect(...)**: Connects to the MySQL database named bloodbank using credentials (host, user, password).

**4. Routes**

1. **@app.route('/')**

* Displays the home page (home.html).
* No data processing, just serves the homepage.

1. **@app.route('/add', methods=['GET', 'POST']**

* Handles the addition of new donors.
* GET: Displays the donor addition form (add\_donor.html).
* POST:
* Extracts donor details from the form (request.form).
* Inserts these details into the donors table using an INSERT SQL query.
* Redirects to the list of donors.

1. **@app.route('/view')**

* Displays all donors from the database (view\_donors.html).
* Executes a SELECT query to fetch donor data.

1. **@app.route('/update/<int:donor\_id>', methods=['GET', 'POST'])**

* Handles updating an existing donor.
* GET:
* Fetches donor data using a SELECT query for the provided donor\_id.
* Displays the form (update\_donor.html) pre-filled with donor details.
* POST:
* Updates the donor record with new data from the form using an UPDATE SQL query.
* Redirects to the list of donors.

1. **@app.route('/delete/<int:donor\_id>')**

* Deletes a donor by their donor\_id.
* Executes a DELETE SQL query and redirects to the list of donors.

**5. Running the App**

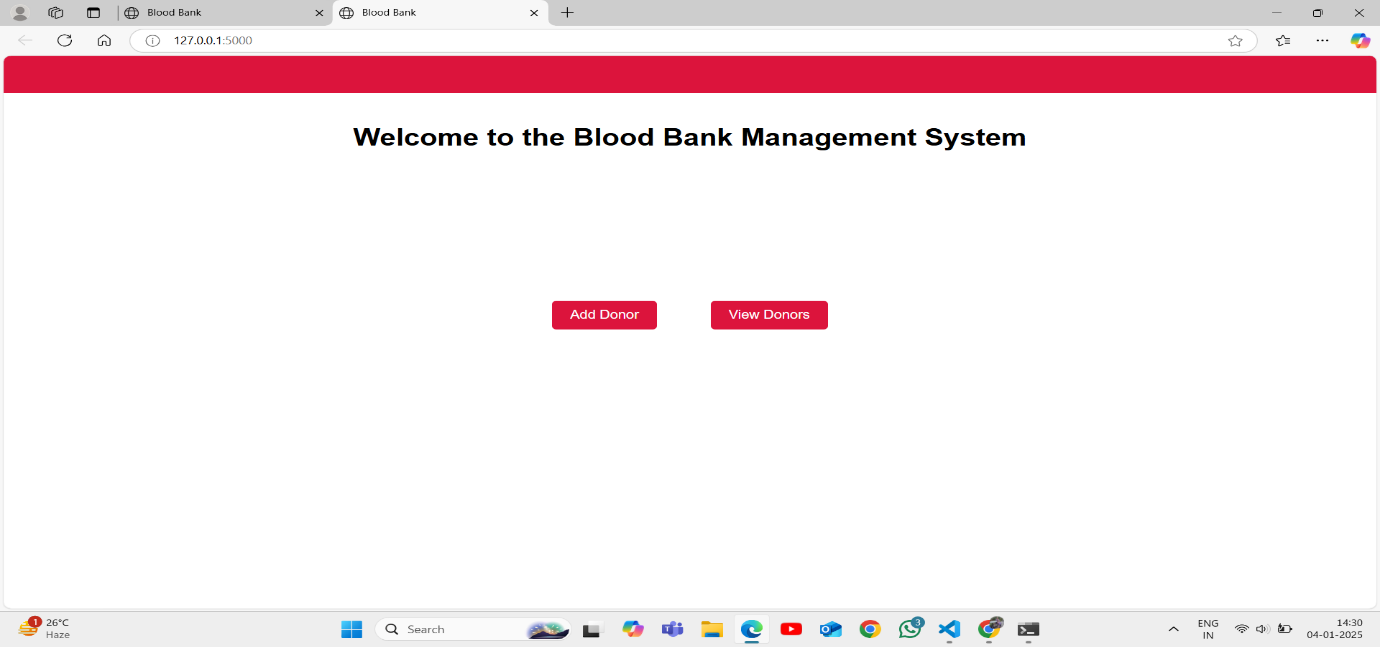
* **app.run(debug=True)**: Runs the Flask app in debug mode, allowing automatic reloading and error debugging.

