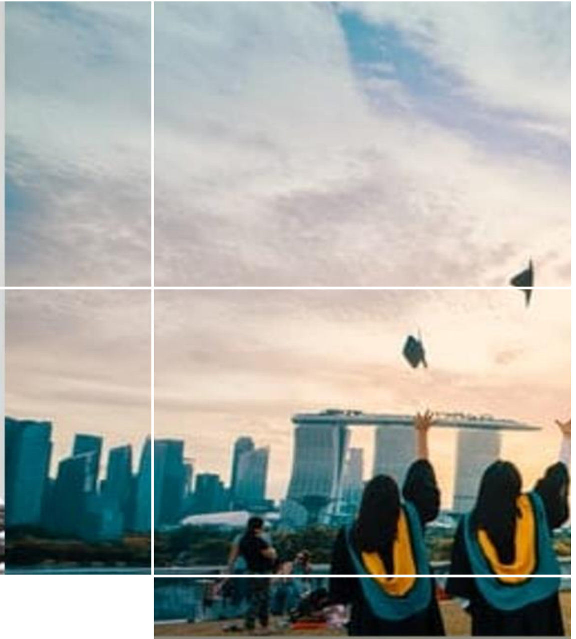


Campus Alert Management System

Group 1:
Mohamed Afeef - 50
Nayana S Babu - 55
Rasha Ameena - 57
Sneha Ann Mathew - 71

AGENDA

1. Literature Review
2. SRS
3. SDD
4. Testing
5. Evaluation and Results
6. Project Plan
7. Conclusion

LITERATURE REVIEW – Existing Solutions

1. Vision-Based Indoor Navigation (ViNav)

- Utilizes smartphone cameras for real-time indoor navigation.
- Replaces GPS, which is unreliable indoors.
- **Limitation:** Not suitable for outdoor or campus-wide coverage.

2. SMS-Based Communication System

- Simple solution using SMS for clearance in institutions.
- **Strength:** Accessible in areas with poor internet.
- **Drawbacks:** Lacks integration, limited scalability, and no automation.

3. COVID-19 Tracking System

- Uses geo-fencing and machine learning for alerts and tracking.
- **Improvement:** Enhanced tracking accuracy.
- **Challenge:** Still limited by GPS signal reliability in some environments.

LITERATURE REVIEW – Institutional Alert Systems

4. Emergency Alert Management System

- Web-based alert platform for campuses.
- Supports emergencies like medical, fire, and natural disasters.
- **Strength:** Real-time alerts to stakeholders.
- **Drawback:** Limited integration with external systems.

Key Gap Identified

- Lack of a **unified, scalable, and integrated** system
  that supports real-time data, location tracking, emergency alerts, and effective user communication.

GAP ANALYSIS

Insights from Literature:

- Systems lack real-time coordination, multi-role access, and location precision.
- GPS reliability and platform integration are consistent concerns.

Need for CAMS:

- A centralized, geolocation-powered alert system.
- Supports **real-time notifications, outbreak detection, and discussion forums.**
- Tailored to institutional settings with role-based access and map-integrated features.

Problems Identified

- No real-time alerts for health issues or emergencies
- Poor communication leads to missed updates
- Privacy concerns in anonymous health reporting
- Low awareness of emergency resources
- Uninformed campus maintenance disrupts schedules

Solution

- Unified platform for health, safety, and event updates
- Real-time alerts and notifications
- Secure anonymous reporting
- Improved student engagement and awareness

Software Requirements Specification – SRS

Functional Requirements

- Disease Spread Alert System
- Maintenance & Repair Information
- Event Management
- Emergency Amenities Access
- Report Handling & Verification
- Approval Mechanism for Health Reports
- Peer Support System

Non-Functional Requirements

- Performance: Fast response, high availability, scalable system
- Security: Data encryption, user authentication, integrity, access control
- Usability: Intuitive UI, consistency, mobile responsive, error handling

Software Requirements Specification – SRS

System Features

- User Authentication
- Emergency Alert Notifications
- Geolocation-Based Disease Monitoring
- Maintenance & Repair Tracking
- Event Notification & Management
- Peer Health Support Groups
- SMS/Push Notification System
- Admin Dashboard
- Club Forum Integration
- Privacy & Security Enforcement

Design & Implementation Constraints

- Regulatory Compliance
- Scalability & Data Handling Limits
- UI Usability Standards
- Data Privacy & Security Protocols
- Tech Limitations: Hardware, API dependencies
- Maintainability & System Interoperability

