**DEVELOPMENT AND IMPLEMENTATION OF MACHINE LEARNING – BASED SECURITY ALERT AND EVACUATION SYSTEM**

**A blue and white logo

Description automatically generated**

By

Azamat Shirinshoev

In partial fulfilment of the requirement for the degree

Bachelor of Science in Computer Science

Department of Computer Science, School of Arts & Science

University of Central Asia, Naryn campus, Kyrgyz Republic

(2023)

# ABSTRACT

This thesis presents the design and implementation of the Campus Emergency Alert System (CEAS), a comprehensive safety solution tailored for the University of Central Asia. The CEAS is engineered to enhance campus safety by integrating advanced technological features that facilitate real-time communication, emergency management, and user localization. The system architecture employs a robust, scalable framework that supports high availability and reliability to handle emergency scenarios effectively.

Functional requirements of the CEAS include prompt communication, real-time emergency notifications, user status updates, and facial recognition, among others. Non-functional requirements focus on system performance metrics such as uptime, response time, and compliance with international security standards. The system's dual-interface design separates user functionalities—such as receiving alerts and sending SOS notifications—from administrative capabilities, which include monitoring user statuses and managing system-wide alerts through an advanced dashboard which was created using Python framework, Django.

Key technologies utilized in the CEAS include Django REST Framework, GPS tracking for precise location services, and machine learning algorithms for facial recognition, cross-platform app and data analytics. This integration of technologies ensures that the system not only meets the immediate needs of safety and communication during emergencies but also adheres to the highest standards of data integrity and user accessibility.

The implementation of CEAS aims to significantly improve the response times to campus emergencies, enhance the coordination of security efforts, and increase the overall safety of students, faculty, and staff. The thesis evaluates the system’s performance and user feedback through a series of tests and surveys, confirming that CEAS meets the outlined requirements and effectively improves the campus safety infrastructure.

# Remove this empty space.

# ACKNOWLEDGEMENTS

My fulfilling journey throughout my bachelor's degree at the University of Central Asia was rarely a solitary endeavor, and the culmination of this experience, my thesis, was no exception.

I would like to express my biggest gratitude to my supervisors Dr. Ayman Aljarbouh, who was involved in every step of this project, and Dr. Muhammad Fayaz, whose expertise and insights guided me throughout this project.

My heartfelt thanks go to my family, for their unwavering support and encouragement. Their belief in me has been a constant source of strength throughout my studies, especially during the most challenging phases of my life.

I extend my sincere thanks to my friends, for their invaluable contributions. Their insightful discussions and constructive feedback enhanced the development of the ideas within this thesis.

**Table of Contents**

[ABSTRACT 2](#_Toc165050712)

[ACKNOWLEDGEMENTS 3](#_Toc165050713)

[Table of Tables 6](#_Toc165050714)

[Tables of Figures 7](#_Toc165050715)

[CHAPTER 1: INTRODUCTION 8](#_Toc165050716)

[1.1. Background 8](#_Toc165050717)

[1.2. Problem Statement 8](#_Toc165050718)

[1.3. Project's Scope 9](#_Toc165050719)

[1.4. Deployment of System 10](#_Toc165050720)

[1.5. Objectives 11](#_Toc165050721)

[CHAPTER 2. LITERATE REVIEW 13](#_Toc165050722)

[2.1. Overview of Campus Emergency Alert Systems 13](#_Toc165050723)

[2.2. Technology Integration: Face Recognition and Mobile App Development 13](#_Toc165050724)

[2.3. Security and Privacy Considerations 14](#_Toc165050725)

[2.4. Similar application comparison 14](#_Toc165050726)

[CHAPTER 3. SYSTEM REQUIREMENTS 17](#_Toc165050727)

[3.1. Functional Requirements: 17](#_Toc165050728)

[3.2. Non-Functional Requirements 19](#_Toc165050729)

[3.3 Target Users and their Needs 21](#_Toc165050730)

[CHAPTER 4. METHODOLOGY AND IMPLEMENTATION 22](#_Toc165050731)

[4.1 Development of API 24](#_Toc165050732)

[4.1.1 Designing the Database 27](#_Toc165050733)

[4.1.2 Implementing the API 28](#_Toc165050734)

[4.1.3 WebSocket Usage for Chat Service and Notifications 31](#_Toc165050735)

[4.1.4 Middleware for Face Recognition Model 32](#_Toc165050736)

[4.1.5 Integration with Mobile Application 34](#_Toc165050737)

[CHAPTER 5. WORKING MECHANISM 36](#_Toc165050738)

