DBMS MINI PROJECT
Final Project Report

Course: UE23CS351A-Database Management System
Academic year: 2025-26

REFUGEE MANAGEMENT SYSTEM

Team Details:

NAME:	SRN:
ANVITA KULKARNI	PES1UG23CS092
ANUSHKA GUPTA	PES1UG23CS088

TABLE OF CONTENTS

1. ABSTRACT:

This project implements a **Refugee Management System** using a relational database. Its purpose is to efficiently track and manage key entities in refugee aid, including **Refugees**, the **Camps** they reside in, the **Volunteers** assisting them, and the **organisations** and **Donors** providing support. The system is designed to manage essential data operations (CRUD), track aid distribution, assign volunteers to camps, and provide critical data insights (like camp occupancy and total aid distributed) through stored procedures, functions, and complex queries.

2. User requirement specification

2.1 Purpose of the Project

The purpose of the Refugee Management System (RMS) is to replace inefficient, manual record-keeping in humanitarian aid camps with a centralised, digital ecosystem. The primary goal is to streamline operations by connecting all stakeholders, that is, Camp Managers, Volunteers, Refugees, and Donors, onto a single platform.

By digitising these processes, the project aims to ensure transparency in aid distribution, optimise resource allocation, and prevent camp overcrowding through intelligent automation. Ultimately, the system serves to dignify the refugee experience by providing them with direct access to their own data and integration progress, while giving administrators the real-time analytics needed to make critical decisions.

2.2 Scope of the Project

The scope of this project encompasses the full lifecycle of refugee camp management, from the initial registration of a refugee to their daily aid consumption and eventual integration or departure. It includes the development of a relational database backend to store complex relationships between camps, inventories, and personnel, as well as a

web-based frontend with Role-Based Access Control (RBAC) for five distinct user types.

Specifically, the system covers the management of camp capacities, volunteer scheduling based on skills, tracking of financial and in-kind donations, and the granular logging of aid distribution (food, medicine, etc.). The scope is limited to the operational data management of the camps and does not extend to external legal processing of asylum applications or physical infrastructure construction.

2.3 Functional Requirements

1. Role-Based Authentication The system shall allow users to log in via a secure portal, automatically redirecting them to their specific dashboard (Admin, Manager, Volunteer, Refugee, or Donor) based on their credentials.

2. Refugee Registration & Auto-Assignment The system shall allow Camp Managers to register new refugees and automatically assign them to the camp with the most available capacity to prevent overcrowding.

3. Camp Capacity Monitoring The system shall provide a real-time dashboard for Administrators to view the current occupancy versus capacity for all camps, highlighting camps that are nearing full status.

4. Volunteer Domain Management The system shall allow Volunteers to list their specific skills (e.g., Medical, Education) and allow Managers to search for and assign volunteers to camps that require those specific skills.

5. Inventory Management & Forecasting The system shall track the quantity of resources (Medicine, Food, Hygiene Kits) at each camp and provide a forecast view to alert Admins when stock levels are critical.

6. Aid Distribution Logging The system shall allow Volunteers or Managers to record the distribution of specific aid items to specific refugees, automatically updating the camp's inventory count upon submission.

7. Refugee Integration Journey The system shall provide a self-service portal for Refugees to view a visual progress bar of their integration milestones (e.g., Registration → Health Check → Skills Training).

8. Personal Aid History The system shall allow Refugees to view a read-only history of all food, medicine, and supplies they have received from the camp for transparency.

9. Donation Tracking The system shall allow Donors to submit donation records (financial or in-kind) and view a history of their past contributions, along with the organisation's impact statistics.

10. Admin Audit Trail The system shall provide a comprehensive audit log for Administrators, detailing recent activities such as new registrations, aid distributed, and login events for security and accountability.

3. List of Software/Tools/Programming languages used:

Programming Languages:

- **SQL (Structured Query Language):** Used for Database definition (DDL), manipulation (DML), creating stored procedures, triggers, and handling complex queries.
- **JavaScript (ES6+):** Used for both client-side application logic (DOM manipulation, form handling) and server-side development (via Node.js).
- **HTML5 & CSS3:** Used for the front-end structure, creating the unified dashboard UI, and styling the application.

Database System:

- **MySQL:** A Relational Database Management System (RDBMS) used for persistent data storage of refugees, camps, and volunteers.

Server Technology:

- **Node.js:** The JavaScript runtime environment used to build the backend API.
- **Express.js:** A minimalist web application framework used for building the REST API endpoints.

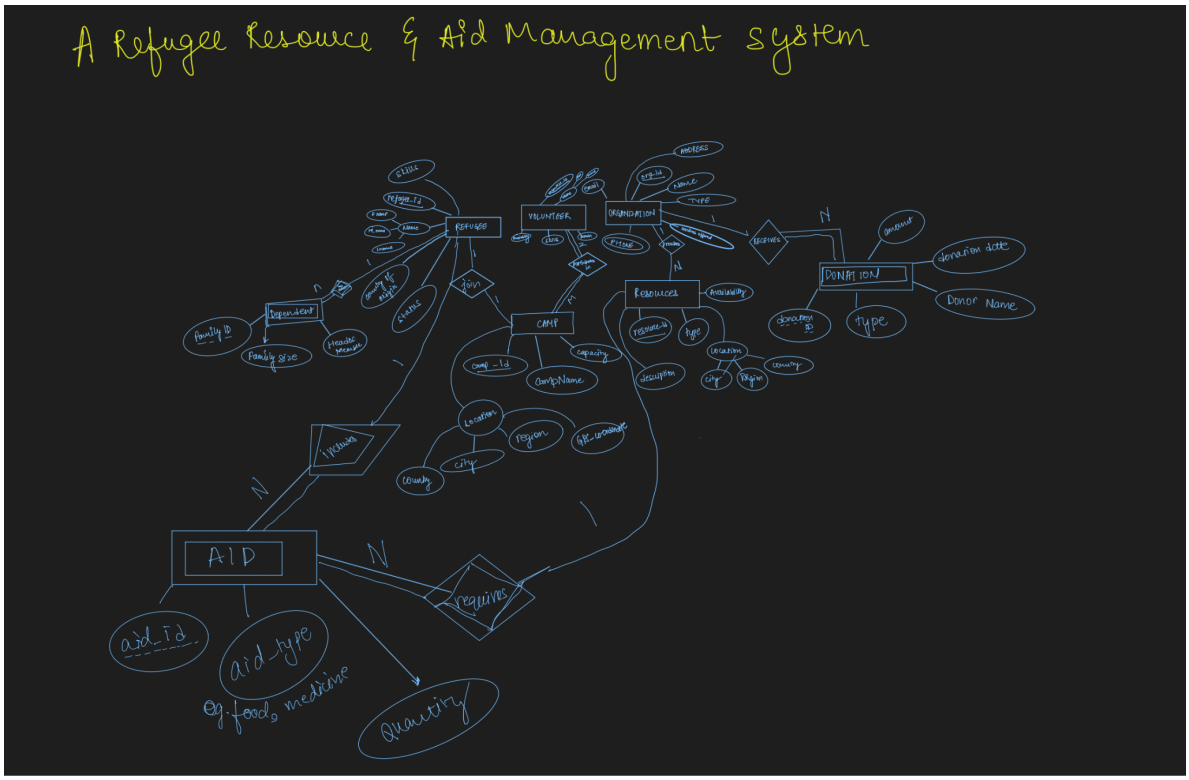
Key Libraries & Modules:

