```
const express = require('express');
const cors = require('cors');
const path = require('path');
const fs = require('fs');
const bcrypt = require('bcrypt');
const SibApiV3Sdk = require('sib-api-v3-sdk'); // Import Brevo SDK
require('dotenv').config(); // Load environment variables from .env

const app = express();
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Serve static files from the 'public' directory
app.use(express.static(path.join(__dirname, 'public')));

// Load existing data from JSON files
let users = [];
let hospitals = [];
let bloodDonors = [];
let ambulances = [];

const loadData = () => {
    if (fs.existsSync('users.json')) {
        users = JSON.parse(fs.readFileSync('users.json'));
    }
    if (fs.existsSync('hospitals.json')) {
        hospitals = JSON.parse(fs.readFileSync('hospitals.json'));
    }
    if (fs.existsSync('donors.json')) {
        bloodDonors = JSON.parse(fs.readFileSync('donors.json'));
```

```
    }
    if (fs.existsSync('ambulances.json')) {
        ambulances = JSON.parse(fs.readFileSync('ambulances.json'));
    }
};

// Save data to JSON files
const saveData = () => {
    fs.writeFileSync('users.json', JSON.stringify(users, null, 2));
    fs.writeFileSync('hospitals.json', JSON.stringify(hospitals, null, 2));
    fs.writeFileSync('donors.json', JSON.stringify(bloodDonors, null, 2));
    fs.writeFileSync('ambulances.json', JSON.stringify(ambulances, null, 2));
};

// Load data when the server starts
loadData();

// Define a simple route
app.get('/', (req, res) => {
    res.send('Medical Emergency App Server is Running!');
});

// User Registration Route
app.post('/register', async (req, res) => {
    const { username, password, email } = req.body; // Add email to registration
    if (username && password && email) {
        const hashedPassword = await bcrypt.hash(password, 10);
        users.push({ username, password: hashedPassword, email }); // Store email
        saveData();
        console.log(`User registered: ${username}`);
        return res.send(`User ${username} registered successfully!`);
```

```
    }
    return res.status(400).send('Failed to register user. Missing username, password, or email.');
});

// User Login Route
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    const user = users.find(u => u.username === username);
    if (user && await bcrypt.compare(password, user.password)) {
        console.log(`User logged in: ${username}`);
        return res.send(`Welcome back, ${username}!`);
    }
    return res.status(401).send('Invalid username or password.');
});

// Hospital Registration Route
app.post('/hospitalRegister', (req, res) => {
    const { hospitalName, specialist, doctorName, experience, location, fees } = req.body;
    if (!hospitalName || !specialist || !doctorName || !experience || !location || fees === undefined)
{
        return res.status(400).send('All fields are required!');
    }

    const hospitalEntry = { hospitalName, specialist, doctorName, experience, location,
consultationFees: fees };
    hospitals.push(hospitalEntry);
    saveData();
    console.log('Registered Hospital:', hospitalEntry);
    res.send('Hospital registered successfully!');
});

// View Registered Hospitals Route
```

```
app.get('/hospitals', (req, res) => {
    res.json(hospitals);
});

// Delete Hospital Route
app.delete('/hospitals/:hospitalName', (req, res) => {
    const { hospitalName } = req.params;
    const index = hospitals.findIndex(h => h.hospitalName === hospitalName);

    if (index !== -1) {
        hospitals.splice(index, 1);
        saveData();
        console.log(`Hospital deleted: ${hospitalName}`);
        return res.send(`Hospital ${hospitalName} deleted successfully.`);
    }

    return res.status(404).send('Hospital not found.');
});

// Blood Donor Registration Route (with blood group dropdown)
app.post('/registerBloodDonor', (req, res) => {
    const { donorName, bloodGroup, location, email } = req.body;
    const bloodGroups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']; // Define available blood groups

    if (!donorName || !bloodGroup || !location || !email) {
        return res.status(400).send('All fields are required!');
    }

    if (!bloodGroups.includes(bloodGroup)) {
        return res.status(400).send('Invalid blood group selected!');
    }
```

```
    const donorEntry = { donorName, bloodGroup, location, email }; // Ensure email is correctly set
    bloodDonors.push(donorEntry);
    saveData();
    console.log('Registered Blood Donor:', donorEntry);
    res.send('Blood donor registered successfully!');
});

// View Blood Donors Route
app.get('/donors', (req, res) => {
    res.json(bloodDonors);
});

// Delete Donor Route
app.delete('/donors/:donorName', (req, res) => {
    const { donorName } = req.params;
    const index = bloodDonors.findIndex(d => d.donorName === donorName);

    if (index !== -1) {
        bloodDonors.splice(index, 1);
