International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Real-Time Disaster Management and Community Alerting System Using Web Technologies

Sushma V.[1], Amrutha B. G[2], Kruthi D[3], Apoorva[4], Chaithra C. N[5]

[1]Assistant Professor, Dept. of Computer Science and Engineering ATME College of Engineering, Mysuru, India
Email: sushma.sush13@gmail.com
[2]Dept. of Computer Science and Engineering ATME College of Engineering, Mysuru, India, Email: aammuaamrutha1905@gmail.com
[3]Dept. of Computer Science and Engineering ATME College of Engineering, Mysuru, India Email: kruthi7676@gmail.com
[4]Dept. of Computer Science and Engineering ATME College of Engineering, Mysuru, India Email: apoorvaa750@gmail.com
[5]Dept. of Computer Science and Engineering ATME College of Engineering, Mysuru, India, Email: chaitracn8@gmail.com

ABSTRACT—

Natural disasters such as floods, earthquakes, cy- clones and landslides continue to cause significant loss of life and infrastructure damage worldwide. A major contributing factor to this impact is the delay in communicating critical informa- tion to affected communities and responders. Existing disaster management systems are often centralized, slow, and limited in their ability to incorporate community-generated, ground-level information in real time. This paper presents RESCUE.AI, a web-based, real-time disaster management and community alert- ing platform that combines crowd-sourced reporting, geospatial visualization, global disaster feeds and first-response guidance.

RESCUE.AI enables citizens to submit incident alerts with location, severity and description; these alerts are immediately broadcast to all connected users using WebSockets (Socket.IO) and displayed on an interactive map built with Leaflet.js. The backend is implemented using Flask and SQLAlchemy with SQLite as the database, while external disaster feeds from GDACS, NASA EONET and USGS can be integrated to provide global situational awareness. A geolocation-based risk module estimates the user's proximity to active alerts using the Haversine formula. The system is lightweight, mobile-friendly and designed to support United Nations Sustainable Development Goal (SDG) 11: Sustainable Cities and Communities. This paper describes the architecture, implementation, and qualitative evaluation of RESCUE.AI, and discusses its potential as a foundation for future AI-enhanced disaster management solutions.

Index Terms—Disaster Management, Real-Time Alerts, Web- Sockets, Community Reporting, GIS Mapping, Leaflet.js, Flask, SDG 11.

I. INTRODUCTION

Disasters are inherently time-critical events where minutes or even seconds can determine the difference between safety and harm. In many developing regions, disaster communi-

cation still depends heavily on official channels, public an- nouncements or social media posts that are not structured, verified, or delivered in a timely way to the right people. Com- munities on the ground often possess the earliest awareness of danger, yet there are limited mechanisms for them to broadcast structured alerts to others in the vicinity.

With the widespread use of smartphones, GPS and web technologies, there is an opportunity to build systems that are both community-driven and real-time. Instead of treating citizens as passive recipients of information, modern platforms can treat them as active sensors who can create, update and verify disaster-related data. When combined with official and global disaster data sources, such systems can significantly increase situational awareness and enable faster response.

RESCUE.AI is designed as a lightweight, real-time disaster management platform that addresses the following needs:

- Allow any user to submit verified alerts with geolocation, severity and description.
- Broadcast new alerts instantly to all connected clients using WebSockets.
- Visualize incidents on an interactive map with severity- based markers.
- Provide a simple risk indication to users based on their proximity to active alerts.
- Supply first-aid guidance that is accessible even under low-connectivity conditions.

The system is intentionally built with a stack that is easy to deploy and maintain in academic and community settings: Flask for the backend, SQLite for storage, Socket.IO for

real-time communication, and Leaflet.js for mapping. The platform also aligns with Google Solution Challenge themes and directly supports UN SDG 11 by promoting safer, more resilient communities.

The rest of this paper is organized as follows. Section II reviews related work and existing solutions. Section III defines the problem and objectives. Section IV describes the system architecture and design. Section V discusses implementation details and algorithms. Section VI presents results and quali- tative evaluation. Section VII concludes with future work.

II. RELATED WORK

Several research studies have focused on improving dis- aster communication, crisis mapping, and real-time alerting systems. This section reviews the most relevant works.

Okolloh introduced one of the earliest and most influential community-driven crisis reporting systems in *"Ushahidi: A Web and Mobile Platform for Crowdsourcing Crisis Infor- mation"* [1]. Ushahidi demonstrated the potential of crowd- sourced reporting during emergencies; however, the platform lacks modern real-time features like WebSockets and requires considerable deployment resources.

