SOFTWARE REQUIREMENTS SPECIFICATION

for

REAL-TIME DISASTER INFORMATION AGGREGATION

Version 1.0

Prepared by : 1. Shivanjali Jagtap (202201040070)
2. Yash Nair (202201040075)
3. Akanksha Patil (202201040085)
4. Yashas Nepalia (202201040082)
5.Trunal Wagh (202301040023)
6.Ravindra Rajhans(202340020)

Submitted to : Santosh Warpe
Lecturer

May 30, 2025

Contents

1 Introduction

1.1 Purpose

This document defines the functional and non-functional requirements for the Real-Time Disaster Information Aggregation System . The system supports emergency responders by collecting, processing, and visualizing real-time disaster data from social media, news portals, and open APIs. It includes modules for data ingestion, NLP-based classification, a user dashboard, and automated alerts. The goal is to improve situational awareness and decision-making during disasters. Deployment details and external service integrations are excluded from this scope.

1.2 Intended Audience and Reading Suggestions

This SRS targets developers, project managers, testers, users (disaster response agencies), and documentation writers involved with the Real-Time Disaster Information Aggregation System.The document starts with an overview for all readers, followed by detailed requirements mainly for developers and testers, and concludes with design and interface details for technical teams and writers.Readers should begin with the overview to understand the problem and key objectives—Data Collection, Categorization, Dashboard, and Alerts—before focusing on sections relevant to their roles.

1.3 Project Scope

The Real-Time Disaster Information Aggregation System is a software solution designed to collect, analyze, and present real-time disaster-related information from diverse sources such as social media, news portals, and open data APIs. Its primary purpose is to enhance situational awareness and support timely decision-making for disaster response agencies.Key benefits include improved data accuracy, faster response coordination, and automated alerts for critical incidents. The system aligns with organizational goals of enhancing public safety, streamlining emergency operations, and leveraging technology for proactive disaster management.This SRS defines the core functionalities of data ingestion, processing, visualization via a user-friendly dashboard, and automated notification services, all aimed at supporting effective disaster response and saving lives.

2 Overall Description

2.1 Product Perspective

The Real-Time Disaster Information Aggregation System is a new, standalone tool created to help improve disaster management. It collects and analyzes live data from sources like social media, news sites, and public APIs. Rather than replacing existing emergency systems, it works alongside them by gathering and processing important information in one place and sending alerts when needed.The system connects to outside data sources through APIs and streaming feeds. Inside, it has different parts that handle collecting data, sorting it, showing it on a dashboard, and sending automated alerts.Because of its flexible design, this system can easily connect with other disaster management tools in the future to share information smoothly.

2.2 Product Functions

The Real-Time Disaster Information Aggregation System provides the following key functions:

- **Data Collection:** Automatically gather disaster-related information from social media platforms, news portals, and open-source APIs in real-time.
- **Data Processing & Categorization:** Filter, analyze, and classify incoming data using Natural Language Processing (NLP) and machine learning to identify relevant disaster events.
- **User Dashboard:** Present real-time updates through an interactive, user-friendly dashboard featuring event feeds, maps, and analytics.
- **Automated Alerts:** Generate and send notifications to users based on customizable criteria for critical disaster events.
- **Reporting & Logging:** Maintain records of collected data, user activities, and system alerts for auditing and analysis.

2.3 User Classes and Characteristics

The Real-Time Disaster Information Aggregation System is built for different types of users with different needs and technical skills:

- **Disaster Response Coordinators:** These are the main users. They watch real-time disaster updates on the dashboard and need quick, accurate information and alerts to make fast decisions. They have a moderate level of technical know-how and understand disaster response.
- **Emergency Management Officials:** They use the system to get an overview of the situation and help coordinate the response. They look at summaries, reports, and alerts but usually don't need deep technical skills since they focus on big-picture planning.
- **Data Analysts:** They check and study the collected data in detail. They use advanced features of the dashboard and need to know both data analysis and disaster-related information.
- **System Administrators:** These users take care of the system's setup, user access, and maintenance. They have strong technical expertise, especially in security and system integration.
- **General Public or Media:** They might have limited access to view some public disaster updates or alerts, but they won't be able to control or change anything in the system.