- **mysql2:** An efficient, promise-based driver used for connecting the Node.js backend to the MySQL database.
- **bcrypt:** A library used for securely hashing and verifying user passwords (security).
- **cors:** Middleware used to enable Cross-Origin Resource Sharing (CORS) between the frontend and backend servers.

Development Tools:

- **npm (Node Package Manager):** Used to manage project dependencies and libraries.
- **VS Code Live Server:** A tool used for serving static front-end files during the development phase.
- **localStorage API:** A browser API used for client-side state persistence (used here as a fallback for demo modes).

4. ER DIAGRAM



5. RELATIONAL SCHEMA

6. DDL COMMANDS

```sql
CREATE DATABASE refugee_management_system;
USE refugee_management_system;

-- ------------------------------------------
-- Table: Camp
-- Purpose: Stores information about refugee camps.
-- ------------------------------------------
CREATE TABLE Camp (
    camp_id INT AUTO_INCREMENT PRIMARY KEY,
    camp_name VARCHAR(255) NOT NULL,
    city VARCHAR(100),
    capacity INT NOT NULL CHECK (capacity > 0), -- Requirement: CHECK constraint
    current_occupancy INT DEFAULT 0 -- Requirement: DEFAULT constraint
);

-- ------------------------------------------
-- Table: Refugee
-- Purpose: Stores details of individual refugees.
-- ------------------------------------------
CREATE TABLE Refugee (
    refugee_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100) NULL,
    last_name VARCHAR(100) NOT NULL,
    date_of_birth DATE,
    country_of_origin VARCHAR(100),
    status ENUM('Registered', 'Asylum Seeker', 'Approved', 'Departed')NOT NULL,
    skills VARCHAR(255),
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    camp_id INT,
    profile_document BLOB,
    FOREIGN KEY (camp_id) REFERENCES Camp(camp_id) ON DELETE SET NULL
);
```

```sql
-- ------------------------------------------
-- Table: Volunteer
-- Purpose: Stores information about volunteers.
-- ------------------------------------------
CREATE TABLE Volunteer (
    volunteer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    domain TEXT, -- e.g., "Medical, Translation, Logistics"
    availability ENUM('Full-time', 'Part-time', 'On-call') DEFAULT 'On-call'

);

-- ------------------------------------------
-- Table: Volunteer_Camp_Assignment (Junction Table)
-- Purpose: Manages the many-to-many relationship between Volunteers and Camps.
-- ------------------------------------------
CREATE TABLE Volunteer_Camp_Assignment (
    assignment_id INT AUTO_INCREMENT PRIMARY KEY,
    volunteer_id INT,
    camp_id INT,
    start_date DATE,
    end_date DATE,
    FOREIGN KEY (volunteer_id) REFERENCES Volunteer(volunteer_id) ON DELETE CASCADE, -- If a volunteer is deleted, their ass
    FOREIGN KEY (camp_id) REFERENCES Camp(camp_id) ON DELETE CASCADE -- If a camp is deleted, its assignments are removed.
);
-- ------------------------------------------
-- Table: Donor
-- Purpose: Stores information about individuals or entities who donate.
-- ------------------------------------------
CREATE TABLE Donor (
    donor_id INT AUTO_INCREMENT PRIMARY KEY,
    donor_name VARCHAR(255) NOT NULL,
    donor_type ENUM('Individual', 'Corporation', 'NGO')
);
```

```sql
-- ------------------------------------------
-- Table: Organization
-- Purpose: Stores details of aid organizations.
-- ------------------------------------------
CREATE TABLE Organization (
    organization_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    type VARCHAR(100), -- e.g., "NGO", "Governmental"
    address TEXT,
    email VARCHAR(255) UNIQUE,
    services_offered LONGTEXT
);

-- ------------------------------------------
-- Table: Organization_Phone
-- Purpose: Stores multiple phone numbers for each organization.
-- ------------------------------------------
CREATE TABLE Organization_Phone (
    organization_id INT,
    phone_number VARCHAR(20) NOT NULL,
    PRIMARY KEY (organization_id, phone_number), -- Composite key prevents duplicate numbers for the same org
    FOREIGN KEY (organization_id) REFERENCES Organization(organization_id) ON DELETE CASCADE -- If the organization is dele
);
```

```sql
-- Table: Donation
-- Purpose: Tracks donations made by donors to organizations.
-- ---------------------------------------------
CREATE TABLE Donation (
    donation_id INT AUTO_INCREMENT PRIMARY KEY,
    donor_id INT,
    organization_id INT,
    amount DECIMAL(12, 2),
    donation_type VARCHAR(100), -- e.g., "Monetary", "In-Kind"
    donation_date DATE,
    FOREIGN KEY (donor_id) REFERENCES Donor(donor_id) ON DELETE SET NULL, -- Keep donation record even if donor is deleted
    FOREIGN KEY (organization_id) REFERENCES Organization(organization_id) ON DELETE CASCADE
);

-- ---------------------------------------------
-- Table: Aid
-- Purpose: Catalog of available aid types.
-- ---------------------------------------------
CREATE TABLE Aid (
    aid_id INT AUTO_INCREMENT PRIMARY KEY,
    aid_type ENUM('Food', 'Medicine', 'Shelter', 'Clothing', 'Hygiene Kit') NOT NULL
);

-- ---------------------------------------------
-- Table: Aid_Distribution (Junction Table)
-- Purpose: Tracks which refugee received what aid, when, and how much.
-- ---------------------------------------------
CREATE TABLE Aid_Distribution (
    distribution_id INT AUTO_INCREMENT PRIMARY KEY,
    refugee_id INT,
    aid_id INT,
    quantity_distributed INT NOT NULL,
    FOREIGN KEY (refugee_id) REFERENCES Refugee(refugee_id) ON DELETE CASCADE,
    FOREIGN KEY (aid_id) REFERENCES Aid(aid_id) ON DELETE CASCADE
);
```