        saveData();
        console.log(`Donor deleted: ${donorName}`);
        return res.send(`Donor ${donorName} deleted successfully.`);
    }

    return res.status(404).send('Donor not found.');
});

// Import and configure Brevo API instance
const apiInstance = new SibApiV3Sdk.TransactionalEmailsApi();
const apiKey = process.env.BREVO_API_KEY; // Securely using the API key from the .env file
```

```
SibApiV3Sdk.ApiClient.instance.authentications['api-key'].apiKey = apiKey;


// Send Blood Donation Alert Route
app.post('/alert', (req, res) => {
   const { bloodGroup, message } = req.body;
   console.log(`Alert request received: Blood Group - ${bloodGroup}, Message - ${message}`);


   const recipientEmails = bloodDonors
      .filter(donor => donor.bloodGroup === bloodGroup)  // Filter donors by blood group
      .map(donor => ({ email: donor.email, name: donor.donorName }));  // Prepare email & name for each donor


   if (recipientEmails.length === 0) {
      return res.status(404).send('No registered donors found to send the alert.');
   }
// Start the server
const port = process.env.PORT || 5000;
app.listen(port, () => {
   console.log(`Server is running on port ${port}`);
});




<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>User Registration</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <style>
```

```
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f8f9fa;
        }
        .registration-container {
            background-color: white;
            padding: 30px;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            width: 400px; /* Fixed width for uniformity */
        }
        .form-group {
            margin-bottom: 20px; /* Increased space between fields */
        }
    </style>




</head>
<body>
    <header>
        <img src="images\logo.png" alt="Logo" class="logo"> <!-- Use the actual file name -->
    </header>



    <div class="registration-container">
        <h1 class="text-center">User Registration</h1>
```

```
    <form id="registrationForm">
      <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" class="form-control" id="username" name="username" required>
      </div>
      <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" class="form-control" id="email" name="email" required>
      </div>
      <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" class="form-control" id="password" name="password" required>
      </div>
      <button type="submit" class="btn btn-primary btn-block">Register</button>
      <p class="mt-3 text-center">Already have an account? <a href="login.html">Login
here</a>.</p>
    </form>
  </div>

  <script>
    document.getElementById('registrationForm').onsubmit = async function(event) {
      event.preventDefault();
      const username = document.getElementById('username').value;
      const email = document.getElementById('email').value; // Get email value
      const password = document.getElementById('password').value;

      const response = await fetch('/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, email, password }), // Include email in the request
      });
```

```
        const result = await response.text();
        alert(result);
        if (response.ok) {
            window.location.href = 'login.html';
        }
    };
</script>

<!-- Cami Button to start the assistant -->
<button id="camiButton" class="camiButton">CAMI</button>

<!-- Stop Button (initially hidden) to stop the assistant -->
<button id="stopButton" style="display:none;">Stop</button>

<!-- This is where the assistant's messages will be displayed -->
<div id="assistantMessages"></div>

<!-- Link to voice-assistant.js -->
<script src="voice-assistant.js"></script>



</body>
</html>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Medical Emergency App</title>
```

```
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <img src="images/logo.png" alt="Logo" class="logo">
    </header>

    <div class="container">
        <h1>Welcome to the Medical Emergency Coordination App</h1>
        <div class="buttons">
            <a href="register.html" class="button">Register</a>
            <a href="login.html" class="button">Login</a>
        </div>
    </div>

    <!-- Cami Button to start the assistant -->
    <button id="camiButton" class="camiButton">CAMI</button>

    <!-- Stop Button (initially hidden) to stop the assistant -->
    <button id="stopButton" style="display:none;">Stop</button>

    <!-- This is where the assistant's messages will be displayed -->
    <div id="assistantMessages"></div>

    <!-- Link to voice-assistant.js -->
    <script src="voice-assistant.js"></script>
```

```
</body>
</html>



<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Login</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f8f9fa;
        }
        .login-container {
            background-color: white;