2.4 Operating Environment

Designed to run on Windows Server 2019 or newer operating systems, with potential support for Linux-based servers to increase compatibility in diverse IT environments.The system integrates with external platforms via RESTful APIs and streaming services to gather disaster-related data from:

- Social media platforms (e.g., Twitter, Facebook)
- Major news portals and RSS feeds
- Open-source data feeds such as government alerts and public safety APIs

It employs MongoDB as the primary database for efficient storage and retrieval of unstructured and semi-structured data, supporting real-time analytics and rapid querying.The system supports modern web browsers, with Chrome as the primary recommended browser for the user dashboard, ensuring responsive and interactive UI/UX across desktops and tablets.Reliable, high-speed internet connectivity is required to enable continuous real-time data ingestion, processing, and timely alert generation.It is designed to coexist smoothly with existing emergency response systems and tools, including GIS mapping software and communication platforms, ensuring no conflicts or performance degradation.Security best practices are implemented, including encrypted data transmission (TLS/SSL), role-based access control, and regular system audits to ensure data privacy and system integrity.

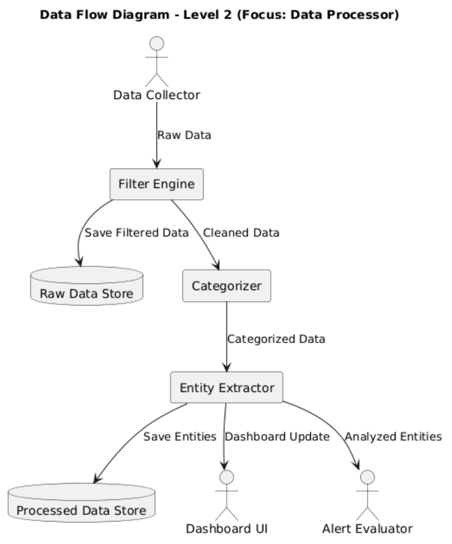


Figure 2.1: Data Flow Diagram - Level 2 (Focus: Data Processor)

2.5 Assumptions and Dependencies

The Real-Time Disaster Information Aggregation System is based on certain assumptions and dependencies that may affect its development and functionality. It assumes the availability of stable internet connectivity for real-time data collection and alert generation. The system relies on continuous access to third-party APIs, including social media platforms like Twitter, news portals, and open data sources. It is also dependent on the compatibility and support of tools such as MongoDB, REST APIs, and modern web browsers.

The platform assumes that users, especially disaster response teams, have basic technical knowledge to navigate the dashboard. Additionally, changes in third-party data access policies or limitations in external services could impact the system's performance. Integration with existing disaster management tools is assumed to be achievable through standard protocols.

For classification of incoming disaster-related data, the system employs a zero-shot BERT algorithm, enabling effective categorization of disaster events without requiring task-specific labeled training data, thus enhancing flexibility and adaptability in real-time scenarios.

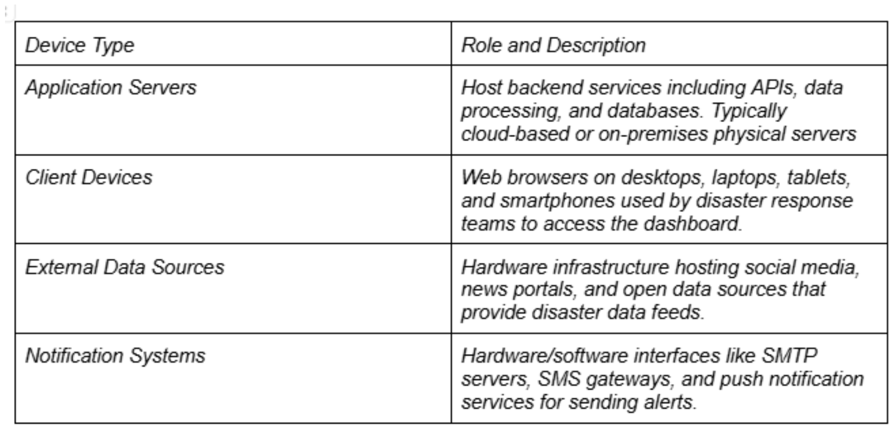
3 External Interface Requirements

3.1 User Interfaces

The system's UI is a responsive, web-based interface built primarily with ReactJS and styled using Material UI or Tailwind CSS, focusing on clarity, accessibility, and fast interactions for disaster response agencies. Streamlit is also used, likely for rapid data dashboards or ML components, either embedded or running alongside the React app. Combining React's flexibility with Streamlit's ease for data apps offers a powerful solution. Best practices include following accessibility standards, optimizing performance, and ensuring mobile-friendliness to support timely decision-making during disasters.

