Mini project report on

Crowdsourced Disaster Management System (CDMS)

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology in

Computer Science &

Engineering

UE22CS351A – DBMS

Project

Submitted by:

T H MANOJ	**PES2UG22CS612**
SUHAS HEGDE	**PES2UG22CS587**

Under the guidance of
Prof. Nivedita Kasturi
Assistant Professor
PES University

AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Crowdsourced Disaster Management System

is a bonafide work carried out by

T H MANOJ	**PES2UG22CS612**
SUHAS HEGDE	**PES2UG22CS587**

In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2024 – DEC. 2024. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Prof. Nivedita Kasturi
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Crowdsourced Disaster Management System** has been carried out by us under the guidance of **Prof. Nivedita Kasturi, Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2024.

T H MANOJ	**PES2UG22CS612**
SUHAS HEGDE	**PES2UG22CS587**

ABSTRACT

The Crowdsourced Disaster Management System is a project that enhances disaster response through community involvement and data integration. It allows users to report disasters with real-time data on location, severity, and details, facilitating immediate prioritization and response. Donors can contribute funds and resources, while relief camps are managed for capacity and resource availability, ensuring efficient support for affected individuals. Users can send SOS alerts for emergency assistance, and volunteers are assigned tasks within camps, streamlining coordination. This system enables transparent resource tracking, responsive disaster relief, and collaborative engagement among donors, volunteers, and communities, strengthening overall resilience during disasters.

TABLE OF CONTENTS

INTRODUCTION

Natural and man-made disasters can have devastating consequences, disrupting lives, infrastructure, and entire communities. Effective disaster response is crucial in mitigating the impacts of these calamities and ensuring the well-being of affected populations. The Crowdsourced Disaster Management System (CDMS) is a comprehensive platform that aims to revolutionize the way communities, relief organizations, and authorities collaborate during times of crisis.

At the heart of the CDMS is the recognition that disaster response requires a multifaceted approach that harnesses the collective efforts and insights of various stakeholders. By leveraging the power of crowdsourcing and data integration, the system enables seamless coordination and efficient resource allocation, ultimately strengthening overall resilience and emergency preparedness.

One of the key features of the Crowdsourced Disaster Management System is its ability to facilitate real-time disaster reporting. Users can report incidents with detailed information on location, severity, and other critical details. This data is then centralized, allowing for immediate prioritization and response by relevant authorities and relief organizations.

The CDMS empowers donors and volunteers to contribute resources and assistance in a transparent and coordinated manner. Donors can pledge financial support or donate essential supplies, while the system ensures the efficient distribution and utilization of these resources. Relief camps are managed for capacity and resource availability, ensuring that affected individuals and communities receive the necessary aid and support.

The platform also provides a mechanism for users to send SOS alerts in emergency situations, triggering immediate response and deployment of resources. Volunteers are assigned tasks within the relief camps, leveraging their skills and expertise to streamline coordination and enhance the overall effectiveness of the relief efforts.

By integrating real-time disaster reporting, resource mobilization, relief camp management, and volunteer coordination, the Crowdsourced Disaster Management System creates a comprehensive ecosystem that fosters collaborative engagement among all stakeholders. This approach aims to enhance disaster resilience, strengthen community preparedness, and provide timely and effective relief during times of crisis.

Through the CDMS, communities, relief organizations, and authorities can work together to mitigate the devastating impacts of natural and man-made disasters, ultimately building a more resilient and responsive world.

The Crowdsourced Disaster Management System's key features include:

- Real-time disaster reporting and prioritization
- Centralized platform for resource mobilization and distribution
- Efficient management of relief camps and volunteer coordination
- Transparent tracking of resources and relief efforts
- Collaborative engagement among donors, volunteers, and communities

By leveraging the power of crowdsourcing and data integration, the Crowdsourced Disaster Management System aims to enhance disaster resilience, strengthen community preparedness, and provide timely and effective relief during times of crisis.

PROBLEM DEFINITION WITH USER REQUIREMENT SPECIFICATIONS

Problem Definition

In the wake of natural and man-made disasters, the need for swift, coordinated, and resource efficient responses has become more pressing than ever. Current disaster management efforts often face numerous challenges that hamper effective response and relief delivery. Some of the primary issues include:

1. **Inefficient Resource Allocation**:
   - Resources like food, medical supplies, and shelter are crucial in disaster scenarios. However, the lack of a centralized system to manage and track these resources results in duplication, wastage, or shortages where they're needed most. This can also hinder the efficient distribution of resources to areas affected by the disaster.
2. **Limited Coordination Among Volunteers**:
   - While many volunteers are willing to help, the lack of a structured task allocation system limits their impact. Without proper assignment and tracking, volunteers may work in overlapping capacities or be unaware of where their efforts are needed, reducing the overall efficiency of the relief efforts.
3. **Insufficient Transparency in Donation Usage**:
   - Donors contribute resources and funds, but often lack visibility into how their contributions are used. This lack of transparency can discourage future contributions and reduce the trust between donors and relief organizations.
4. **Overloaded Relief Camps**:
   - During disasters, relief camps quickly reach full capacity. Without real-time monitoring of camp capacities, affected individuals might be directed to already full camps, causing further stress and potential safety issues.
5. **Inadequate Communication Channels for Affected Individuals**:
   - Individuals affected by a disaster may need to send SOS alerts or emergency messages for immediate assistance. Current systems often lack a direct and quick mechanism for individuals to communicate their exact needs and locations.

The Crowdsourced Disaster Management System aims to address these challenges by creating an integrated platform that leverages community involvement, real-time reporting, and efficient management of resources and volunteers to ensure rapid, transparent, and organized disaster response and relief.

User Requirement Specifications

To effectively serve the various stakeholders involved in disaster management, the system is designed with the following user requirements in mind:

1. **Disaster Reporting and Monitoring**:
   - **Requirement**: Users must be able to report new disasters by providing information such as location, severity, and description of the disaster. Reports should be timestamped automatically.
   - **Functionality**: The system will allow users (including volunteers, disaster relief agencies, and the public) to report disasters. Each report should be stored in a database and made accessible for monitoring.
2. **Donation Management**:
   - **Requirement**: The system should facilitate the donation of funds or resources and allow donors to specify the disaster they wish to support. o **Functionality**: Donors can contribute by specifying their donor ID, the disaster ID, amount, and resources donated. The system will also timestamp each donation and store it, ensuring transparency and accountability.
3. **Relief Camp Management**:
   - **Requirement**: Relief camp managers must be able to record camp location, capacity, and occupancy levels and link camps to specific disasters.
   - **Functionality**: Each relief camp's information (location, capacity, current occupancy, associated disaster ID) will be updated in real time, providing a clear view of available space at each camp. This information can be used to direct victims to appropriate locations.
4. **Resource Management at Relief Camps**:
   - **Requirement**: Relief camps need to manage resources like food, medical supplies, and water by tracking resource names, quantities, and usage.
   - **Functionality**: Camp managers can log resources and update their quantities based on availability and usage. This data will help ensure that supplies are allocated appropriately and restocked as necessary.

5. **SOS Alert Mechanism**:
    - **Requirement**: Individuals in distress should have a way to send SOS alerts with location information, and disaster reports with high severity will be considered as an SOS Alert.
    - **Functionality**: Affected users can submit SOS alerts that include location and a custom message describing their immediate needs. Alerts are timestamped and sent to relevant agencies and volunteers for rapid response.
6. **User Account Management and Roles**:
    - **Requirement**: All users, including volunteers, donors, should have accounts with role-based access and permissions.
    - **Functionality**: The system will provide registration and login functionalities, allowing users to create accounts and select their roles. Roles will determine the permissions and actions available to each user.
7. **Data Privacy and Security**:
    - **Requirement**: Sensitive user information, such as personal details and donation amounts, must be protected.
    - **Functionality**: The system will include security measures such as data encryption, user authentication, and secure password storage to protect user information.

LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES USED

- **Front-End Development**:

- **React**: JavaScript library used for building dynamic and responsive user interfaces, ensuring a seamless user experience for reporting disasters, managing resources, and sending SOS alerts.
- **HTML5 & CSS3**: Core web technologies used alongside React for structuring and styling the front end, ensuring a user-friendly layout and design.

- **Back-End Development**:

- **Node.js**: JavaScript runtime used to run server-side applications, providing the framework to handle asynchronous operations and manage concurrent requests efficiently.
- **Express.js**: Node.js web application framework used to build the backend API, manage routing, handle requests, and interact with the database.

