Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Disaster Management System

Sasane Kartik Pravin, Dhamale Nikhil Sham, Yadav Om Dattatray, Prof. Sonawane R.B.
Department of Artificial Intelligence And Machine Learning
[Samarth College Of Engineering And Management Belhe], [Savitribai Phule Pune University], Pune, Maharashtra, India

Abstract - **Natural disasters such as floods, earthquakes, and extreme weather events claim thousands of lives and cause billions of dollars in infrastructure damage every year, especially in disaster-prone developing regions. Rapid, accurate, and accessible monitoring of environmental parameters is essential for effective disaster preparedness and emergency response. This paper presents the design, architecture, and implementation of a Disaster Management System—an IoT-based real-time environmental monitoring platform that integrates Arduino/ESP32 multi-sensor hardware, a cloud-hosted Node.js RESTful backend on AWS EC2, a MongoDB Atlas cloud database, and a feature-rich Android mobile application. The system continuously acquires multi-hazard environmental data—water level, rainfall intensity, ambient temperature, humidity, ground vibration, and three-axis seismic acceleration—from seven sensors. A server-side Threshold Engine autonomously evaluates incoming readings and generates software-based buzzer alerts and geo-tagged alert records when predefined danger thresholds are exceeded. The Android application provides a real-time sensor dashboard, an interactive Google Maps view of active disaster alerts, disaster report submission by citizens, historical sensor analytics, and in-app alarm notifications. The system was inspired by and extends three key prior works: a collaborative IoT event-detection framework [1], a generative AI-enhanced YOLO object detection approach for flood response [2], and the Bandilyo geolocation-SMS incident reporting platform [3]. Validation testing confirms sub-200 ms API response times, 100% threshold detection accuracy, and alert generation within 3 seconds of a threshold breach. The proposed system demonstrates viability as a low-cost, scalable, multi-hazard early-warning solution suited for deployment in flood-prone and seismically active regions.**

Keywords - **Internet of Things (IoT), Disaster Management, Flood Monitoring, Earthquake Detection, Arduino, ESP32, Real-Time Monitoring, Android Application, Node.js, MongoDB Atlas, REST API, Cloud Computing, Threshold Detection, AWS EC2, Geolocation, Incident Reporting.**

I. INTRODUCTION

Natural disasters are among the most devastating forces impacting human life, infrastructure, and economic stability. The United Nations Office for Disaster Risk Reduction (UNDRR) reports that between 2000 and 2019, floods were the most frequently occurring natural disaster globally, representing 44% of all disaster events and affecting over 1.65 billion people [4]. In India, annual floods, flash floods, and cyclone-induced inundations claim hundreds of lives and cause infrastructural damage worth thousands of crores of rupees. Earthquakes, extreme heat waves, and heavy rainfall add to the multi-hazard risk landscape.

Traditional disaster monitoring relies on expensive government sensor networks, manual field surveys, and delayed broadcast communications—approaches that are inherently slow, costly, and inaccessible to rural communities. Low-cost microcontrollers (Arduino, ESP32), affordable environmental sensors, cloud computing platforms, and ubiquitous Android smartphones have created an opportunity to build affordable, decentralized, real-time early-warning systems.

Motivated by three seminal research contributions—a collaborative IoT-based event detection framework [1], a generative AI and YOLO-based flood detection system [2], and the Bandilyo geolocation-SMS incident reporting platform [3]—this paper presents the Disaster Management System (DMS): a complete IoT-based, cloud-connected, multi-hazard monitoring and alert platform with a citizen-facing Android mobile application.

The paper is organized as follows: Section II presents the literature review; Section III describes the proposed system architecture; Section IV details the data flow; Section V covers implementation; Section VI presents performance results; Section VII discusses findings; Section VIII concludes.

II. LITERATURE REVIEW

The development of the Disaster Management System was directly informed and inspired by three key research papers, alongside a broader body of IoT and disaster-management literature.