3.2 Hardware Interfaces

This section describes the logical and physical characteristics of the interfaces between the software product and hardware components it interacts with. The disaster data aggregation system operates in a distributed environment, requiring interaction with servers, client devices, network hardware, and external data sources.



Device Type	*Role and Description*
Application Servers	*Host backend services including APIs, data processing, and databases. Typically cloud-based or on-premises physical servers*
Client Devices	*Web browsers on desktops, laptops, tablets, and smartphones used by disaster response teams to access the dashboard.*
External Data Sources	*Hardware infrastructure hosting social media, news portals, and open data sources that provide disaster data feeds.*
Notification Systems	*Hardware/software interfaces like SMTP servers, SMS gateways, and push notification services for sending alerts.*

Figure 3.1: Supported Device Types

4 System Features

This section details the major functional features of the disaster information aggregation system. Each feature is organized with a description, priority level, stimulus/response behavior, and detailed functional requirements. These features aim to support disaster response agencies by improving situational awareness, streamlining response activities, and enhancing real-time decision-making.

4.1 Data Collection Engine

4.1.1 Description and Priority

The **Data Collection Engine** is a foundational feature of the disaster response software system, responsible for automatically gathering relevant disaster-related information from multiple real-time sources. These sources include social media platforms like **Twitter**, news aggregation APIs such as **NewsAPI**, and official government open data feeds that provide alerts and environmental updates.

The primary objective of this component is to ensure a continuous and comprehensive flow of real-time information into the system, enabling timely situational awareness for users and supporting further analysis and classification.

4.1.2 Stimulus/Response Sequences

- A user or a scheduled `cron` job triggers the data collection task.
- The system establishes API connections to all configured data sources.
- Upon receiving data, the system validates the structure and integrity of each record.
- Validated data is stored in the processing pipeline for classification and analysis.
- When a new data source is added or enabled by the user, the system updates the fetch pipeline automatically to include the new endpoint.

4.1.3 Functional Requirements

- **REQ-1.1:** The system shall connect to the **Twitter API** using relevant disaster-related hashtags and keywords to fetch real-time tweets.
- **REQ-1.2:** The system shall retrieve the latest news articles from **NewsAPI** at 15-minute intervals, focusing on disaster-related topics.

- **REQ-1.3:** The system shall connect to multiple **government open data sources** and parse **JSON** or **XML** formats to extract public weather and emergency bulletins.
- **REQ-1.4:** The system shall implement a **retry mechanism** for failed API calls and log such failures, with retries attempted every 5 minutes.
- **REQ-1.5:** The system shall support **dynamic addition or modification** of data sources via a configuration file or administrative user interface.

4.2 Categorization and Data Processing

4.2.1 Description and Priority

The **Categorization and Data Processing** module transforms unstructured or semi-structured incoming data into actionable insights. It employs **Zero-Shot BERT**, a transformer-based language model, to classify data into disaster types—such as *earthquake*, *cyclone*, or *flood*—and to assign severity levels based on predefined scoring rules. The system ensures that only meaningful and verified information proceeds to the dashboard by filtering out irrelevant or low-confidence entries.

4.2.2 Stimulus/Response Sequences

- As new data flows in from the Collection Engine, it enters the processing pipeline.
- Zero-Shot BERT models infer disaster-related categories without the need for task-specific training.
- A rule-based scoring system assigns severity levels based on contextual impact factors such as casualties or geographic location.
- Non-relevant or low-confidence entries are discarded.
- When a user accesses the dashboard, only pre-processed and validated data is displayed.

4.2.3 Functional Requirements

- **REQ-2.1:** The system shall use Zero-Shot BERT models to identify and infer disaster-related labels (e.g., type, location, and event details) from raw text data.
- **REQ-2.2:** The system shall apply a Zero-Shot learning approach to classify each incoming data point into a disaster category.
- **REQ-2.3:** The system shall compute a severity score for each incident using a rule-based scoring algorithm, taking into account impact factors like casualties or affected region.

- **REQ-2.4:** The system shall filter out data points deemed irrelevant based on a configurable threshold.

4.3 Interactive User Dashboard

4.3.1 Description and Priority

The **Interactive User Dashboard** provides a visual interface for end-users, particularly disaster response agencies, to monitor and interact with processed disaster data. It presents information in real-time, categorized by *type*, *severity*, and *location.* Users can perform searches, filter data using specific criteria, view metadata, and configure notification preferences. A map-based interface enhances spatial awareness by visualizing affected areas.

