**Project Name:** **Emergency Response Network**

**Team ID : LTVIP2025TMID27092**

**Team Leader: S. Vijay Kumar**

**Team Member: C. Prathibha**

**Team Member: K. Siddesh**

**Team Member: P. Sudarshan**

**Team Member: V. Vamshi**

**Emergency Response Network**

The ERN backend is built using **Node.js, Express.js, and MongoDB (Mongoose)** to handle emergency requests, assign responders and volunteers, and manage real-time data flow. APIs are developed to allow seamless integration with mobile applications, web dashboards, and IoT devices.

**Key functionalities include:**

* Emergency request registration
* Real-time alerts to response teams
* Automated assignment of responders based on proximity and availability
* Incident status updates

**Scenario Based Case Study**

**Scenario: Fire Emergency in a Residential Area**

* A fire breaks out in a residential area. A resident reports the emergency via a mobile app.
* The request is received by the backend and logged into the database.
* The system identifies the nearest fire stations and available fire trucks.
* Responders are notified with location tracking.
* Authorities receive live updates on the incident status.

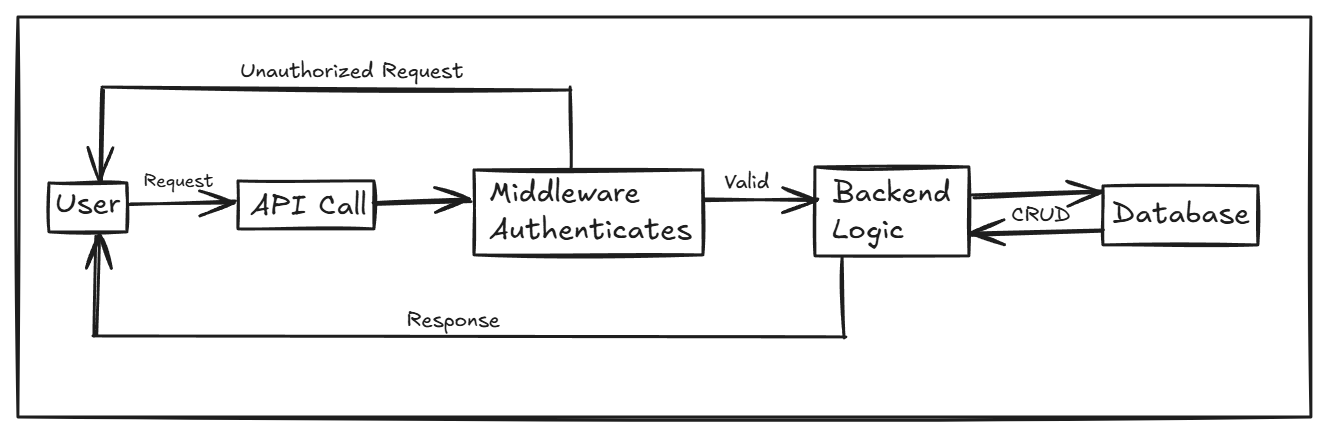
This case study demonstrates how ERN streamlines emergency response through efficient data handling.

**Technical Architecture**

The system follows a **Microservices-based REST API architecture**, ensuring modularity and scalability.

