SOFTWARE REQUIREMENTS SPECIFICATION

for

REAL-TIME DISASTER INFORMATION AGGREGATION

Version 1.0

Prepared by : Kartikey Sapkal
Aryan Tamboli
Onkar Katkamwar
Madake Gahinath
Siddesh Amrutkar
Anant Patil

Submitted to : MIT AOE
Lecturer

May 30, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to outline the software requirements for a Real-Time Disaster Information Aggregation System. This system will assist emergency response teams by gathering, categorizing, and presenting disaster-related data in real time from various sources.

1.2 Document Conventions

Font Style and Size

- **Headings:** Bold, size 14, Times New Roman.
- **Subheadings:** Bold, size 12, Times New Roman.
- **Body Text:** Regular, size 12, Times New Roman.

Numbering System

- Sections and subsections are numbered hierarchically (e.g., 1, 1.1, 1.1.1).

Language

- This document uses formal and technical English appropriate for software engineering documentation.

Terminology and Notations

- **SRS:** Software Requirements Specification
- **API:** Application Programming Interface
- **NLP:** Natural Language Processing (used for classifying and analyzing disaster-related data)
- **ML:** Machine Learning (used for disaster event detection and prediction)
- **GUI:** Graphical User Interface
- **JSON/XML:** Common formats used for data exchange

1.3 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) document is intended for the following stakeholders:

- **Developers:** To understand the functional and non-functional requirements and implement the system accordingly.
- **Project Managers:** To oversee the project planning, monitor progress, allocate resources, and ensure timely delivery based on clearly defined requirements.
- **Testers and Quality Assurance Teams:** To design test plans and validate that the system meets all outlined requirements and use cases.
- **Clients and End Users (Students, Teachers, Directors, and Office Staff):** To verify that their needs and expectations are correctly interpreted and will be addressed by the proposed system.
- **System Designers and Architects:** To develop the architecture of the system and ensure that it aligns with the requirements defined here.

Reading Suggestions

- **Section 1 – Introduction:** Recommended for all readers to gain a high-level understanding of the project goals, scope, and purpose.
- **Section 2 – Overall Description:** Important for developers, testers, and designers to understand user roles, system environment, and constraints.
- **Section 3 – System Features:** Crucial for developers, testers, and users to grasp the functional capabilities and interactions within the system.
- **Appendices (if any):** Useful for all technical readers to get clarity on definitions, acronyms, and supporting data such as diagrams or database schema.

This document assumes that the reader has a basic understanding of web applications and information systems. For a better grasp, familiarity with role-based systems and database-driven applications will be helpful.

1.4 Project Scope

The scope of the project titled **Real-Time Disaster Information Aggregation** is to develop a robust and scalable system that collects, processes, and disseminates disaster-related information from various sources in real time. The system is intended to assist authorities, emergency responders, and the general public in making informed decisions during natural or man-made disasters.

The key objectives and boundaries of the system include:

- Aggregation of real-time data from diverse sources such as social media, news APIs, meteorological services, and crowd-sourced inputs.
- Classification and filtering of disaster-related information to eliminate noise and ensure relevance.
- Visualization of affected areas through an interactive and intuitive user interface using maps and data dashboards.
- Notification and alert mechanisms for timely dissemination of critical information to relevant stakeholders.
- Secure data handling and scalability to support high volumes of real-time inputs during peak disaster periods.

The project focuses on building a prototype that demonstrates core functionalities such as data ingestion, classification, visualization, and alerting. Future enhancements may include integration with AI-based prediction models, multilingual support, and support for voice-based queries or reporting.

This version does not currently cover physical response coordination (e.g., deployment of rescue teams), satellite communication integration, or offline data access features. These aspects can be considered for later phases based on feedback and technological feasibility.

2 Overall Description

2.1 Product Perspective

The **Real-Time Disaster Information Aggregation System** is designed as a standalone web-based application that aggregates disaster-related data from multiple online and sensor-based sources. It can be integrated with existing emergency response frameworks and platforms to enhance situational awareness. The system adopts a modular architecture to facilitate future scalability and extension.

2.2 Product Functions

The major functions of the system include:

- Real-time data collection from APIs, social media, and weather services.
- Filtering and classification of relevant disaster-related content.
- Visual representation of affected regions using map interfaces.
- Notification and alert system to disseminate urgent updates.

2.3 User Classes and Characteristics

The primary users of the system include:

- **General Public:** Individuals seeking real-time updates on disaster events.
- **Disaster Management Authorities:** Government and emergency personnel monitoring disaster zones.
- **NGOs and Volunteers:** Organizations coordinating relief efforts.
- **Researchers and Analysts:** Users analyzing data patterns for study or planning.

Users may vary in technical skills; therefore, the interface is designed to be intuitive and accessible.

2.4 Operating Environment

The application will be deployed on cloud infrastructure, accessible through modern web browsers (e.g., Chrome, Firefox, Edge). It will require:

- Internet connection for real-time data access.
- Server with Python/flask backend.
- Frontend built using typescript or equivalent frameworks.