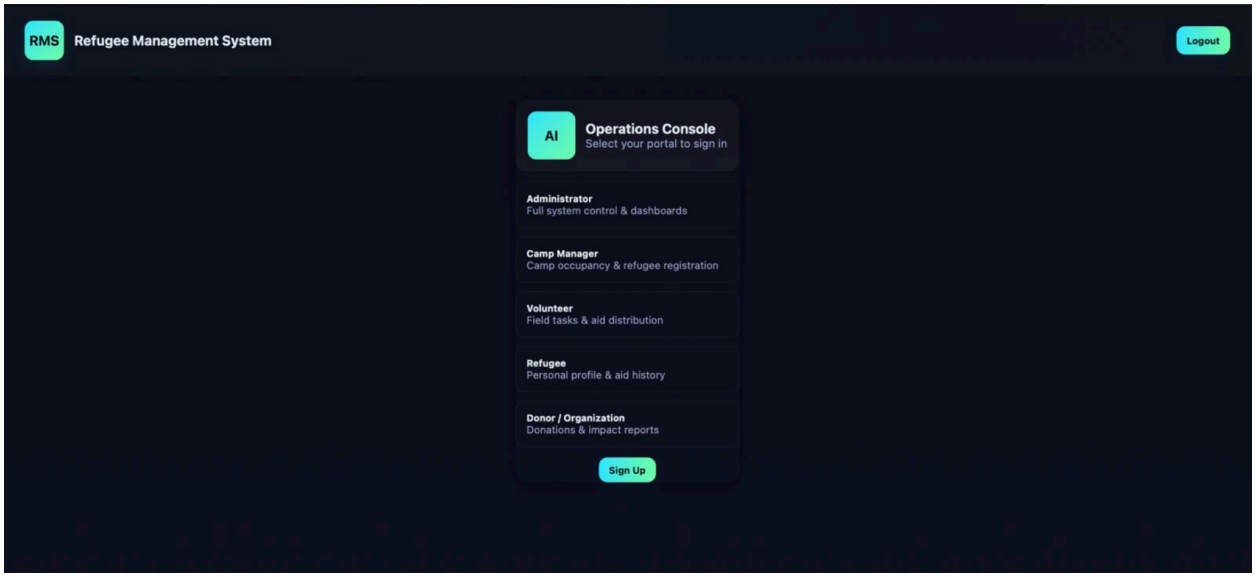
```sql
-- ---------------------------------------------
-- Table: Users
-- Purpose: Central user authentication and role management
-- ---------------------------------------------
CREATE TABLE Users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('Admin', 'Manager', 'Volunteer', 'Refugee', 'Donor', 'Org') NOT NULL,

    -- Foreign key references to respective profile tables
    volunteer_id INT NULL,
    refugee_id INT NULL,
    donor_id INT NULL,
    organization_id INT NULL,

    -- Optional metadata for audit/logging
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL,

    FOREIGN KEY (volunteer_id) REFERENCES Volunteer(volunteer_id) ON DELETE CASCADE,
    FOREIGN KEY (refugee_id) REFERENCES Refugee(refugee_id) ON DELETE CASCADE,
    FOREIGN KEY (donor_id) REFERENCES Donor(donor_id) ON DELETE CASCADE,
    FOREIGN KEY (organization_id) REFERENCES Organization(organization_id) ON DELETE CASCADE
);
```

7. CRUD Operations

The create, read, update & delete operations are performed below

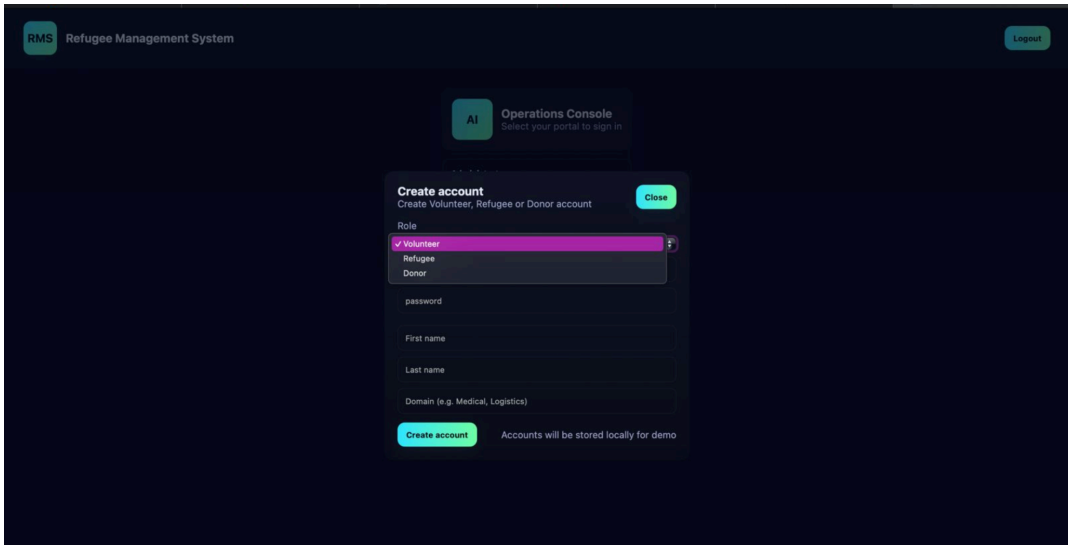
Landing Page & Portal Selection

- Multi-role portal buttons (Admin, Manager, Volunteer, Refugee, Donor)
- Login modal
- Sign-up button