**Technology Stack:**

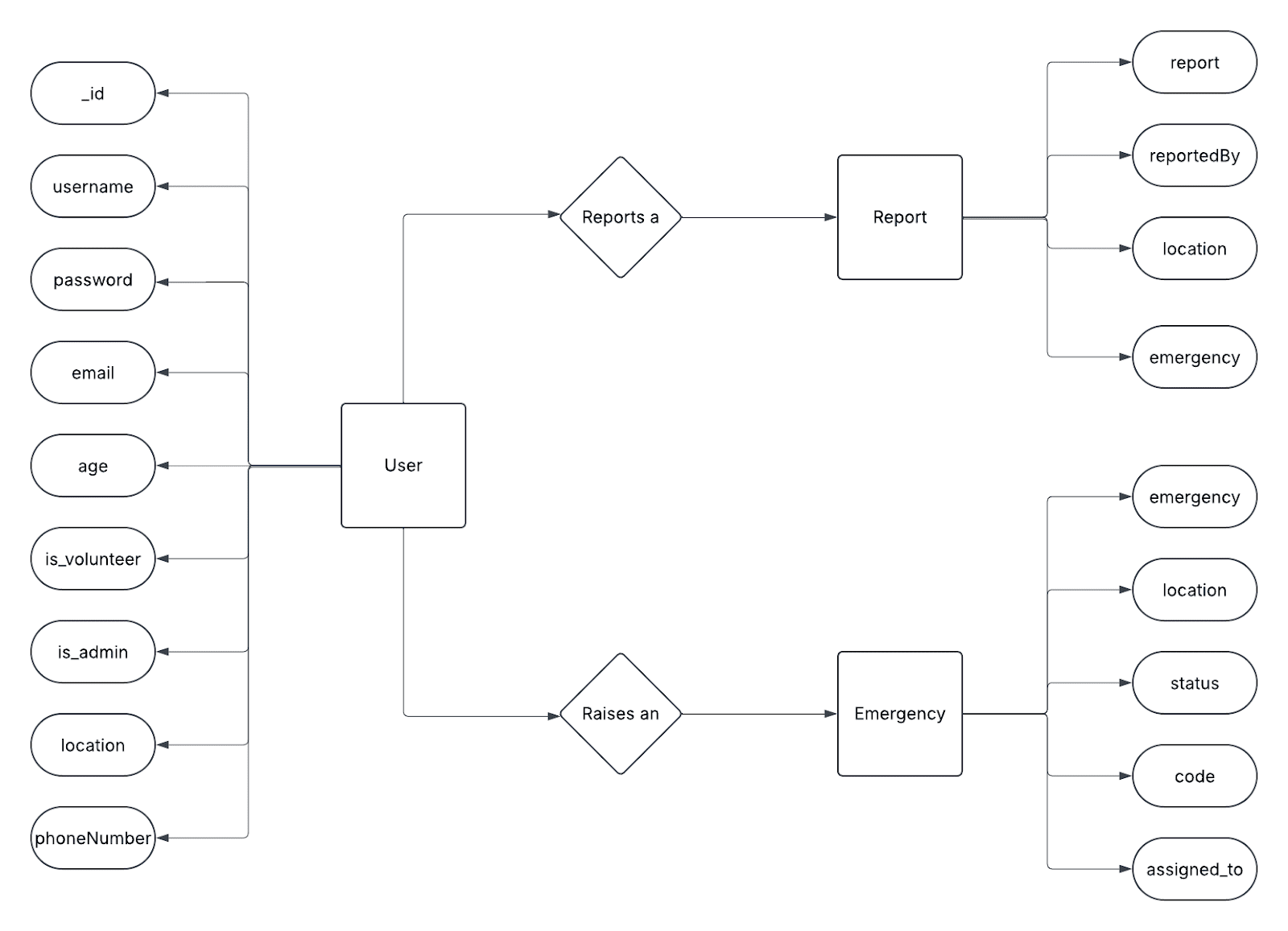
* **Backend:** Node.js, Express.js
* **Database:** MongoDB (Mongoose ODM)
* **Authentication:** JWT-based authentication
* **API Testing:** Postman / Thunderclient



**ER-Diagram**

The **Entity-Relationship Diagram (ERD)** represents database structure, including entities like:

* **Users:** Stores users, victims, volunteers and admin details
* **Emergencies:** Stores reported emergencies
* **Reports:** Stores reported incidents

**Key Features**

* **Automated Emergency Response:** Alerts responders based on proximity
* **Real-time Updates:** Updates on responder locations and incident status.
* **User Authentication & Role Management:** Secure access for users, volunteers, and admins.
* **Incident Logging & Reports:** Maintains history for future analysis.
* **Multi-Platform API Integration:** Supports mobile, web, and IoT applications.