            padding: 30px;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            width: 400px; /* Fixed width for uniformity */
        }
        .form-group {
            margin-bottom: 20px; /* Increased space between fields */
        }
    </style>
```

```
</head>
<body>
  <header>
    <img src="images\logo.png" alt="Logo" class="logo">
  </header>

<!-- Cami Button to start the assistant -->
<button id="camiButton" class="camiButton">CAMI</button>

<!-- Stop Button (initially hidden) to stop the assistant -->
<button id="stopButton" style="display:none;">Stop</button>

<!-- This is where the assistant's messages will be displayed -->
<div id="assistantMessages"></div>

<!-- Link to voice-assistant.js -->
<script src="voice-assistant.js"></script>


  <div class="login-container">
    <h1 class="text-center">User Login</h1>
    <form id="loginForm">
      <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" class="form-control" id="username" name="username" required>
      </div>
      <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" class="form-control" id="password" name="password" required>
      </div>
      <button type="submit" class="btn btn-primary btn-block">Login</button>
```

```
        <p class="mt-3 text-center">Forgot your password? <a href="forgot_password.html">Reset it here</a>.</p>
      </form>
    </div>

    <script>
      document.getElementById('loginForm').onsubmit = async function(event) {
        event.preventDefault();
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;

        const response = await fetch('/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ username, password }),
        });

        const result = await response.text();
        alert(result);
        if (response.ok) {
          window.location.href = 'dashboard.html';
        }
      };
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hospital Registration</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <img src="images\logo.png" alt="Logo" class="logo">
    </header>

  <!-- Cami Sphere as the button -->
  <div id="cami-sphere" style="position: fixed; bottom: 20px; right: 20px; width: 60px; height: 60px; background-color: #4CAF50; border-radius: 50%; display: flex; justify-content: center; align-items: center; color: white; cursor: pointer; font-size: 16px; text-align: center; z-index: 9999;">
    <span style="font-weight: bold;">Cami</span>
</div>

<!-- Manual Stop Button -->
<button id="stop-btn" style="position: fixed; bottom: 20px; left: 20px; background-color: red; color: white; padding: 10px 15px; border: none; border-radius: 5px; cursor: pointer; z-index: 9999;">
    Stop Cami
</button>

<!-- Link to the JS file that includes Cami's functionality -->
<script src="/cami.js"></script>



    <div class="container">
        <h1>Register a New Hospital</h1>
        <form id="hospitalRegistrationForm">
```

```
<label for="hospitalName">Hospital Name:</label>
<input type="text" id="hospitalName" name="hospitalName" required>

<label for="specialist">Specialist:</label>
<input type="text" id="specialist" name="specialist" required>

<label for="doctorName">Doctor's Name:</label>
<input type="text" id="doctorName" name="doctorName" required>

<!-- Dropdown for Experience -->
<label for="experience">Experience:</label>
<select id="experience" name="experience" required>
    <option value="" disabled selected>Select Years of Experience</option>
    <option value="0">0 years</option>
    <option value="1">1 year</option>
    <option value="2">2 years</option>
    <option value="3">3 years</option>
    <option value="4">4 years</option>
    <option value="5">5 years</option>
    <option value="6">6 years</option>
    <option value="7">7 years</option>
    <option value="8">8 years</option>
    <option value="9">9 years</option>
    <option value="10">10 years</option>
    <option value="11">11 years</option>
    <option value="12">12 years</option>
    <option value="13">13 years</option>
    <option value="14">14 years</option>
    <option value="15">15 years</option>
    <option value="16">16 years</option>
    <option value="17">17 years</option>
```

```
            <option value="18">18 years</option>
            <option value="19">19 years</option>
            <option value="20">20 years</option>
            <!-- Broader options for experience -->
            <option value="21">21 years+</option>
            <option value="25">25 years+</option>
            <option value="30">30 years+</option>
            <option value="35">35 years+</option>
            <option value="40">40 years+</option>
            <option value="45">45 years+</option>
            <option value="50">50 years+</option>
        </select>

        <label for="location">Location:</label>
        <input type="text" id="location" name="location" required>

        <label for="fees">Consultation Fees (₹):</label>
        <input type="number" id="fees" name="fees" required min="0">