2.5 Design and Implementation Constraints

- Real-time processing must maintain low-latency performance.
- System should comply with data privacy regulations.
- External APIs may impose rate limits or licensing restrictions.
- Deployment is limited to environments supporting containerized applications (e.g., Docker).

2.6 Assumptions and Dependencies

- External data sources (e.g., APIs) are assumed to be reliable and updated frequently.
- Users have stable internet connectivity.
- Third-party libraries and APIs used in the system will continue to be supported.
- The system assumes basic disaster classification logic (e.g., fire, flood, earthquake) can be implemented using keyword or NLP techniques.

3 External Interface Requirements

3.1 User Interfaces

The system will offer a responsive and intuitive web-based graphical user interface (GUI) accessible via desktop and mobile browsers. Key elements include:

- **Dashboard:** Displays real-time maps, alerts, and summaries of disaster events.
- **Search and Filter:** Allows users to search disaster information based on location, type, and time.
- **Notifications:** Pop-up alerts and banners for critical incidents.
- **Feedback Module:** A form interface for users to submit ground-level disaster updates.

3.2 Hardware Interfaces

The system will be hosted on cloud infrastructure, and there are no special hardware interfaces required at the client side. However, the system will be designed to interface with the following:

- **Disaster Monitoring Sensors:** (Optional) APIs from IoT-based sensors or weather stations that provide structured data feeds.
- **Mobile Devices:** Devices with GPS sensors for location-based feedback submission.

3.3 Software Interfaces

The system will interact with several external and internal software components:

- **APIs:** RESTful APIs from services like OpenWeatherMap, Google Maps, Twitter, and government disaster management systems.
- **Data Processing Modules:** Python modules for data collection, filtering,clustering, and NLP-based classification.

3.4 Communications Interfaces

The system will use standard web-based communication protocols:

- **HTTP/HTTPS:** For client-server communication.

- **Email Gateway APIs:** For pushing alerts and notifications to registered users.
- **JSON/XML:** Data interchange formats for internal and external API integrations.

4 System Features

4.1 Real-Time Disaster Data Aggregation

- **Description:** Aggregates real-time data from multiple sources, including news APIs, social media platforms, weather APIs, and official government sources.
- **Functional Requirements:**
  - Collect data periodically from predefined APIs.
  - Normalize and preprocess incoming data.
  - Store relevant disaster-related information in the database.

4.2 Disaster Detection and Classification

- **Description:** Uses rule-based and machine learning techniques to identify and classify types of disaster such as floods, earthquakes, fires, etc.
- **Functional Requirements:**
  - Apply keyword matching and NLP models and transformers to classify the type of disaster.
  - Tag each event with severity and geolocation.

4.3 Dynamic User Dashboard

- **Description:** A web interface that presents real-time disaster events on a map along with filters, summaries, and trend analytics.
- **Functional Requirements:**
  - Display disaster events with icons and color codes.
  - Allow filtering based on location, type, and severity.
  - Display graphs showing trends over time.

4.4 User Alert and Notification System

- **Description:** Notifies registered users about relevant disasters based on their location and preferences.

- **Functional Requirements:**
  - Send real-time push notifications via emails
  - Allow users to customize alert preferences.
  - Ensure timely and reliable delivery of alerts.

4.5 Data Export and Reporting

- **Description:** Generates downloadable reports for disaster trends, frequency, and response metrics.
- **Functional Requirements:**
  - Provide CSV or PDF report downloads.
  - Include statistics and visual charts.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

The system must be capable of handling real-time data ingestion from multiple APIs simultaneously without significant delay. It should support concurrent user sessions, enabling at least 1000 users to access the dashboard and receive updates concurrently. The system should display updated information with no more than a 5-second delay from data ingestion.

5.2 Security Requirements

The system will ensure the privacy of all collected data by preventing unauthorized access or misuse. Proper measures will be taken to maintain the confidentiality and accuracy of disaster-related information. Additionally, the system will be designed to optimize resource usage to avoid unnecessary processing or bandwidth consumption, ensuring efficient performance even under high load.

5.3 Resource Optimization

The system is designed to be lightweight and efficient by utilizing APIs for real-time data retrieval. This approach reduces the need for storing large datasets locally, leading to lower RAM consumption and faster performance. The architecture promotes efficient use of computing resources to ensure smooth operation even on systems with limited capacity.

5.4 Software Quality Attributes

The software will emphasize:

- **Reliability:** System must have 99.9% uptime.
- **Scalability:** The system should scale horizontally to handle increased data sources and users.
- **Maintainability:** Modular code design to allow easy updates and integration of new features or APIs.
- **Usability:** The dashboard interface should be intuitive, responsive, and accessible across devices.

5.5 Business Rules

- Only verified government and trusted news sources will be used for real-time data.
- Critical alerts will only be issued if verified by at least two independent sources or validated user reports.
- User-submitted data will be flagged and moderated before appearing on the public interface.