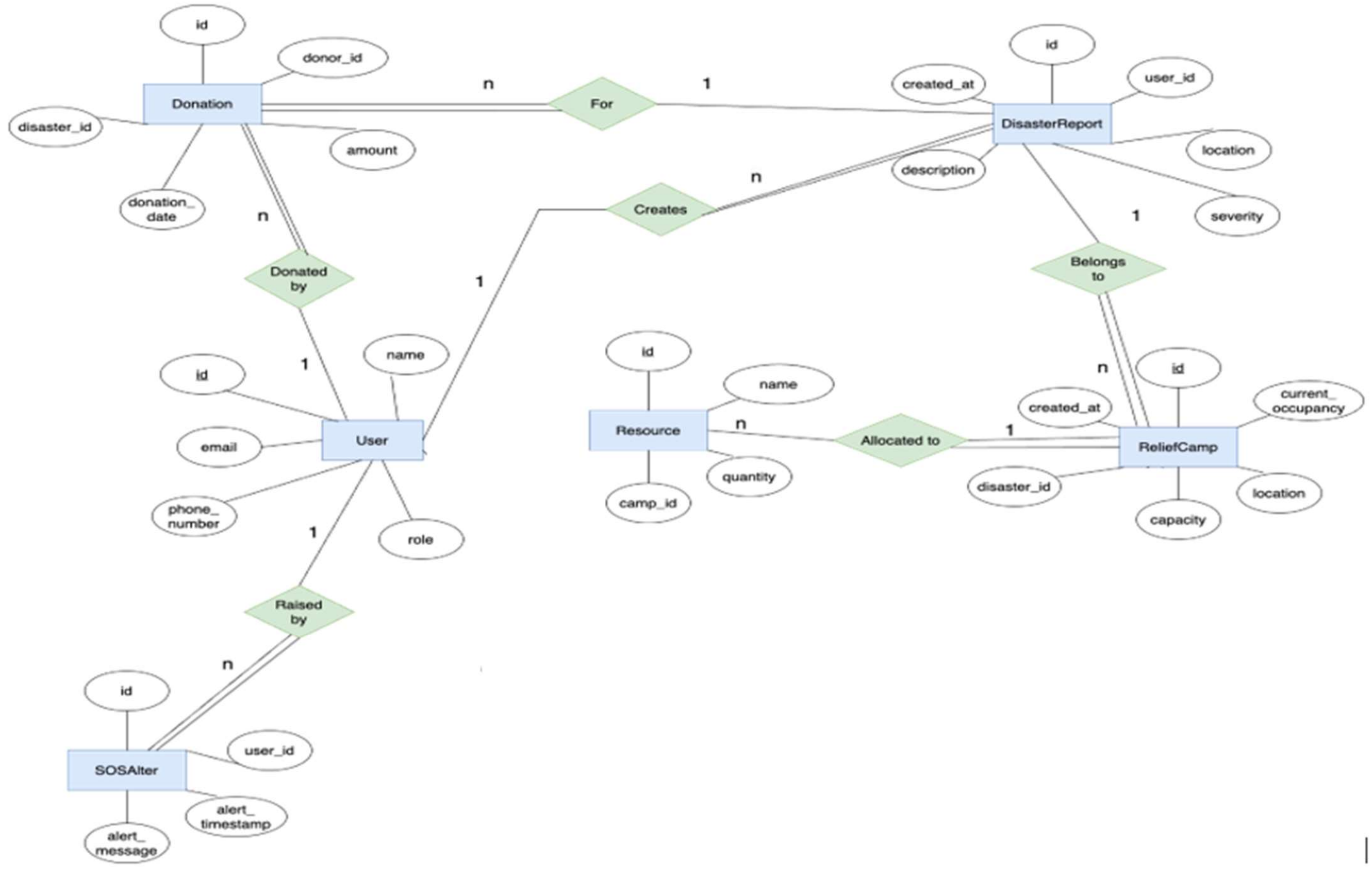
- **Database Management**:

- **SQL Database (MySQL)**: Relational database management system used to store structured data, such as user information, disaster reports, donations, and relief camp details. SQL is used for data retrieval, storage, and manipulation.

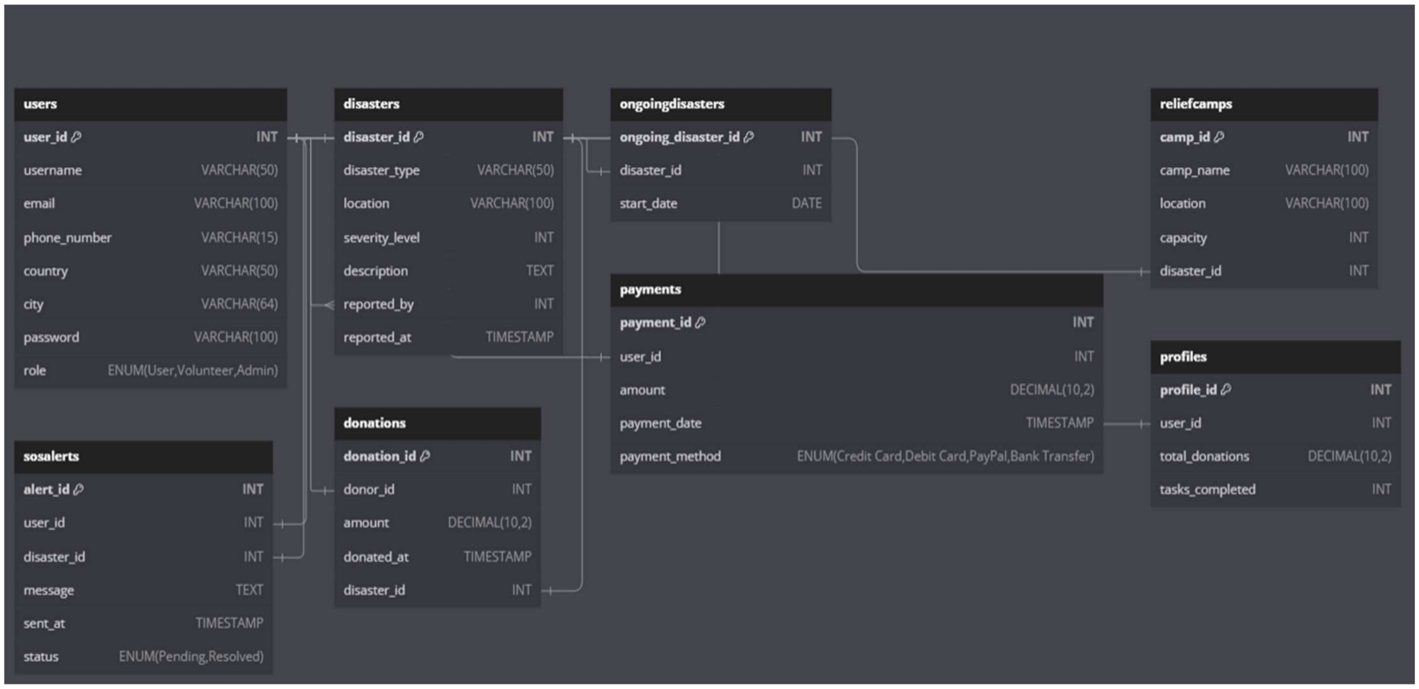
- **Development Environment & Tools**:

- **Visual Studio Code**: IDE used for coding, debugging, and version control integration.
- **Git**: Version control system used for tracking changes in source code, enabling collaboration and version management.
- **GitHub:** Version control hosting and collaboration.

ER Diagram:


Donation
id
donor_id
disaster_id
amount
donation_date
For
n
1
DisasterReport
id
created_at
user_id
location
description
severity
Creates
n
1
Donated by
n
1
Belongs to
1
n
User
id
name
email
phone_number
role
Resource
id
name
quantity
camp_id
Allocated to
n
1
ReliefCamp
id
created_at
current_occupancy
disaster_id
location
capacity
Raised by
1
n
SOSAlter
id
user_id
alert_message
alert_timestamp

ER TO RELATIONAL MAPPING



users	
user_id	INT
username	VARCHAR(50)
email	VARCHAR(100)
phone_number	VARCHAR(15)
country	VARCHAR(50)
city	VARCHAR(64)
password	VARCHAR(100)
role	ENUM(User,Volunteer,Admin)

disasters	
disaster_id	INT
disaster_type	VARCHAR(50)
location	VARCHAR(100)
severity_level	INT
description	TEXT
reported_by	INT
reported_at	TIMESTAMP

ongoingdisasters	
ongoing_disaster_id	INT
disaster_id	INT
start_date	DATE

reliefcamps	
camp_id	INT
camp_name	VARCHAR(100)
location	VARCHAR(100)
capacity	INT
disaster_id	INT

payments	
payment_id	INT
user_id	INT
amount	DECIMAL(10,2)
payment_date	TIMESTAMP
payment_method	ENUM(Credit Card,Debit Card,PayPal,Bank Transfer)

profiles	
profile_id	INT
user_id	INT
total_donations	DECIMAL(10,2)
tasks_completed	INT

donations	
donation_id	INT
donor_id	INT
amount	DECIMAL(10,2)
donated_at	TIMESTAMP
disaster_id	INT

sosalerts	
alert_id	INT
user_id	INT
disaster_id	INT
message	TEXT
sent_at	TIMESTAMP
status	ENUM(Pending,Resolved)

DDL Statements:

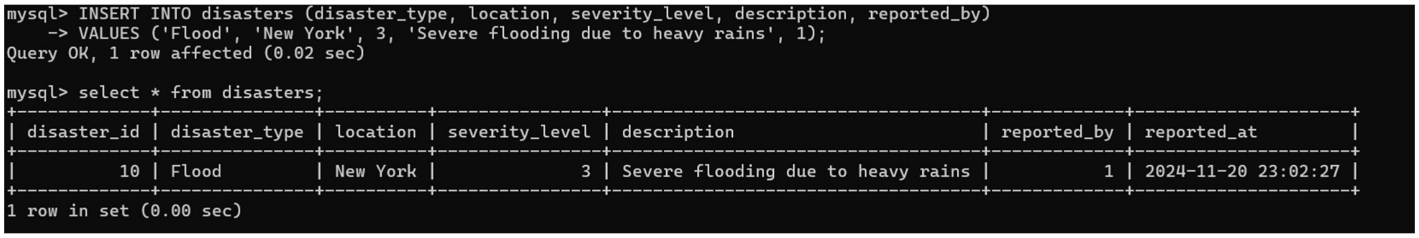
Create Statements:

```
CREATE DATABASE IF NOT EXISTS disastermanagementsystem;

USE disastermanagementsystem;

-- Table: disasters
CREATE TABLE disasters (
    disaster_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    disaster_type VARCHAR(50) NOT NULL,
    location VARCHAR(100) NOT NULL,
    severity_level INT NOT NULL,
    description TEXT DEFAULT NULL,
    reported_by INT DEFAULT NULL,
    reported_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Table: donations
CREATE TABLE donations (
    donation_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    donor_id INT DEFAULT NULL,
    amount DECIMAL(10,2) NOT NULL,
    donated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    disaster_id INT DEFAULT NULL
);

-- Table: ongoingdisasters
CREATE TABLE ongoingdisasters (
    ongoing_disaster_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    disaster_id INT NOT NULL,
    start_date DATE NOT NULL
);

-- Table: payments
CREATE TABLE payments (
    payment_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    user_id INT DEFAULT NULL,
    amount DECIMAL(10,2) NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    payment_method ENUM('Credit Card','Debit Card','PayPal','Bank Transfer') NOT NULL
);

-- Table: profiles
CREATE TABLE profiles (
    profile_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL UNIQUE,
    total_donations DECIMAL(10,2) DEFAULT 0.00,
    tasks_completed INT DEFAULT 0
);

-- Table: reliefcamps
CREATE TABLE reliefcamps (
    camp_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    camp_name VARCHAR(100) NOT NULL,
    location VARCHAR(100) NOT NULL,
    capacity INT NOT NULL,
    disaster_id INT DEFAULT NULL
);

-- Table: sosalerts
CREATE TABLE sosalerts (
    alert_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    user_id INT DEFAULT NULL,
    disaster_id INT DEFAULT NULL,
    message TEXT NOT NULL,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    status ENUM('Pending','Resolved') DEFAULT 'Pending'
);

-- Table: users
```

```
CREATE TABLE users (
    user_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone_number VARCHAR(15) NOT NULL,
    country VARCHAR(50) NOT NULL,
    city VARCHAR(64) NOT NULL,
    password VARCHAR(100) NOT NULL,
    role ENUM('User','Volunteer','Admin') DEFAULT 'User' NOT NULL
);
```

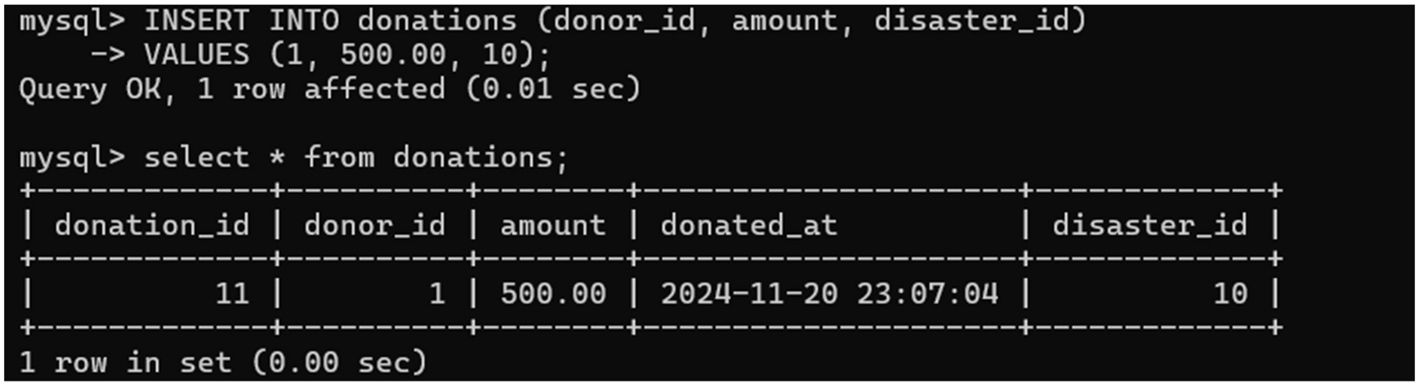
DML STATEMENTS (CRUD OPERATION SCREENSHOTS)

INSERT INTO disasters (disaster_type, location, severity_level, description, reported_by) VALUES ('Flood', 'New York', 3, 'Severe flooding due to heavy rains', 1);



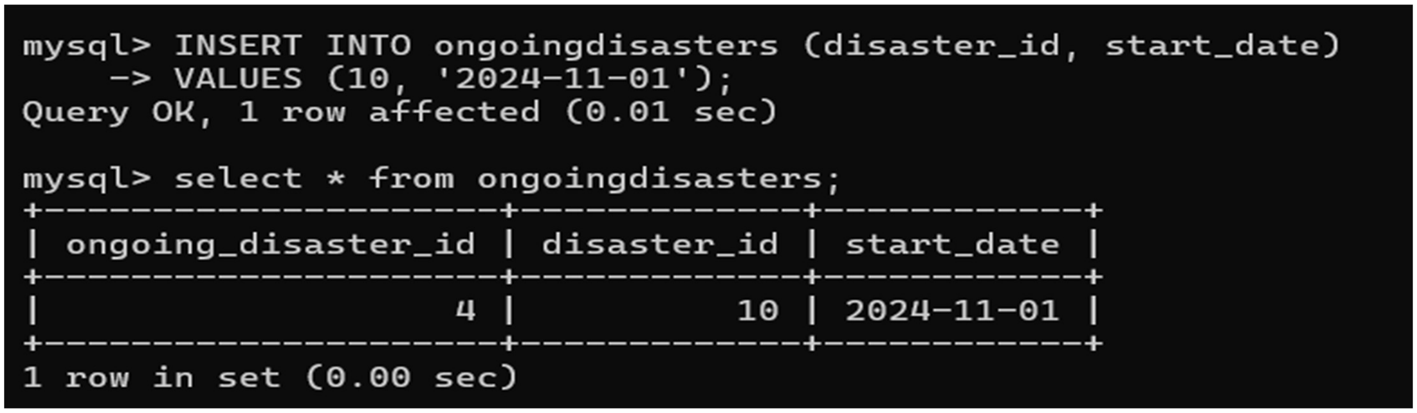
```
mysql> INSERT INTO disasters (disaster_type, location, severity_level, description, reported_by)
    -> VALUES ('Flood', 'New York', 3, 'Severe flooding due to heavy rains', 1);
Query OK, 1 row affected (0.02 sec)

mysql> select * from disasters;
+-------------+---------------+----------+----------------+------------------------------------+-------------+---------------------+
| disaster_id | disaster_type | location | severity_level | description                        | reported_by | reported_at         |
+-------------+---------------+----------+----------------+------------------------------------+-------------+---------------------+
|          10 | Flood         | New York |              3 | Severe flooding due to heavy rains |           1 | 2024-11-20 23:02:27 |
+-------------+---------------+----------+----------------+------------------------------------+-------------+---------------------+
1 row in set (0.00 sec)
```

INSERT INTO donations (donor_id, amount, disaster_id) VALUES (1, 500.00, 2);



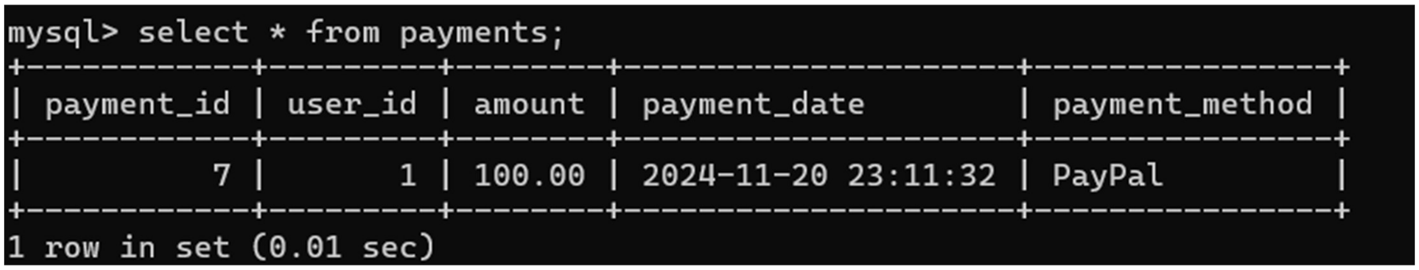
```
mysql> INSERT INTO donations (donor_id, amount, disaster_id)
    -> VALUES (1, 500.00, 10);
Query OK, 1 row affected (0.01 sec)

mysql> select * from donations;
+-------------+----------+--------+---------------------+-------------+
| donation_id | donor_id | amount | donated_at          | disaster_id |
+-------------+----------+--------+---------------------+-------------+
|          11 |        1 | 500.00 | 2024-11-20 23:07:04 |          10 |
+-------------+----------+--------+---------------------+-------------+
1 row in set (0.00 sec)
```