SOFTWARE DESIGN DOCUMENT (SDD)

1. System Architecture

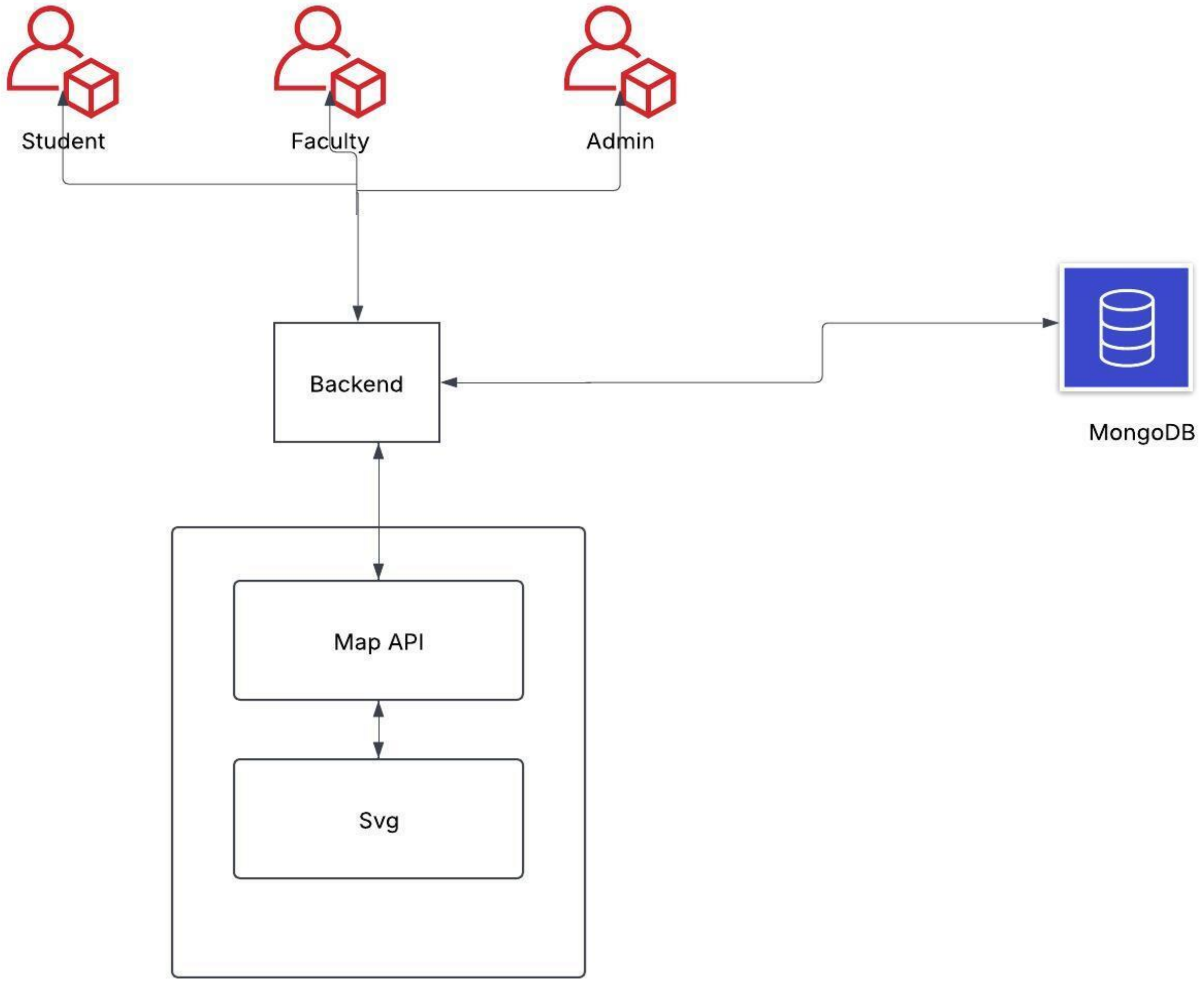
CAMS is composed of three main modules: **Frontend (React.js)**, **Backend (Node.js + Express.js)**, and a **Geolocation System** integrated with **OpenStreetMap and SVG**.

