**CHAPTER THREE**

**METHODOLOGY OF STUDY**

**3.1 EXISTING METHODOLOGY**

The existing methodology of a campus emergency response system includes the use of multiple communication channels like text alerts, emails, and security notice messages for quick information dissemination. It involves established emergency protocols for various scenarios and regular training and drills for the campus community.

**3.2 OVERVIEW OF PROPOSED METHODOLOGY**

The proposed web-based campus emergency response application includes a centralized dashboard for real-time monitoring, secure user authentication with role-based access, and instant alerts via push notifications. It features tools for incident reporting and tracking, collaboration between campus security and local authorities, and resource management for emergencies. Additionally, it offers analytics and reporting capabilities to evaluate response times and improve overall campus safety.

**3.3. METHOD OF DATA COLLECTION**

1. Response Verification: As respondents finish the surveys, information is automatically gathered and kept in a safe, centralized database.
2. Reminders: Keep an eye on real-time progress and response rates to enable prompt interventions (like reminders) if response rates fall short of expectations.
3. Progress tracking to boost participation, send follow-up reminders to those who did not answer.
4. Automated Data Capture Check answers when needed, especially for surveys that call for authentication (such category-only feedback).

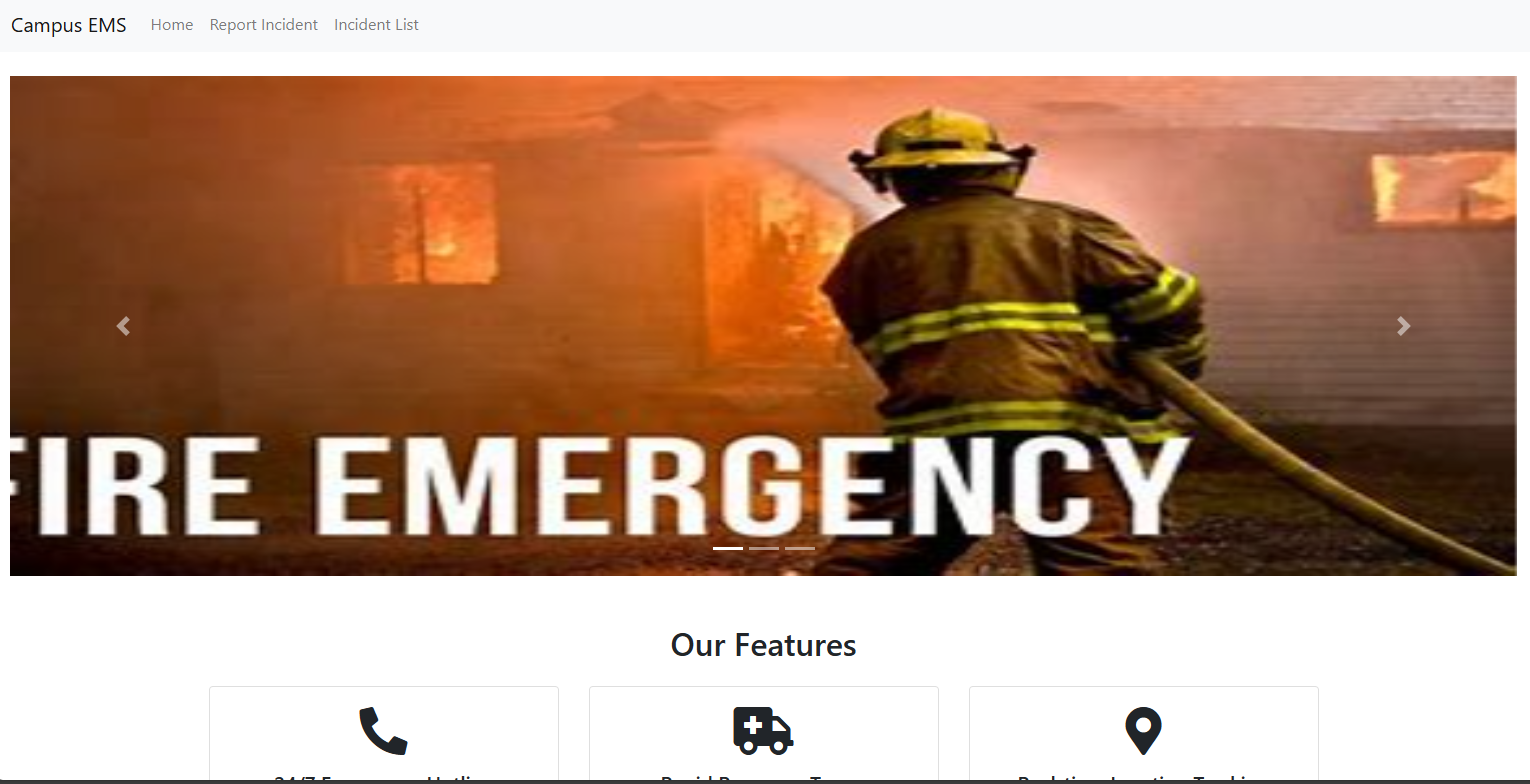


Fig 3.1 Campus Emergency Response System

Source: From the Application design

**3.4 SYSTEM DESIGN**

The proposed digital campus emergency and response system consists of a comprehensive, scalable, and secure architecture designed to facilitate efficient data collection, analysis, and reporting. The system comprises the following key components:

**Front-End (User Interface):**

**Web and Mobile Interfaces:** Provides user-friendly and responsive interfaces for survey participation and viewing results.

**Back-End (Server-Side):** Application Server: Manages business logic, user authentication, and survey distribution using Django Rest Framework.

**Database Server:** Stores survey data and user information securely using SQLite.

**Security Layer:** Authentication and Authorization: Ensures secure access through role-based controls and encryption.

**Key Modules**

1. Surveys Management: Instruments for designing, modifying, and sending surveys.
2. Data Collection: Real-time, automated data collection with privacy safeguards.
3. Data Analysis and Reporting: Tools for data interpretation and visualization, including analytics engines.