[5.1 System Workflow Overview 36](#_Toc165050739)

[5.2 Mobile Application Functionality 36](#_Toc165050740)

[5.3 Admin Dashboard Functionality 37](#_Toc165050741)

[CHAPTER 7. CONCLUSION 39](#_Toc165050742)

[REFERENCES: 41](#_Toc165050743)

**List of Tables**

Table 2.1 …………………………………………………………………………………… 15

Table 3.1 …………………………………………………………………………………… 17

Table 3.2 …………………………………………………………………………………… 19

Table 4.1 …………………………………………………………………………………… 22

Table 4.2 …………………………………………………………………………………… 24

List of Figures

Figure 4.1 …………………………………………………………………………………… 26

Figure 4.2 …………………………………………………………………………………… 28

Figure 4.3 …………………………………………………………………………………… 30

Figure 5.1 …………………………………………………………………………………… 36

Figure 5.2 …………………………………………………………………………………… 37

Figure 5.3 …………………………………………………………………………………… 37

Figure 5.4 …………………………………………………………………………………… 38

**STATEMENT OF AUTHORSHIP**

I certify that this work has not been accepted in substance for any degree and is not concurrently

being submitted for any degree other than that of Bachelor of Science in Computer Science being

studied at the Department of Computer Science, School of Arts & Science, University of Central

Asia, Kyrgyz Republic. I also declare that this work is the result of my own findings and

investigations except where otherwise identified by references and that I have not plagiarized

another’s work.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Azamat Shirinshoev

**CHAPTER 1: INTRODUCTION**

## Background

University of Central Asia (UCA) is educational institution with campuses in Tajikistan and Kyrgyzstan. In addition to these campuses, UCA operates SPCE (School of Professional and Continuous Education) that also focuses on the education sector which extends its educational reach into other countries as well. As the number of students, instructors and staff continues to grow, ensuring the safety and well-being of all campus members are critical. The Campus Emergency Alert System project intends to meet this urgent need by creating a comprehensive and technologically sophisticated solution for improving campus safety during emergency scenarios.

## Problem Statement

The current scenario: a fire breaks out unexpectedly at a university building. The existing system relies on sirens to inform users of the emergency and instruct immediate evacuation. While effective in broadcasting alerts, this method lacks the capability to provide real-time information about the safety status of individuals on campus.

The Proposed Enhancement: The Campus Emergency Alert System would enable administrators to issue a real-time emergency alert, alerting all users (students, instructors, and staff) via their mobile devices [2]. Users get push notifications notifying them of the severity of the issue and providing instructions for immediate action [1]. The system uses cutting-edge surveillance technology, such as cameras with face recognition capabilities, to automatically monitor and update the status of people. This degree of automation guarantees that persons at risk are recognized quickly, allowing administrators to concentrate on emergency response coordination.

Moreover, the system extends its functionality beyond the campus borders. If a user finds themselves in a dangerous situation off-campus, they can activate an "in danger" signal through the app, which shares their real-time location with campus security. This feature not only keeps the individual connected with campus safety measures but also allows security personnel to manage off-campus emergencies effectively.

## Project's Scope

With the goal of improving campus safety, the Campus Emergency Alert System (CEAS) offers a complete, real-time emergency management solution via the use of contemporary software structures and technologies. The specific elements and technology that define the project's scope are listed below:

#### 1.3.1 Technology Stack

* **Mobile App Development:**
  + **React Native Expo:** Utilized for developing mobile applications, Expo is an open-source platform that allows the creation of native apps for multiple operating systems using a single codebase in JavaScript and React. This promotes faster deployment and easier maintenance.
  + **Expo Notifications:** This library integrates with the mobile application to handle full push notification services, ensuring timely communication with users.
* **Server-Side Technologies:**
  + **Django Channels:** Employs Django Channels to manage real-time Web Sockets, which are essential for real time data transfer and maintaining persistent connections between the server and client applications.
  + **REST API:** A RESTful API will be implemented to handle Create, Read, Update, and Delete (CRUD) operations, allowing efficient management of user data and system interactions. This API will serve as the backbone for data exchanges between the mobile app, admin dashboard, and the server.
  + **Redis & Celery:** Redis is used as a message broker to queue messages before they are handled by Celery, which processes asynchronous tasks and manages the task queue. This combination enhances the efficiency and reliability of WebSocket communications, ensuring messages are processed quickly and in an orderly fashion.
* **Integration with Notification Services:**
  + **Firebase Cloud Messaging (FCM) and Apple Push Notification Service (APNs):** These services ensure that notifications are delivered consistently across various device ecosystems, enhancing the reliability of the system.