4.3.2 Stimulus/Response Sequences

- When a user logs into the system, the dashboard displays a high-level summary of disaster events with default filters applied.
- Clicking on a specific event opens a detailed view showing all associated metadata, such as source, type, and timestamp.
- Users can update their alert preferences, which are saved and immediately reflected in the system's notification service.

4.3.3 Functional Requirements

- **REQ-3.1:** The dashboard shall display categorized disaster events, organized by type, region, and severity.
- **REQ-3.2:** Users shall be able to perform keyword-based searches and apply date-range filters on the displayed data.
- **REQ-3.3:** Clicking a data item shall open a detailed view that includes full metadata and source references.
- **REQ-3.4:** The dashboard shall integrate a map-based visualization tool, such as `Leaflet.js` or the `Google Maps API`, to display geolocated events.
- **REQ-3.5:** Logged-in users shall have access to configure personal alert settings, including notification types and thresholds.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

The software solution must operate efficiently under various conditions to ensure timely, reliable disaster information delivery. Performance expectations include:

- **Real-time Data Processing:** The system shall process and categorize incoming disaster-related data streams from social media, news portals, and open sources within 5 seconds of receipt to maintain situational awareness.
- **High Throughput Data Collection:** The system shall support collecting and ingesting at least 10,000 data points per minute without performance degradation, ensuring scalability during major disaster events when information volume spikes.
- **Low Latency Notifications:** Critical alerts generated by the system must be delivered to end users within 10 seconds after detection to enable rapid response.
- **Dashboard Responsiveness:** The user dashboard shall load and update information within 3 seconds, even when displaying large datasets or filtering by multiple criteria.
- **Fault Tolerance and Recovery:** The system must maintain performance under partial failures, retrying failed data fetches or alert deliveries without impacting user experience.

Rationale: Fast and scalable processing is essential because disaster situations generate huge, fast-changing data. Delays in data handling or alerting could cause missed opportunities for early interventions, increasing risks to lives and property.

5.2 Safety Requirements

Given the critical nature of disaster response, the system must minimize risks that could arise from inaccurate or delayed information. Safety requirements include:

- **Accuracy and Reliability:** The system must ensure high accuracy (>90%) in filtering and categorizing disaster-related information, minimizing false alarms and missed alerts.
- **Data Integrity:** All collected data and user settings must be stored and transmitted securely, with mechanisms (e.g., encryption, checksums) to prevent data corruption or tampering.

- **User Privacy and Security:** User information, including alert preferences and locations, must be protected following relevant regulations (e.g., GDPR) to prevent unauthorized access.
- **Fail-safe Alerting:** In case of system failures, backup alert mechanisms (e.g., retry queues, fallback notification channels) must ensure critical alerts are still delivered.
- **Compliance:** The system design must comply with applicable disaster management policies and safety regulations, including local government guidelines for emergency communications.

Rationale: The system's purpose is to support life-saving decisions; hence, any malfunction or misinformation can cause serious harm. The safety requirements ensure trustworthiness, protect users, and align with legal and ethical standards.

5.3 Security Requirements

The system must ensure strong security and privacy protections due to the sensitive nature of disaster-related data and user information.

- **User Authentication:** The system shall require secure user authentication using multi-factor authentication (MFA) to ensure only authorized disaster response personnel and users can access sensitive data and system functions.
- **Role-Based Access Control (RBAC):** Access to data and system features shall be governed by RBAC, limiting user permissions based on roles (e.g., admin, analyst, viewer).
- **Data Encryption:** All data in transit must be encrypted using industry-standard protocols (e.g., TLS 1.2 or higher). Sensitive data at rest shall be encrypted using AES-256 or equivalent.
- **Data Privacy Compliance:** The system must comply with relevant data privacy laws and regulations, such as GDPR, CCPA, or local jurisdiction rules.
- **Audit Logging:** The system shall maintain detailed audit logs of all user actions and system events related to data access, modifications, and alerts. Logs should be tamper-proof and retained for a minimum of 1 year.
- **Protection Against Attacks:** The system must include safeguards against common security threats such as SQL injection, cross-site scripting (XSS), distributed denial of service (DDoS) attacks, and unauthorized data scraping.

