EXPNO: 10 BLOOD BANK MANAGEMENT SYSTEM

DATE:

Description:

A **Blood Bank Management System (BBMS)** is a software solution that efficiently manages the collection, storage, and distribution of blood. It tracks donor information, monitors blood inventory levels, and handles requests from hospitals. Key features include donor registration, blood donation tracking, inventory management, expiry alerts, and secure reporting. The system improves operational efficiency, ensures timely blood supply, reduces wastage, and enhances donor engagement, making it essential for maintaining a reliable blood bank.

Introduction:

The **Blood Bank Management System (BBMS)** is a software solution that streamlines the management of blood donations, inventory, and distribution. It helps track donor details, monitor blood stocks, and facilitate timely blood delivery to hospitals. By automating processes like donation tracking and inventory management, the BBMS enhances efficiency, reduces errors, and ensures a reliable supply of blood for patients in need.

System architecture:

Technology stack

Frontend: HTML, CSS, JavaScript (React.js, Angular)

Backend: Java (Spring Boot), Python (Django, Flask), or Node.js (Express)

Database: MySQL, PostgreSQL, or MongoDB

Cloud/Hosting: AWS, Google Cloud, or Heroku

APIs: RESTful API for communication

Security: SSL/TLS, encryption, role-based access control

File structure:

```
/dbms-project
├── /frontend          # Frontend files (React/Angular)
│   ├── /components    # UI components
│   ├── /pages         # Different app pages
│   ├── /services      # API calls
│   ├── App.js         # Main app file
│   └── package. json  # Frontend dependencies
├── /backend           # Backend files (Node.js/Java/Spring)
│   ├── /controllers   # API logic
│   ├── /models        # DB schemas
│   ├── /routes        # API routes
│   ├── /config        # Config files (DB, server)
│   └── server.js      # Server setup
├── /database          # DB-related files (migrations, seeds)
├── /docs              # Documentation
├── .env               # Environment variables
└── README.md          # Project overview
```

Database structure:

```sql
CREATE TABLE Donors (
    donor_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(15) NOT NULL,
    blood_group VARCHAR(5) NOT NULL,
    last_donation_date DATE,
    medical_history TEXT,
    status ENUM('active', 'inactive') DEFAULT 'active'
);
```

Features:

1. Donor Management:

- **Donor Registration:** Register new donors with personal details, blood type, and medical history.
- **Donor Eligibility Check:** Track eligibility based on the time since last donation and health conditions.
- **Donor History:** Maintain a history of each donor's donation records, including blood group, donation dates, and medical conditions.
- **Donor Notifications:** Send reminders to donors for future donations based on eligibility.

2. Blood Inventory Management:

- **Inventory Tracking:** Manage and monitor the stock of blood and its components (whole blood, plasma, red blood cells, platelets).
- **Expiry Management:** Track the expiry dates of blood bags and generate alerts when blood is nearing its expiration.
- **Blood Group Classification:** Categorize inventory by blood type and Rh factor (e.g., A+, O-, AB+).
- **Stock Alerts:** Notify when stock is low or when blood bags are nearing expiry.

3. Blood Donation and Collection:

- **Donation Scheduling:** Allow donors to schedule appointments for blood donation.
- **Donation Tracking:** Track the quantity of blood donated, the type of blood, and the donor's information.
- **Blood Collection:** Record details of each donation and link it to the appropriate inventory.

4. Blood Requests and Distribution:

- **Hospital Requests:** Hospitals can submit blood requests with required blood group, quantity, and urgency.
- **Request Fulfillment:** Track and manage the fulfillment of blood requests from hospitals or clinics.
- **Delivery Tracking:** Monitor the delivery and transfer of blood to hospitals.
- **Request Prioritization:** Prioritize requests based on urgency and blood availability.

5. Transaction Management:

- **Blood Transactions:** Keep a record of all transactions involving blood (e.g., donations, blood distributed to hospitals).
- **Transaction History:** Maintain a log of all blood movements, including quantities used and reasons for use.

6. Reporting and Analytics:

- **Inventory Reports:** Generate reports on blood stock levels, expiration dates, and usage.
- **Donation Statistics:** Track donation trends, donor frequency, and blood type availability.
- **Hospital Request Reports:** Analyze the volume of blood requested by hospitals over time.
- **Performance Analytics:** Generate insights on the system's overall performance, such as donor engagement, blood usage, and waste.

7. User and Role Management:

- **Role-Based Access Control (RBAC):** Assign roles (e.g., admin, manager, staff) and restrict access to sensitive data based on roles.
- **User Management:** Add, update, and delete user accounts for staff managing the system.

9. Communication and Notification System:

- **Email/SMS Notifications:** Notify donors about upcoming donation drives, appointments, and eligibility.
- **Reminder Notifications:** Send reminders for when the blood needs replenishing or when blood donations are needed.

10. Backup and Data Recovery:

- **Data Backup:** Regular backups of system data to ensure data integrity and prevent loss in case of system failure.
- **Disaster Recovery:** Plan for data restoration and recovery in case of technical issues or server failures.

