Disaster Alert System (DAS)

Setup Guide

Assignment 1: Collaborative Software Development

Group 17

January 2026

Contents

1 Introduction

The **G17 Disaster Alert System (DAS)** is a comprehensive web-based platform designed to monitor, analyze, and alert users about potential environmental hazards and disasters in real-time. The system provides a full-stack solution with a React-based frontend, Flask backend API, and MongoDB database for persistent data storage.

1.1 Key Features

- **Real-time Monitoring:** Continuous tracking and visualization of disaster alerts.
- **Instant Notifications:** Multi-channel alert delivery via Email and SMS (Twilio integration).
- **User Dashboard:** Interactive web interface for monitoring status, history, and trends.
- **Admin Authorization:** Role-based access control with authorized user management.
- **Geolocation Awareness:** Location-based alert filtering and radius-based notification delivery.
- **RESTful API:** Comprehensive API endpoints for alert creation, retrieval, and user management.

2 Project Structure

2.1 Directory Overview

```
G17_Disaster_Alert_System/
          Backend/                      # Flask backend application
                app.py                   # Main Flask application
                Dockerfile               # Docker image for backend
                requirements.txt          # Python dependencies
                __pycache__/
          Frontend/                     # React + TypeScript
   frontend
                src/
                      App.tsx             # Main app component
                      main.tsx            # Entry point
                      pages/              # Page components
                      components/         # Reusable components
                      contexts/           # React contexts
                      hooks/              # Custom React hooks
                Dockerfile               # Docker image for
   frontend
                package.json             # Node.js dependencies
                vite.config.ts           # Vite build configuration
```

```
18                tailwind.config.ts      # Tailwind CSS
    configuration
19        tests/                          # Test suite
20              backend/                   # Backend unit &
    integration tests
21              frontend/                  # Frontend tests
22        reports/                        # Documentation & reports
23        docker-compose.yml              # Docker Compose
    orchestration
24        .env                            # Environment variables
25        README.md                       # Project overview
```

Listing 1: Project Directory Layout

3 System Requirements

Before installing the system, ensure your environment meets the following requirements:

3.1 Hardware Requirements

- **Processor:** Intel/AMD dual-core or equivalent (2+ GHz recommended).
- **RAM:** Minimum 4 GB (8 GB recommended for smooth Docker operation).
- **Storage:** At least 2 GB free disk space.
- **Internet Connection:** Required for API calls (OpenStreetMap, Twilio).

3.2 Software Requirements

3.2.1 For Docker Setup (Recommended)

- **Docker:** Version 20.10+ (`https://www.docker.com`)
- **Docker Compose:** Version 1.29+ (usually included with Docker Desktop)
- **Git:** For cloning the repository

3.2.2 For Local Setup (Without Docker)

- **Python:** Version 3.9+ (`https://www.python.org`)
- **Node.js:** Version 16+ (`https://nodejs.org`)
- **Bun:** Package manager for Node (Optional, can use npm instead)
- **MongoDB:** Version 5.0+ (`https://www.mongodb.com`)
- **Git:** Version control system
- **OS:** Windows 10/11, macOS 10.15+, or Ubuntu 20.04+

4 Installation Guide

4.1 Step 1: Clone the Repository

```
git clone https://github.com/Davda-James/G17_Disaster_Alert_System.git
cd G17_Disaster_Alert_System
```

Listing 2: Clone Repository

4.2 Step 2: Configure Environment Variables

1. A `.env` file must exist in the root directory.
2. Kindly ensure to have it or create it from the template and fill in the required variables.

```
# MongoDB Configuration
MONGO_URI=mongodb://mongo:27017/my_database

# JWT Configuration
JWT_SECRET_KEY=your-secret-key-change-this

# Twilio SMS Configuration (Optional)
TWILIO_ACCOUNT_SID=your_twilio_sid
TWILIO_AUTH_TOKEN=your_twilio_token
TWILIO_NUMBER=+1234567890

# SMTP Configuration (Optional, for email notifications)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your_email@gmail.com
SMTP_PASSWORD=your_app_password
FROM_EMAIL=notifications@disasteralert.com
SMTP_USE_TLS=True
```

Listing 3: Required .env Configuration

5 Setup: Docker Approach (Recommended)

This is the recommended approach as it isolates dependencies and simplifies deployment.