INSERT INTO ongoingdisasters (disaster_id, start_date) VALUES (2, '2024-11-01');



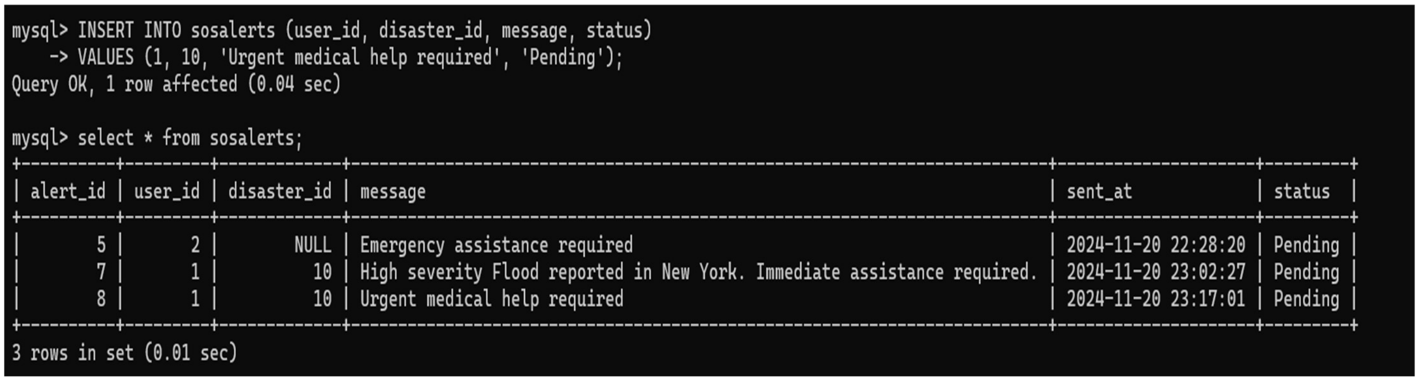
```
mysql> INSERT INTO ongoingdisasters (disaster_id, start_date)
    -> VALUES (10, '2024-11-01');
Query OK, 1 row affected (0.01 sec)

mysql> select * from ongoingdisasters;
+---------------------+-------------+------------+
| ongoing_disaster_id | disaster_id | start_date |
+---------------------+-------------+------------+
|                   4 |          10 | 2024-11-01 |
+---------------------+-------------+------------+
1 row in set (0.00 sec)
```

INSERT INTO payments (user_id, amount, payment_method) VALUES (1, 100.00, 'PayPal');



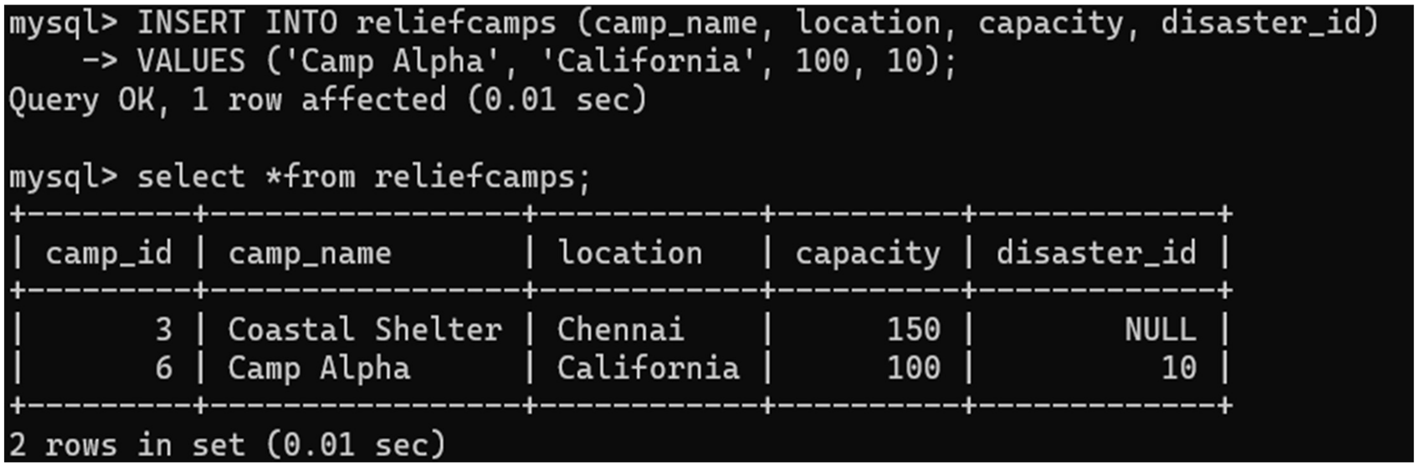
```
mysql> select * from payments;
+------------+---------+--------+---------------------+----------------+
| payment_id | user_id | amount | payment_date        | payment_method |
+------------+---------+--------+---------------------+----------------+
|          7 |       1 | 100.00 | 2024-11-20 23:11:32 | PayPal         |
+------------+---------+--------+---------------------+----------------+
1 row in set (0.01 sec)
```

INSERT INTO sosalerts (user_id, disaster_id, message, status) VALUES (1, 2,
'Urgent medical help required', 'Pending');



```
mysql> INSERT INTO sosalerts (user_id, disaster_id, message, status)
    -> VALUES (1, 10, 'Urgent medical help required', 'Pending');
Query OK, 1 row affected (0.04 sec)

mysql> select * from sosalerts;
+----------+---------+-------------+-----------------------------------------------------------------------+---------------------+---------+
| alert_id | user_id | disaster_id | message                                                               | sent_at             | status  |
+----------+---------+-------------+-----------------------------------------------------------------------+---------------------+---------+
|        5 |       2 |        NULL | Emergency assistance required                                         | 2024-11-20 22:28:20 | Pending |
|        7 |       1 |          10 | High severity Flood reported in New York. Immediate assistance required. | 2024-11-20 23:02:27 | Pending |
|        8 |       1 |          10 | Urgent medical help required                                          | 2024-11-20 23:17:01 | Pending |
+----------+---------+-------------+-----------------------------------------------------------------------+---------------------+---------+
3 rows in set (0.01 sec)
```

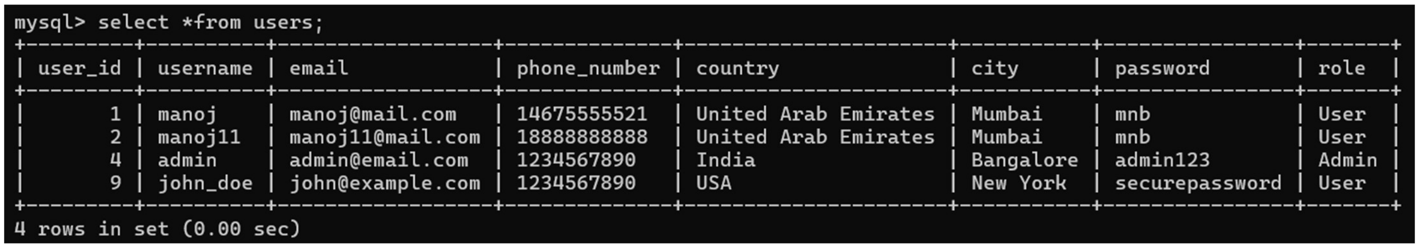
INSERT INTO reliefcamps (camp_name, location, capacity, disaster_id) VALUES
('Camp Alpha', 'California', 100, 2);



```
mysql> INSERT INTO reliefcamps (camp_name, location, capacity, disaster_id)
    -> VALUES ('Camp Alpha', 'California', 100, 10);
Query OK, 1 row affected (0.01 sec)

mysql> select *from reliefcamps;
+---------+-----------------+------------+----------+-------------+
| camp_id | camp_name       | location   | capacity | disaster_id |
+---------+-----------------+------------+----------+-------------+
|       3 | Coastal Shelter | Chennai    |      150 |        NULL |
|       6 | Camp Alpha      | California |      100 |          10 |
+---------+-----------------+------------+----------+-------------+
2 rows in set (0.01 sec)
```

Update and Delete:

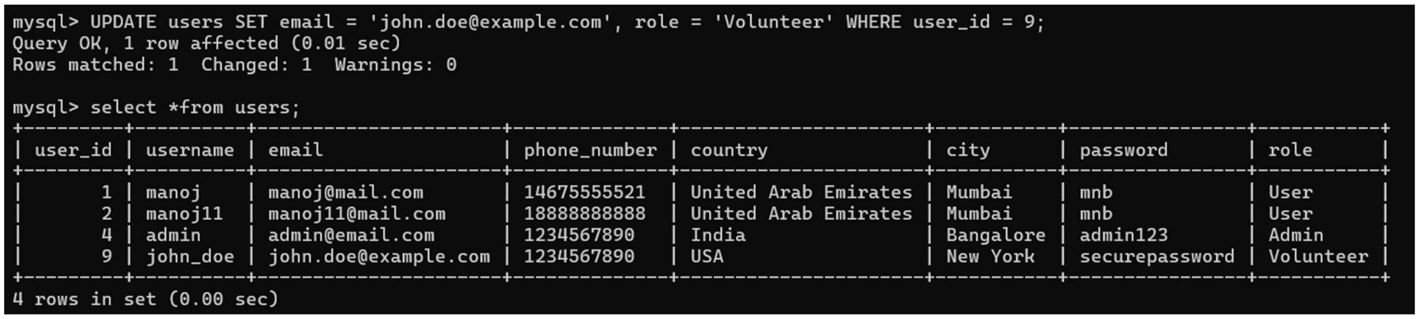
INSERT INTO users (username, email, phone_number, country, city, password, role) VALUES ('john_doe', 'john@example.com', '1234567890', 'USA', 'New York', 'securepassword', 'User');



```
mysql> select *from users;
+---------+----------+-------------------+--------------+----------------------+-----------+----------------+-------+
| user_id | username | email             | phone_number | country              | city      | password       | role  |
+---------+----------+-------------------+--------------+----------------------+-----------+----------------+-------+
|       1 | manoj    | manoj@mail.com    | 14675555521  | United Arab Emirates | Mumbai    | mnb            | User  |
|       2 | manoj11  | manoj11@mail.com  | 18888888888  | United Arab Emirates | Mumbai    | mnb            | User  |
|       4 | admin    | admin@email.com   | 1234567890   | India                | Bangalore | admin123       | Admin |
|       9 | john_doe | john@example.com  | 1234567890   | USA                  | New York  | securepassword | User  |
+---------+----------+-------------------+--------------+----------------------+-----------+----------------+-------+
4 rows in set (0.00 sec)
```

UPDATE User

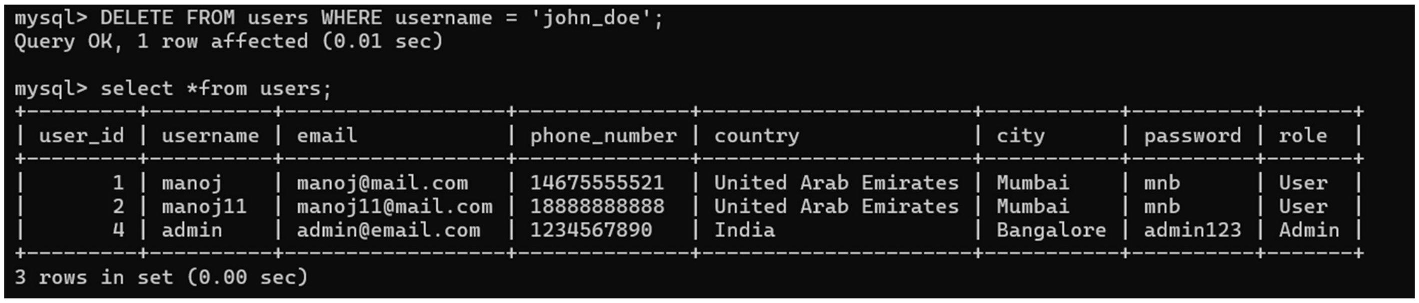
UPDATE users SET email = 'john.doe@example.com', role = 'Volunteer' WHERE user_id = 1;



```
mysql> UPDATE users SET email = 'john.doe@example.com', role = 'Volunteer' WHERE user_id = 9;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select *from users;
+---------+----------+----------------------+--------------+----------------------+-----------+----------------+-----------+
| user_id | username | email                | phone_number | country              | city      | password       | role      |
+---------+----------+----------------------+--------------+----------------------+-----------+----------------+-----------+
|       1 | manoj    | manoj@mail.com       | 14675555521  | United Arab Emirates | Mumbai    | mnb            | User      |
|       2 | manoj11  | manoj11@mail.com     | 18888888888  | United Arab Emirates | Mumbai    | mnb            | User      |
|       4 | admin    | admin@email.com      | 1234567890   | India                | Bangalore | admin123       | Admin     |
|       9 | john_doe | john.doe@example.com | 1234567890   | USA                  | New York  | securepassword | Volunteer |
+---------+----------+----------------------+--------------+----------------------+-----------+----------------+-----------+
4 rows in set (0.00 sec)
```

DELETE User

DELETE FROM users WHERE user_id = 1;



```
mysql> DELETE FROM users WHERE username = 'john_doe';
Query OK, 1 row affected (0.01 sec)

mysql> select *from users;
+---------+----------+------------------+--------------+----------------------+-----------+----------+-------+
| user_id | username | email            | phone_number | country              | city      | password | role  |
+---------+----------+------------------+--------------+----------------------+-----------+----------+-------+
|       1 | manoj    | manoj@mail.com   | 14675555521  | United Arab Emirates | Mumbai    | mnb      | User  |
|       2 | manoj11  | manoj11@mail.com | 18888888888  | United Arab Emirates | Mumbai    | mnb      | User  |
|       4 | admin    | admin@email.com  | 1234567890   | India                | Bangalore | admin123 | Admin |
+---------+----------+------------------+--------------+----------------------+-----------+----------+-------+
3 rows in set (0.00 sec)
```

QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

Join:

1)

```
-- JOIN query to find high severity disasters without SOS alerts

SELECT d.* FROM disasters d

LEFT JOIN sosalerts s ON d.disaster_id = s.disaster_id

WHERE d.severity_level = 3 AND s.alert_id IS NULL;

-- Simple SELECT query to get user-specific disasters

SELECT * FROM disasters WHERE reported_by = ? ORDER BY reported_at DESC;

-- Simple SELECT query to get all disasters

SELECT * FROM disasters ORDER BY reported_at DESC;
```

2)

```
-- JOIN query to get relief camps with disaster information

SELECT rc.*, d.disaster_type

FROM reliefcamps rc

JOIN disasters d ON rc.disaster_id = d.disaster_id

ORDER BY rc.camp_id DESC;
```

3)

```
-- JOIN query to get payment history with disaster details

SELECT p.payment_id, p.amount, p.payment_method, p.payment_date,
      d.disaster_id, dis.disaster_type, dis.location
FROM payments p
JOIN donations d ON p.user_id = d.donor_id
JOIN disasters dis ON d.disaster_id = dis.disaster_id
WHERE p.user_id = ?
ORDER BY p.payment_date DESC;
```

4)

```
-- JOIN query to get donation details with disaster info
SELECT d.donation_id, d.donor_id, d.amount, d.donated_at,
      dis.disaster_type, dis.location, dis.severity_level
FROM donations d
JOIN disasters dis ON d.disaster_id = dis.disaster_id
ORDER BY d.donated_at DESC
LIMIT 10;
```

5)

```
-- JOIN query in stored procedure to get high severity disasters
CREATE PROCEDURE GetHighSeverityDisasters()
BEGIN
   SELECT d.disaster_id, d.disaster_type, d.location,
         d.severity_level, d.description,
         sa.status as sos_status, sa.sent_at as sos_created_at
   FROM disasters d
   LEFT JOIN sosalerts sa ON d.disaster_id = sa.disaster_id
```

```
    WHERE d.severity_level = 3
    ORDER BY d.reported_at DESC;
END;
```

6)

```
-- JOIN query to get SOS alerts with disaster details
SELECT sa.*, d.disaster_type, d.location
FROM sosalerts sa
JOIN disasters d ON sa.disaster_id = d.disaster_id
WHERE d.severity_level = 3
ORDER BY sa.sent_at DESC;
```

Aggregate Function:

```
-- AGGREGATE function to get total donations
CREATE FUNCTION GetTotalDonationsForDisaster(disaster_id_param INT)
RETURNS DECIMAL(10,2)
BEGIN
    SELECT COALESCE(SUM(amount), 0.00)
    INTO total
    FROM donations
    WHERE disaster_id = disaster_id_param;
    RETURN total;
END;
```

Nested Query:

```
-- NESTED query to find camps with low resources for high severity disasters
SELECT rc.id AS camp_id, rc.location, rc.capacity, rc.current_occupancy
FROM ReliefCamp rc
WHERE rc.disaster_id IN (
   SELECT id FROM DisasterReport WHERE severity = 'High'
   ORDER BY timestamp
) AND r.quantity < 300;
```

This nested query is used to display the relief camps where the severity of the disaster is “High” and the resources for this particular camp is running low.