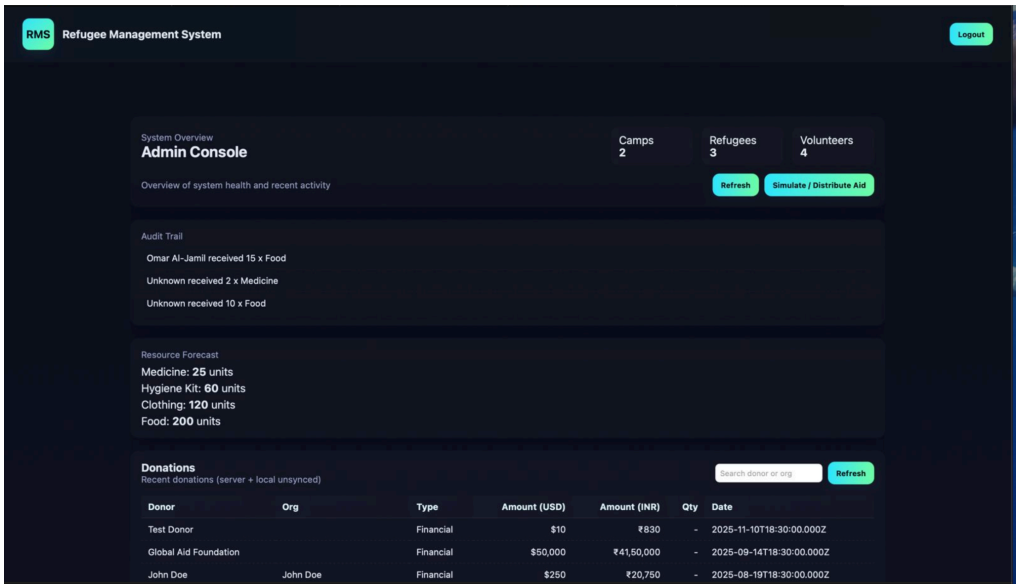
User Authentication

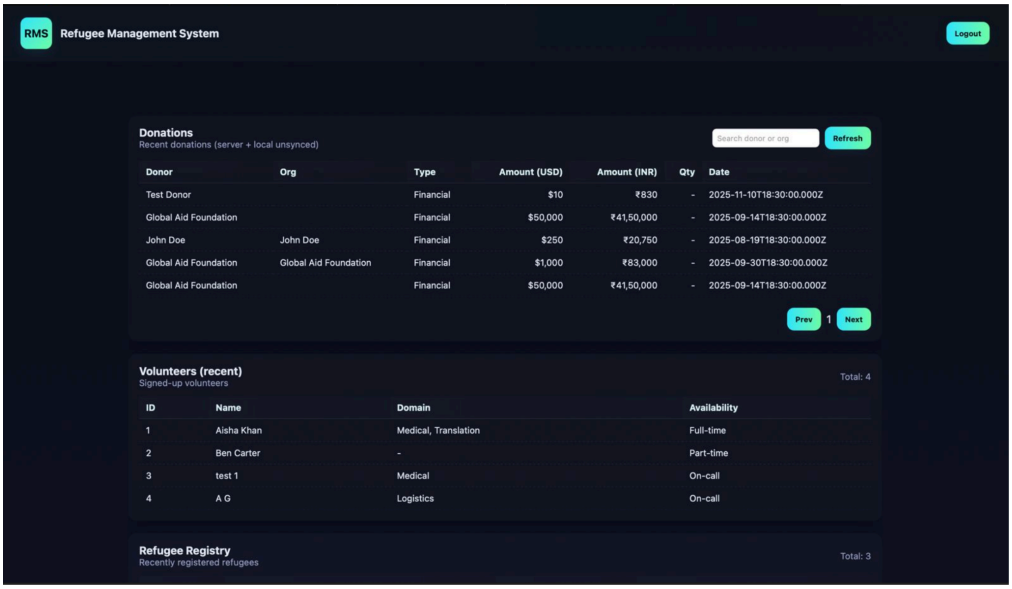
- Login modal with username/password
- Role-based authentication
- Sign-up form with dynamic fields based on role selection

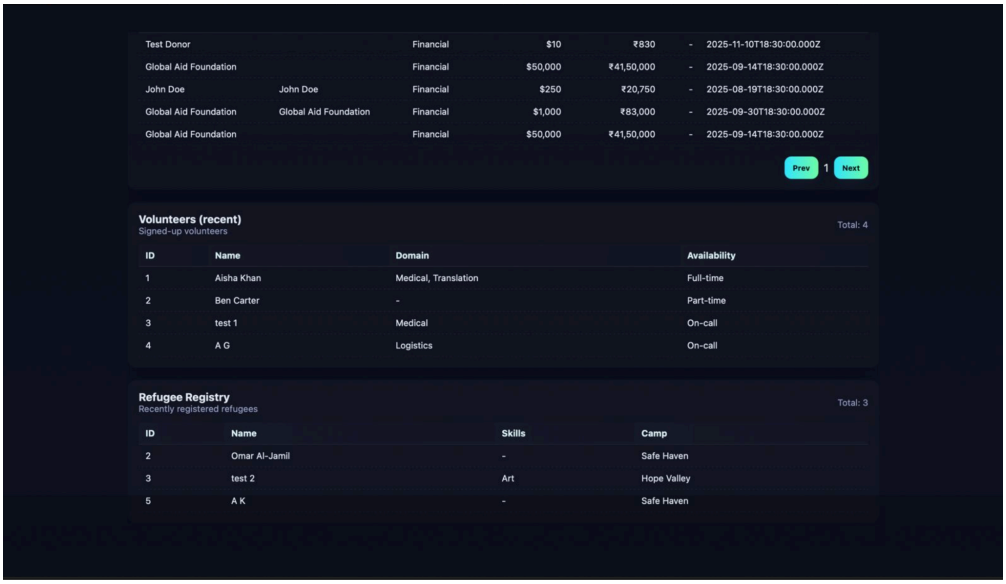


Admin Dashboard

- KPI cards (Camps, Refugees, Volunteers counts)
- Audit trail log
- Resource forecast with stock levels
- Refresh and Simulate Aid buttons

