TECHNICAL REPORT (TOC, LOF, LOT, LOE & REFERENCES)

SmartSOS: AWS-Based Location-Aware Emergency Assistance and Management System

Authors: Areesha Hameed Khan, Saleh bint e Bilal
Supervisors: Dr. Zunnurain, Umair Makhdom
Affiliation: Information Technology University of Punjab, Lahore
Date: 2 January 2026

Contents

1. Literature Review

1.1 Emergency Management and Location-Based Services

Location-based services (LBS) have been widely recognized as critical components in emergency management, enabling applications to locate users and route requests to the nearest responders. Several commercial and government solutions use GPS for navigation and incident reporting, but many still focus primarily on call routing rather than rich data capture and evidence management.

Research on emergency alert systems highlights the importance of capturing real-time context, including coordinates, time, and multimedia, to improve situational awareness for responders. Work on IoT-based panic buttons and earthquake or disaster alert platforms shows that cloud platforms can reliably ingest and process such data at scale.

1.2 Cloud-Based Architectures for Web Applications

AWS prescriptive guidance and architecture blogs describe standard three-tier web architectures consisting of presentation, application, and data tiers deployed within a VPC. These designs commonly use load balancers, EC2 instances or containers for the application layer, RDS for relational storage, and S3 for static assets and backups. Serverless patterns using API Gateway and Lambda are also common, but SmartSOS focuses on EC2 to align with lab experience and course outcomes.

Best practices emphasize network isolation, IAM-based access control, encryption, and monitoring using CloudWatch to achieve secure and observable deployments. These guidelines strongly influence the SmartSOS architecture.

1.3 Gaps in Existing Solutions

While many emergency apps send SMS with GPS coordinates or allow users to call hotlines, fewer solutions offer an integrated platform combining web UI, structured incident workflow, evidence upload, and intelligent guidance. Systems deployed in developed regions may also not be directly suited to local infrastructure and constraints in cities like Lahore. SmartSOS addresses this gap by combining modern web technologies with an AWS-backed architecture tailored for a university-level yet production-like implementation.

2. System Requirements and Analysis

2.1 Functional Requirements

Requirement	Description
FR1	Users shall register and log in to the system.

FR2	Users shall submit emergency reports specifying type (Police, Ambulance, Fire).
FR3	The system shall automatically capture the user's GPS location (with permission).
FR4	Users shall upload one or more photos as evidence for an incident.
FR5	The system shall store incident details and evidence metadata in the database.
FR6	The system shall send email/SMS alerts to configured responders on new incidents.
FR7	Users shall view a history of their submitted incidents and statuses.
FR8	The system shall provide a chat-like interface for first-aid and safety guidance.
FR9	Admin users shall view incidents and update their status.

2.2 Non-Functional Requirements

Requirement	**Description**
NFR1	Security – All communication must use HTTPS, and AWS IAM and security groups must enforce least-privilege access.
NFR2	Scalability – The system should support horizontal scaling of EC2 instances through an Auto Scaling Group and an Application Load Balancer.
NFR3	Availability – RDS should be deployed in Multi-AZ configuration where possible, and S3 should provide durable storage for photos.
NFR4	Usability – The React SPA should be responsive and usable on mobile and desktop browsers.

NFR5	Maintainability – The codebase should be modular, with clear separation between frontend and backend.
NFR6	Observability – CloudWatch must log key application events and metrics.

2.3 Feature-Actor Mapping

SmartSOS is designed for three primary user roles:

- Citizens/Patients: Can submit emergency reports, upload evidence, request guidance, and view their incident history.
- Nurses/Rescuers: Can respond to incidents, update status, and access incident details.
- Administrators: Can manage all incidents, view system health, and configure responder contacts.

3. System Design and AWS Architecture

3.1 High-Level Architecture

SmartSOS adopts a three-tier architecture deployed within an AWS VPC. (See Figure 1 for the architecture diagram.)

- Presentation Tier: A React SPA accessed via web or mobile browsers. Static assets can be served either from NGINX on EC2 or from an S3 static website fronted by CloudFront.
- Application Tier: One or more EC2 instances running Node.js/Express and exposing RESTful APIs behind an Application Load Balancer.
- Data Tier: Amazon RDS (MySQL/PostgreSQL) stores users, incidents, and evidence metadata; Amazon S3 stores incident photos and other unstructured files.

3.2 Network and Security Design

The VPC uses a CIDR block such as 10.0.0.0/16 and is divided into public and private subnets across multiple Availability Zones.

- Public subnets host the ALB and NAT Gateway.
- Private subnets host EC2 instances and RDS.
- Internet Gateway provides outbound internet access via NAT for private subnets.
- Security groups allow HTTPS from the internet to ALB, HTTP/HTTPS from ALB to EC2, and database ports only from EC2 to RDS.
- IAM roles grant EC2 instances scoped permissions to access S3, RDS, SNS, and CloudWatch.