STORED PROCEDURE, FUNCTIONS AND TRIGGERS

Functions/Procedures:

-- Function to get total donations for a disaster

```
CREATE FUNCTION IF NOT EXISTS GetTotalDonationsForDisaster(disaster_id_param INT)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

    DECLARE total DECIMAL(10,2);

    SELECT COALESCE(SUM(amount), 0.00)

    INTO total

    FROM donations

    WHERE disaster_id = disaster_id_param;

    RETURN total;

END;
```

-- Function to check relief camp capacity

```
CREATE FUNCTION IF NOT EXISTS CheckCampCapacity(camp_id_param INT)

RETURNS BOOLEAN

DETERMINISTIC

BEGIN

    DECLARE current_count INT;

    DECLARE max_capacity INT;
    SELECT COUNT(*) INTO current_count
    FROM volunteers
    WHERE camp_id = camp_id_param;

    SELECT capacity INTO max_capacity
    FROM reliefcamps
    WHERE camp_id = camp_id_param;

    RETURN current_count < max_capacity;
END;
```

-- Procedure to get high severity disasters with pending alerts

```
CREATE PROCEDURE IF NOT EXISTS GetHighSeverityDisasters()
BEGIN
    SELECT
        d.disaster_id,
        d.disaster_type,
        d.location,
        d.severity_level,
        d.description,
        sa.status as sos_status,
        sa.sent_at as sos_created_at
    FROM disasters d
    LEFT JOIN sosalerts sa ON d.disaster_id = sa.disaster_id
    WHERE d.severity_level = 3
    ORDER BY d.reported_at DESC;
END;
```

-- Procedure to assign volunteer to camp with capacity check

```
CREATE PROCEDURE IF NOT EXISTS AssignVolunteerToCamp(
    IN p_user_id INT,
    IN p_camp_id INT,
    IN p_task_description TEXT
)
BEGIN
    DECLARE camp_full BOOLEAN;
    SELECT NOT CheckCampCapacity(p_camp_id) INTO camp_full;

    IF camp_full THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Camp is at full capacity';
    ELSE
        INSERT INTO volunteers (user_id, camp_id, task_description)
        VALUES (p_user_id, p_camp_id, p_task_description);
    END IF;
END;
```

Trigger:

-- Trigger for creating SOS alerts for high severity disasters

```
CREATE TRIGGER IF NOT EXISTS after_disaster_insert

AFTER INSERT ON disasters

FOR EACH ROW

BEGIN

   IF NEW.severity_level = 3 THEN

      INSERT INTO sosalerts (user_id, disaster_id, message, status)

      VALUES (

         NEW.reported_by,

         NEW.disaster_id,

         CONCAT('High severity ', NEW.disaster_type, ' reported in ', NEW.location, '. Immediate assistance required.'),

         'Pending'

      );

   END IF;

END;
```

-- Trigger for handling SOS alerts when severity is updated

```
CREATE TRIGGER IF NOT EXISTS after_disaster_update

AFTER UPDATE ON disasters

FOR EACH ROW

BEGIN

   IF NEW.severity_level = 3 AND OLD.severity_level != 3 THEN

      INSERT INTO sosalerts (user_id, disaster_id, message, status)
```

```
        VALUES (
            NEW.reported_by,
            NEW.disaster_id,
            CONCAT('High severity ', NEW.disaster_type, ' reported in ', NEW.location, '. Immediate assistance required.'),
            'Pending'
        );
    ELSEIF NEW.severity_level != 3 AND OLD.severity_level = 3 THEN
        DELETE FROM sosalerts WHERE disaster_id = NEW.disaster_id;
    END IF;
END;
```

-- Trigger for cascade deletion

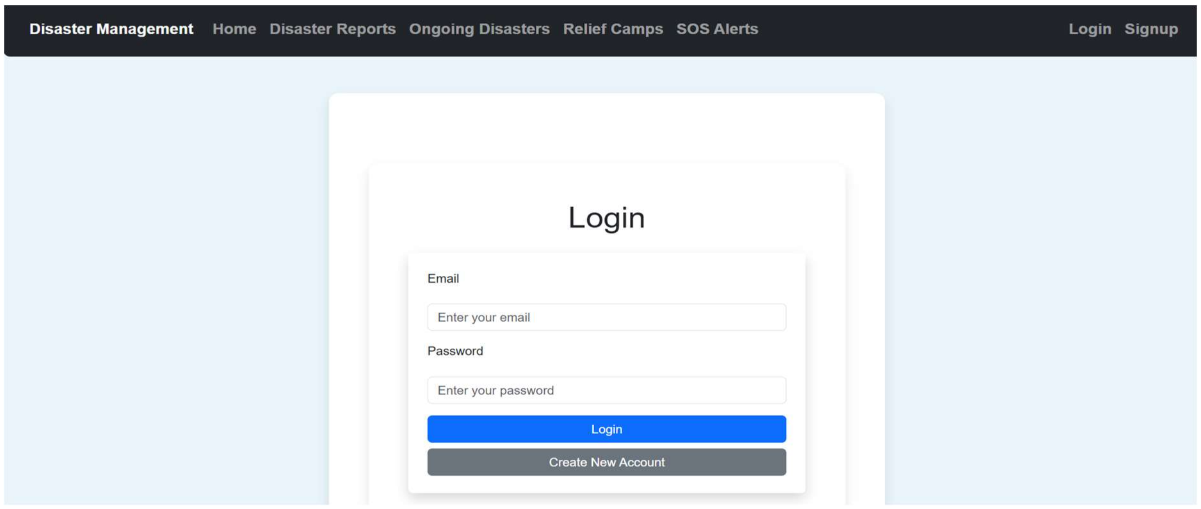
```
CREATE TRIGGER IF NOT EXISTS before_disaster_delete
BEFORE DELETE ON disasters
FOR EACH ROW
BEGIN
    DELETE FROM sosalerts WHERE disaster_id = OLD.disaster_id;
    DELETE FROM reliefcamps WHERE disaster_id = OLD.disaster_id;
    DELETE FROM donations WHERE disaster_id = OLD.disaster_id;
END;
```

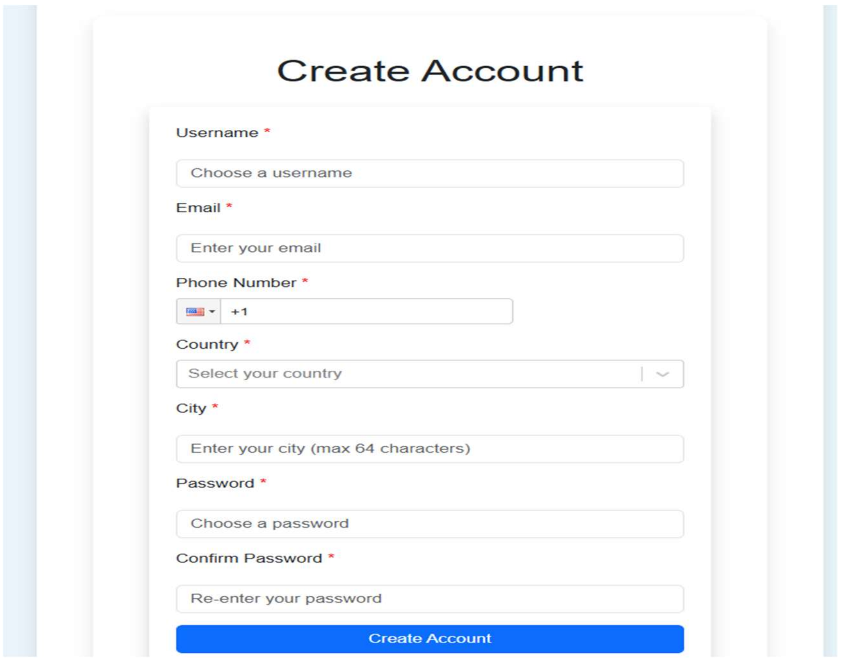
FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)

GitHub link:

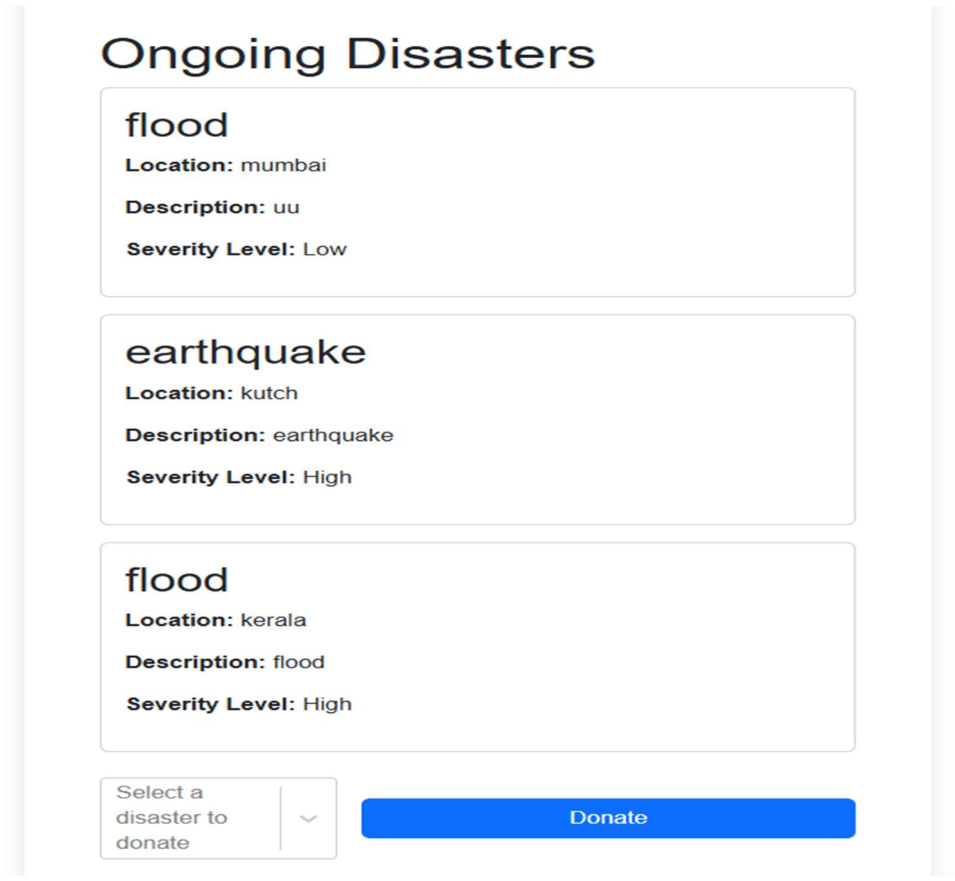
https://github.com/THManoj/Crowdsourced-Disaster-Relief-System.git