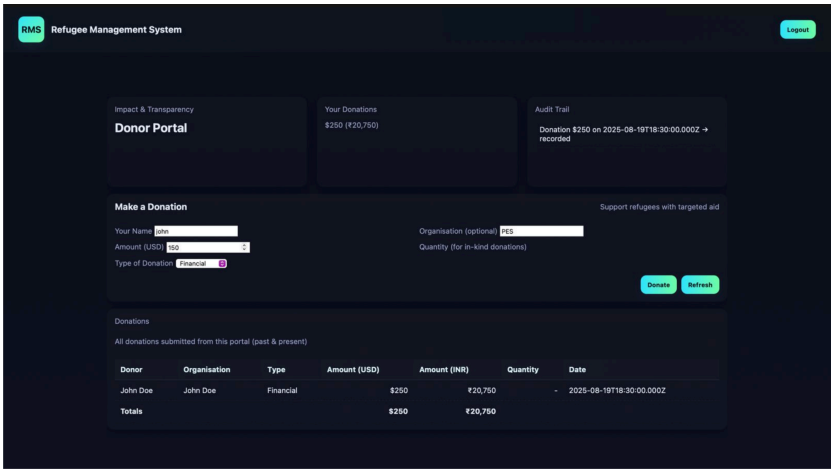




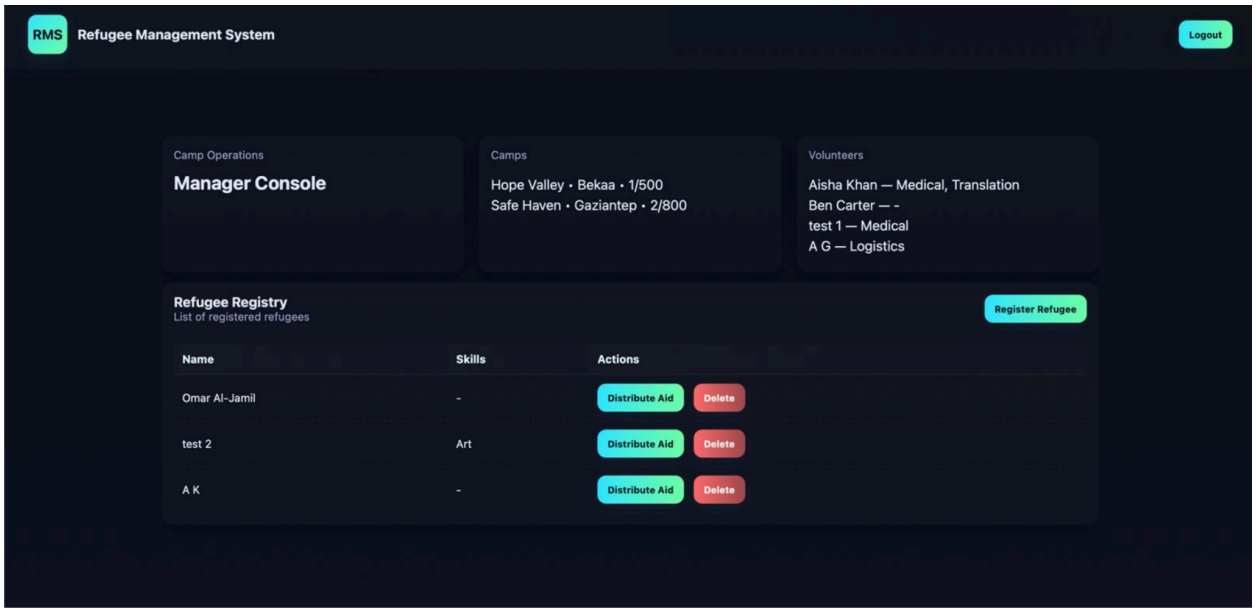


Donor Portal - Donation Form

- Donor information fields
- Donation type dropdown
- Conditional fields (Amount for Financial, Quantity for In-kind)
- USD to INR conversion
- Donate and Refresh buttons