3.3 Application Logic and Data Model

The backend follows a layered architecture:

- API Layer: Express routes handle endpoints for authentication, incidents, evidence, and guidance.
- Service Layer: Business logic for incident workflows, SNS notifications, and guidance rules.
- Data Access Layer: Uses an ORM or query builder to interact with RDS.

Core Database Tables:

Users - id (PK) - name - email - phone - role (citizen, rescuer, admin) - password_hash - created_at	Incidents - id (PK) - user_id (FK) - type (police, ambulance, fire) - description - latitude - longitude - status (NEW, ACKNOWLEDGED, RESOLVED) - created_at - updated_at
Evidence - id (PK) - incident_id (FK) - s3_key - file_type - uploaded_at	GuidanceRules - id (PK) - category - condition - message

3.4 AWS Services Mapping

Feature	**AWS Services**
User Authentication	EC2 + RDS (session management)
Emergency Report	EC2 + RDS + SNS
Photo Upload	EC2 + S3
Incident Status Tracking	EC2 + RDS
First-Aid Guidance	EC2 + RDS (rule store) or in-memory rules

Notification Delivery	SNS with email/SMS subscriptions
Logging & Monitoring	CloudWatch Logs & Metrics
Network Isolation	VPC, Security Groups, IAM

4. Implementation

4.1 Environment Setup

1. Create a new AWS account or use the provided student account.
2. Configure an IAM admin user and enable MFA.
3. Create a VPC with CIDR 10.0.0.0/16, two public subnets and two private subnets across different Availability Zones.
4. Attach an Internet Gateway and configure route tables and a NAT Gateway for private subnets.

4.2 Database and Storage

1. Provision an Amazon RDS MySQL/PostgreSQL instance in private subnets with Multi-AZ enabled if available.
2. Create an S3 bucket for evidence photos with private access, enabling encryption at rest and blocking public access.
3. Optionally create another S3 bucket for static frontend assets.

4.3 Backend (Node.js/Express on EC2)

1. Launch an EC2 instance (e.g., Amazon Linux 2) in a private subnet with an IAM role that allows S3, RDS, SNS, and CloudWatch access.
2. Install Node.js, configure environment variables (DB credentials, S3 bucket name, SNS topic ARN).
3. Implement REST APIs:
   - POST /api/incidents – create incident, store in RDS, upload location, queue SNS notification.
   - POST /api/incidents/:id/evidence – accept multipart uploads, store files in S3, save metadata in RDS.
   - GET /api/incidents – list user incidents.
   - POST /api/guidance – accept high-level scenario input and return rule-based guidance.
4. Configure ALB and an Auto Scaling Group for EC2 instances.

4.4 Frontend (React SPA)

1. Scaffold a React app with pages/components for login, SOS form, evidence upload, guidance chat, and incident history.
2. Use browser geolocation APIs to capture latitude/longitude with user consent.
3. Implement photo capture via file input or camera on mobile and send to backend.
4. Call backend APIs using HTTPS and display responses in a clean UI with role-appropriate views.

4.5 Notifications with Amazon SNS

1. Create an SNS topic and subscribe test email addresses and/or phone numbers.
2. In the backend, publish messages to SNS on new incident creation, including key information and a link to the web dashboard.

4.6 Monitoring with CloudWatch

1. Configure EC2 to send application logs to CloudWatch Logs.
2. Set up CloudWatch Alarms for high CPU, error rates, or low healthy host counts behind the ALB.

5. Testing and Evaluation

5.1 Test Plan

Design black-box test cases for each functional requirement:

Test Case	**Expected Result**
Submit incident with valid data	Incident stored in RDS, notification sent via SNS
Submit incident with missing fields	Validation error returned
Upload valid image files	S3 object created, evidence record inserted in RDS
Request guidance for "severe bleeding"	Appropriate rule-based first-aid response returned
View incident history as citizen	Only own incidents displayed
Update incident status as admin	Status updated in RDS, visible to citizen

Additional testing includes:

- Usability tests on different devices and browsers.
- Basic performance tests by generating multiple concurrent requests.
- Security tests such as SQL injection prevention and authentication bypass attempts.

5.2 Results

Summarize:

- Number of incidents successfully created and stored.
- Notification delivery confirmation via SNS.
- Response times observed under test load.
- Any limitations found (e.g., geolocation not available on some devices).

6. Conclusion and Future Work

SmartSOS demonstrates that a three-tier AWS architecture can effectively support a location-aware, evidence-driven emergency assistance system using widely available web technologies. The implementation shows how React, Node.js/Express, EC2, RDS, S3, SNS, VPC, IAM, and CloudWatch can be combined to address real-world challenges in emergency communication, bystander guidance, and evidence management.