**PRE REQUISITES**

Before setting up the project, ensure you have:

* **Node.js (Latest LTS version) Installed**
* **MongoDB Database Setup (Local or Cloud-based like MongoDB Atlas)**
* **Postman(App) or Thunderclient(VS Code Extension) for API Testing**
* **Code Editor (VS Code or Any Preferred IDE)**
* **Basic Understanding of JavaScript, Node.js, and MongoDB**

**Roles and Responsibility**

The **Emergency Response Network (ERN)** involves three primary roles: **Users, Volunteers, and Admins**. Each role has distinct responsibilities to ensure efficient emergency handling and response coordination.

1.**Users (Emergency Reporters)**

These are individuals who report emergencies, such as victims, witnesses, or concerned citizens.

**Responsibilities:**

         Register and authenticate via the system.

         Report emergencies by providing details such as location, type of incident, and severity.

2**.      Volunteers (Responders & Emergency Personnel)**

Volunteers are first responders such as medical professionals, firefighters, police officers, or trained community helpers.

**Responsibilities:**

         Register and authenticate in the system as verified responders.

         Receive emergency alerts based on location and availability.

**3.      Admins (System Managers & Authorities)**

Admins oversee the system's operation, ensure data integrity, and manage users and volunteers.

**Responsibilities:**

* Manage system users (approve, block, or verify responders).
* Oversee emergency reports and ensure appropriate response.

**Application Flow**

1. **Users or applications send emergency requests via API.**
2. **The system validates and logs the emergency or incident report.**
3. **Based on severity and location, volunteers respond.**
4. Responders receive alerts and track incidents in real-time.
5. Authorities and admins monitor ongoing incidents via API responses**.**

### Project Structure

A screenshot of a computer program

AI-generated content may be incorrect.

**Project Setup and Configuration**

* Initialize the project with **Node.js & Express.js**.
* Set up **package.json** and install necessary dependencies (express, mongoose, dotenv, jsonwebtoken, mongoose, cookie-parser, bcrypt).

A screen shot of a computer program

AI-generated content may be incorrect.

* Configure **environment variables (JWT\_SECRET, MONGODB\_URI)** and basic routing structure.

**Backend Development**

* Create **RESTful APIs** for handling emergency requests, user authentication, and incident updates.

A screen shot of a computer code

AI-generated content may be incorrect.

* Implement **JWT-based authentication** for security.
* Define **controllers and middleware** for data validation and request handling.

**Database**

* Design **MongoDB schema** for users, reports and emergencies.

A screen shot of a computer program

AI-generated content may be incorrect.

* **Users Collection in MongoDB Compass**

A screenshot of a computer

AI-generated content may be incorrect.

* **Emergencies Collection in MongoDB Compass**

A screenshot of a computer

AI-generated content may be incorrect.

* **Reports Collection in MongoDB Compass**

A screenshot of a computer

AI-generated content may be incorrect.