#### 1.3.2 System Functionalities

* **Real-Time Emergency Alerts:**
  + During emergencies, administrators can send alerts that reach all users instantly, providing crucial information and instructions.
* **User Safety Features:**
  + **Face Recognition Middleware:** Incorporates facial recognition technology to automatically update user statuses, aiding rapid identification and assistance.
  + **Location-Based Services:** Enables real-time tracking and updates, crucial for accurate location monitoring during emergencies.
  + **SOS Alert and Communication:** Users who find themselves in danger, especially when off-campus, can send an SOS alert through the app. This alert activates real-time location tracking for the admin, allowing them to not only view the user's current location but also initiate direct communication through an integrated chat feature. This ensures continuous protection and enables security personnel to manage emergencies effectively, both on and off-campus.

## Deployment of System

Deployment is a crucial step in project development, ensuring the web application, face recognition middleware, and mobile app are accessible to users. It involves the installation, configuration, updates, and activation of the applications that bring your software system to life [13].

**Web Application and Middleware Deployment:**

* **Deployment Platform:** The web application and face recognition middleware are deployed on Render.com, a cloud service provider that offers automated scaling, high availability, and streamlined setup processes. This platform supports both our database and Redis instance, ensuring efficient data handling and caching capabilities.
* **Web Server Configuration:** We utilize Gunicorn with Uvicorn workers for serving the web application. Gunicorn acts as a WSGI HTTP server, while Uvicorn provides support for ASGI, allowing asynchronous processing which is essential for handling WebSocket connections used by Django Channels in our system.
* **Command for Running the Web Application on Render.com:**
* python -m gunicorn campus\_alert\_system.asgi:application -k uvicorn.workers.UvicornWorker
  + This command starts the web application using Gunicorn with Uvicorn workers, optimizing it for concurrent connections and real-time data transmission which is vital for our real-time alerting and user status tracking functionalities.

**Mobile Application Deployment:**

* **Distribution:** React Native Expo compiles the mobile application into native code for both Android and iOS platforms, ensuring a wide reach across various devices. Users may readily get the built application as it is released via the Google Play Store and will be released to the Apple App Store.

## Objectives

In every university the security of the faculty, students, and staff is vital. The project aims to improve campus security by developing a comprehensive Campus Emergency Alert System with several features that can preserve and accelerate emergency response.

Enhance Campus Safety: Creating a system that will increase campus safety by allowing quick communication and cooperation during emergencies.

Real-time Alerting: Sending real-time notification to all users through app.

User Status Tracking: Providing a function for the users that can update their status in case of an emergency or if they are in emergency so the security team can react immediately.

Integration with Face Recognition Model: Implementing the face recognition technology to automatically update the user’s status when the users move to safe area.

Admin Control: Giving admin access to trigger emergencies, monitor user statuses, and take safety measures.

User-Admin Communication: Enabling users to chat with admin for assistance.

Location Tracking: Enabling location tracking functionality on admin interface when the user will send “SOS” alert, which encourages fast decision-making and response.

Multi-language Support: Providing support in English and Russian.

Documentation and Training: Creating detailed user and admin documentation and training them to use the system.

Security and Privacy: Protecting user data and privacy using strong security measures that comply with data protection laws.

Usability: Creating user-friendly interface for users and admin.

Feedback and Improvement: Collecting feedback from the users and constantly improving the system.

# **CHAPTER 2. LITERATE REVIEW**

## 2.1. Overview of Campus Emergency Alert Systems

Institutions worldwide put a high importance on campus safety, and as technology advances, emergency warning and response techniques have experienced tremendous modification [5]. Modern emergency alert systems have emerged as vital instruments for enhancing campus security and preparedness in response to this expanding concern [1]. This literature review focuses on the current, technology-driven tactics used by campus emergency warning systems, with a particular emphasis on user interactions, real-time updates, and the incorporation of novel technologies for monitoring safety in real-time. In line with these advancements, the integration of intelligent systems in campuses significantly contributes to the improvement of students' quality of life by efficiently using information technologies in emergency scenarios [12].

For example, universities and colleges have huge and varied student, instructor, and staff populations. Natural disasters, fires, medical crises, and even security threats are all feasible under these conditions [6]. In these kinds of emergencies, prompt communication, coordination, and reactivity are necessary in order to guarantee the health and safety of everyone on campus. Historically, emergency alert systems relied on loudspeakers, sirens, and text-based notifications [4]. These methods, while effective for their time, were limited in reach and personalization. Often, these systems lacked real-time information, which could lead to misinformation during critical events [1]. Furthermore, the efficiency of these systems is dependent on users' quick compliance with notifications, which is impacted by variables such as subjective norm and information quality trust [5].