Palen and Liu, in their widely cited work *"Citizen Com- munications in Crisis: Anticipating a Future of ICT-Supported Public Participation"* [2], emphasized the shift from tradi- tional one-way disaster communication models to participa- tory, citizen-driven systems. This work showed that individuals often possess the earliest situational awareness during crises, reinforcing the need for platforms like RESCUE.AI that em- power public reporting.

Global disaster alert systems such as GDACS have been detailed in the report *"The Global Disaster Alert and Coor- dination System (GDACS): Alerts and Coordination for Rapid Response"* [3]. GDACS combines satellite data and official government feeds to provide reliable early warnings. How- ever, it lacks micro-level, user-generated input from affected communities.

NASA developed the *"Earth Observatory Natural Event Tracker (EONET)"* [4] to aggregate global natural events including wildfires, storms, and earthquakes. While scientif- ically accurate, EONET is designed primarily for researchers, and not optimized for interactive, community-centered incident reporting.

USGS provides highly accurate earthquake detection through the *"Earthquake Hazards Program"* [5], which offers near-real-time seismic data. These official systems are essential but are limited to scientific measurements and do not incorporate human-sourced observations.

Real-time communication technologies have been explored by Belqasmi et al. in *"Real-Time Web Applications Using WebSockets: A Review of Technologies and Architectures"* [6]. Their analysis confirms that WebSockets drastically reduce latency compared to traditional HTTP polling, making them suitable for emergency alert broadcasting. However, the work is technological and does not present an integrated disaster platform.

GIS-based visualization has been studied in depth by Man- sourian et al. in *"A GIS-Based Framework for Real-Time Disaster Management"* [7]. Their framework highlights the role of spatial mapping for emergency planning but relies on complex GIS systems that may not be feasible for lightweight community deployment.

Mobile disaster alert systems have also been proposed. Vi- jayalakshmi and Nadhamuni developed an early smartphone- based alert model in *"Android Based Disaster Alert Notifica- tion System Using GPS and GSM"* [8]. While effective for alert dissemination, it lacks real-time map visualization and crowdsourced reporting.

Overall, the literature indicates a gap for systems that integrate:

- community-generated reporting,
- real-time WebSocket broadcasting,
- GIS-based visualization,
- and external global disaster feeds.

RESCUE is designed to address this multi-dimensional gap with a lightweight, accessible, and real-time disaster communication platform.

III. Problem Definition and Objectives

A. *Problem Statement*

The core problem addressed in this work can be stated as: *How can we design and implement a web-based platform that enables real-time, community-driven disaster reporting and visualization, while integrating geolocation, severity- based risk assessment and external disaster feeds, in a way that is lightweight, accessible and suitable for deployment in*

resource-constrained settings?

B. Objectives

The main objectives of RESCUE.AI are:

- To provide a platform where users can submit disaster alerts with key attributes such as title, description, lati- tude, longitude, location name and severity.
- To broadcast newly created alerts in real time to all connected users using WebSockets.
- To visualize alerts on an interactive map with severity- based markers.
- To estimate a simple risk indicator for users based on their distance to active alerts.
- To provide offline-friendly first-aid and safety tips within the same interface.
- To integrate live disaster news feeds from external sources, where available.

IV. System Design and Architecture

A. Overall Architecture

RESCUE.AI follows a client–server architecture with real- time communication. The main components are:

Web Client: A browser-based interface built with HTML, CSS and JavaScript that renders the map, forms and live data.

- **Backend Server:** A Flask application providing REST APIs and Socket.IO events.
- **Database:** SQLite database storing alerts and relevant metadata.
- **Real-Time Layer:** Flask-SocketIO managing WebSocket connections.
- **External Services:** Optional integration modules for GDACS, NASA EONET and USGS feeds.

Fig. 1 illustrates the high-level architecture of the system.

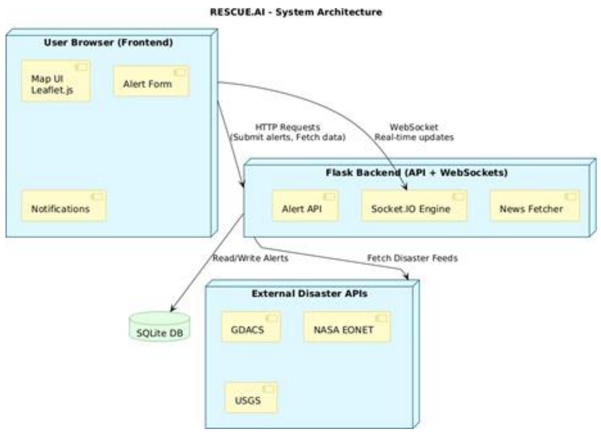


Fig. 1. High-level system architecture of RESCUE.AI.

