1. INTRODUCTION

Natural disasters like floods and cyclones cause severe damage to life and property. Timely warnings help reduce risks, but traditional methods are often slow or ineffective. Using the MERN stack, a web-based system can provide accurate, real-time alerts and serve as a centralized platform for disaster information.

1.1 PURPOSE

The purpose of this document is to design and develop an **online Flood and Cyclone Early Warning System** that delivers timely alerts, weather updates, and safety guidelines to users. The system will:

- Provide real-time flood and cyclone alerts based on meteorological data.
- Ensure easy accessibility through a web-based platform.
- Allow users to receive critical notifications and safety instructions.

1.2 DOCUMENT CONVENTIONS

This document uses the following conventions:

- Database
- Application Programming Interface
- User Interface
- MongoDB, Express.js, React.js, Node.js
- Global Positioning System

1.3 INTENDED AUDIENCE AND READING SUGGESTIONS

This project is a prototype for a flood and cyclone early warning system.The system is useful for disaster management authorities, local communities, and individuals in disaster-prone areas. Readers are suggested to focus on the system's alert mechanism, user interface, and real-time data handling to understand its functionality.

1.4 PROJECT SCOPE

The purpose of the online Flood and Cyclone Early Warning System is to provide timely alerts, weather updates, and safety guidelines to users in disaster-prone areas. The system will use a centralized database and real-time data to generate and deliver warnings through a web-based platform. Its main goal is to help authorities and communities prepare in advance, minimize risks, and ensure a simple, user-friendly experience for all users.

1.5 REFERENCES

https://www.earthobservatory.nasa.gov/images/8233/cyclone-sidr-floods-bangladesh

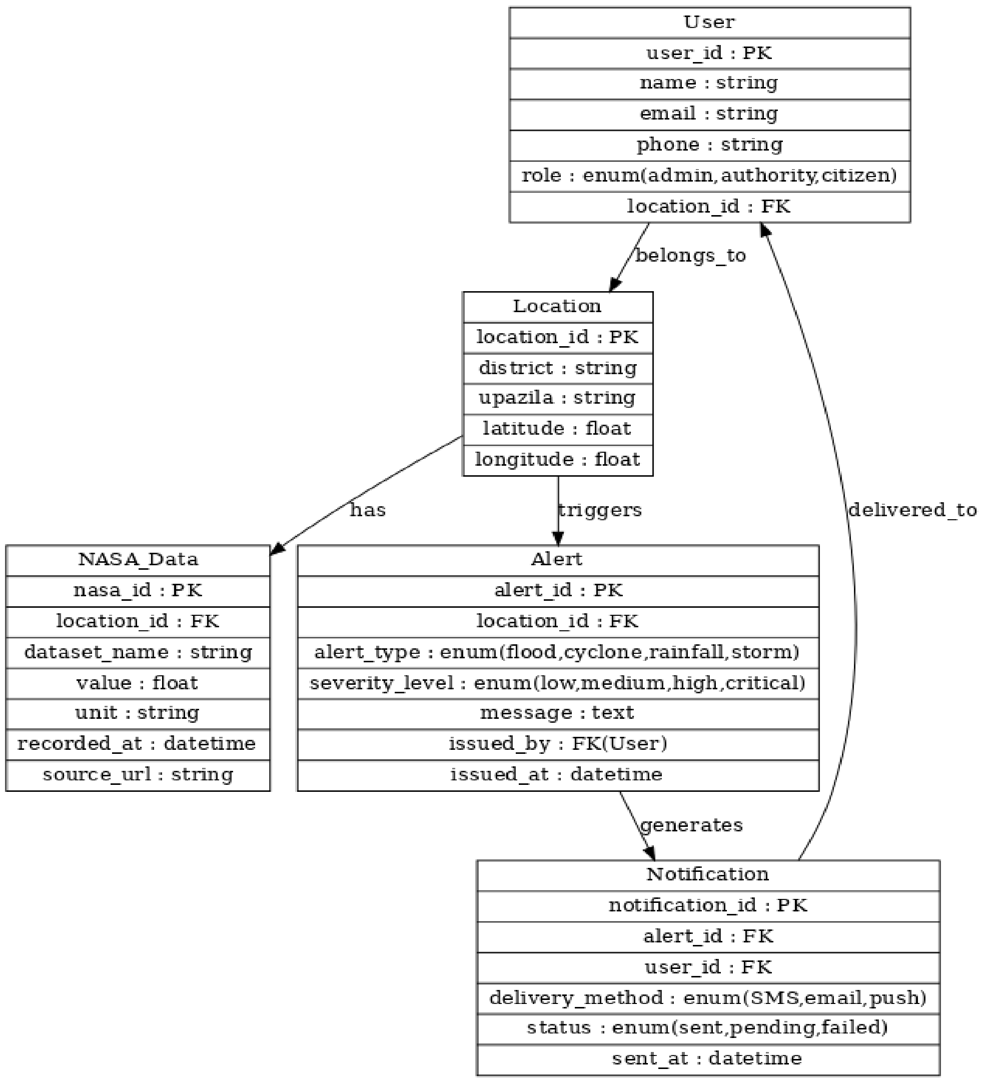
2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