4. Efficiency Optimization: To improve responsiveness, cache and query optimization are used.
5. Maintenance and Support
6. User Support: Documentation and a help desk for users.

Overview User Interface

(Web Responsive Design Interface)

Application Server

(Python Django Server)

User Interface

(Web Responsive Design Interface)

Scalability, Performance & Maintenance

Key Modules

- Survey Management - Data Collection - Data Analysis & - User Management & Reporting

Security Layer Database Server

(Encryption, Access (Survey Data, User Info) Control, Compliance)

Fig 3.2 Showing System Design Use Case Diagram of the application

This diagram illustrates the architecture's layers and key components, showing the flow from the user interface through the back-end processing and data management, with a focus on security, integration, and scalability.

**3.5. DATABASE STRUCTURE**

Data Structure for a Digital Campus Emergency and Response System.

The data structure for the digital campus Emergency and response system is designed to efficiently store and manage various types of information, including user data, survey details, responses, and analysis results. The structure is typically organized into several key entities, each with its attributes, ensuring data integrity and accessibility.

User

(Report Cases)

Location 1

DB

Location 1

Location 1

Feedback

Update

Fig 3.3 Database Structure for Campus Emergency Response System

**3.5.1.** **INPUT STRUCTURE**

The input structure defines how data is formatted and received by the system, particularly during survey creation, response submission, and user management. This structure ensures data consistency, validation, and ease of processing. Here's an overview of the key input structures:

**User Entity**

Attributes:

UserID: Unique identifier for each user (Primary Key).

Username: User's login name.

Password: Hashed password for secure authentication.

Email: User's email address.

**2. Emergency Entity**

Attributes:

id: Unique identifier for each survey (Primary Key).

Incidence type: type of the incidence.

reported: posted by.

description: Brief description of the incidence.

location: location of the populace.

responses: Foreign key relationship with response table

Date created: Date incidence was created

**3. Incidence Type**

Attributes:

id: Unique identifier for each response (Primary Key).

name: name of the incidence

Relationships Between Entities

incidence type – incidence: This is a one-to-one relationship that connects the incidence type to incidence specific

Emergency Contact – incidence: This is a one-to-one relationship that connects the Emergency Contact to incidence.