B. Module Description

The system is divided into the following modules:

1) Alert Management Module: Handles creation, storage, retrieval and resolution of alerts. It exposes REST endpoints such as:

- GET /api/alerts – list alerts.
- POST /api/alerts – create a new alert.
- PATCH /api/alerts/{id}/resolve – mark an alert as resolved.

2) *Real-Time Broadcast Module:* Upon successful creation or update of an alert, the server emits Socket.IO events (e.g., new_alert, alert_resolved) to all connected clients.

3) *Map and Visualization Module:* The frontend uses Leaflet.js to render an interactive map. Each alert is repre- sented by a circle marker whose color corresponds to the severity level (low, medium, high, critical). Popups display details such as title, description, reporter, and timestamp.

4) *Geolocation and Risk Module:* The browser attempts to obtain the user's geolocation via the Geolocation API. The system then computes the distance between the user's location and nearby alerts using the Haversine formula:

$d = 2R \arcsin$

$s_{\sin 2} \Delta\phi + \cos(\phi\ ) \cos(\phi\ ) \sin 2\ \Delta\lambda\ !$, (1)

where R is the Earth's radius, $\phi 1$ and $\phi 2$ are latitudes, and $\Delta\phi$, $\Delta\lambda$ are differences in latitude and longitude.

5) First-Aid Help Module: Static or semi-static content provides first-aid steps for common scenarios such as bleeding, cardiac arrest, burns and earthquake safety. This information is displayed as cards in the UI and can be extended in future to be AI-powered.

6) News Feed Module: This module calls external APIs (where available) to fetch recent disaster events and renders them as a news list on the frontend.

V. IMPLEMENTATION

A. Technology Stack

Table I summarizes the technologies used in RESCUE.AI.

Layer	**Technologies**
Frontend	HTML5, CSS3, JavaScript, Leaflet.js,Font Awesome
Backend	Python, Flask, Flask-SocketIO, Flask-CORS
Database	SQLite, SQLAlchemy ORM
Real-Time	Socket.IO (WebSockets)
External Feeds	GDACS RSS, NASA EONET API,USGS GeoJSON (optional)

B. Backend Implementation

The backend exposes REST APIs and Socket.IO events. The core Alert model includes fields such as id, title, description, latitude, longitude, severity, location_name, username, and created_at. On startup, the database tables are created using SQLAlchemy.

When a new alert is submitted via POST /api/alerts, the server:

1) Validates mandatory fields.

2) Inserts the alert into the database.

3) Emits a new_alert Socket.IO event with the alert payload.

Similarly, marking an alert as resolved triggers an alert_resolved event.

C. Frontend Implementation

The frontend is structured as a single-page layout with:

- A disaster map panel.
- An alert submission form.
- Live alerts list.
- First-aid help section.
- News feed section.

JavaScript modules are responsible for: Initializing the Leaflet map.

- Fetching existing alerts and adding markers.
- Listening for Socket.IO events and updating the UI.

- Handling form submission and showing status messages.

Computing and displaying geolocation-based risk feed- back.

D. Sequence of Operations

Fig. 2 illustrates the sequence of operations when a user submits a new alert and other users receive it in real time.