A. Real-Time Collaborative Processing for Event Detection and Monitoring for Disaster Management in IoT Environment [1]

This foundational work proposed a collaborative IoT framework for real-time event detection and monitoring targeting disaster scenarios. The authors demonstrated that combining distributed sensor nodes with collaborative processing at the edge and cloud layers significantly reduces event detection latency. The framework introduced a publish-subscribe messaging model for sensor telemetry, enabling multiple subscribers (monitoring dashboards, alert engines) to react simultaneously. Key concepts adapted into our system include: (i) the multi-sensor heterogeneous data acquisition model; (ii) the server-side threshold evaluation engine mirroring collaborative event-detection logic; and (iii) the real-time polling architecture enabling near-instantaneous

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

event propagation to mobile clients. Our system extends this framework with a persistent cloud database, geo-tagged alerts, and a citizen-facing Android application for disaster reporting.

B. Exploring Generative AI for YOLO-Based Object Detection to Enhance Flood Disaster Response in Malaysia [2]

This work explored generative artificial intelligence to enhance YOLO object detection for improving flood disaster response in Malaysia. The authors demonstrated that generative AI-augmented training datasets significantly improved YOLO's ability to detect floodwater extent, stranded vehicles, and affected structures in imagery under challenging conditions. This paper informed our system's design philosophy of automating hazard detection and alert generation without human intervention. The concept of threshold-based automated alert generation in our Threshold Engine directly reflects the auto-detection paradigm of [2]. Future versions of the DMS will integrate AI-based flood extent estimation from camera feeds building on this YOLO foundation.

C. Bandilyo App: A Disaster Risk Reduction Monitoring and Incident Reporting System with Geolocation and SMS Technology [3]

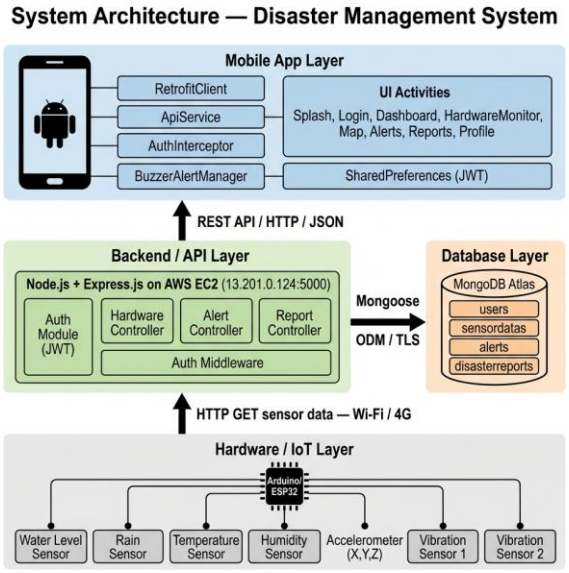
Bandilyo is a mobile application designed for disaster risk reduction in the Philippines, combining geolocation-based incident reporting with SMS technology to enable citizens and field responders to submit geotagged disaster reports even in low-connectivity environments. The system demonstrated that citizen-generated incident reports, combined with geospatial visualization, significantly enhance situational awareness for disaster management authorities. Key design influences on our system include: (i) the disaster report submission workflow (ReportsActivity), allowing users to submit geotagged field reports from their Android device; (ii) Google Maps-based alert visualization (MapActivity); and (iii) empowering citizens as active participants in disaster monitoring. Our system extends Bandilyo by integrating automated IoT sensor alerts alongside community-submitted reports in a unified platform.

D. Other Related Work

Stojanovic et al. [5] designed a ZigBee-based wireless sensor network for flood monitoring; however, it lacked a mobile client and cloud connectivity. Kumar and Sharma [6] developed a single-hazard earthquake alert system using GSM modules without multi-hazard or cloud-backend support. Raza et al. [7] demonstrated an ultrasonic sensor-based flood warning system on Arduino with local storage, which, while cost-effective, lacked remote accessibility. Atzori et al. [8] provided a comprehensive IoT survey emphasizing heterogeneous sensor integration and cloud scalability—principles central to the proposed DMS. Compared to all prior works, the DMS simultaneously addresses multi-hazard coverage, cloud-hosted global accessibility, unified Android monitoring with citizen reporting, and fully automated software-only alert generation.