**3.5.2 OUTPUT STRUCTURE**

**1. Response Entity**

Attributes:

id: Unique identifier for each response (Primary Key).

Incidence type: Identifier of the related survey (Foreign Key).

description: Brief description of the purpose.

date\_created: Date and time when the response was submitted.

Response status: Actual responses, stored in a format such as text format, including answers to each question.

**2. Admin User Entity**

Attributes:

1. id: The user id has a unique key called the primary key
2. name: Name of the admin user
3. email: email of the admin user (This is used in-terms of administering privileges and roles)

**CHAPTER FOUR**

**SYSTEM TESTING AND IMPLEMENTATION**

**4.0 INTRODUCTION**

System testing and implementation are crucial phases in the development lifecycle of the automated campus emergency and response system. In these stages, the system's suitability for users, planned functionality, and deployment readiness are checked. In this introduction, the objectives, protocols, and methods employed in these phases are summarized.

**4.1 CHOICE OF PROGRAMMING, LANGUAGES AND DATABASE**

Selection of Database and Programming Languages Choosing the right database systems and programming languages is essential to creating a reliable, scalable, and effective digital campus Emergency and response system. System needs, scalability, security, developer experience, and integration skills are some of the variables that influence the decision. The suggested database systems and programming languages for this project are summarized as follows:

**1. Programming Languages**

Front-End Development:

i. HTML5 and CSS3 (Boostrap):

* 1. Rationale: HTML5 and CSS3 are essential for structuring and styling the user interface. They provide the foundation for building accessible and responsive designs that work across various devices and screen sizes.
  2. Benefits: Wide browser support, flexibility in design, and ease of use.

ii. JavaScript:

1. Rationale: JavaScript is the standard language for web development, providing rich interactivity and responsive user interfaces integrated with bootstrap for.
2. Benefits: High performance, large community support, extensive libraries and tools, and easy integration with other technologies.

Back-End Development:

i. Python (with Django Framework):

* 1. Rationale: Python is known for its simplicity, readability, and versatility. Django, a high-level Python web framework, is well-suited for developing secure and maintainable web applications. It provides built-in functionalities for authentication, URL routing, ORM (Object-Relational Mapping), and more.
  2. Benefits: Rapid development, strong security features, comprehensive documentation, and a large ecosystem of packages and tools.

**2. Database Systems**

Relational Database:

1. SQLite:
   1. Rationale: SQL is a widely-used open-source relational database, known for its reliability, ease of use, and performance. It is suitable for applications with a large volume of read-heavy workloads.
   2. Benefits: Extensive documentation, wide community support, and integration with various platforms and tools.

The digital campus emergency and response system's database system and programming language selection is influenced by factors including security, ease of development, scalability, and the capacity to manage intricate data exchanges. Utilizing technologies such as CSS, HTML, JavaScript, Python, and MySQL, the system can offer a reliable, easy-to-use, and effective platform for gathering and evaluating ideas on campus.

**4.2 SYSTEM REQUIREMENTS**

To handle massive amounts of survey replies and real-time data analytics, the system needs a scalable design with secure authentication and data encryption. It should ensure a consistent user experience across devices by supporting multi-platform accessibility, which includes web and mobile interfaces.

**4.2.1 HARDWARE REQUIREMENT AND SPECIFICATIONS**

To enable scalability and performance, the system needs a cloud-based server infrastructure with at least 4 CPU cores, 8 GB of RAM, and 100 GB of SSD storage. A local PC with a multi-core processor, 256 GB SSD, and 16 GB RAM is advised for development and testing. Sustaining data availability and integrity requires backup systems and high-speed internet access.

**4.2.2 SOFTWARE REQUIREMENTS AND SPECIFICATIONS**

The system requires a Window or Linux-based server environment (e.g., Window 10,

Ubuntu) for hosting the application, with Python and Django for backend development. A

front-end stack comprising HTML5, CSS3, and JavaScript is recommended. The system also needs PostgreSQL, SQLite or MySQL for relational data storage. The system can run on any local server like Xampp or Wampp also integrated into the cloud using a flexible python cloud hosting called pythonanywhere.