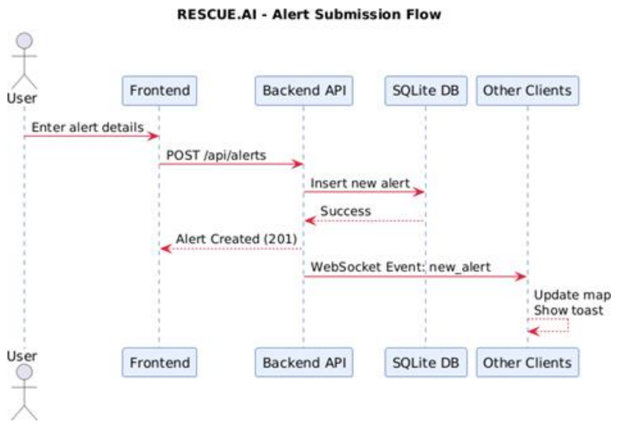


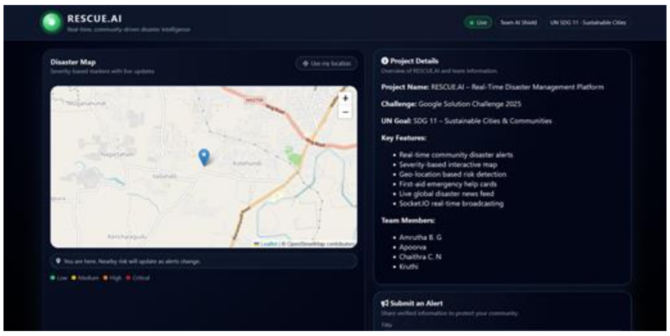
Fig. 2. Sequence of alert submission and broadcast in RESCUE.AI.

VI. Results and Discussion

This section presents the functional validation, performance results, and usability assessment of the RESCUE.AI system. Multiple screenshots of the implemented platform are included to illustrate key interface components and real-time behavior.

A. System Dashboard Overview

Fig. 3 shows the primary dashboard of RESCUE.AI, which integrates the real-time disaster map, project details, team information, and severity legend. The interface is designed to be clean, intuitive, and responsive on both desktop and mobile devices.



Project Details

Project Name: RESCUE.AI – Real-Time Disaster Management Platform

Challenge: Google Solution Challenge 2025

UN Goal: SDG 11 – Sustainable Cities & Communities

Key Features:

- Real-time community disaster alerts
- Severity-based interactive map
- Geo-location based risk detection
- First-aid emergency help cards
- Live global disaster news feed
- Socket.IO real-time broadcasting

Team Members:

- Amrutha B. G
- Apoorva
- Chaithra C. N
- Kruthi

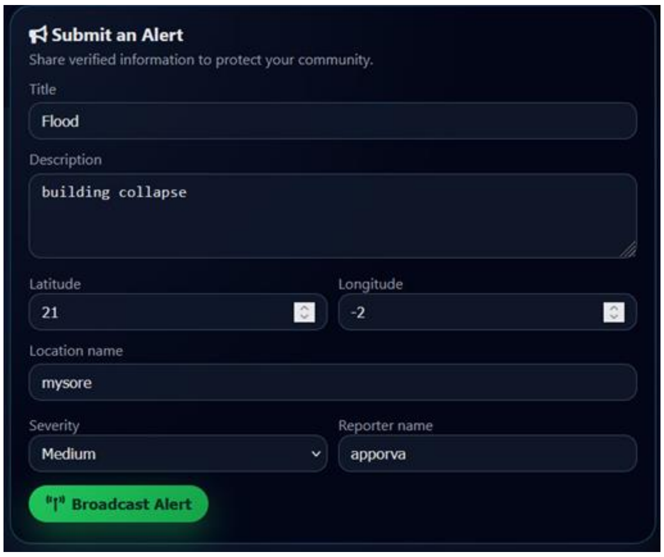
Fig. 3. Main dashboard of RESCUE.AI displaying the real-time disaster map and project information.

B. Functional Validation

The system was tested locally with multiple browsers and devices connected to the same backend. The following func- tional aspects were validated:

- Alerts submitted from one client appear on all connected clients without requiring a page refresh.
- Map markers update in real time when new alerts are created or resolved.
- Geolocation requests succeed on modern browsers, and risk messages update when alerts occur near the user’s location.
- First-aid cards and news feeds render correctly and are scrollable.

Fig. 4 illustrates the alert submission interface used for creating structured disaster alerts.



Submit an Alert
Share verified information to protect your community.

Title: Flood

Description: building collapse

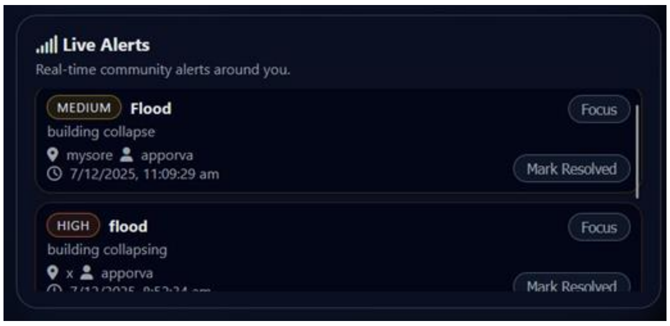
Latitude: 21
Longitude: -2

Location name: mysore

Severity: Medium
Reporter name: apporva

Fig. 4. Alert submission module where users enter incident details including severity, location, and description.