5.4 Software Quality Attributes

To ensure the system is reliable, maintainable, and user-friendly, the following quality attributes will be prioritized:

- **Availability:** The system shall have an uptime of 99.9%, ensuring it is accessible during disaster events without significant downtime.
- **Reliability:** The system must reliably deliver alerts and dashboard updates with a failure rate below 1% during peak load times.
- **Usability:** The user interface shall be intuitive and easy to navigate, enabling users with minimal technical training to operate the system effectively. Usability tests should target a System Usability Scale (SUS) score above 80.
- **Maintainability:** The codebase shall follow modular design principles and be documented thoroughly to enable rapid bug fixes and feature enhancements. Average time to fix critical bugs should be under 48 hours.
- **Scalability:** The system shall scale horizontally to handle spikes in data volume and user load, supporting up to 100,000 concurrent users during major disaster events.

6 Other Requirements

Appendix A: Glossary

Disaster Event: Any natural or man-made incident (e.g., earthquake, flood, fire) that impacts public safety or infrastructure.

Severity Threshold: A set limit that defines how severe a disaster must be to trigger automatic alerts. It helps focus on important events and avoid unnecessary notifications.

Push Notification: A quick message sent directly to a user's device to inform them immediately about critical disaster events, helping them stay updated in real time.

Role-Based Access Control (RBAC): A security system that gives users access and permissions based on their roles, ensuring they only see or do what's appropriate for their job.

Streamlit: An open-source Python framework used to build interactive, user-friendly web applications quickly and easily. It is commonly used for creating data-driven interfaces and dashboards with minimal coding, ideal for displaying real-time disaster data and visualizations on the frontend of the system.

Zero-Shot BERT: A transformer-based NLP model that can classify text into categories without prior labeled examples by leveraging pre-trained language understanding, useful for disaster data classification when labeled data is limited or unavailable.

Appendix B: Diagrams

Use Case Diagram

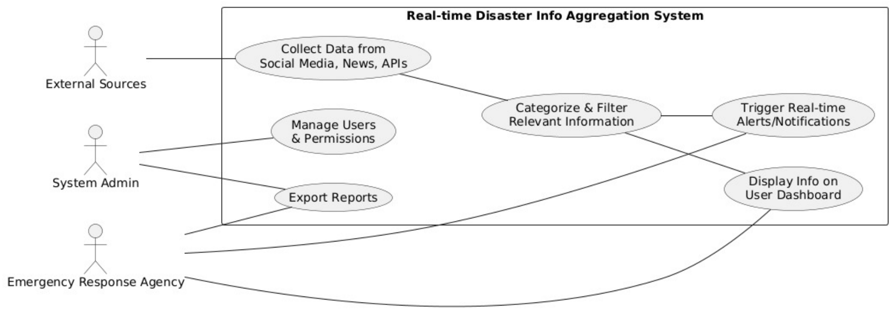

Real-time Disaster Info Aggregation System
External Sources
System Admin
Emergency Response Agency
Collect Data from Social Media, News, APIs
Manage Users & Permissions
Export Reports
Categorize & Filter Relevant Information
Trigger Real-time Alerts/Notifications
Display Info on User Dashboard