## 2.2. Technology Integration: Face Recognition and Mobile App Development

Emergency alert systems have evolved over time into complex platforms capable of giving users real-time messages and critical information during crises [6]. These systems use several technologies, including mobile apps, push alerts, and surveillance tools, to give a complete and timely reaction to emerging crises [2, 4]. Push alerts have shown to be more successful in immediately informing a big audience [8]. They enable administrators to create notifications and users to change their statuses in real time. When combined with video surveillance and facial recognition, real-time systems may provide more tailored warnings depending on the user's proximity to the danger zone, as well as automatically updating the status of persons [5, 7]. The switch to these systems was greatly impacted by the incidents such as Virginia Tech shooting, which made it clear how important it is to communicate quickly and clearly [8].

## 2.3. Security and Privacy Considerations

The focus on user engagement is a major change in modern systems. Allowing users to update their safety status has two benefits: it gives people a feeling of control during crises and provides administrators with a more accurate view of the situation on the ground [1, 2]. The incorporation of chat services bridges the communication gap between administrators and users even further, allowing for more focused help and real-time feedback [1]. GPS and Wi-Fi triangulation, for example, play critical roles in current emergency warning systems [9]. Administrators can better focus resources and advise users away from risk zones by knowing their real-time position [10]. While safety is important, people have also raised worries about privacy [5]. This requires a balance between them. In this context, intelligent campuses, which leverage information technologies to create a safe and efficient environment, play a critical role in improving the operation and efficacy of these systems [5].

It's becoming more and more important for technology to help protect students and teachers as schools deal with more and more hazards [11]. Campus emergency systems of the future will have real-time tracking, state reports that users can control, chat features, and services that are based on where the user is [3, 6]. With technological advancements, it is inevitable that new challenges and issues arise in tandem. The necessity for continuous examination and evaluation of these systems arises from the imperative to ensure their sustained functionality, user-friendliness, and adherence to privacy considerations. The growth of these systems, especially in reaction to major incidents like the Virginia Tech shooting, emphasizes the need of speedy and efficient system in emergency circumstances [8].

## 2.4. Similar application comparison

There are various applications designed to enhance the safety within educational institutions. To understand the advantage of the proposed Campus Emergency Alert System the comparison test has been conducted. Below is the table of similar applications, focusing on key features, usability, and technological integration.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Applications**  **Features** | **CEAS** | **Campus Safety Monitoring** | **Rave Guardian App** | **SchoolPass App** | **Campus Alert System** | **KMSAFE APP** | **Total** |
| **Emergency Notifications** | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ | 6 |
| **User Status Update** | ✔️ | ❌ | ❌ | ✔️ | ❌ | ✔️ | 3 |
| **Facial Recognition Integration** | ✔️ | ✔️ | ❌ | ❌ | ❌ | ❌ | 2 |
| **Admin-User Chat Feature** | ✔️ | ❌ | ✔️ | ❌ | ❌ | ❌ | 2 |
| **User Location Display** | ✔️ | ❌ | ✔️ | ❌ | ❌ | ❌ | 2 |
| **Individual Emergency Alerts** | ✔️ | ❌ | ✔️ | ✔️ | ❌ | ❌ | 3 |
| **Two-Tier Hierarchy (Admin & User)** | ✔️ | ❌ | ✔️ | ✔️ | ❌ | ✔️ | 4 |
| **Integration with Monitoring System** | ✔️ | ✔️ | ❌ | ✔️ | ❌ | ❌ | 3 |

Table 2.1. Analysis of similar applications

The description of the features and the links to the similar applications are added here:

Links to the apps:

1. Campus Safety Monitoring: <https://doi.org/10.1109/DCABES52998.2021.00036/>;
2. Rave Guardian App: <https://play.google.com/store/apps/details?id=com.ravemobilesafety.raveguardian&hl=en_US/>;
3. SchoolPass App: <https://schoolpass.com/solutions/emergency-attendance/>;
4. Campus Alert System: <https://play.google.com/store/apps/details?id=cos.campusalertsystem/>;
5. KMSAFE APP: <https://doi.org/10.1109/BioSMART54244.2021.9677778/>;
6. Campus Emergency Alert System (CEAS): <https://campusalertsystem.onrender.com/>