        <button type="submit">Register Hospital</button>
    </form>
</div>

<script>
    document.getElementById('hospitalRegistrationForm').onsubmit = async function(event) {
        event.preventDefault();
        const hospitalName = document.getElementById('hospitalName').value;
        const specialist = document.getElementById('specialist').value;
        const doctorName = document.getElementById('doctorName').value;
        const experience = document.getElementById('experience').value;
        const location = document.getElementById('location').value;
```

```
        const fees = document.getElementById('fees').value;

        const response = await fetch('/hospitalRegister', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ hospitalName, specialist, doctorName, experience, location, fees }),
        });

        const result = await response.text();
        alert(result);
        if (response.ok) {
            window.location.href = 'dashboard.html';
        }
    };
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registered Hospitals</title>
  <link rel="stylesheet" href="style.css">
  <style>
    body {
      margin: 0;
      padding: 20px;
```

```
    font-family: Arial, sans-serif;
}

.container {
    max-width: 800px;
    margin: 0 auto;
}

h1 {
    text-align: center;
    margin-bottom: 20px;
}

/* Make the table container scrollable if the list is long */
.table-container {
    max-height: 400px; /* Adjust the height as needed */
    overflow-y: auto;
    border: 1px solid #ccc;
}

table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    border: 1px solid #ddd;
    padding: 8px;
    text-align: left;
}
```

```
        th {
            background-color: #f4f4f4;
            position: sticky; /* Keeps the header fixed while scrolling */
            top: 0;
        }

        .delete-btn {
            background-color: #ff4d4d;
            color: white;
            border: none;
            padding: 5px 10px;
            cursor: pointer;
        }

        .delete-btn:hover {
            background-color: #ff0000;
        }
    </style>
</head>
<body>
    <header>
        <img src="images\logo.png" alt="Logo" class="logo">
    </header>

  <!-- Cami Sphere as the button -->
  <div id="cami-sphere" style="position: fixed; bottom: 20px; right: 20px; width: 60px; height: 60px; background-color: #4CAF50; border-radius: 50%; display: flex; justify-content: center; align-items: center; color: white; cursor: pointer; font-size: 16px; text-align: center; z-index: 9999;">
    <span style="font-weight: bold;">Cami</span>
</div>

<!-- Manual Stop Button -->
```

```
<button id="stop-btn" style="position: fixed; bottom: 20px; left: 20px; background-color: red; color: white; padding: 10px 15px; border: none; border-radius: 5px; cursor: pointer; z-index: 9999;">
  Stop Cami
</button>

<!-- Link to the JS file that includes Cami's functionality -->
<script src="/cami.js"></script>



  <div class="container">
    <h1>Registered Hospitals</h1>
    <div class="table-container">
      <table>
        <thead>
          <tr>
            <th>Hospital Name</th>
            <th>
              Specialist:
              <select id="specialistFilter" onchange="filterHospitals()">
                <option value="">All</option>
                <option value="Cardiologist">Cardiologist</option>
                <option value="Dermatologist">Dermatologist</option>
                <option value="Pediatrician">Pediatrician</option>
                <option value="Orthopedic">Orthopedic</option>
                <option value="General Surgeon">General Surgeon</option>
                <option value="Neurologist">Neurologist</option>
                <option value="Gynecologist">Gynecologist</option>
                <option value="Psychiatrist">Psychiatrist</option>
                <option value="ENT Specialist">ENT Specialist</option>
                <option value="Endocrinologist">Endocrinologist</option>
                <option value="Oncologist">Oncologist</option>
```

```
                    <option value="Urologist">Urologist</option>
                    <option value="Radiologist">Radiologist</option>
                    <option value="Pathologist">Pathologist</option>
                    <option value="Dental Surgeon">Dental Surgeon</option>
                    <option value="General Physician">General Physician</option>
                    <!-- Add more specialists as needed -->
                  </select>
                </th>
                <th>Doctor's Name</th>
                <th>Experience (years)</th>
                <th>Location</th>
                <th>Consultation Fees (₹)</th>
                <th>Action</th>
              </tr>
            </thead>
            <tbody id="hospitalList"></tbody>
          </table>
        </div>
      </div>

      <script>
        async function fetchHospitals() {