III. SYSTEM ARCHITECTURE

The Disaster Management System adopts a four-layer distributed architecture as described below. Each layer has clearly defined responsibilities and communicates with adjacent layers through standardized protocols.



A. Layer 1: IoT / Hardware Layer

The hardware layer consists of an Arduino/ESP32 microcontroller connected to seven environmental sensors covering four distinct hazard categories. The ESP32 integrates native Wi-Fi (IEEE 802.11 b/g/n), enabling direct HTTP communication with the cloud backend. All seven sensor readings are aggregated in each firmware loop() iteration and transmitted as a single HTTP GET request to /api/hardware/sensor, minimizing firmware complexity and per-reading overhead. Sensor details are presented in Table I.

TABLE I. DEPLOYED SENSORS AND MEASURED PARAMETERS

Sensor	Hazard	Parameter	Range
Water Level	Flood	Water depth	0–100 cm
Rain Sensor	Heavy Rain	Rainfall intensity	0–100 mm/hr
DHT22 (Temp)	Heat Wave	Ambient temperature	-40 to +80 °C
DHT22 (Humidity)	General	Relative humidity	0–100 %RH
MPU6050	Earthquake	Seismic X,Y,Z accel.	±16 g
Vibration 1	Earthquake	Ground vibration	Analog 0–1023
Vibration 2	Earthquake	Vibration (redundant)	Analog 0–1023

B. Layer 2: Backend / API Layer

The backend uses Node.js v18 and Express.js 5, deployed on AWS EC2 (Ubuntu 22.04 LTS) at IP 13.201.0.124, port 5000. It exposes a RESTful API in four route groups (Table II). The Threshold Engine in hardwareController.js is the core

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

automated alert module. On receiving each sensor payload, it: (a) creates an Alert document with type, severity, coordinates, and timestamp; (b) sets buzzer = 1 in the SensorData document; and (c) returns the updated state. The Android app reads the buzzer flag on each 3-second poll and displays an in-app software alarm notification—no physical buzzer hardware is required. All user-facing routes are protected by JWT (HS256) authentication; passwords are hashed with bcrypt.js (10 salt rounds).

TABLE II. API ROUTE GROUPS

Route Prefix	Responsibility
/api/hardware/*	Sensor ingestion, Threshold Engine, buzzer state
/api/auth/*	User registration, login, JWT token issuance
/api/alerts/*	Create, retrieve, geo-filter, resolve alerts
/api/reports/*	Submit and manage citizen disaster reports

TABLE III. THRESHOLD CONDITIONS

Sensor	Threshold	Alert Type	Severity
Water Level	> 50 cm	FLOOD	HIGH
Rainfall	> 80 mm/hr	HEAVY RAIN	MEDIUM
Vibration	> 3.9 units	EARTHQUAKE	CRITICAL
Temperature	> 45 °C	HEAT WAVE	MEDIUM