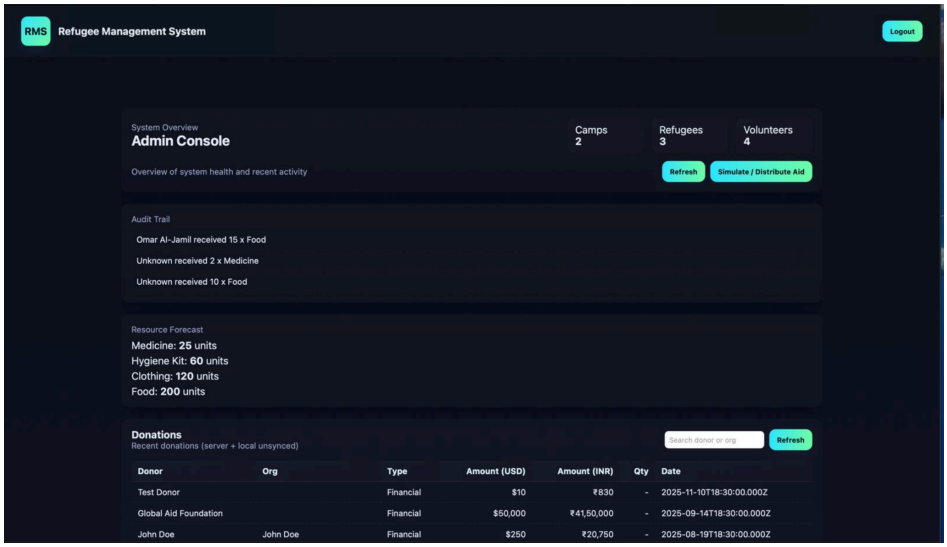
Camp Manager - Refugee Registration

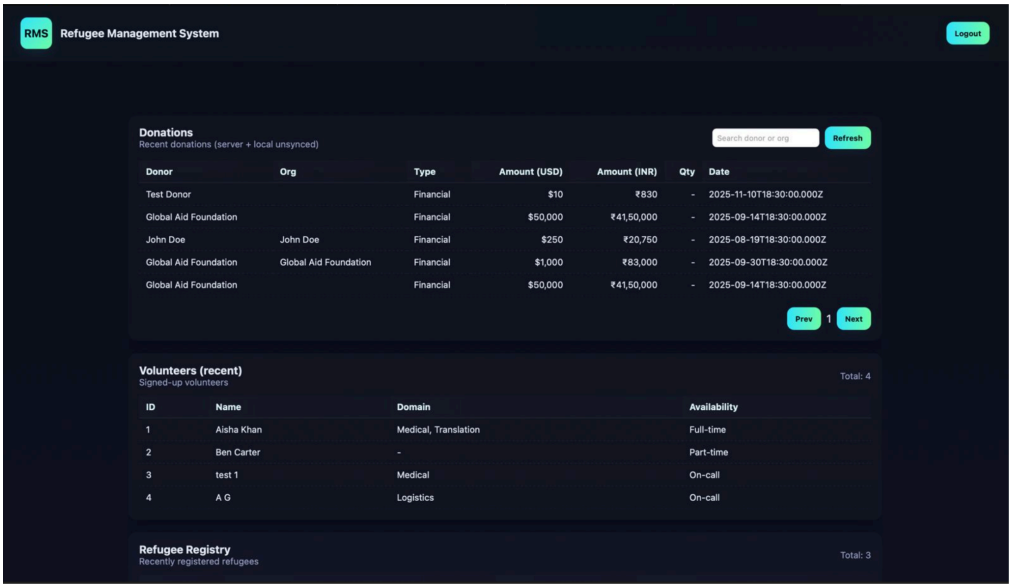


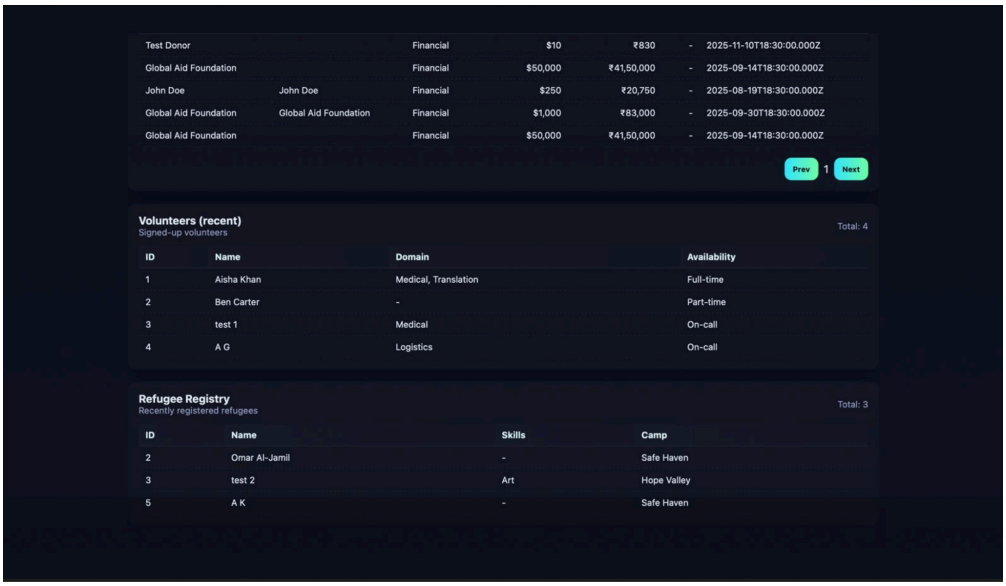
8. List of functionalities/features of the application

Admin Dashboard

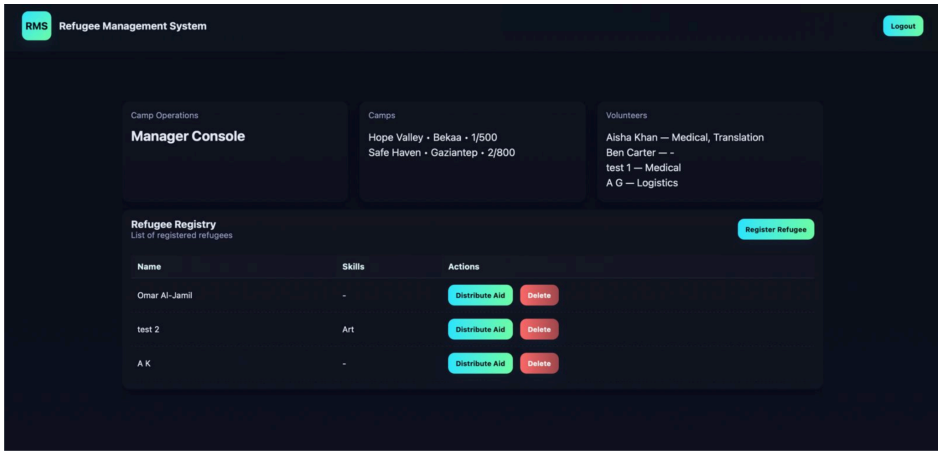
- KPI cards (Camps, Refugees, Volunteers counts)
- Audit trail log
- Resource forecast with stock levels
- Refresh and Simulate Aid buttons