Architecture Flow:

1. Frontend serves as the user interface, handling login, reports, events, chat, and real-time map view.
2. Backend manages logic, authentication, and interacts with MongoDB for data handling.
3. Geolocation System overlays SVG markers (e.g., infected zones, alerts) on campus maps using coordinates from the database.
4. Cloudinary API manages media like profile pictures and event flyers.

Tech Stack:

- **Frontend:** React
- **Backend:** Node.js, Express.js
- **Database:** MongoDB
- **SMS Alerts:** Textbelt
- **Maps & Locations:** OpenStreetMap API
- **Visual Assets:** Inkscape (SVG Editing)



System Interaction Flow

- Users access the system through a web interface for activities like login, dashboard viewing, chat, eport submission etc.
- The frontend sends HTTP requests to the backend for tasks like report submission and marker retrieval.
- The backend processes incoming requests, interacts with the MongoDB database, and returns the necessary data.
- The frontend uses geolocation data to dynamically render and update SVG overlays on the campus map.
- Emergency or medical alerts are pushed in real time, instantly updating map markers and notifying users.

2. Application Architecture Design

CAMS follows a **three-tier architecture**:

- **Presentation Layer**: React frontend with modular pages
- **Business Logic Layer**: Express backend handling authentication, access control, and alert logic
- **Data Layer**: MongoDB for persistent storage

Integrations:

- **OpenStreetMap + SVG:** Location rendering on a scalable campus map.
- **Cloudinary:** Media storage for profile and event images.
- **Textbelt:** Sends emergency SMS notifications.

3. GUI Design

Interface Description

- CAMS provides a **role-based unified interface** accessible to all user roles: Guest, General User, Authorized User, Administrator.
- UI is organized into four major subsystems:
    - **Alerting System**
    - **Event System**
    - **Severity Identification**
    - **Geolocation System**

User Role Interfaces

- **Guest**: View-only access (social events, contact info).
- **General User**: Submit health/repair alerts, join support groups, view emergency markers.
- **Authorized User**:
  - Faculty: Verify health alerts, post updates.
  - Club Rep: Manage social events.
- **Administrator**: Full system control, role assignment, peer group integrity enforcement.

Module Interfaces

- **Alert System**:
  - Health Alerts: Verified via vouching by faculty; mapped via health markers.
  - Repair Alerts: Mapped via repair markers; managed by admin.
  - Social Events: Created by club reps; shown using social markers.
- **Role-Based Authorization**: Role access assigned on registration and enforced throughout the platform.
- **Severity Identification**: Calculates risk (cases × severity level); auto-alerts if threshold exceeded.
- **Geolocation System**:
  - Uses OpenStreetMap with SVG overlays.
  - All markers (health, repair, social) dynamically rendered based on stored coordinates.

4. Database Design

- **Database Used**: MongoDB (document-based, flexible schema)
- **Data Entities**:
- `users`: Stores user details (students, staff, admin)
- `alerts`: Stores all types of alerts (fire, medical, repair)
- `responses`: Captures user replies to alerts
- `communication_logs`: Tracks alert delivery statuses
- `type_of_marker`: Defines categories for map markers (e.g., infected zone)

Scope of testing

Scope of Web Portal

- Admin login and dashboard access
- Marker management (infection, repair, event locations)
- Event creation and updates
- Real-time health and repair alerts
- Report generation (infected cases, repair progress)

Scope of Mobile Device

- Access to campus maps with dynamic markers in mobile platform
- Receive SMS and in-app notifications
- Submit infection reports
- Create an account using email
- Login with all roles of users
- Rendering of image files
- Add image files to the cloud platform
- Profile
- Limitations to the types of image file formats in the device

Scope of testing

Scope of Admin Panel

- Create, edit, delete repair/infection markers
- Approve user-reported infection cases
- Manage user roles
- Send SMS alerts via TextBelt SMS Gateway
- Create an account via backend
- Login as an admin
- Upload image files
- Edit information about the file
- display the file
- Approve and reject the messages in peer support groups to limit violations
- Create external and internal users

Testing Strategies

- Functional Testing-Validate all features as per SRS, Ensure complete user role functionality
- Usability Testing-Test all 5 roles (Guest, User, Faculty, Club Rep, Admin), Verify role-specific access and interface clarity
- Performance Testing-Test load times under- Large geospatial datasets, Image uploads, Multiple simultaneous users (Vercel deployment)
- Cross-Browser/Device Testing-Check compatibility on-Android and Windows OS and browsers Chrome, Firefox, Edge, Safari, Validate limited screen responsiveness
- Security Testing- Role-based access control enforcement, Password encryption in database, No unauthorized access or modifications
- Regression Testing: After bug fixes or new feature addition
- Retesting: Re-execute failed test cases post-bug fix