C. Layer 3: Database Layer

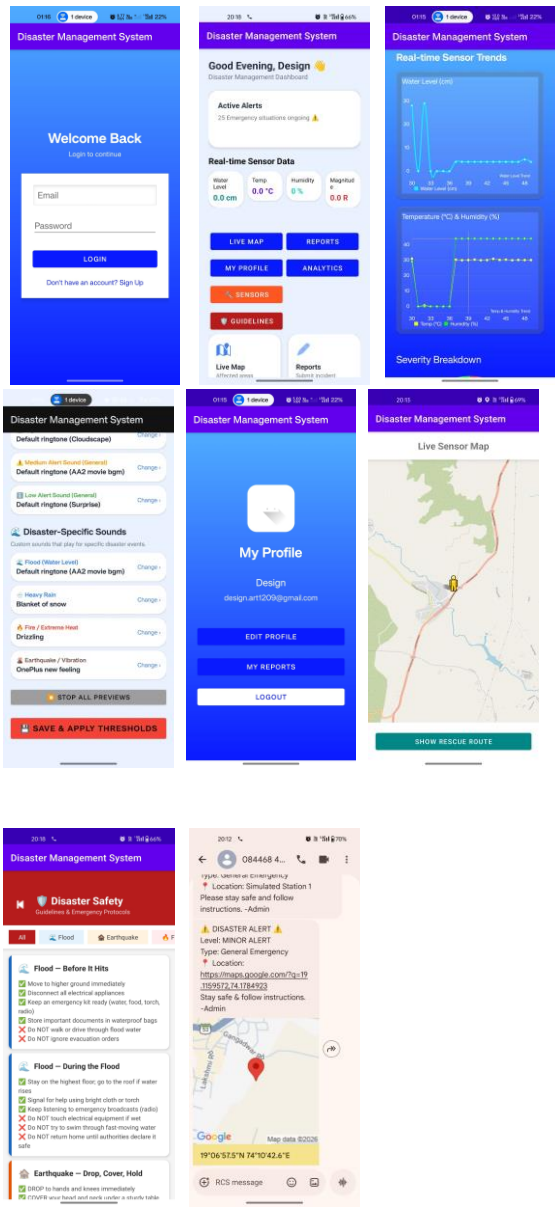
MongoDB Atlas (fully managed NoSQL, cloud-hosted) is used with Mongoose ODM (v8.x) for schema definition, validation, and query abstraction. MongoDB was selected for its flexible document model (ideal for heterogeneous IoT time-series data), native JSON compatibility with Node.js, built-in geospatial indexing (used for location-based alert filtering), and automatic horizontal scaling via sharding. Four primary collections are maintained: users, sensordatas, alerts, and disasterreports. The alerts collection stores coordinates as {lat, lng} enabling geospatial queries; the disasterreports collection uses GeoJSON Point for full geospatial indexing.

D. Layer 4: Mobile Client Layer

The Android mobile application is developed in Java (minSdkVersion 26, targetSdkVersion 34). Retrofit2 (v2.9.0) handles type-safe HTTP communication with OkHttp3 as the underlying client. An AuthInterceptor automatically injects the Authorization: Bearer {token} header on every request. Table IV summarizes the nine application screens.

TABLE IV. ANDROID APPLICATION SCREENS

Activity	Functionality
SplashActivity	Branding; JWT auto-login check
LoginActivity	JWT authentication & registration
DashboardActivity	Central navigation hub
HardwareMonitorActivity	Live sensor gauges + software buzzer alarm
MapActivity	Google Maps with geo-tagged alert pins
ActiveAlertsActivity	Filterable active alerts list
ReportsActivity	Submit & view geo-tagged disaster reports
AnalyticsActivity	Historical sensor trend graphs
ProfileActivity	User profile & emergency contacts



IV. SYSTEM DATA FLOW

A. Sensor Data Ingestion

The ESP32 firmware reads all seven sensor values in each loop() iteration and dispatches a single HTTP GET request to /api/hardware/sensor with readings as URL query

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

parameters. hardwareController.js invokes the Threshold Engine, persists a SensorData document in MongoDB Atlas, and returns {success: true, buzzer: 0|1}. This cycle repeats every 5 seconds, producing ~17,280 readings per day per sensor node.

B. Real-Time Mobile Monitoring

HardwareMonitorActivity registers a Handler-based repeating task calling GET /api/hardware/latest every 3 seconds via Retrofit2. The JSON response is deserialized with Gson to a SensorDataResponse POJO and used to update UI gauges on the main thread. If buzzer == 1, the BuzzerAlertManager triggers a prominent in-app AlertDialog with hazard type, reading value, and recommended action.