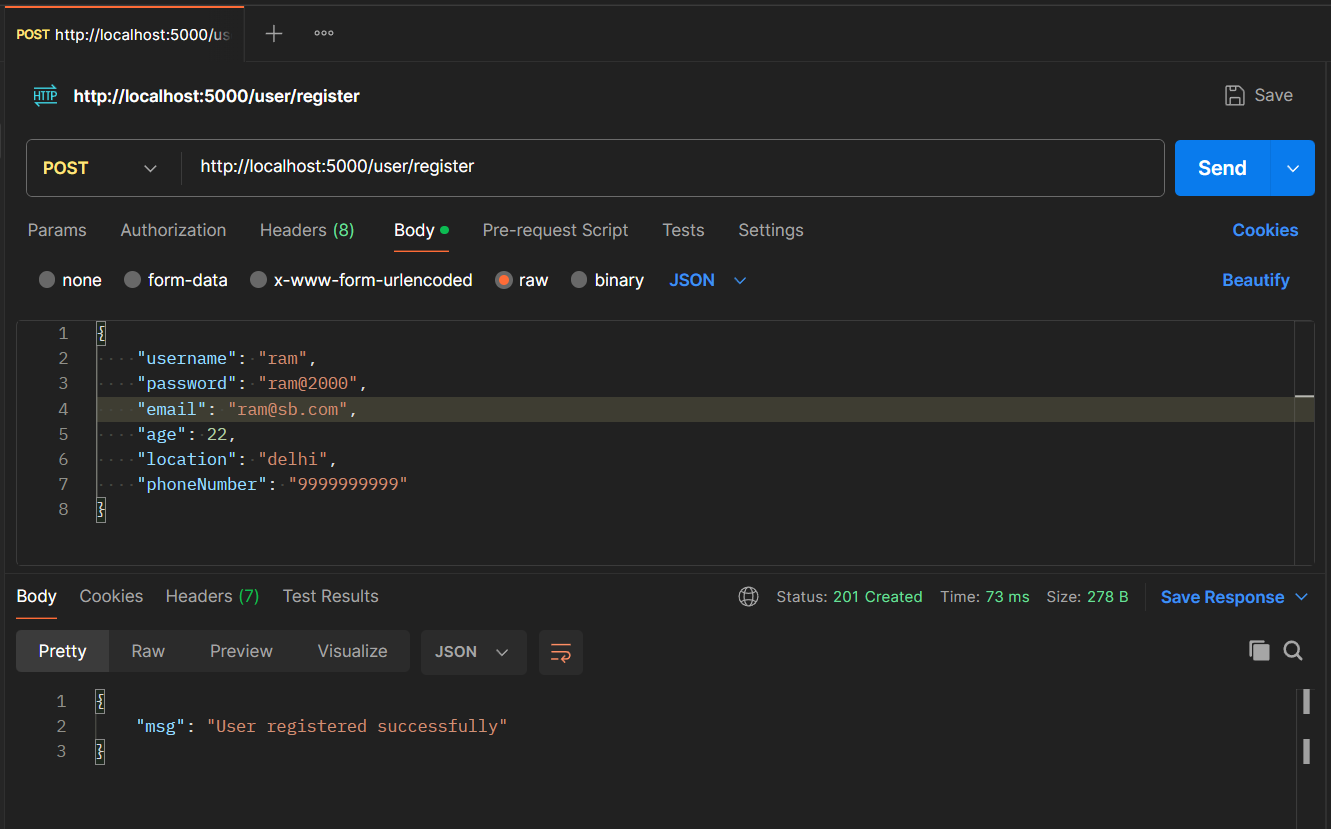
* Implement **Mongoose models** and database operations (CRUD).
* Set up **MongoDB Atlas (if using cloud storage)**.

**API Testing**

* Use **Postman(Download Application) or Thunderclient(VS Code Extension)** to test API endpoints.
* Validate responses for:
  + Creating and updating reports, emergencies
  + Alerting volunteers and admins
  + Authentication and role-based access
* Debug issues and optimize performance.