          const response = await fetch('/hospitals');
          const hospitals = await response.json();
          const hospitalList = document.getElementById('hospitalList');

          hospitalList.innerHTML = ''; // Clear existing list
          hospitals.forEach(hospital => {
            const listItem = document.createElement('tr'); // Create table row

            listItem.innerHTML = `
```

```
            <td>${hospital.hospitalName}</td>
            <td>${hospital.specialty}</td>
            <td>${hospital.doctorName}</td>
            <td>${hospital.experience}</td>
            <td>${hospital.location}</td>
            <td>${hospital.consultationFees}</td>
            <td><button class="delete-btn">Delete</button></td>
        `;

        const deleteButton = listItem.querySelector('.delete-btn');
        deleteButton.onclick = async () => {
            const delResponse = await fetch(`/hospitals/${hospital.hospitalName}`, {
                method: 'DELETE'
            });
            const delResult = await delResponse.text();
            alert(delResult);
            fetchHospitals(); // Refresh list after deletion
        };

        hospitalList.appendChild(listItem);
    });
}

function filterHospitals() {
    const selectedSpecialist = document.getElementById('specialistFilter').value;
    const hospitalRows = document.querySelectorAll('#hospitalList tr');

    hospitalRows.forEach(row => {
        const specialistCell = row.cells[1].innerText; // Assuming specialty is in the second cell
        if (selectedSpecialist === '' || specialistCell === selectedSpecialist) {
            row.style.display = '';
```

```
            } else {
                row.style.display = 'none';
            }
        });
    }

    fetchHospitals();
  </script>
</body>
</html>

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Register Blood Donor</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <img src="images\logo.png" alt="Logo" class="logo">
  </header>

 <!-- Cami Sphere as the button -->
 <div id="cami-sphere" style="position: fixed; bottom: 20px; right: 20px; width: 60px; height: 60px; background-color: #4CAF50; border-radius: 50%; display: flex; justify-content: center; align-items: center; color: white; cursor: pointer; font-size: 16px; text-align: center; z-index: 9999;">
  <span style="font-weight: bold;">Cami</span>
</div>

<!-- Manual Stop Button -->
```

```
<button id="stop-btn" style="position: fixed; bottom: 20px; left: 20px; background-color: red; color: white; padding: 10px 15px; border: none; border-radius: 5px; cursor: pointer; z-index: 9999;">
  Stop Cami
</button>

<!-- Link to the JS file that includes Cami's functionality -->
<script src="/cami.js"></script>

  <div class="container">
    <h1>Register as a Blood Donor</h1>
    <form id="donorRegistrationForm">
      <label for="donorName">Name:</label>
      <input type="text" id="donorName" name="donorName" required>

      <label for="bloodGroup">Blood Group:</label>
      <select id="bloodGroup" name="bloodGroup" required>
        <option value="">Select Blood Group</option>
        <option value="A+">A+</option>
        <option value="A-">A-</option>
        <option value="B+">B+</option>
        <option value="B-">B-</option>
        <option value="AB+">AB+</option>
        <option value="AB-">AB-</option>
        <option value="O+">O+</option>
        <option value="O-">O-</option>
      </select>

      <label for="location">Location:</label>
      <input type="text" id="location" name="location" required>

      <label for="email">Email:</label>
```

```
        <input type="email" id="email" name="email" required>

        <button type="submit">Register Donor</button>
    </form>
</div> <!-- Closing container div -->

<script>
    document.getElementById('donorRegistrationForm').addEventListener('submit', async function(event) {
        event.preventDefault(); // Prevent the form from submitting the traditional way

        const donorData = {
            donorName: this.donorName.value,
            bloodGroup: this.bloodGroup.value,
            location: this.location.value,
            email: this.email.value
        };

        try {
            const response = await fetch('/registerBloodDonor', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(donorData)
            });

            const result = await response.text();
            alert(result); // Alert the user with the response from the server
            if (response.ok) {
                this.reset(); // Reset form fields on success
```

```
            }
        } catch (error) {
            console.error('Error registering donor:', error);
            alert('Error registering donor. Please try again.');
        }
    });
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registered Blood Donors</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <img src="images\logo.png" alt="Logo" class="logo"> <!-- Use the actual file name -->