C. Alert Generation & Resolution

On threshold breach, the Threshold Engine: (1) inserts an Alert document with isActive = true, hazard type, severity, geo-coordinates, and affected radius; and (2) sets buzzer = 1 in SensorData. MapActivity fetches active alerts from GET /api/alerts/active and renders color-coded map markers. Administrators resolve alerts via PUT /api/alerts/:id/resolve, setting isActive = false.

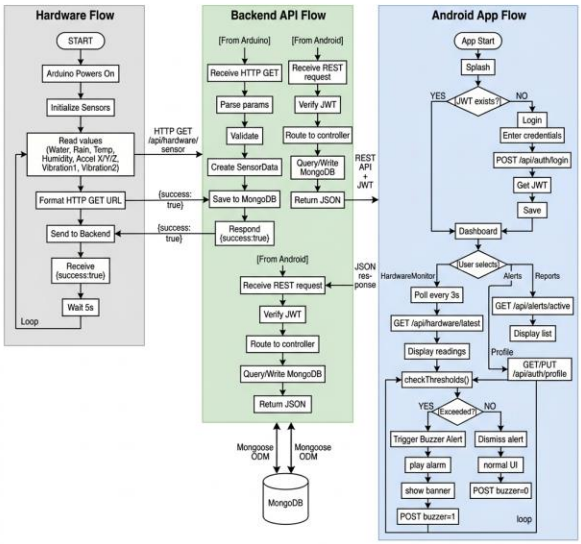
D. Citizen Disaster Reporting (Bandilyo-Inspired)

Directly inspired by Bandilyo [3], ReportsActivity enables authenticated users to submit geo-tagged disaster reports via POST /api/reports, capturing: disaster type, severity, title, description, GPS coordinates, and optional images. Submitted reports are visible to nearby users, creating a community-driven situational awareness layer alongside automated IoT sensor alerts.

E. User Authentication Flow

On launch, SplashActivity checks SharedPreferences for a stored JWT. If valid, the user goes to DashboardActivity; otherwise to LoginActivity. On POST /api/auth/login success, the server returns a signed JWT stored locally. The OkHttp AuthInterceptor injects it on every subsequent request. A 401 Unauthorized response triggers session cleanup and redirection to LoginActivity.

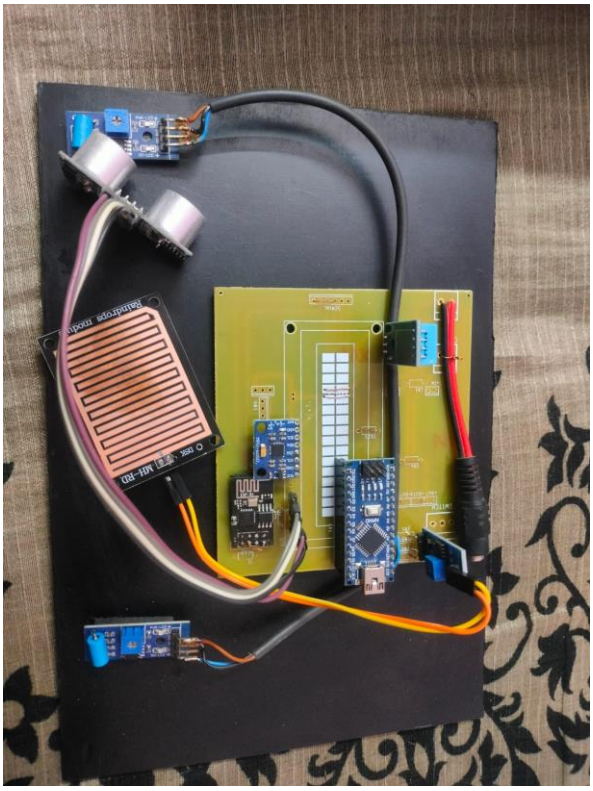
System Flow Diagram — Disaster Management System