5.1 Prerequisites

- Docker Desktop installed and running
- 4+ GB RAM available for containers

5.2 Installation Steps

5.2.1 Step 1: Verify Docker Installation

```
docker --version
docker-compose --version
```

Listing 4: Check Docker Installation

Expected output: `Docker version 20.10+` and `Docker Compose version 1.29+`

5.2.2 Step 2: Build and Start Containers

```
# From project root directory
docker-compose up -d --build

# Verify all services are running
docker-compose ps
```

Listing 5: Start Docker Services

5.2.3 Step 3: Check Container Logs

```
# Check backend logs
docker-compose logs -f das_backend

# Check frontend logs
docker-compose logs -f das_frontend

# Check database logs
docker-compose logs -f das_mongo
```

Listing 6: View Container Logs

5.3 Docker Service Details

Table 1: Docker Compose Services

Service	Container Name	Port	URL
MongoDB	das_mongo	27017	mongodb://mongo:27017
Flask Backend	das_backend	5000	http://localhost:5000
React Frontend	das_frontend	8080	http://localhost:8080

5.4 Accessing the Application

After containers are running:

- **Frontend (React Dashboard):** `http://localhost:8080`
- **Backend API:** `http://localhost:5000`

6 Setup: Local Development (Without Docker)

Use this approach for local development with more control over individual components.

6.1 Prerequisites

- Python 3.9+
- Node.js 16+
- MongoDB 5.0+ running locally
- Git

6.2 Part A: MongoDB Setup

6.2.1 On Windows

```
# Download MongoDB Community Edition from:
# https://www.mongodb.com/try/download/community

# After installation, MongoDB service should auto-start
# Verify connection:
mongosh  # or mongo for older versions
```

Listing 7: MongoDB Setup on Windows

6.2.2 On macOS

```
# Install via Homebrew
brew tap mongodb/brew
brew install mongodb-community

# Start MongoDB service
brew services start mongodb-community

# Verify connection
mongosh
```

Listing 8: MongoDB Setup on macOS

6.2.3 On Linux (Ubuntu)

```
# Install MongoDB
wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt
    -key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
    focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d
    /mongodb-org-5.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org

# Start service
sudo systemctl start mongod
sudo systemctl enable mongod

# Verify connection
mongosh
```

Listing 9: MongoDB Setup on Linux

6.3 Part B: Backend Setup (Flask)

6.3.1 Step 1: Create Virtual Environment

```
cd Backend

# On Windows
python -m venv venv
.\venv\Scripts\activate

# On macOS/Linux
python3 -m venv venv
source venv/bin/activate
```

Listing 10: Setup Python Environment

6.3.2 Step 2: Install Dependencies

```
pip install -r requirements.txt
```

Listing 11: Install Backend Dependencies

6.3.3 Step 3: Configure Environment

1. Ensure `.env` file in root directory has:

```
MONGO_URI=mongodb://localhost:27017/my_database
JWT_SECRET_KEY=your-secret-key
# ... other variables

```

Listing 12: Backend .env Configuration

2. Update `MONGO_URI` to use `localhost` instead of `mongo` service name

6.3.4 Step 4: Run Flask Server

```
# From Backend directory (with venv activated)
python app.py

# Expected output:
# * Running on http://0.0.0.0:5000
# * Press CTRL+C to quit
```

Listing 13: Start Flask Backend

6.4 Part C: Frontend Setup (React + TypeScript)

6.4.1 Step 1: Navigate to Frontend Directory

```
cd Frontend
```

Listing 14: Setup Frontend

6.4.2 Step 2: Install Dependencies

```
# Option 1: Using npm
npm install

# Option 2: Using Bun (faster)
bun install
```

Listing 15: Install Frontend Dependencies

6.4.3 Step 3: Start Development Server

```
# Using npm
npm run dev

# Using Bun
bun run dev

# Expected output:
# VITE v4.x.x ready in xxx ms
#     Local: http://localhost:5173
#     press h + enter to show help
```

Listing 16: Start React Development Server

6.5 Accessing Local Application

- **Frontend:** `http://localhost:5173` (or `http://localhost:8080` depending on config)
- **Backend API:** `http://localhost:5000`

7 Default Admin Account

The system automatically creates a default admin user on first run:

```
Email:    admin@gov.org
Password: admin123
Status:   Authorized User
```

Listing 17: Default Admin Credentials

- This account is created/verified automatically on backend startup
- If account already exists, authorization status is ensured to be `True`

8 Configuration Details

8.1 Backend Configuration (Flask)

Key environment variables in `.env`:

- **MONGO_URI:** MongoDB connection string
  - Docker: `mongodb://mongo:27017/my_database`
  - Local: `mongodb://localhost:27017/my_database`
- **JWT_SECRET_KEY:** Secret for JWT token generation (change in production!)
- **TWILIO_*:** SMS notification credentials
- **SMTP_*:** Email notification credentials