After submission, alerts appear immediately in the “Live Alerts” panel as shown in Fig. 5. Users may optionally mark alerts as resolved.



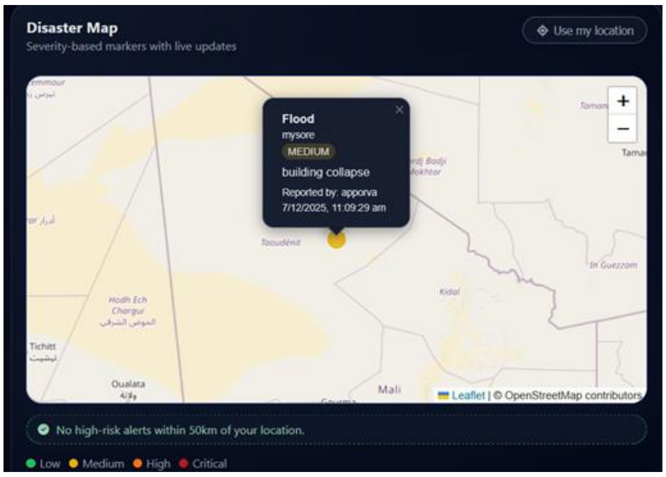
Live Alerts
Real-time community alerts around you.

MEDIUM **Flood**
building collapse
mysore apporva
7/12/2025, 11:09:29 am

HIGH **flood**
building collapsing
x apporva

Fig. 5. Real-time alerts panel showing incoming community alerts with severity categorization.

The alert is simultaneously plotted on the interactive map (Fig. 6), where users can click markers to view full details including time and reporter information.



Disaster Map
Severity-based markers with live updates

Flood
mysore
MEDIUM
building collapse
Reported by: apporva
7/12/2025, 11:09:29 am

No high-risk alerts within 50km of your location.

● Low ● Medium ● High ● Critical

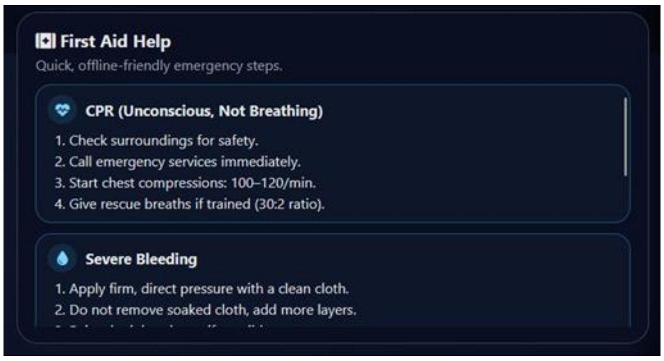
Fig. 6. Map visualization of an alert with severity-based marker and detailed popup.

C. *Performance Observations*

During evaluation, the round-trip latency between sub- mitting an alert and seeing it reflected across other clients remained under a few hundred milliseconds. Socket.IO effi- ciently maintains persistent WebSocket connections and auto- matically falls back to HTTP polling when needed.

SQLite provided reliable performance for this prototype. For higher loads, SQLAlchemy allows seamless migration to database engines such as PostgreSQL or MySQL.

The first-aid help module (Fig. 7) performed well, provid- ing a static yet crucial resource for immediate offline-ready guidance.


First Aid Help
Quick, offline-friendly emergency steps.

CPR (Unconscious, Not Breathing)
1. Check surroundings for safety.
2. Call emergency services immediately.
3. Start chest compressions: 100–120/min.
4. Give rescue breaths if trained (30:2 ratio).

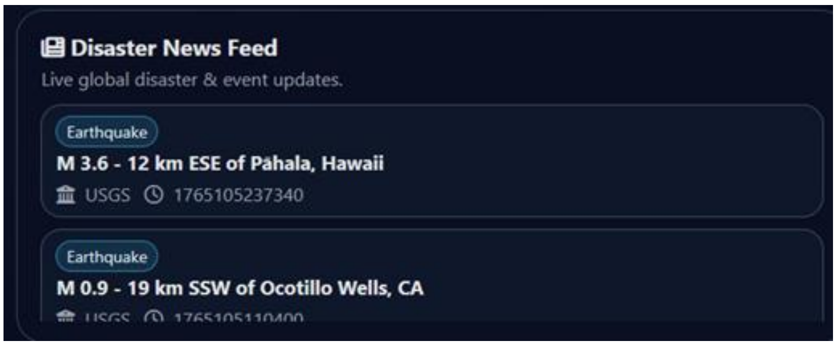
Severe Bleeding
1. Apply firm, direct pressure with a clean cloth.
2. Do not remove soaked cloth, add more layers.