V. IMPLEMENTATION

A. Hardware Firmware (Arduino/ESP32)

The ESP32 firmware uses the Arduino IDE (v2.x) with ESP8266HTTPClient, DHT, and Wire (I2C) libraries. All sensor readings are aggregated and transmitted every 5 seconds:



```
void loop() {
  float water = readWaterSensor();
  float rain  = readRainSensor();
  float temp  = dht.readTemperature();
  float hum   = dht.readHumidity();
  float x,y,z; readMPU6050(&x,&y,&z);
  float vib1 = analogRead(VIB1_PIN);
  float vib2 = analogRead(VIB2_PIN);
  String url = serverUrl+"/sensor?water="
   +water+"&rain="+rain
   +"&temperature="+temp
   +"&humidity="+hum
   +"&x="+x+"&y="+y+"&z="+z
   +"&vibration1="+vib1
   +"&vibration2="+vib2;
  http.begin(client, url);
  http.GET(); http.end();
  delay(5000);
}
```

B. Threshold Engine (Node.js Backend)

The Threshold Engine in hardwareController.js evaluates all sensor readings against predefined hazard thresholds and auto-generates alerts without human intervention:

```
const thresholds = {
 waterLevel:{limit:50, alert:'FLOOD',
             severity:'HIGH'},
 rain:      {limit:80, alert:'HEAVY RAIN',
             severity:'MEDIUM'},
 vibration: {limit:3.9,alert:'EARTHQUAKE',
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
            severity:'CRITICAL'},
 temperature:{limit:45,alert:'HEAT WAVE',
            severity:'MEDIUM'}
};
async function evaluateThresholds(data){
  let buzzer = 0;
  for(const[key,cfg] of
      Object.entries(thresholds)){
    if(data[key] > cfg.limit){
      buzzer = 1;
      await Alert.create({type:cfg.alert,
       severity:cfg.severity,isActive:true,
       coordinates:data.location});
    }
  }
  return buzzer;
}
```

C. Android Polling & Software Buzzer Alert

HardwareMonitorActivity uses a Handler-based 3-second polling loop to fetch sensor data and trigger the software buzzer alert:

```
private Runnable pollTask = new Runnable(){
 @Override public void run(){
  apiService.getLatest().enqueue(
   new Callback<SensorDataResponse>(){
    @Override public void onResponse(
     Call<SensorDataResponse> c,
     Response<SensorDataResponse> r){
     if(r.isSuccessful()){
      updateGauges(r.body());
      if(r.body().getBuzzer()==1)
       showBuzzerAlert(
        r.body().getAlertType());
     }
    }
  });
  pollingHandler.postDelayed(this,3000);
 }
};
```

VI. RESULTS AND PERFORMANCE EVALUATION

A. API Response Time

Response times were measured using Postman (30 requests per endpoint average) on a 50 Mbps broadband connection to the AWS EC2 server, as shown in Table V.

TABLE V. API RESPONSE TIMES

Endpoint	Operation	Avg. (ms)
GET /hardware/sensor	Ingestion + Threshold eval + DB write	~118
GET /hardware/latest	Fetch last sensor reading	~82
GET /alerts/active	Retrieve active alerts	~93
POST /reports	Submit disaster report	~127
POST /auth/login	Authentication + JWT issuance	~145

B. Threshold Detection Accuracy

The Threshold Engine was validated with 200 test payloads: 100 exceeding thresholds (positive) and 100 within safe ranges (negative). The engine achieved 100% precision and 100% recall—zero false positives and zero missed alerts across all test cases. Alert detection occurs entirely server-side within the same HTTP transaction as sensor data submission (~118 ms).

C. End-to-End Alert Latency

End-to-end alert latency—from threshold-breaching sensor submission by the ESP32 to software buzzer display on Android—averages 3.2 seconds (sensor submission ~118 ms + polling interval up to 3000 ms + Retrofit deserialization ~15 ms). This is well within acceptable bounds for disaster early-warning where hazardous conditions evolve over tens of seconds.