11. Integration with Other Systems:

- **Hospital Management Systems (HMS):** Integration with other healthcare systems to automatically process blood requests from hospitals.
- **Payment Systems:** Integration for processing payments or handling donation-related financial records (if applicable).

12. Mobile and Web Application:

- **Donor App:** A mobile or web app for donors to register, track their donation history, and schedule appointments.
- **Admin Portal:** A web-based admin dashboard to manage donors, blood inventory, and hospital requests.

Installation:

1. Clone the Repository:

bash
CopyEdit
git clone https://github.com/your-username/blood-bank-management-system.git
cd blood-bank-management-system

2. Backend Installation:

- Navigate to the backend directory:

bash
CopyEdit
cd backend

- Install dependencies:

bash
CopyEdit
npm install

- Configure your **MySQL** database (config/db.js) and create a database (blood_bank_db).
- Start the backend server:

bash
CopyEdit
npm start

3. Frontend Installation:

- Navigate to the frontend directory:

bash
CopyEdit
cd frontend

- Install dependencies:

bash
CopyEdit
npm install

- Set API URL in .env file (REACT_APP_API_URL=http://localhost:5000).
- Start the frontend server:

bash
CopyEdit
npm start

4. Database Setup:

- Create the database and tables in MySQL:

sql
CopyEdit
CREATE DATABASE blood_bank_db;
USE blood_bank_db;
-- Run table creation scripts

5. Test:

- Visit http://localhost:3000 (frontend) and http://localhost:5000 (backend) to test the system.

This covers the basic installation steps. Adjust configurations as needed.

Administration log-in:

```
CREATE TABLE Admins (
    admin_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

Adding records and data:

```
<?php
$conn = new mysqli("localhost", "root", "", "blood_bank_db");
$sql = "INSERT INTO BloodInventory (blood_group, quantity, blood_type, date_received)
        VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("siss", $_POST['blood_group'], $_POST['quantity'], $_POST['blood_type']
$stmt->execute();
echo "Record added";
?>
```

Viewing records:

```php
<?php
$result = $conn->query("SELECT * FROM BloodInventory ORDER BY id DESC");
?>
```

Assigning blood groups and donors:

```php
<?php
// Get blood record ID from URL if it exists
$blood_id = isset($_GET['blood_id']) ? (int)$_GET['blood_id'] : 0;

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Process form submission here (e.g., update or add record)
$blood_id = (int)$_POST['blood_id'];
$storage_location = $conn->real_escape_string($_POST['storage_location']);

if (empty($storage_location)) {
    $error = "Please enter a storage location";
} else {
    // Check if the storage location is already assigned
    $check = $conn->query("SELECT id FROM BloodInventory WHERE
        storage_location = '$storage_location' AND id != $blood_id");

    if ($check->num_rows > 0) {
        $error = "Storage location is already assigned to another record";
    } else {
        // Update the record
        $query = "UPDATE BloodInventory SET storage_location =
            '$storage_location' WHERE id = $blood_id";
        if ($conn->query($query)) {
            echo '<script>showMessage("Storage location assigned
                successfully!", "success");</script>';
        } else {
            $error = "Error: " . $conn->error;
        }
    }
}