The Flood and Cyclone Early Warning System is a web-based application built using the MERN stack. It integrates real-time weather data, processes it, and delivers alerts to users through a centralized platform. The system provides authorities with a tool to broadcast warnings and helps communities stay informed and prepared. It is designed as a prototype for academic purposes but can be expanded for real-world use with broader data sources and scalability.

2.2 PRODUCT FEATURES

The major features of the airline database system as shown in below entity–relationship model (ER model)



2.3 USER CLASS and CHARACTERISTICS

The disaster alert/early warning system will support **three types of users** with different privileges: **Citizens, Authorities, and Admins**.

1. Citizens (Customer role equivalent)

- **Characteristics**: General population living in flood/cyclone/rainfall/storm-prone areas. They usually interact with the system through a mobile app, SMS alerts, or web portal.
- **Functions available**:
    - Receive alerts about disasters in their location.
    - View latest weather/NASA data (e.g., rainfall, river level).
    - Subscribe/unsubscribe to alert notifications.
    - View history of alerts and personal notification log.
    - Confirm receipt of alerts (optional feedback).
    - Access safety guidelines and emergency contact information.

2. Authorities (Employee role equivalent)

- **Characteristics**: Local government agencies, disaster management officials, and first responders responsible for monitoring and taking preventive measures. They have access to both citizen functions and additional monitoring/management tools.

- **Functions available**:

  - **Citizen Functions** (as above).
  - View **all citizens** who are registered in a given location (to assess population impact).
  - Monitor **live NASA/environmental data** for specific locations.
  - View **active alerts** and their severity levels.
  - Get reports of **alerts delivered successfully/failed**.
  - Track the **status of notifications** (sent/pending/failed).
  - Generate summary reports of **disaster frequency and impact** for a given location.

3. Admins (Administrative role equivalent)

- **Characteristics**: Central system operators and IT staff who manage the database, system configuration, and data feeds from NASA and other sources. They have full privileges.

- **Functions available**:
    - **priority Functions** (as above).
    - Add/Delete a location (district/upazila).
    - Add new NASA dataset sources (e.g., rainfall, cyclone tracking).
    - Define/update **thresholds** for triggering alerts.
    - Add/Delete **alerts** manually (override).
    - Update **alert severity levels** or **messages**.
    - Add/Delete **users** (citizens, authorities).
    - Configure delivery methods (SMS gateway, email server, push service).
    - Monitor **system health** and ensure data consistency across sites.

2.4 OPERATING ENVIRONMENT

The operating environment for the Flood & Cyclone Early Warning System is listed below:

- Distributed Database

- Client/Server System
- Operating System: Cross-Platform (Windows / Linux / macOS)
- Database: MongoDB (NoSQL, distributed database)
- Platform: MERN Stack (MongoDB, Express.js, React.js, Node.js)

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

- **Global Schema:** Unified schema with Users, Locations, NASA_Data, Alerts, and Notifications.
- **Fragmentation Schema:** Data fragmented by location/region for scalability.
- **Allocation Schema:** Fragments distributed across MongoDB clusters or shards.
- **Query Implementation:** Implemented using MongoDB queries (MQL).
- **Global Query Response:** NASA_Data aggregated across regions; Alerts combined with Users for targeted

notifications.

- **Database Implementation:** MongoDB (NoSQL, distributed); can run centralized (local server) or distributed (cloud).