**4.3 RESULT INTERFACE**

The result interface presents survey data through interactive dashboards, featuring cards, and tables for easy analysis. Admin can view all responses and activities within the application. The interface also includes visual tools like word clouds for qualitative data, enhancing the user experience in data interpretation.

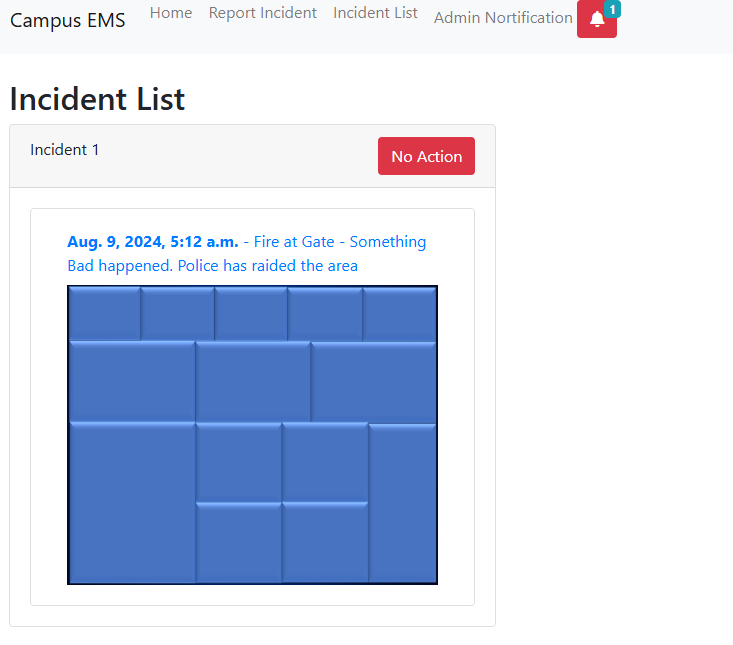


Fig 4.1 Incidence List View

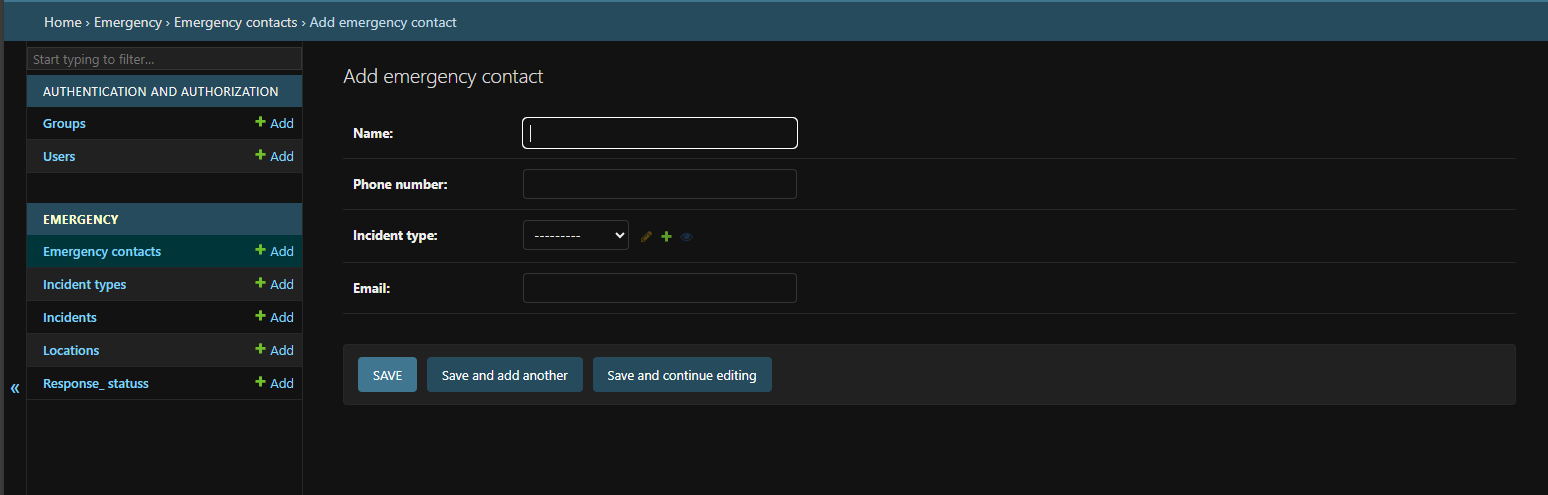


Fig 4.2 Admin Dashboard View

**4.4** **SYSTEM TESTING AND IMPLEMENTATION**

To guarantee functionality, security, and usability, the system is put through extensive testing, which includes user acceptability testing and integration testing. Phased rollout is the method of implementation, with a pilot group being included before complete distribution. Support and monitoring are given on an ongoing basis to resolve problems and improve performance after launch.

**4.5 DESCRIPTION/DISCUSSION OF FINDINGS**

The efficiency of data gathering and user interaction were greatly enhanced by the installation of the digital campus emergency and response system. Deeper insights into campus sentiment were made possible by the system's real-time analytics capabilities and user-friendly interface. Furthermore, the switch from manual to digital processes decreased administrative burden and errors, producing more trustworthy and useful data for decision-making.

**4.6 SYSTEM DOCUMENTATION**

System documentation includes detailed guides on installation, configuration, and usage, and troubleshooting tips. It provides comprehensive information on system architecture, data models, and user interfaces to assist both developers and end-users. Regular updates ensure that documentation remains accurate and reflective of system changes.

**4.6.1 HOW TO LOAD THE SOFTWARE**

To load the software, use the following steps:

1. Create a folder in a desktop environment
2. Copy the software from the initial location to current folder created
3. Download Python into your PC and add to environmental variable
4. Inside the folder called emergency, configure the settings by adding an os path

**4.6.2 HOW TO RUN THE SOFTWARE**

Apply the following steps to run the software:

**Locally**

1. Open your command prompt and CD into the folder location of the campus\_ems folder inside the parent folder