Login and signup: (Insert)





Donor view:



Ongoing Disasters

flood

Location: mumbai

Description: uu

Severity Level: Low

earthquake

Location: kutch

Description: earthquake

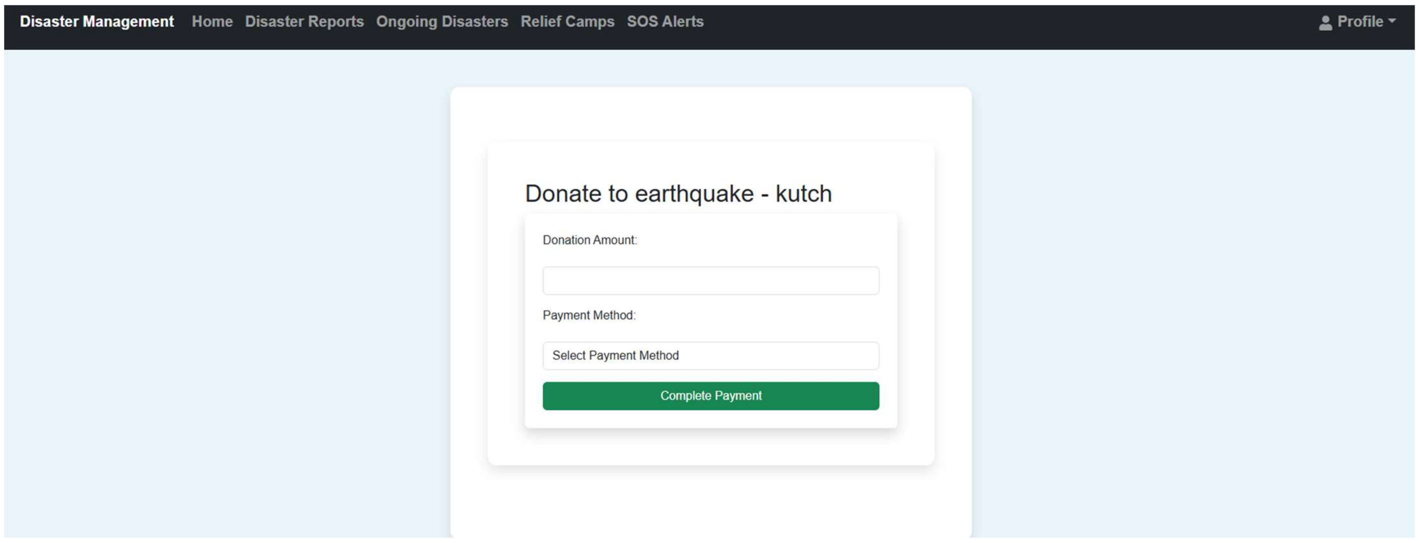
Severity Level: High

flood

Location: kerala

Description: flood

Severity Level: High



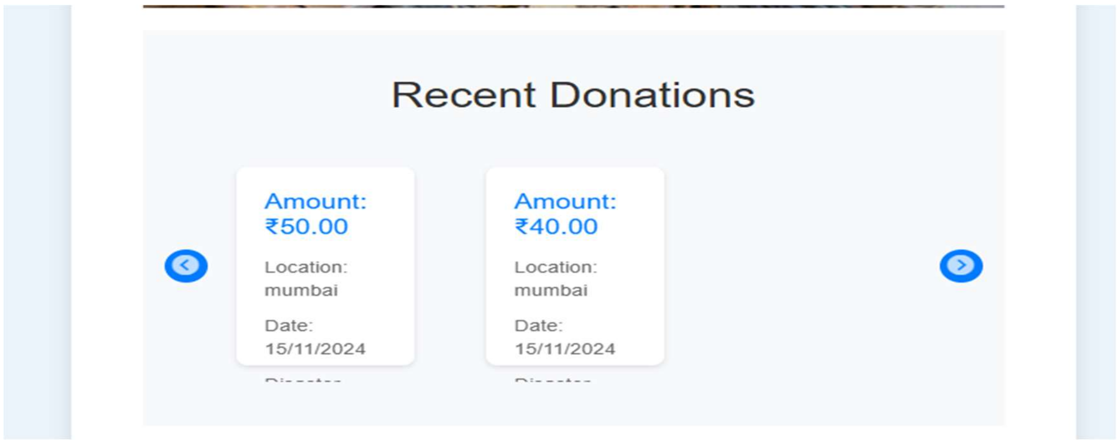
Donate to earthquake - kutch

Donation Amount:

Payment Method:

Select Payment Method

Shows recent donation and ongoing disasters (Join)



Recent Donations

Amount:
₹50.00

Location:
mumbai

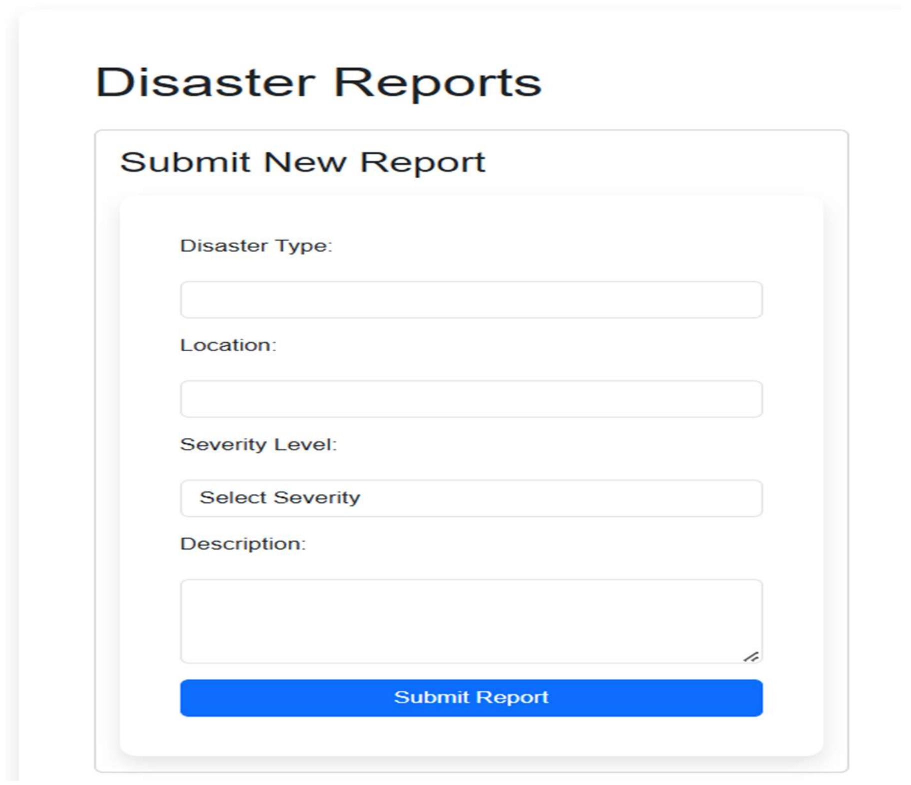
Date:
15/11/2024

Amount:
₹40.00

Location:
mumbai

Date:
15/11/2024

Form for disaster reporting : (Insert)



Disaster Reports

Submit New Report

Disaster Type:

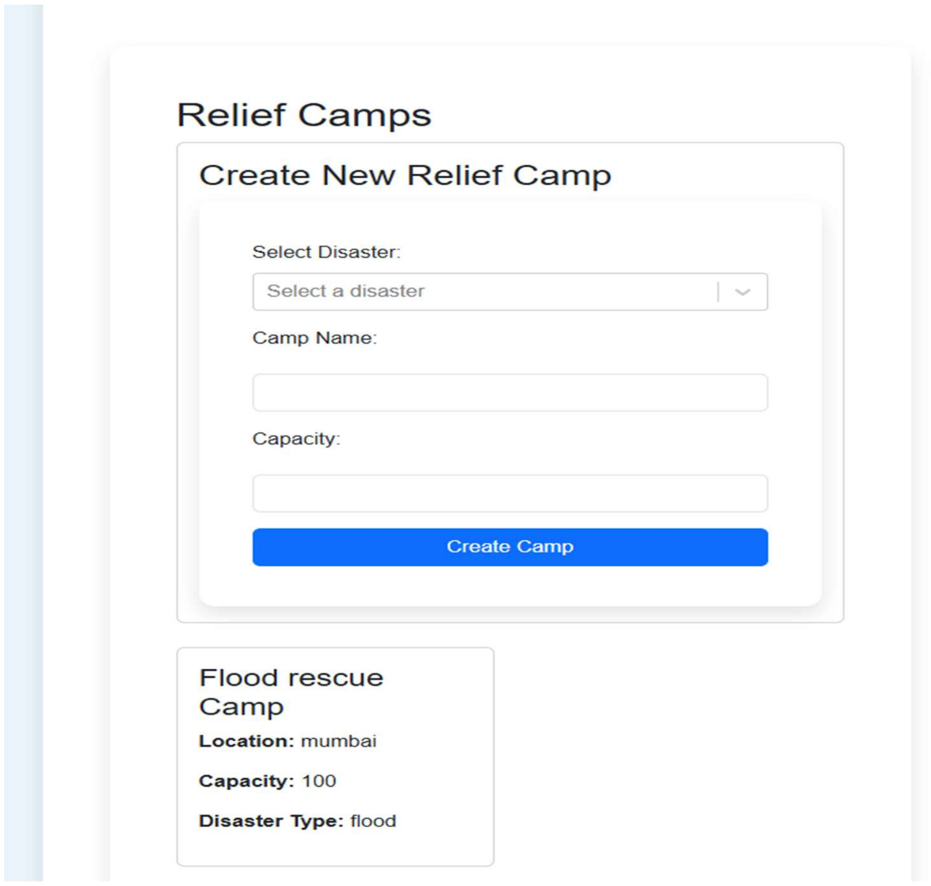
Location:

Severity Level:

Select Severity

Description:

Shows relief camps available for specific disasters



Relief Camps

Create New Relief Camp

Select Disaster: Select a disaster

Camp Name:

Capacity:

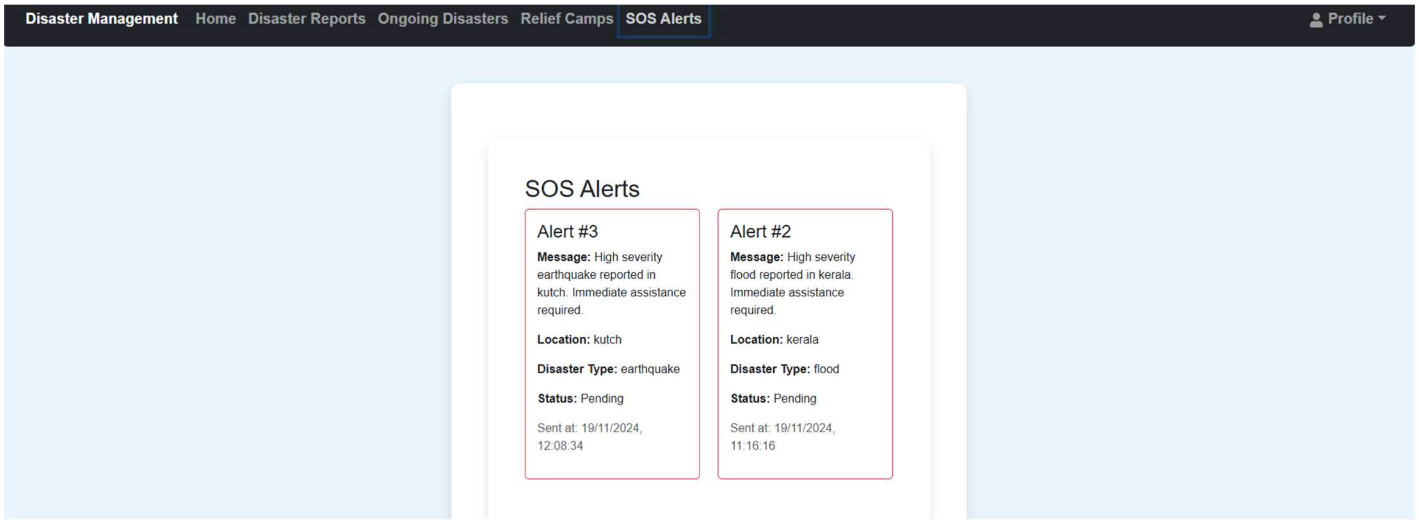
Flood rescue Camp

Location: mumbai

Capacity: 100

Disaster Type: flood

SOS Alerts:



SOS Alerts

Alert #3

Message: High severity earthquake reported in kutch. Immediate assistance required.

Location: kutch

Disaster Type: earthquake

Status: Pending

Sent at: 19/11/2024, 12:08:34

Alert #2

Message: High severity flood reported in kerala. Immediate assistance required.

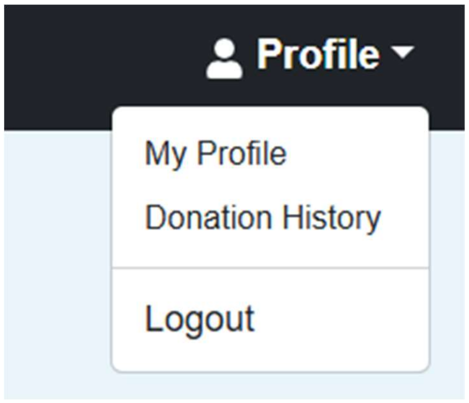
Location: kerala

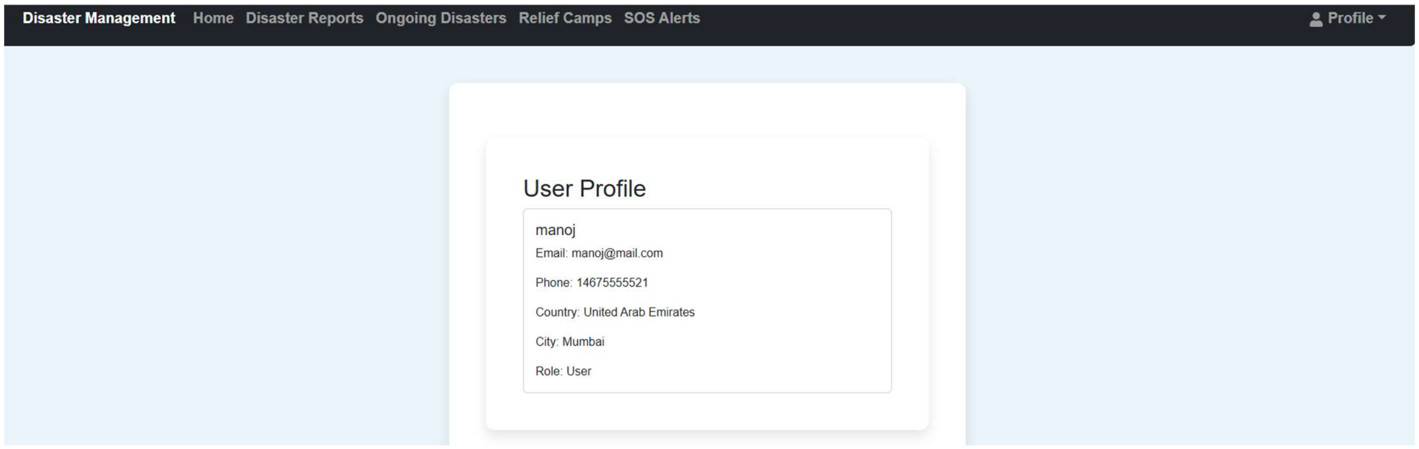
Disaster Type: flood

Status: Pending

Sent at: 19/11/2024, 11:16:16

Profile and Donation History:





User Profile

manoj

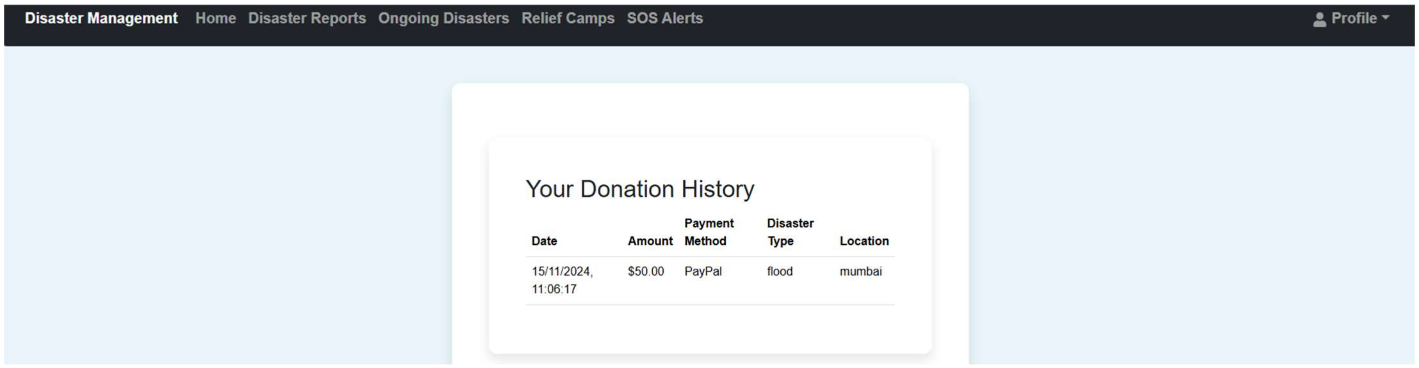
Email: manoj@mail.com

Phone: 14675555521

Country: United Arab Emirates

City: Mumbai

Role: User



Your Donation History

Date	Amount	Payment Method	Disaster Type	Location
15/11/2024, 11:06:17	$50.00	PayPal	flood	mumbai

REFERENCES/BIBLIOGRAPHY

1. www.google.com
2. www.youtube.com
3. https://dev.mysql.com/doc/
4. https://www.geeksforgeeks.org/

GitHub link:

https://github.com/THManoj/Crowdsourced-Disaster-Relief-System.git