  </header>

  <!-- Cami Sphere as the button -->
  <div id="cami-sphere" style="position: fixed; bottom: 20px; right: 20px; width: 60px; height: 60px; background-color: #4CAF50; border-radius: 50%; display: flex; justify-content: center; align-items: center; color: white; cursor: pointer; font-size: 16px; text-align: center; z-index: 9999;">
    <span style="font-weight: bold;">Cami</span>
  </div>
```

```
  <!-- Manual Stop Button -->
  <button id="stop-btn" style="position: fixed; bottom: 20px; left: 20px; background-color: red;
color: white; padding: 10px 15px; border: none; border-radius: 5px; cursor: pointer; z-index: 9999;">
    Stop Cami
  </button>

  <!-- Link to the JS file that includes Cami's functionality -->
  <script src="/cami.js"></script>




  <div class="container">
    <h1>Registered Blood Donors</h1>
    <table class="blood-donor-table">
      <thead>
        <tr>
          <th>Name</th>
          <th>Blood Group</th>
          <th>Location</th>
          <th>Email</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody id="donorTableBody">
        <!-- Rows will be populated here dynamically -->
      </tbody>
    </table>
  </div>

  <script>
    async function loadDonors() {
      const response = await fetch('/donors');
```

```
    const donors = await response.json();
    const donorTableBody = document.getElementById('donorTableBody');

    donors.forEach(donor => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${donor.donorName}</td>
            <td>${donor.bloodGroup}</td>
            <td>${donor.location}</td>
            <td>${donor.email}</td> <!-- Display email correctly -->
            <td>
                <button onclick="deleteDonor('${donor.donorName}')">Delete</button>
            </td>
        `;
        donorTableBody.appendChild(row);
    });
}

async function deleteDonor(donorName) {
    const response = await fetch(`/donors/${donorName}`, {
        method: 'DELETE'
    });

    if (response.ok) {
        alert(`Donor ${donorName} deleted successfully.`);
        location.reload(); // Reload the page to update the donor list
    } else {
        alert('Failed to delete donor. Donor may not exist.');
    }
}
```

```
        // Load donors when the page is loaded
        window.onload = loadDonors;
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Send Blood Donation Alert</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <img src="images\logo.png" alt="Logo" class="logo"> <!-- Use the actual file name -->
    </header>

    <!-- Cami Sphere as the button -->
    <div id="cami-sphere" style="position: fixed; bottom: 20px; right: 20px; width: 60px; height: 60px; background-color: #4CAF50; border-radius: 50%; display: flex; justify-content: center; align-items: center; color: white; cursor: pointer; font-size: 16px; text-align: center; z-index: 9999;">
        <span style="font-weight: bold;">Cami</span>
    </div>

    <!-- Manual Stop Button -->
    <button id="stop-btn" style="position: fixed; bottom: 20px; left: 20px; background-color: red; color: white; padding: 10px 15px; border: none; border-radius: 5px; cursor: pointer; z-index: 9999;">
        Stop Cami
```

```
    </button>

    <!-- Link to the JS file that includes Cami's functionality -->
    <script src="/cami.js"></script>



    <div class="container">
        <h1>Send Blood Donation Alert</h1>
        <form id="bloodDonationAlertForm">
            <label for="alertMessage">Alert Message:</label>
            <textarea id="alertMessage" name="alertMessage" rows="4" required></textarea>

            <label for="bloodGroup">Blood Group:</label>
            <select id="bloodGroup" name="bloodGroup" required>
                <option value="" disabled selected>Select the Blood Group</option>
</select>

            <button type="submit">Send Alert</button>
        </form>
    </div>

    <script>
        document.getElementById('bloodDonationAlertForm').onsubmit = async function(event) {
            event.preventDefault();
            const alertMessage = document.getElementById('alertMessage').value;
            const bloodGroup = document.getElementById('bloodGroup').value;

            const response = await fetch('/alert', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ bloodGroup, message: alertMessage }),
```

```
        });

        const result = await response.text();
        alert(result);
      };
  </script>
</body>
</html>



// Initialize Speech Recognition
const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;
const recognition = new SpeechRecognition();
recognition.lang = 'en-US';
recognition.continuous = true;
recognition.interimResults = false;

let isListening = false;
let assistantSpeaking = false;
let hasIntroduced = false; // Flag for introduction

// Start Cami interaction on button click
document.getElementById('camiButton').onclick = function () {