**Results:**

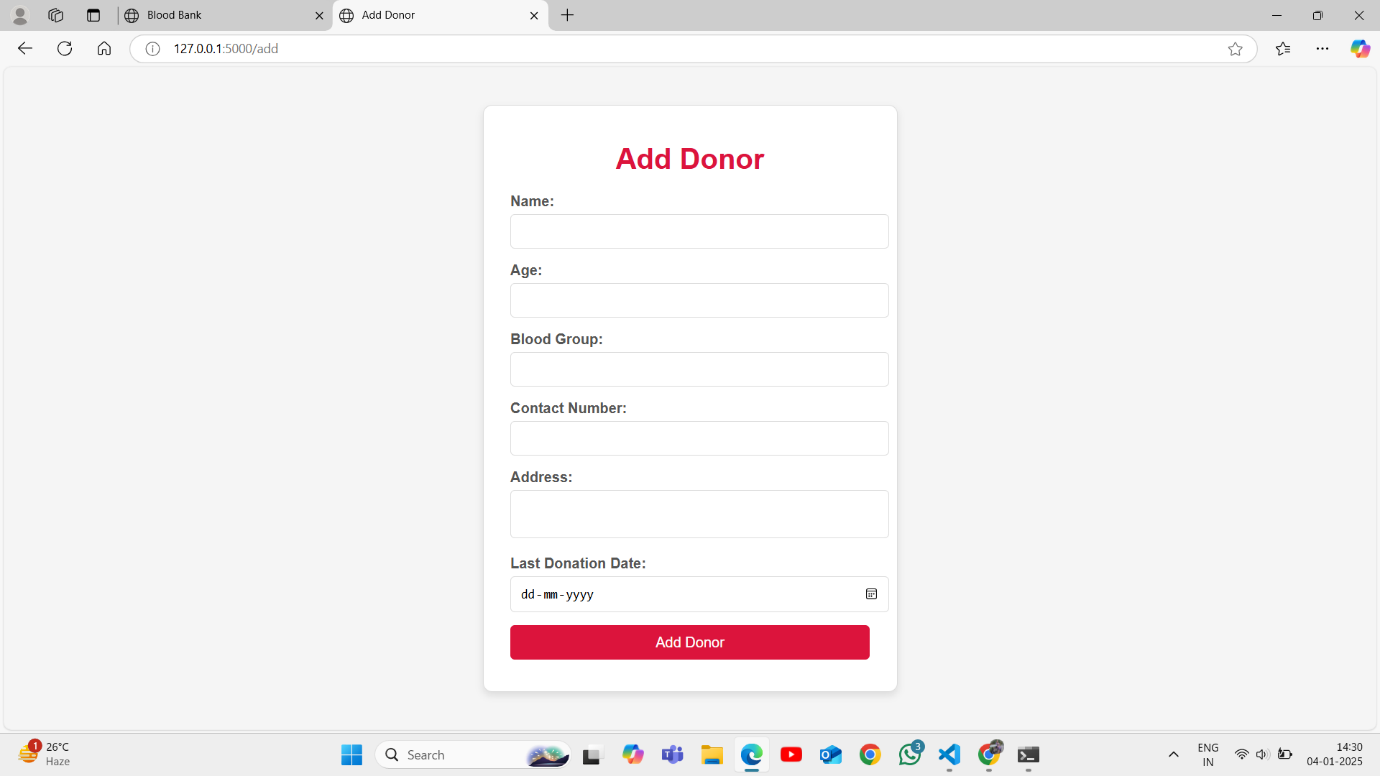
The application successfully fulfils the following requirements:

* Allows authorities to manage blood donor information.
* Supports CRUD operations (Create, Read, Update, Delete) for donor records.
* Ensures easy navigation through a simple and intuitive interface.
* The MySQL database stores donor data securely and supports querying to view, add, edit, or delete records.