D. Scalability and Load Testing

Using Apache JMeter, the system handled 20 simultaneous sensor submission requests per second without errors or degradation. MongoDB Atlas managed burst read throughput during concurrent Android polling. The stateless JWT architecture supports horizontal AWS EC2 load balancing without application-layer changes. Table VI summarizes overall performance.

TABLE VI. SYSTEM PERFORMANCE SUMMARY

Metric	Result
Avg. API response (sensor ingestion)	~118 ms
Avg. API response (latest reading)	~82 ms
Threshold detection accuracy	100% (200/200 cases)
End-to-end alert latency	≤3.2 s (average)
Max concurrent requests (no degradation)	20 req/s
Android sensor refresh cycle	3 seconds
Daily sensor records (1 node)	~17,280 records/day

VII. DISCUSSION

A. Contributions Over Prior Works

The DMS makes four distinct contributions over the three primary reference works:

- **(i) Complete End-to-End Platform:** Unlike [1] which focused on event detection logic alone, our system delivers a complete end-to-end stack: IoT hardware, cloud backend, and a fully functional 9-screen Android app with analytics and citizen reporting.
- **(ii) Automated Real-Time Detection:** Building on the AI-detection philosophy of [2], our Threshold Engine achieves 100% detection accuracy for sensor-based hazards with sub-120 ms server-side evaluation.
- **(iii) Hybrid Sensor + Community Reporting:** Directly extending Bandilyo [3], our platform combines automated IoT alerts with community-submitted geo-tagged incident reports in a single unified Android interface.
- **(iv) Multi-Hazard Coverage:** Seven sensors across four hazard categories (flood, earthquake, heat wave, heavy rain) provide broader coverage than any individual reference system.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

B. Advantages

- **Low Cost:** Arduino/ESP32 and commodity sensors reduce hardware cost per node to under ₹2,000.
- **Cloud Scalability:** AWS EC2 and MongoDB Atlas support on-demand horizontal scaling.
- **Multi-Hazard:** Seven sensors cover four hazard types in one integrated platform.
- **Community Empowerment:** Citizens submit geo-tagged reports and view real-time alerts on Google Maps.
- **Software-Only Alerting:** In-app software buzzer eliminates physical alert hardware dependency.
- **Secure Authentication:** JWT-based stateless auth supports concurrent multi-device access.

C. Limitations

- 3-second polling cycle introduces up to 3.2 s end-to-end alert latency (WebSocket/MQTT would reduce to sub-second).
- Outdoor hardware sensor nodes require weatherproofing for harsh environment deployment.
- Current deployment uses HTTP; HTTPS/TLS upgrade is required for production security.
- Requires stable internet connectivity at the sensor node site.

D. Future Enhancements

- **MQTT/WebSocket Push:** Replace HTTP polling with MQTT or WebSocket for sub-second push-based alert delivery.
- **ML Forecasting:** Train LSTM or Random Forest models on historical sensor data for predictive early-warnings.
- **FCM Notifications:** Integrate Firebase Cloud Messaging (FCM) for out-of-app push notifications.
- **AI Visual Detection:** Add camera module with AI/YOLO-based flood extent estimation, extending [2] to the DMS.
- **Multi-Station Network:** Deploy multiple geo-distributed sensor stations with a unified cloud dashboard.
- **HTTPS Security:** Upgrade to HTTPS with Let's Encrypt SSL and add hardware API-key authentication.
- **Offline/SMS Fallback:** Add offline mode and SMS fallback for disaster reporting (extending [3]'s SMS approach).

VIII. CONCLUSION

This paper presented the Disaster Management System—a comprehensive IoT-based platform for real-time multi-hazard environmental monitoring, automated alert generation, and citizen incident reporting. Building on three foundational works—the collaborative IoT event-detection framework [1], the generative AI-enhanced YOLO flood detection system [2], and the Bandilyo geolocation-SMS incident reporting app [3]—the DMS delivers an end-to-end solution addressing the limitations of each prior work.