2. Located the campus\_ems sub-folder and run *pip install django* to install Django rest frame-work
3. Download and configure Xampp Server. Now, run *python manage.py makemigrations* to load the admin, make current data migrations
4. Now, run *python manage.py* migrate to migrate the data into the database
5. Configure your static file and templates.
6. After all configurations, run *python manage.py runserver* to run the application
7. Using the inbuilt SQLite for Database and Django Server for local hosting

**CHAPTER FIVE**

**SUMMARY, CONCLUSION AND RECOMMENDATION**

**5.1 SUMMARY**

A campus emergency and response system is designed to ensure the safety and security of students, faculty, and staff in a campus environment. It typically involves the integration of various technologies and procedures to efficiently detect, respond to, and manage emergencies such as natural disasters, medical emergencies, fires, or security threats. The system's goal is to develop a system for quick dissemination of emergency information within and around the campus environment.

**5.2 CONCLUSION**

Implementing a comprehensive campus emergency and response system is crucial for maintaining a safe and secure environment within educational institutions. By utilizing advanced technologies and well-coordinated procedures, campuses can effectively manage emergencies, minimize harm, and ensure the well-being of everyone on campus. The success of such a system depends on proper planning, regular training, and continuous improvement based on feedback and incident analysis.

**5.3 RECOMMENDATION**

To enhance campus safety, it's recommended to implement a multi-channel alert system that includes SMS, email, apps, and public address systems for quick notifications. Regular drills and training should be conducted to ensure everyone on campus is familiar with emergency procedures. Advanced monitoring technologies like surveillance cameras and sensors should be installed for real-time threat detection. Clear communication protocols need to be established for effective reporting and coordination during emergencies. It's also important to develop and regularly update comprehensive evacuation plans for various scenarios. Finally, the system should be continuously evaluated and improved based on feedback and incident analysis.