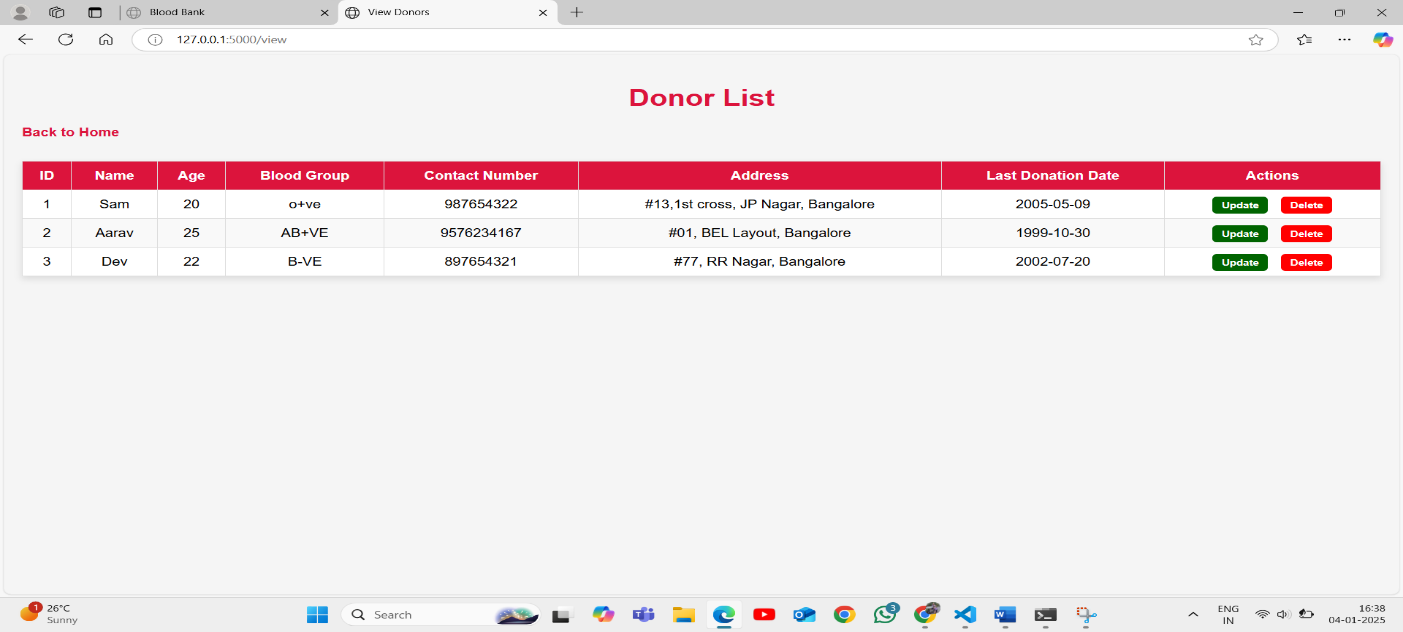
1. HOME PAGE



1. ADD DONOR



1. VIEW LIST



Challenges Faced:

* **Database Integration**: Configuring MySQL and ensuring proper data insertion, update, and deletion was initially challenging, especially with data types and query syntax.
* **Frontend Responsiveness**: Designing a responsive and user-friendly frontend interface for managing donor information across devices took significant effort, especially for form handling and data display.
* **Data Integrity**: Ensuring that the data entered through the forms was valid (e.g., correct contact numbers and valid dates) was a critical challenge to maintain data integrity.

Future Enhancements:

The system can be further enhanced with the following features:

* **User Authentication**: Implement role-based authentication to allow different access levels (e.g., Admin for adding/updating records, User for viewing only).
* **Mobile Application**: Develop a mobile app for on-the-go access to manage donor records.
* **Search and Filter**: Implement a search functionality to easily find donors based on criteria like blood group or contact details.
* **Email Notifications**: Add functionality to send email reminders to donors about upcoming donation opportunities.
* **Data Analytics**: Generate reports and insights, such as donation frequency or common blood groups, to improve the blood donation process.