8.2 Frontend Configuration (React)

- **API Base URL:** Backend should be accessible at `http://localhost:5000`
- **Port:** Default Vite dev server runs on `localhost:5173`
- **Build Output:** Production build outputs to `dist/` directory

8.3 Database Configuration (MongoDB)

- **Database Name:** `my_database`
- **Collections Auto-created:**
  - `users`: User accounts and profiles
  - `alerts`: Disaster alerts and notifications
- **Volume Persistence:** Docker uses `mongo_data` volume for data persistence

9 Testing

9.1 Running Backend Tests

```
cd tests/backend

# Run all tests
pytest

# Run with verbose output
pytest -v

# Run specific test file
pytest test_alerts.py

# View coverage report
pytest --cov=../Backend
```

Listing 18: Run Backend Tests

9.2 Running Frontend Tests

```
cd Frontend

# Run tests with Vitest
npm run test

# Run with coverage
npm run test:coverage

# Watch mode
npm run test:watch
```

Listing 19: Run Frontend Tests

10 Common Issues and Troubleshooting

10.1 Docker Issues

10.1.1 Issue: Containers fail to start

- **Solution 1:** Check Docker daemon is running

```
docker ps

```

- **Solution 2:** Rebuild without cache

```
docker-compose down
docker-compose up -d --build

```

10.1.2 Issue: MongoDB connection refused

- **Check .env:** Ensure `MONGO_URI=mongodb://mongo:27017/my_database` (not localhost)
- **Check network:** Services must be on same Docker network

```
docker network ls
docker network inspect g17_disaster_alert_system_das_network

```

10.1.3 Issue: Port already in use

- **Find process:**

```
# Windows
netstat -ano | findstr :5000

# macOS/Linux
lsof -i :5000

```

- **Kill process or change port** in `docker-compose.yml`

10.2 Local Setup Issues

10.2.1 Issue: ModuleNotFoundError in Python

- **Solution:**

```
# Ensure virtual environment is activated
# Windows
.\venv\Scripts\activate

# macOS/Linux
source venv/bin/activate

# Then reinstall
pip install -r requirements.txt

```

10.2.2 Issue: npm/bun packages not found

- **Solution:**

```
cd Frontend
rm -rf node_modules package-lock.json  # or bun.lockb
npm install  # or bun install

```

10.2.3 Issue: MongoDB connection fails

- **Check if MongoDB is running:**

```
mongosh  # or mongo

```

- **If not running:**

```
# macOS
brew services start mongodb-community

# Linux
sudo systemctl start mongod

# Windows - MongoDB service should auto-start
sc query MongoDB

```

11 API Endpoints

11.1 Authentication

- `POST /api/signup` - Register new user
- `POST /api/login` - User login
- `GET /api/me` - Get current user profile

11.2 User Management

- `PUT /api/user/<user_id>` - Update user profile

11.3 Alerts

- `POST /api/alerts` - Create new alert
- `GET /api/alerts` - Retrieve alerts (with filtering)

Query parameters for `GET /api/alerts`:

- `time`: `24h`, `7d`, `30d` (default: `30d`)
- `type`: Alert type filter (default: `all`)

12 Production Deployment

12.1 Docker Compose for Production

```
# Use environment file for secrets
docker-compose -f docker-compose.yml --env-file .env.production up -d

# Monitor logs
docker-compose logs -f

# Scale services (if needed)
docker-compose up -d --scale backend=2
```

Listing 20: Production Deployment

13 Useful Resources

- **Flask Documentation:** `https://flask.palletsprojects.com/`
- **React Documentation:** `https://react.dev`
- **MongoDB Documentation:** `https://docs.mongodb.com/`
- **Docker Documentation:** `https://docs.docker.com/`
- **Vite Guide:** `https://vitejs.dev/`
- **Twilio API:** `https://www.twilio.com/docs/`

A Quick Reference Commands

A.1 Docker Commands

```
# Quick start
docker-compose up -d --build

# Check status
docker-compose ps

# View logs
docker-compose logs -f das_backend

# Stop all
docker-compose down

# Cleanup everything
docker-compose down -v
```

A.2 Local Setup Commands

```
# Backend startup
cd Backend
source venv/bin/activate  # or .\venv\Scripts\activate on Windows
python app.py

# Frontend startup (new terminal)
cd Frontend
npm run dev  # or bun run dev

# MongoDB verification
mongosh
> show dbs
> use my_database
> db.users.find()
```

A.3 Testing Commands

```
# Backend tests
cd tests/backend
pytest -v

# Frontend tests
cd Frontend
npm run test
```