Testing Environment

Deployment & Technical Stack

- Operating Systems: Windows 11, macOS
- Browsers Tested: Google Chrome, Firefox, Safari (Latest)
- Hardware: Intel i5 processor, 8GB RAM minimum
- Server: AWS cloud-hosted backend
- Frontend Hosting: Vercel (for testing and deployment)
- Database: MongoDB Atlas
- Geolocation: OpenStreetMap with SVG overlay
- Media Handling: Cloudinary
- SMS Gateway: Textbelt
- Authentication: JWT-based login system

Testing Environment

Platforms & Access

- Testing conducted on:
- Laptops and desktops with Windows OS
- Mobile devices running Android OS
- Uses dummy test data including users, infection alerts, repair logs, and events
- JWT authentication tested for session integrity and security

Browser & Device Support

- Cross-browser testing on: Chrome, Firefox, Edge, Safari (latest)
- Device testing includes: Android smartphones/Windows desktops/laptops
- Limited screen responsiveness testing done for variable screen sizes

Testing Environment

QA Tools Used

- TestRail: Test case documentation and management
- Jira: Bug tracking and test task assignment
- Snagit: Screenshots and screen recording for bug reports
- Postman & Browser DevTools: API and network traffic testing

Testing Approach

- Manual testing can be used to test all the functionalities of the web application.
- Automated testing can be used to test the performance and load of the web application.
- Responsive testing will be done to ensure the web application is compatible with different devices and screen sizes.

Deliverables

- **Bug Reports**
  Severity Levels:
  - Critical: System crash/data loss
  - Major: Non-critical flow failure
  - Minor: Incorrect results
  - Trivial: UI/textual errors
- **Test reports from the test cases after deployment.**
  - Concurrent rendering of multiple markers at same location
  - multiple entries of the same user in different roles
  - Tracking of the geolocation of the user at hand
  - Concurrency control between users
  - Outbreak false alarms
  - Simulation of false reports by users
- **Website feedback:** Ensures the site meets its goals and performs well with the target audience.

Evaluations and Results

A. Functional Testing

- All core modules (Alerting, Events, Geolocation, Severity, Access Control) tested and passed.
- Alert Workflow: Simulated alerts verified via vouching and triggered peer groups.
- Event System: Events posted/viewed with map integration and filtering.
- Outbreak Detection: Severity logic triggered alerts as expected.
- Geolocation Mapping: SVG + OpenStreetMap markers rendered accurately.

B. Usability Testing

- Tested across all user roles (Guest, Student, Faculty, Club Rep, Admin)
- Functional and UI responsiveness achieved ~80% success rate
- Navigation found to be easier

Evaluations and Results

C. Security & Reliability

- Vouching system discarded 24% invalid alerts, reducing misinformation.
- Role-based access blocked all unauthorized actions.
- No breaches or bypasses were observed.

D. Limitations & Observations

- Indoor GPS accuracy slightly affected; improvement via vision-based tools suggested.
- Admin bottleneck in registrations and moderation—recommend AI for support.
- Risk of low engagement due to complex UI or alert fatigue.

E. Performance Analysis

- Loading delays from large datasets & high-res graphics.
- Scalability issues under high SMS volume and geospatial data growth.
- API compatibility and internet speed affect availability.
- Error rates rise with DB strain and incorrect vouching logic.

Project Plan

	Start	End	Optimization	Testing
Frontend	20th Feb	30th March	1st – 21st April	11th – 21st April
Backend	8th March	20th April	1st – 21st April	11th– 21st April
Map API	20th Feb	4th April	1st – 21st April	11th – 21st April

Conclusion

- Implementation Completed:
  Developed a fully functional campus alert system with geolocation, alerts, and role-based access.
- Prototype Tested:
  Conducted functional, usability, performance, and security testing across all user roles.
- Key Outcomes:
  - Core modules worked as intended
  - Alerts delivered within 5–10 seconds
  - 80%+ usability effectiveness
  - Role restrictions and vouching validated
- Areas for Improvement:
  - Improve scalability for larger user base
  - Enhance indoor GPS accuracy and UI responsiveness

THANK YOU