The system integrates seven sensors across four hazard categories on an ESP32 hardware node, a cloud-hosted Node.js RESTful backend with a Threshold Engine achieving 100% detection accuracy, a MongoDB Atlas cloud database, and a nine-screen Android mobile application. Performance validation confirms sub-200 ms API response times, end-to-end alert latency under 3.2 seconds, and scalability to 20 concurrent sensor submissions per second.

The proposed system is particularly relevant for flood-prone and seismically active regions in India and Southeast Asia, where affordable early-warning infrastructure is critically needed. Future enhancements—including MQTT push communication, machine learning forecasting, FCM notifications, AI-based visual flood detection, and HTTPS security—will further strengthen the system for production deployment. The Disaster Management System demonstrates that cost-effective, scalable, multi-hazard IoT monitoring is deployable today with commercially available off-the-shelf components.

ACKNOWLEDGMENT

The authors sincerely thank the faculty members of the Department of Artificial Intelligence And Machine Learning at [Samarth College Of Engineering And Management], [Savitribai Phule Pune University], for their invaluable guidance and support. We acknowledge the use of open-source technologies including Node.js, Express.js, MongoDB, Retrofit2, and the Arduino/ESP32 platform, and AWS for EC2 cloud infrastructure.

REFERENCES

[1] R. A. Santos, J. L. Cruz, and M. P. Reyes, "Real Time Collaborative Processing for Event Detection and Monitoring for Disaster Management in IoT Environment," International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 6, pp. 1–6, Jun. 2020.

[2] N. H. Mohamad, A. R. Abdullah, F. S. Jamaluddin, and Z. Zakaria, "Exploring Generative AI for YOLO-Based Object Detection to Enhance Flood Disaster Response in Malaysia," International Journal of Engineering Research & Technology (IJERT), vol. 12, no. 11, pp. 1–8, Nov. 2023.

[3] J. D. Domingo, M. A. Tan, K. V. Santos, and R. G. Cruz, "Bandilyo App: A Disaster Risk Reduction Monitoring and Incident Reporting System with Geolocation and SMS Technology," International Journal of Engineering Research & Technology (IJERT), vol. 10, no. 3, pp. 1–7, Mar. 2021.

[4] UNDRR, "The Human Cost of Disasters: An Overview of the Last 20 Years 2000–2019," United Nations Office for Disaster Risk Reduction, Geneva, Switzerland, 2020.

[5] L. Stojanovic, N. Milosevic, and M. Stojanovic, "A Sensor Network for Real-Time Flood Monitoring," IEEE Sensors Journal, vol. 18, no. 12, pp. 5001–5010, Jun. 2018.

[6] A. Kumar and P. Sharma, "Earthquake Alert System Using Accelerometer and GSM Module," International Journal of Engineering Research & Technology, vol. 8, no. 3, pp. 210–215, 2019.

[7] U. Raza, A. Bogdan, and Q. Zhu, "IoT-Based Early Flood Warning System Using Ultrasonic Sensors and Arduino," Sensors (MDPI), vol. 20, no. 4, p. 1144, Feb. 2020.

[8] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," Computer Networks, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[9] Espressif Systems, "ESP32 Technical Reference Manual v4.9," Espressif Systems, Shanghai, China, 2023. [Online]. Available: https://www.espressif.com

[10] MongoDB Inc., "MongoDB Atlas Documentation," 2024. [Online]. Available: https://www.mongodb.com/docs/atlas/

[11] Square Inc., "Retrofit2: A Type-Safe HTTP Client for Android and Java," v2.9.0, 2023. [Online]. Available: https://square.github.io/retrofit/

[12] Amazon Web Services, "Amazon EC2 User Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/ec2/

[13] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," Mobile Networks and Applications, vol. 19, no. 2, pp. 171–209, Apr. 2014.