Kenya Red Cross Disaster Management System - Backend Documentation

Overview

The backend provides APIs to support:

- User location reporting through USSD.
- Emergency signal registration with geographical coordinates.
- Real-time management and storage of disaster-related data.

The backend is built with **Node.js**, using **Express.js** for the server, **PostgreSQL** (with Supabase for online hosting), and **Knex.js** for database queries.

---

1. Install Dependencies

Ensure you have Node.js and npm installed, then run:

3. Configure Environment Variables

Create a .env file in the root directory with the following content:

For local PostgreSQL:

env

DATABASE_URL=postgres://<username>:<password>@<host>:<port>/<database>

PORT=3000

For Supabase:

env

DATABASE_URL=postgres://<supabase-username>:<supabase-password>@<supabase-host>:5432/<supabase-database>

PORT=3000

Project Structure

bash

project-directory/

|

├── db.js # Database connection setup

├── server.js # Main server file

├── routes/

│ ├── users.js # Routes for user-related operations

│ ├── emergencies.js # Routes for handling emergency signals

├── .env # Environment variables

├── package.json # Node.js dependencies

├── README.md # Project documentation

└── schema.sql # SQL file for setting up the database

Key Features

1. API Endpoints

Endpoint	Method	Description
/api/users/report	POST	Accepts USSD-based user location reports.
/api/emergencies/register	POST	Registers an emergency signal with location coordinates.
/api/emergencies/all	GET	Retrieves all registered emergencies.

---

2. API Request

Register Emergency Signal

- **Endpoint:** /api/emergencies/register
- **Method:** POST
- **Body (JSON):**

json

Example of this include:

```
{
  "signal_frequency": "100.7 FM",
  "location": {
    "latitude": -1.102,
    "longitude": 36.904
  }
```

}

- **Response:**

json

```
{
  "status": "success",
  "message": "Emergency signal registered successfully"
}
```

Supabase Integration

1. Database URL

The Supabase connection string in .env replaces the local PostgreSQL configuration.

2. Using Supabase Dashboard

- Monitor your data directly in the Supabase dashboard.
- Use the SQL editor for queries and managing database schemas.

3. Live Queries

For real-time updates (e.g., live signal tracking), Supabase's real-time API can be integrated. Contact Supabase support to enable this feature.

How It Works

1. Server Initialization

The server.js initializes the backend, sets up Express routes, and connects to the database using db.js.

2. Handling Requests

- **User Reports**: Handles USSD data to extract and store user location.
- **Emergency Signals**: Logs the signal frequency and geographical location into the database.

3. Database Connection

The db.js file uses Knex.js to interact with the PostgreSQL database.

Security and Data Protection

1. Environment Variables

Sensitive data (e.g., database credentials) is stored in the .env file. Ensure this file is:

- Excluded from version control by adding it to .gitignore.
- Properly configured on deployment platforms like Vercel or Heroku.

2. Supabase Role-Based Access Control

- Ensure proper role-based access control (RBAC) is configured in Supabase.
- Use row-level security (RLS) policies for fine-grained access.

3. Encryption

- All communication with Supabase is encrypted using SSL.
- User passwords and sensitive data should be hashed before storage.

4. Data Privacy

- Avoid logging sensitive information.
- Implement data retention policies for older reports.