Future enhancements may include:

- Integrating mobile push notifications via Firebase Cloud Messaging.
- Real-time location tracking for responders and civilians.
- Machine-learning-based prioritization of incidents based on severity.
- Integration with official emergency service APIs and dispatch systems.
- Additional security hardening, penetration testing, and deployment automation using CI/CD pipelines.
- Infrastructure-as-code using AWS CloudFormation or Terraform for reproducible deployments.

7. References

"Building a three-tier architecture on a budget," AWS Architecture Blog, 2024. [Online]. Available:

https://aws.amazon.com/blogs/architecture/building-a-three-tier-architecture-on-a-budget/

"Creating a Multi-Tier Architecture with S3, EC2, and RDS: A Step-by-Step Guide," dev.to, 2024. [Online]. Available:

https://dev.to/heritageolaleye/creating-a-multi-tier-architecture-with-s3-ec2-and-rds-a-step-by-step-guide-320g

"Create a web server and an Amazon RDS DB instance," AWS Documentation. [Online]. Available:

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html

"Location-Based Services for Emergencies," GIM International, 2024. [Online]. Available:

https://www.gim-international.com/content/article/location-based-services-for-emergencies

"Category: Amazon Simple Notification Service (SNS)," AWS Architecture Blog. [Online]. Available:

https://aws.amazon.com/blogs/architecture/category/messaging/amazon-simple-notification-service-sns/

"Web & mobile apps – AWS Prescriptive Guidance," AWS. [Online]. Available:

https://aws.amazon.com/prescriptive-guidance/patterns/websitesandwebapps-pattern-list.html

"Technical Documentation in Software Development: Types, Best Practices and Tools," AltexSoft. [Online]. Available:

https://www.altexsoft.com/blog/technical-documentation-in-software-development-types-best-practices-and-tools/

"A simple emergency alert solution with AWS IoT Button," AWS Blog, 2018. [Online]. Available:

https://aws.amazon.com/blogs/publicsector/grandma-emergency-button-a-simple-emergency-alert-solution-with-aws-iot-button/

"Building a Three-Tier Architecture on AWS," YouTube, 2025. [Online]. Available:

https://www.youtube.com/watch?v=JUK43Dw9LN8

"How to Deploy a 3-Tier Architecture on AWS," YouTube, 2023. [Online]. Available:

https://www.youtube.com/watch?v=amiIcyt-J2A

8. Appendices

Appendix A: API Endpoint Specifications

Authentication

- POST /api/auth/register – User registration
- POST /api/auth/login – User login
- POST /api/auth/logout – User logout

Incidents

- POST /api/incidents – Create new incident
- GET /api/incidents – Get user's incidents (with filters)
- GET /api/incidents/:id – Get incident details
- PUT /api/incidents/:id/status – Update incident status (admin only)

Evidence

- POST /api/incidents/:id/evidence – Upload evidence photo
- GET /api/incidents/:id/evidence – List evidence for incident
- DELETE /api/evidence/:id – Delete evidence file (admin/owner only)

Guidance

- POST /api/guidance – Request first-aid/safety guidance
- GET /api/guidance/categories – Get available guidance categories

Appendix B: Database Schema (SQL)

```
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  phone VARCHAR(20),
  role ENUM('citizen', 'rescuer', 'admin') DEFAULT 'citizen',
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);

CREATE TABLE incidents (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  type ENUM('police', 'ambulance', 'fire') NOT NULL,
  description TEXT,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  status ENUM('NEW', 'ACKNOWLEDGED', 'RESOLVED') DEFAULT 'NEW',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE evidence (
  id INT PRIMARY KEY AUTO_INCREMENT,
  incident_id INT NOT NULL,
  s3_key VARCHAR(500) NOT NULL,
  file_type VARCHAR(50),
  uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (incident_id) REFERENCES incidents(id) ON DELETE CASCADE
);

CREATE TABLE guidance_rules (
  id INT PRIMARY KEY AUTO_INCREMENT,
  category VARCHAR(50) NOT NULL,
  condition TEXT NOT NULL,
  message TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Appendix C: Environment Configuration (.env Example)

```
# Database
DB_HOST=smartsos-rds.xxxxx.us-east-1.rds.amazonaws.com
DB_PORT=3306
DB_USER=admin
DB_PASSWORD=your_password
DB_NAME=smartsos

# S3
AWS_S3_BUCKET_NAME=smartsos-evidence-photos
AWS_REGION=us-east-1

# SNS
AWS_SNS_TOPIC_ARN=arn:aws:sns:us-east-1:123456789:smartsos-alerts

# Application
NODE_ENV=production
API_PORT=3000
JWT_SECRET=your_jwt_secret_key
FRONTEND_URL=https://smartsos.example.com
```