3. SYSTEM FEATURES

3.1 DESCRIPTION and PRIORITY:

The **Flood & Cyclone Early Warning System** maintains real-time information on weather data (from NASA datasets), alerts, and user notifications.

- **High Priority**: Timely and accurate alerts are critical to saving lives and preventing disaster-related damage in affected regions.

3.2 STIMULUS/RESPONSE SEQUENCES

1. **User views alerts for their location**

    - **Stimulus**: User logs in or accesses the dashboard.

  - **Response**: Displays all active alerts relevant to the user’s location.

2. **System generates alert from NASA data**

   - **Stimulus**: New weather data indicates potential flood or cyclone.

   - **Response**: System creates an alert and sends notifications to all affected users.

3. **User subscribes to notifications**

   - **Stimulus**: User selects notification preferences (email, SMS, push).

   - **Response**: System delivers alerts based on the chosen communication channel.

3.3 FUNCTIONAL REQUIREMENTS

DISTRIBUTED DATABASE

- The database stores NASA weather data, alerts, and user information in a **distributed manner**, allowing scalable access and processing.

- Data can be **sharded or replicated** across clusters based on location or dataset type for faster queries and high availability.

3.4 CLIENT/SERVER SYSTEM

- The system follows a **client/server architecture**:
    - **Client**: React.js front-end interface for users to view alerts and notifications.
    - **Server**: Node.js + Express.js back-end handles API requests, processes NASA data, generates alerts, and communicates with MongoDB.
- All data resides on the **server side**, while the client executes the application interface.

4. EXTERNAL INTERFACE REQUIREMENTS

4.1 USER INTERFACES

- Front-end software: React.js (responsive web interface for users to view alerts and notifications).

- Back-end software: Node.js with Express.js (REST APIs for alert generation, user management, and data handling).

4.2 HARDWARE INTERFACES

- Operating system: Windows, Linux, or macOS.
- A browser that supports HTML5, CSS3, and JavaScript.

4.3 SOFTWARE INTERFACES

Software used	**Description**
Operating system	Cross-platform OS (Windows, Linux, macOS) for best compatibility and deployment flexibility.
Database	MongoDB (NoSQL) to store NASA datasets, alerts, and user information.
React.js	Front-end framework for building interactive user interfaces and real-time alert dashboards.
Node.js + Express.js	Backend runtime and framework to implement APIs, process data, and manage notifications.

4.4 COMMUNICATION INTERFACES

The system supports all modern web browsers.

- The system is web-based and accessible through any modern web browser.

- Alerts and notifications are delivered via multiple channels such as **email, SMS, or push notifications**.

5. NONFUNCTIONAL REQUIREMENTS

5.1 PERFORMANCE REQUIREMENTS

- The system should fetch and process NASA weather data efficiently to generate timely alerts.

- Alerts must be delivered to users within seconds after being issued.

- The application should handle multiple users simultaneously without performance degradation.

- The database should support fast read/write operations for storing and retrieving user, location, and alert data.

- The web interface should be responsive on both desktop and mobile browsers.

5.2 NORMALIZATION

- The database is designed to reduce redundancy by storing each piece of information only once.

- Tables are broken into smaller, thematic tables (Users, Locations, NASA_Data, Alerts, Notifications) to avoid modification anomalies.

- The database is normalized up to **3NF**, which is sufficient for preventing insertion, update, and deletion anomalies while maintaining performance.

5.3 SAFETY REQUIREMENTS

- Regular database backups will be maintained to prevent data loss from hardware failures or catastrophic events.

- In case of a failure, the database can be restored from backups and any committed operations can be reapplied from transaction logs to recover the most recent state.

5.4 SECURITY REQUIREMENTS

- User authentication and role-based access control will secure sensitive operations (e.g., issuing alerts).
- Data transmission between front-end and back-end will be encrypted (HTTPS).
- Notification data and user information will be stored securely in the database with proper access restrictions.

5.5 SOFTWARE QUALITY ATTRIBUTES

- **Availability:** Alerts should be available and delivered in real-time to users during extreme weather events.
- **Correctness:** Alerts should accurately reflect the location, type, and severity of floods or cyclones.
- **Maintainability:** The system should support regular updates for backend logic, front-end UI, bug fixes, and security patches.
- **Usability:** The application should be user-friendly and accessible on multiple devices, including desktops and mobile browsers.