  if (!isListening && !assistantSpeaking) {
    isListening = true;
    document.getElementById('stopButton').style.display = 'block';
    startInteraction();
  }
};
```

```
// Introduction and listening starter
function startInteraction() {
    if (!hasIntroduced) {
        speak("Hello, I am Cami, your assistant. How can I assist you today?");
        hasIntroduced = true;
    }
    setTimeout(() => {
        startListening();
    }, 1000); // Small delay before listening
}

// Start listening
function startListening() {
    if (!assistantSpeaking && !recognition.active) {
        recognition.start();
    }
}

// Speak function with callback
function speak(response) {
    const speech = new SpeechSynthesisUtterance(response);
    assistantSpeaking = true;
    const voice = window.speechSynthesis.getVoices().find(voice => voice.name === "Google UK
English Female");
    speech.voice = voice || null;

    speech.onend = function () {
        assistantSpeaking = false;
        if (isListening) {
            startListening(); // Resume listening after speaking
        }
```

```
    };
    window.speechSynthesis.speak(speech);
}

// Handle voice recognition results
recognition.onresult = function (event) {
    const userInput = event.results[0][0].transcript.toLowerCase().trim();
    console.log("User said: " + userInput);
    handleUserInput(userInput);
};

// Process user input with detailed responses
function handleUserInput(input) {
    let response = "I didn't quite catch that. Could you try again?";

    if (/log ?in/.test(input)) {
        response = "Redirecting you to the login page.";
        navigateTo("/login", response);
    } else if (/register( user)?/.test(input)) {
        response = "Redirecting you to the user registration page.";
        navigateTo("/register", response);
    } else if (/register hospital/.test(input)) {
        response = "Redirecting you to the hospital registration page.";
        navigateTo("/hospitalregister", response);
    } else if (/view hospitals/.test(input)) {
        response = "Here are the available hospitals.";
        navigateTo("/hospitals", response);
    } else if (/register blood donor/.test(input)) {
        response = "Redirecting you to the blood donor registration page.";
        navigateTo("/donor_register", response);
    } else if (/view blood donors/.test(input)) {
```

```
        response = "Showing the list of registered blood donors.";
        navigateTo("/view_donors", response);
    } else if (/send alert/.test(input)) {
        response = "Sending alert via SMS.";
        speak(response);
        // Add SMS functionality here if required
    } else if (/register ambulance/.test(input)) {
        response = "Redirecting you to the ambulance registration page.";
        navigateTo("/ambulance_register", response);
    } else if (/view ambulances/.test(input)) {
        response = "Here is the list of registered ambulances.";
        navigateTo("/ambulance_list", response);
    } else if (/book ambulance/.test(input)) {
        response = "Redirecting you to the ambulance booking page.";
        navigateTo("/Ambulance_booking", response);
    } else if (/log ?out/.test(input)) {
        response = "Logging you out.";
        navigateTo("/logout", response);
    } else if (/stop/.test(input)) {
        stopAssistant();
    } else {
        speak(response); // Speak generic response if command not matched
    }
}

// Function to handle navigation with delay for response
function navigateTo(url, response) {
    console.log("Attempting to navigate to: " + url);
    speak(response);
    setTimeout(() => {
        console.log("Redirecting now to: " + url);
```

```
        window.location.href = `http://localhost:5000${url}`; // Added full path
    }, 1000); // Delay for speech to finish before navigation
}

// Stop assistant completely
function stopAssistant() {
    recognition.stop();
    isListening = false;
    assistantSpeaking = false;
    hasIntroduced = false; // Reset introduction for next session
    speak("Alright, stopping now.");
    document.getElementById('stopButton').style.display = 'none';
}

// Stop button functionality
document.getElementById('stopButton').onclick = stopAssistant;

// Handle errors during speech recognition
recognition.onerror = function (event) {
    console.error("Speech recognition error: ", event.error);
    if (event.error === 'not-allowed' || event.error === 'service-not-allowed') {
        alert("Microphone access is blocked. Please allow microphone access in your browser settings.");
        stopAssistant();
    } else if (event.error === 'no-speech') {
        speak("I didn't catch that. Could you please repeat?");
    }
};

// Reset flags when recognition ends
recognition.onend = function () {
```

```
    if (isListening && !assistantSpeaking) {
        setTimeout(() => recognition.start(), 500); // Small delay to prevent overlap
    }
};
```