Fig. 7. First-aid guidance module offering quick and essential emergency response steps.

D. Usability and Impact

A major design objective was interface simplicity. The system separates critical functionalities into distinct sections: map, alert submission, first-aid help, and disaster news. This structure minimizes cognitive load during emergencies.

Fig. 8 shows the integrated disaster news feed, which retrieves updates from global sources such as USGS. This allows users to gain broader awareness beyond their immediate region.


Disaster News Feed
Live global disaster & event updates.

Earthquake
M 3.6 - 12 km ESE of Pāhala, Hawaii
USGS 1765105237340

Earthquake
M 0.9 - 19 km SSW of Ocotillo Wells, CA

Fig. 8. Global disaster news feed populated with real-time earthquake data from USGS.

By integrating community alerts and global feeds, RES- CUE.AI serves as a unified situational awareness platform. It can support universities, municipalities, volunteer groups, and campus emergency response units. Furthermore, the platform can be used for mock drills, training programs, and academic research.

Overall, the results demonstrate that RESCUE.AI is an ef- fective, lightweight, and scalable disaster management solution suitable for practical deployment and further enhancement with AI-driven capabilities.

VII. CONCLUSION AND FUTURE WORK

This paper presented RESCUE.AI, a real-time, web-based disaster management and community alerting system built using Flask, Socket.IO, Leaflet.js and SQLite. The platform enables citizens to submit and receive alerts in real time, visualize incidents on an interactive map, access first-aid instructions and view disaster-related news. The design em- phasizes simplicity, extensibility and alignment with SDG 11 goals for sustainable and resilient communities.

Future work includes:

- Integrating AI-based models to automatically classify alerts and estimate risk severity.
- Adding user authentication and role-based access (e.g., verified responders).
- Developing a mobile application or progressive web app (PWA) for offline support.
- Deploying and evaluating the system in real community or institutional pilot studies.

ACKNOWLEDGMENT

The authors would like to thank the Department of Com- puter Science and Engineering, ATME College of Engineering, Mysuru, for their academic support and guidance during the development of this project.

REFERENCES

[1] O. Okolloh, “Ushahidi: A Web and Mobile Platform for Crowdsourcing Crisis Information,” Harvard Humanitarian Initiative, 2009.

[2] L. Palen and S. Liu, “Citizen Communications in Crisis: Anticipating a Future of ICT-Supported Public Participation,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 727–736, 2007.

[3] Global Disaster Alert and Coordination System (GDACS), “The Global Disaster Alert and Coordination System: Alerts and Coordination for Rapid Response,” United Nations Joint Research Centre, 2005.

[4] NASA EONET, “Earth Observatory Natural Event Tracker (EONET),” Available: https://eonet.gsfc.nasa.gov/.

[5] United States Geological Survey (USGS), “Earthquake Hazards Pro- gram,” Available: https://earthquake.usgs.gov/.

[6] F. Belqasmi, R. Glitho and N. El-Maddahi, “Real-Time Web Applica- tions Using WebSockets: A Review of Technologies and Architectures,” IEEE Internet Computing, vol. 22, no. 2, pp. 45–52, 2018.

[7] A. Mansourian, H. Rajabifard, A. Valadan Zoej and I. Williamson, “Us- ing GIS for Real-Time Disaster Management,” Journal of Computers, Environment and Urban Systems, vol. 30, no. 2, pp. 102–118, 2006.

[8] P. Vijayalakshmi and A. Renuga, “Android Based Disaster Alert Notifi- cation System Using GPS and GSM,” International Journal of Advanced Research in Computer Engineering & Technology, vol. 4, no. 3, pp. 730–734, 2015.

[9] Leaflet.js, “Mobile-Friendly Interactive Maps,” Available: https:// leafletjs.com/.

[10] Flask Framework, “The Pallets Projects,” Available: https://flask. palletsprojects.com/.

[11] Flask-SocketIO, “Real-Time Applications with WebSockets,” Available: https://flask-socketio.readthedocs.io/.

[12] Ushahidi Platform, “Crisis Mapping Tools,” Available: https://www. ushahidi.com/.

[13] United Nations, “Sustainable Development Goal 11: Sustainable Cities and Communities,” Available: https://sdgs.un.org/goals/goal11.