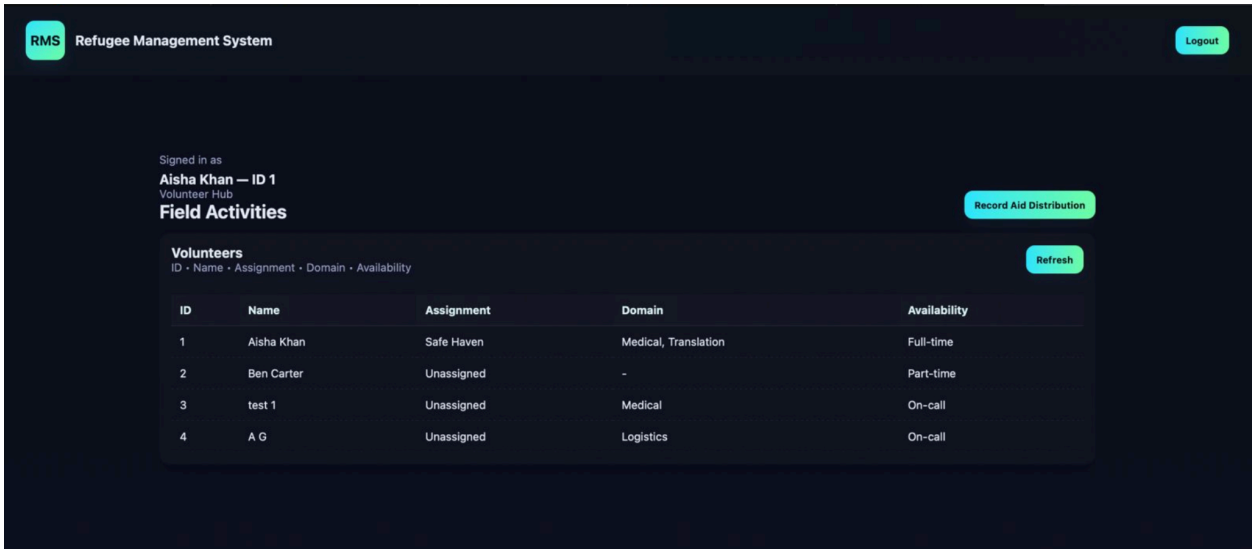


Camp Manager - Refugee Registration



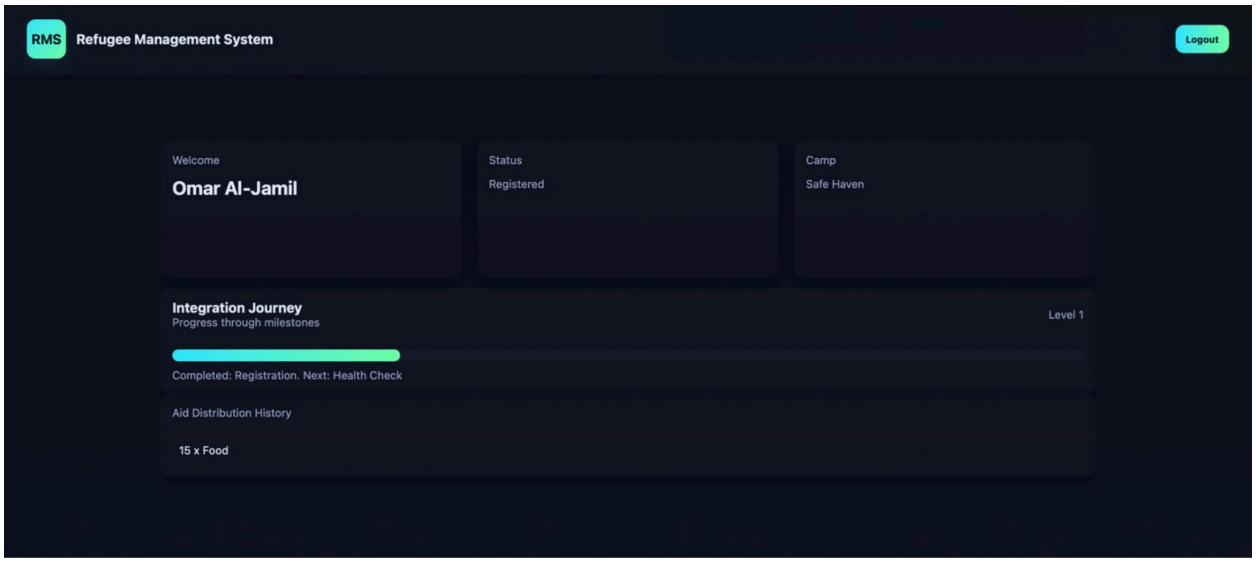
Volunteer Dashboard

- Personal volunteer information
- Full volunteer roster table
- Record Aid Distribution button
- Shows assignments and availability

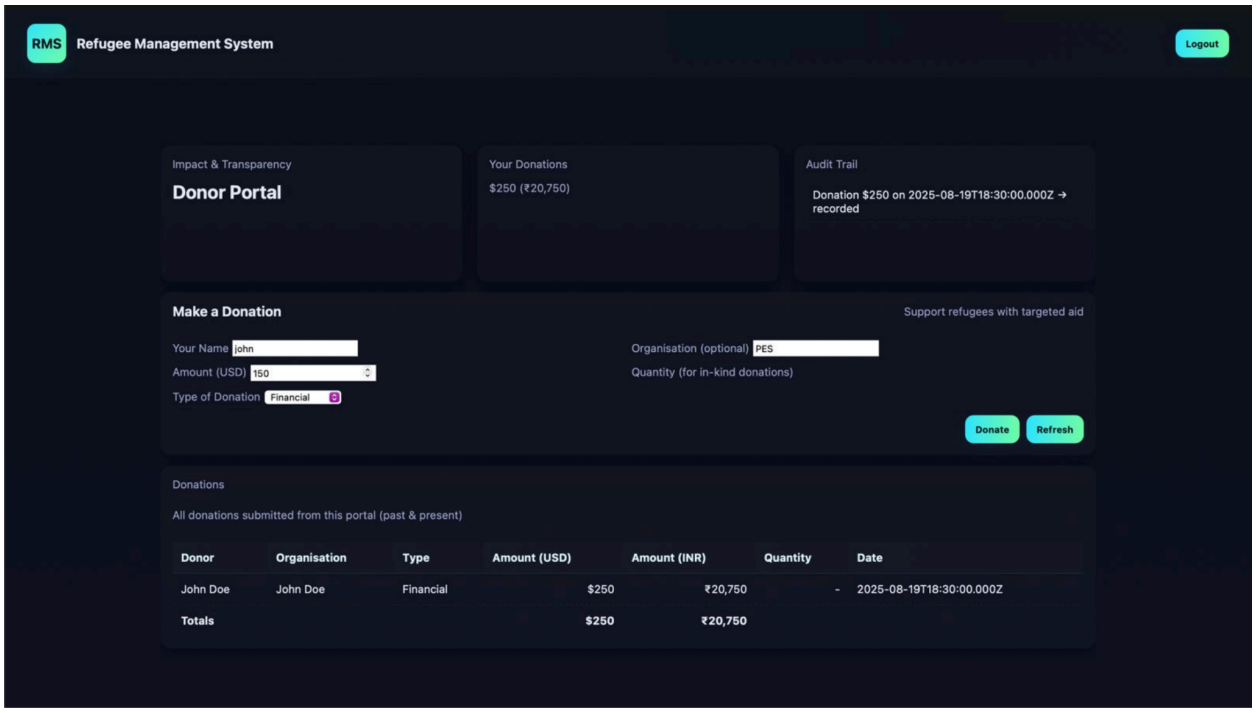


Refugee Portal

- Personal profile card (Name, Status, Camp)
- Progress tracker with visual bar
- Milestone checklist (Registration, Health Check, Skills Training, Employment)
- Level indicator (1-3)



Donor Portal
- Donor information fields
- Donation type dropdown
- Conditional fields (Amount for Financial, Quantity for In-kind)
- USD to INR conversion
- Donate and Refresh buttons



9. Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

All of these components are present and functional within your test.sql file:

- **Trigger:** after_refugee_camp_change (Updates Camp occupancy on new refugee insertion).
- **Stored Procedure:** AddNewRefugee (Inserts a new refugee and automatically assigns them to the camp with the most available capacity).
- **Function:** GetTotalAidDistributed (Calculates the total quantity of a specified type of aid distributed).

- **Join Query (Example):** Retrieves volunteers with a 'Medical' domain assigned to 'Safe Haven' camp.
- **Aggregate Query (Example):** Calculates the total monetary donations grouped by donor name.
- **Nested Query (Example):** Retrieves refugees who have **not** received 'Medicine' aid.

10. Code snippets for invoking the Procedures/Functions/Trigger

1.Procedures:

```
app.post('/api/refugees', async (req, res) => {
  const [result] = await conn.query(
    'CALL sp_RegisterRefugee(?, ?, ?, ?, ?, ?, ?, ?, @refugee_id)',
    [p.first_name, p.middle_name, p.last_name, ...]
  );
  const [[{ '@refugee_id': refugeeId }]] = await conn.query('SELECT @refugee_id');
```

```
app.post('/api/donations', async (req, res) => {
  const [result] = await conn.query(
    'CALL sp_RecordDonation(?, ?, ?, ?, ?, ?, ?, @donation_id)',
    [p.donor_id, p.donor_name, p.organization_name, ...]
  );
  const [[{ '@donation_id': donationId }]] = await conn.query('SELECT @donation_id');
```

```
app.get('/api/camps/:id/refugees', async (req, res) => {
  const campId = parseInt(req.params.id);
  try {
    const [rows] = await pool.query('CALL sp_GetRefugeesByCamp(?)', [campId]);
    res.json({ success: true, refugees: rows[0] });
  } catch (err) {
    console.error('Get refugees by camp error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.get('/api/donations/range', async (req, res) => {
  const { start_date, end_date } = req.query;
  if (!start_date || !end_date) return res.status(400).json({ error: 'missing_dates' });

  try {
    const [rows] = await pool.query('CALL sp_GetDonationsByDateRange(?, ?)', [start_date, end_date]);
    res.json({ success: true, donations: rows[0] });
  } catch (err) {
    console.error('Get donations by date range error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.delete('/api/refugees/:id', async (req, res) => {
  await conn.query('CALL sp_DeleteRefugee(?)', [id]);
```