**Outputs (API Testing)**

**Register Route**

****

**Method:** POST **URL:** /user/register

**Description:** Register a new user

**Login Route**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Method:** POST **URL:** /user/login

**Description:** Login a user

**Update Route**

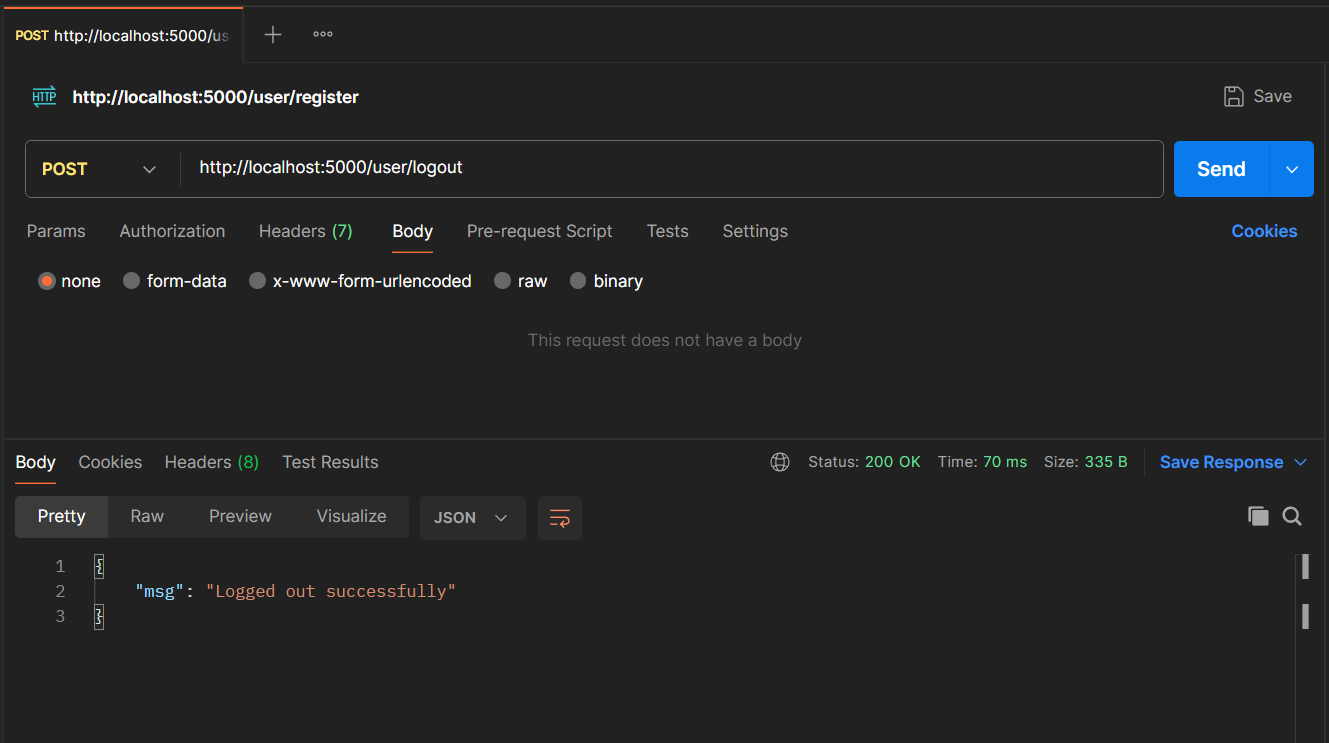
**A screenshot of a computer program

AI-generated content may be incorrect.**

**Method:** POST **URL:** /user/update

**Description:** Update a user’s profile data

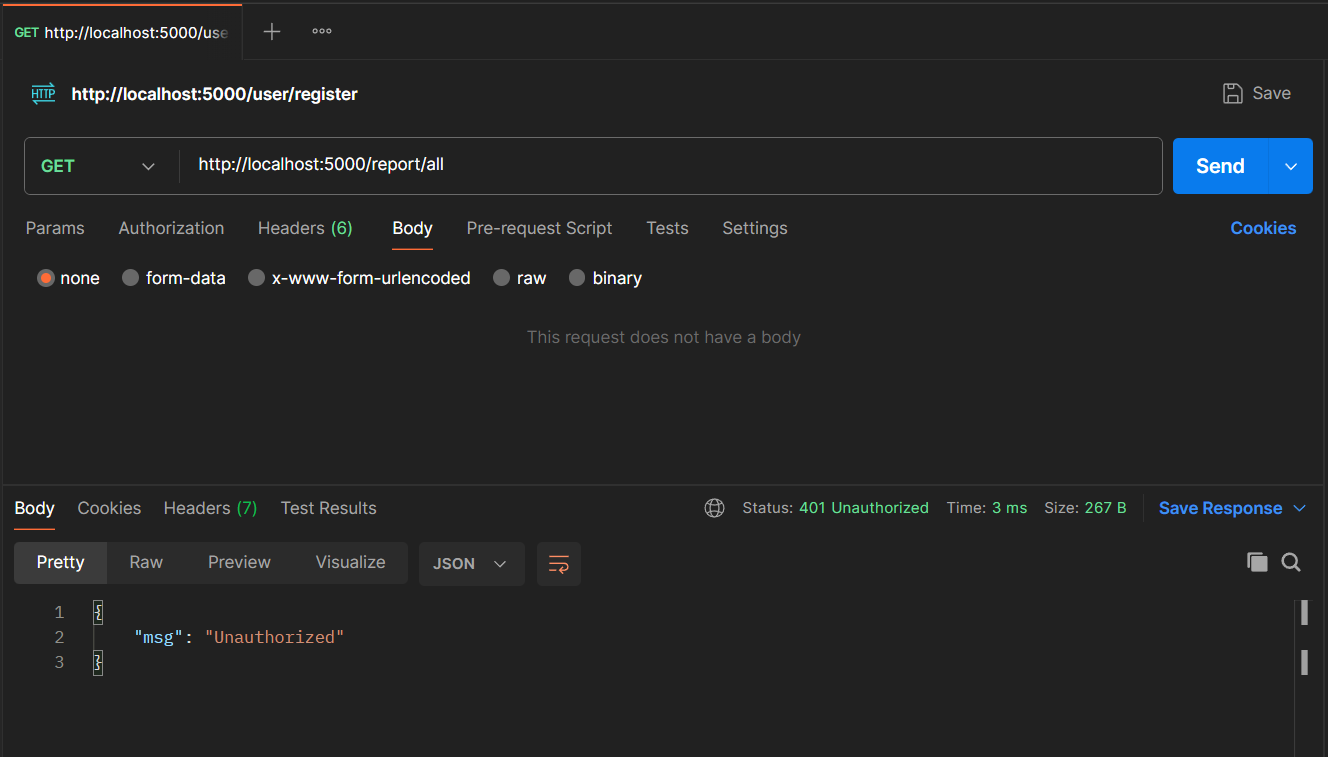
**Logout Route**

****

**Method:** POST **URL:** /user/logout

**Description:** Logout a user

**Unauthorized Request (Not Logged In)**

****

**Method:** GET **URL:** /report/all

**Description:** Unauthorized (when requested without logging in)

**Create Report Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** POST **URL:** /report/create

**Description:** Create a new report

**Get – Nearby Reports Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /report/near/<location>

**Description:** Get nearby reports(pass the location in url as shown in the image above)

**Get – All Report Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /report/all

**Description:** Get all reports

**Filter Reports Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /report/filter?<query-parameters-with-values>(see in the image above)

**Description:** Filter reports based on query

**Create Emergency Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** POST **URL:** /emergency/create

**Description:** Create an emergency

**Update Emergency Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** PUT **URL:** /emergency/update

**Description:** Update an emergency

**Get – All Emergency Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /emergency/all

**Description:** Get all emergencies

**Get – Nearby Emergency Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /emergency/near/<location>

**Description:** Get emergencies near a location; mention location in url(as shown in image)

**Filter Emergency Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /emergency/filter?<query=value>

**Description:** Filter emergencies based on query parameter values(as shown in image above)

**Get Alerts Route**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Method:** GET **URL:** /alert/any

**Description:** Get all the alerts near by(based on requested user location)

**Conclusion**

The **Emergency Response Network** provides a robust backend solution for managing crisis situations effectively. With easy reporting, requesting help in emergencies, volunteers to help and secure data management, this system enhances emergency response efficiency, ensuring timely help for those in need.