Figure 1: Use Case Diagram

Activity Diagram

Sequence Diagram and Class Diagram

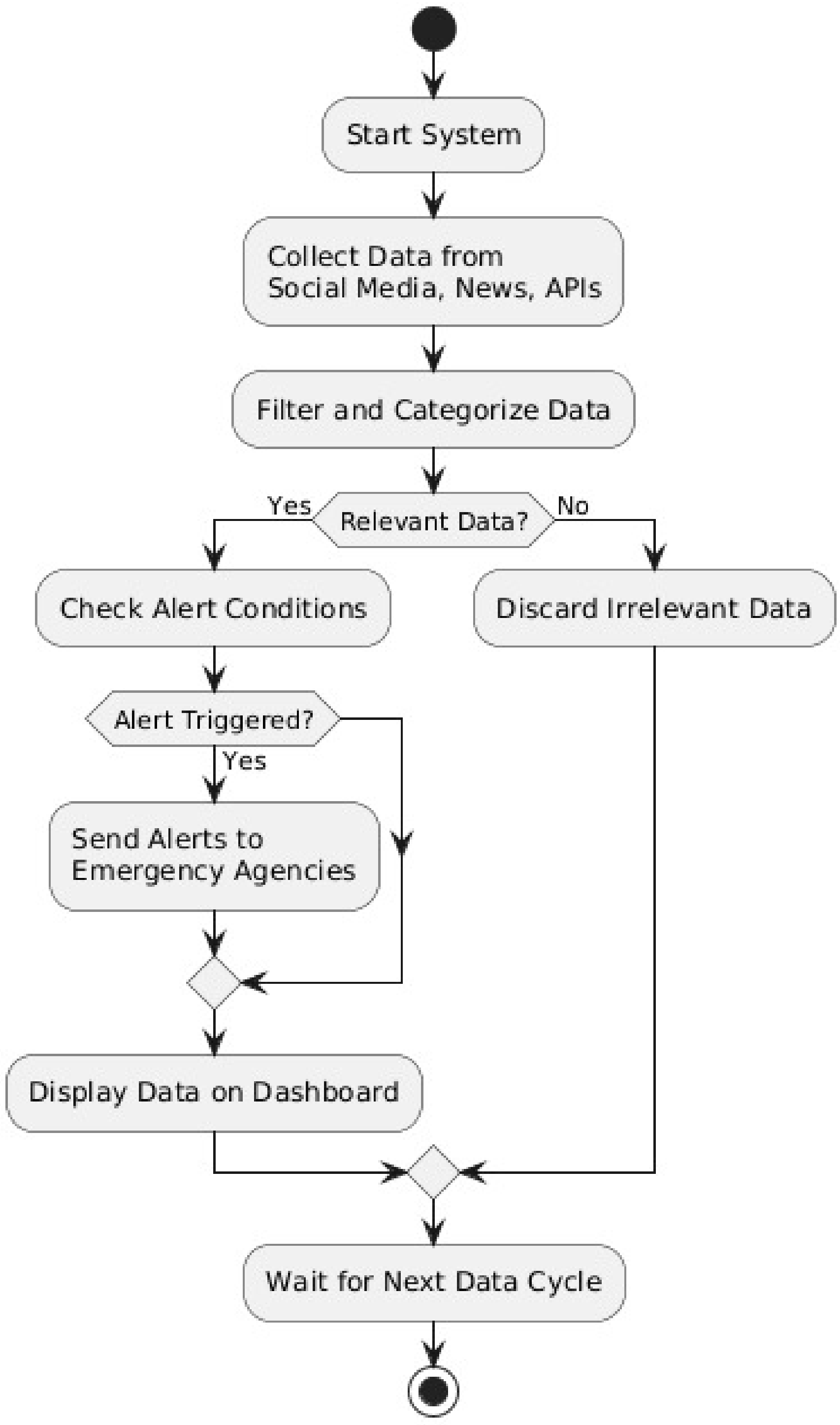

Start System
Collect Data from Social Media, News, APIs
Filter and Categorize Data
Relevant Data?
Yes
No
Check Alert Conditions
Discard Irrelevant Data
Alert Triggered?
Yes
Send Alerts to Emergency Agencies
Display Data on Dashboard
Wait for Next Data Cycle

Figure 2: Activity Diagram

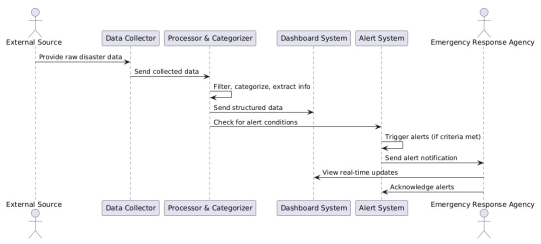


Figure 3: Sequence Diagram

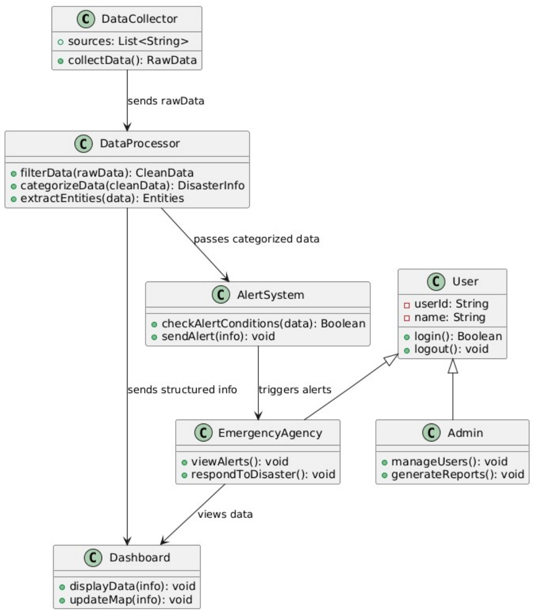


Figure 4: Class Diagram