2. Functions

```
app.get('/api/donors/:id/total', async (req, res) => {
  const donorId = parseInt(req.params.id);
  try {
    const [rows] = await pool.query(
      'SELECT donor_id, donor_name, fn_CalculateTotalDonations(donor_id) AS total_donated FROM Donors WHERE donor_id = ?',
      [donorId]
    );
    if (rows.length === 0) return res.status(404).json({ error: 'not_found' });
    res.json({ success: true, donor: rows[0] });
  } catch (err) {
    console.error('Get donor total error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.get('/api/donations/converted', async (req, res) => {
  try {
    const [rows] = await pool.query(`
      SELECT
        donation_id,
        amount AS usd_amount,
        fn_ConvertUSDtoINR(amount) AS inr_amount,
        donation_date,
        donation_type
      FROM Donations
      WHERE donation_type = 'Financial' AND amount IS NOT NULL
    `);
    res.json({ success: true, donations: rows });
  } catch (err) {
    console.error('Get converted donations error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.get('/api/camps/occupancy', async (req, res) => {
  try {
    const [rows] = await pool.query(`
      SELECT
        camp_id,
        camp_name,
        city,
        capacity,
        current_occupancy,
        fn_GetCampOccupancyRate(camp_id) AS occupancy_rate
      FROM Camps
      ORDER BY occupancy_rate DESC
    `);
    res.json({ success: true, camps: rows });
  } catch (err) {
    console.error('Get camp occupancy error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.get('/api/camps/:id/count', async (req, res) => {
  const campId = parseInt(req.params.id);
  try {
    const [[result]] = await pool.query(
      'SELECT fn_CountRefugeesByCamp(?) AS refugee_count',
      [campId]
    );
    res.json({ success: true, camp_id: campId, refugee_count: result.refugee_count });
  } catch (err) {
    console.error('Count refugees error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

```
app.get('/api/aid/statistics', async (req, res) => {
  try {
    const [rows] = await pool.query(`
      SELECT
        aid_id,
        aid_type,
        stock AS current_stock,
        fn_GetTotalAidDistributed(aid_id) AS total_distributed
      FROM Aid
    `);
    res.json({ success: true, aid_statistics: rows });
  } catch (err) {
    console.error('Get aid statistics error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

3. Triggers

```
app.post('/api/refugees', async (req, res) => {
  const [result] = await conn.query(
    'CALL sp_RegisterRefugee(?, ?, ?, ?, ?, ?, ?, ?, @refugee_id)',
    [p.first_name, p.middle_name, p.last_name, p.date_of_birth, ...]
  );
  // ⚡ trg_ValidateRefugeeAge fires HERE automatically
  // It validates date_of_birth before insert
}
```

```
app.post('/api/donations', async (req, res) => {
  const [result] = await conn.query(
    'CALL sp_RecordDonation(?, ?, ?, ?, ?, ?, ?, @donation_id)',
    [p.donor_id, p.donor_name, p.organization_name, amount, ...]
  );
  // ⚡ trg_AuditDonation fires HERE automatically
  // It logs the donation to Audit_Log table
}
```

```
app.post('/api/aid/distribute', async (req, res) => {
  await conn.query('CALL sp_DistributeAid(?, ?, ?, @distribution_id)', [refugee_id, aid_id, quantity]);
  // ⚡ trg_ValidateAidQuantity fires HERE automatically
  // It rejects negative or zero quantities
}
```

```
app.delete('/api/refugees/:id', async (req, res) => {
  await conn.query('CALL sp_DeleteRefugee(?)', [id]);
}
```

```
app.get('/api/audit-log', async (req, res) => {
  try {
    const [rows] = await pool.query(
      'SELECT * FROM Audit_Log ORDER BY timestamp DESC LIMIT 50'
    );
    res.json({ success: true, logs: rows });
  } catch (err) {
    console.error('Get audit log error', err);
    res.status(500).json({ error: 'server_error' });
  }
});
```

11. More SQL queries

```
DELIMITER $$

CREATE PROCEDURE AddNewRefugee(
    IN p_first_name VARCHAR(100),
    IN p_last_name VARCHAR(100),
    IN p_date_of_birth DATE,
    IN p_country_of_origin VARCHAR(100)
)
BEGIN
    DECLARE v_camp_id INT;

    -- Find the camp with the most available capacity (capacity - current_occupancy)
    SELECT camp_id INTO v_camp_id
    FROM Camp
    WHERE capacity > current_occupancy
    ORDER BY (capacity - current_occupancy) DESC
    LIMIT 1;

    -- Insert the new refugee with the assigned camp_id
    INSERT INTO Refugee (first_name, last_name, date_of_birth, country_of_origin, status, camp_id)
    VALUES (p_first_name, p_last_name, p_date_of_birth, p_country_of_origin, 'Registered', v_camp_id);

END$$

DELIMITER ;



DELIMITER $$
```

```
CREATE TRIGGER after_refugee_camp_change
AFTER INSERT ON Refugee
FOR EACH ROW
BEGIN
    -- Increase occupancy when a refugee is added to a camp
    IF NEW.camp_id IS NOT NULL THEN
        UPDATE Camp SET current_occupancy = current_occupancy + 1 WHERE camp_id = NEW.camp_id;
    END IF;
END$$

DELIMITER ;


DELIMITER $$

CREATE FUNCTION GetTotalAidDistributed(
    p_aid_type ENUM('Food', 'Medicine', 'Shelter', 'Clothing', 'Hygiene Kit')
)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total_quantity INT;

    SELECT SUM(ad.quantity_distributed) INTO total_quantity
    FROM Aid_Distribution ad
    JOIN Aid a ON ad.aid_id = a.aid_id
    WHERE a.aid_type = p_aid_type;

    RETURN IFNULL(total_quantity, 0);
END$$

DELIMITER ;
```

12 . Git Repo Link:

https://github.com/Anushkag01/Refugee-Management-System.git