// Get unassigned blood records
$unassigned = $conn->query("SELECT * FROM BloodInventory WHERE
    storage_location IS NULL OR id = $blood_id");
```

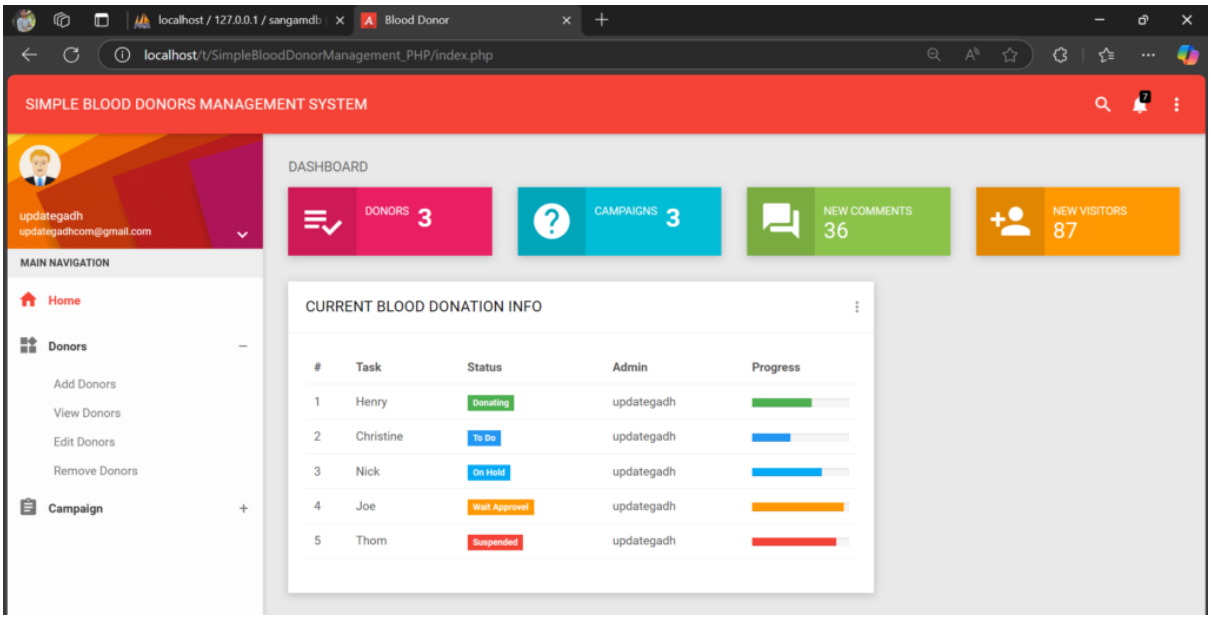
Editing records:

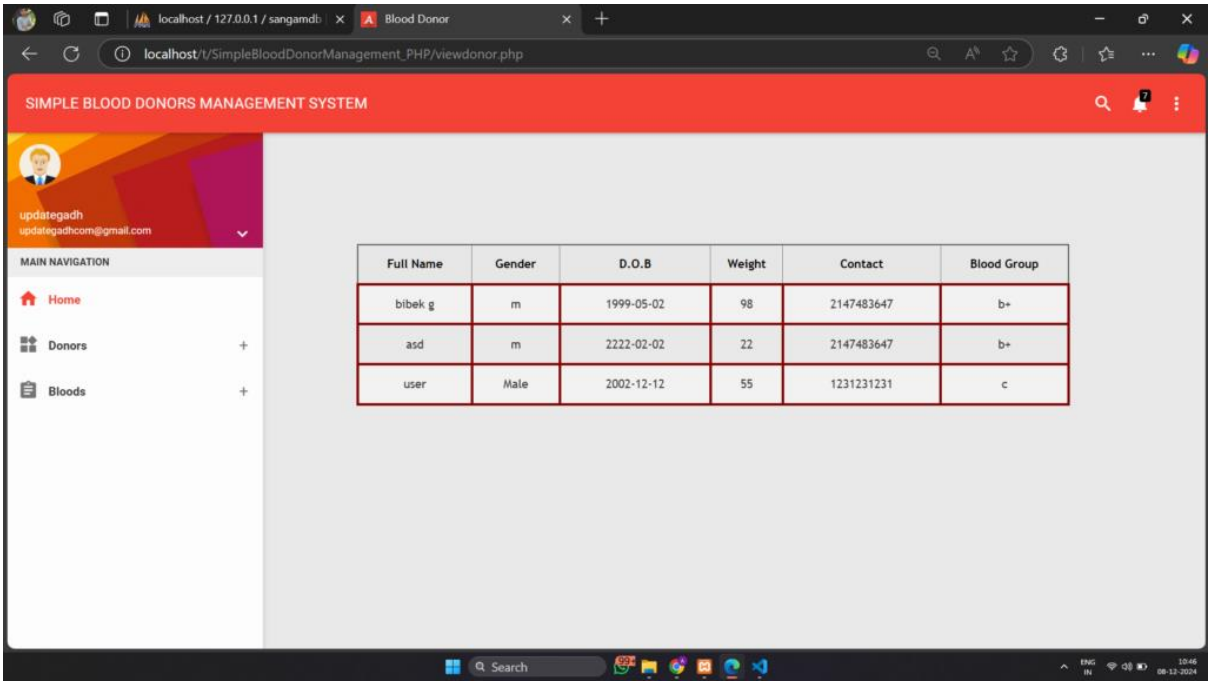
```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Escape input data to prevent SQL injection
    $blood_group = $conn->real_escape_string($_POST['blood_group']);
    $quantity = (int)$_POST['quantity'];

    // Validate that the fields are not empty
    if (empty($blood_group) || empty($quantity)) {
        $error = "Please fill in all fields";
    } else {
        // Update the blood record in the database
        $query = "UPDATE BloodInventory SET blood_group = '$blood_group',
            quantity = $quantity WHERE id = $blood_id";

        if ($conn->query($query)) {
            echo '<script>showMessage("Blood record updated successfully!",
                "success");</script>';

            // Update the blood record data after successful update
            $blood_record['blood_group'] = $blood_group;
            $blood_record['quantity'] = $quantity;
        } else {
            $error = "Error: " . $conn->error;
        }
    }
}
?>
```

Deleting records:

```php
<?php
// Handle blood record deletion
if (isset($_GET['delete'])) {
    $blood_id = (int)$_GET['delete'];

    // Delete the blood record from the database
    if ($conn->query("DELETE FROM BloodInventory WHERE id = $blood_id")) {
        echo '<script>alert("Blood record deleted successfully!"); window
            .location.href = "view_blood_records.php";</script>';
    } else {
        echo '<script>alert("Error deleting record: ' . $conn->error . '");
            window.location.href = "view_blood_records.php";</script>';
    }
}
?>
```

Output:





Result:

The **Blood Bank Management System** efficiently tracks and manages blood donations, ensuring accurate inventory and easy access to blood records. It allows users to add, update, or delete blood records, helping maintain an up-to-date database. The system improves blood availability by streamlining operations and providing real-time updates.