Feature Description:

1. Emergency Notification: Is pressed when there is an emergency, which sends notification to all the users to request to update statuses.
2. User Status Update: Users can update their status using the app. Administrators need this capability for real-time safety checks to determine who is safe, in danger, and unresponsive.
3. Facial Recognition Integration: Automates user status updates via face recognition. This can validate identities without human input during a crisis, improving security and reaction.
4. Admin-User Chat Feature. Enables users and administrators to communicate directly. This helps to provide real-time information and personalized instruction during an emergency.
5. User Location Display: The system can track the real time user location when they are in danger.
6. Individual Emergency Alert: Enables users to send a personal SOS alert when they are in danger. Alert notifies administrators of the user's situation and exact location, prompting an urgent response.
7. Two-Tier Hierarchy (Admin & User): This feature differentiates between two tiers of system access: user and administrator.
8. Integration with Monitoring System: This feature easily integrates the Campus Emergency Alert System with an advanced monitoring system available via the admin dashboard. This connection provides administrators with real-time data on user statuses, allowing for fast scenario evaluation and decision-making.

# **CHAPTER 3. PROPOSED METHODOLOGY**

Our project is mainly comprised of six modules namely

A diagram with colorful text

Description automatically generated with medium confidence

## 3.1. Functional Requirements:

|  |  |
| --- | --- |
| Functional requirement No. | Functional Requirement |
| FR 1 | Multi-language support (English, Russian) |
| FR 2 | Authentication, Authorization, Registration of the user, reset password |
| FR 3 | Real-time Emergency Notifications |
| FR 4 | Access to Emergency Resources |
| FR 5 | Location Tracking |
| FR 6 | Chat Service between users and admin |
| FR 7 | SOS alert notification from the mobile app |
| FR 8 | Face Recognition Model to automate user status update |
| FR 9 | User status update from the mobile app |
| FR 10 | Display graphs, map and table with statistics in admin dashboard |

Table 3.1. Functional Requirements

**FR1: Multi-language support (English, Russian)**

The system offers multi-language support to accommodate the diverse linguistic backgrounds of the campus community. Users are able to easily switch between English and Russian across all user interfaces, notifications, and resources, ensuring accessibility and comprehensibility for all users.

**FR2: Authentication, Authorization, Registration of the user, reset password.**

The system provides a robust security framework that allows users to register, authenticate, and manage their accounts securely. Features includes registration with email and password, secure login processes, multi-factor authentication options, and the ability to reset forgotten passwords, ensuring user accounts are protected and private.

**FR3: Real-time Emergency Notifications**

In the event of an emergency, the system can send real-time notifications to all registered users swiftly. These notifications include critical details about the emergency, such as the nature of the event, and recommended actions, ensuring that all users receive timely and accurate information to respond appropriately.

**FR4: Access to Emergency Resources**

The system provides seamless access to vital emergency resources. This includes digital maps showing safe routes and hazard zones, contact information for emergency services, and step-by-step guidelines for different types of emergencies. These resources should be easily accessible within the app, helping users navigate crises effectively and safely.

**FR5: Location Tracking**

The system is capable of real-time location tracking of users within the campus boundaries. This feature allows administrators to view the real time location of the users on a map interface, when the user will send SOS alert to admin, facilitating quicker response actions during emergencies.

**FR6: Chat Service between Users and Admin**

A built-in chat service enables direct and secure communication between users and administrators. This service supports instant messaging to help manage emergency situations and provide support and guidance when needed.

**FR7: SOS Alert Notification from the Mobile App**

Users can send an SOS alert through the mobile app with just a few taps. This alert immediately notifies the system administrators and triggers the emergency protocol to provide swift assistance to the user in distress.

**FR8: Face Recognition Model to Automate User Status Update**

The system includes a facial recognition feature to automate the process of updating user statuses during an emergency. This technology will help in quickly identifying and accounting for users without requiring manual check-ins, enhancing the speed and accuracy of response efforts.

**FR9: User Status Update from the Mobile App**

Users can manually update their status via the mobile app. This feature allows individuals to report their safety or need for help, contributing to more efficient emergency management and resource allocation.

**FR10: Display Graphs, Maps, and Table with Statistics in Admin Dashboard**

The admin dashboard is equipped with tools to display various forms of data visualization such as graphs, maps, and tables. These visual tools provide statistics on user activities, emergency response efficiency, and other relevant data, aiding administrators in making informed decisions and assessing the effectiveness of the system.

## 3.2. Non-Functional Requirements

|  |  |
| --- | --- |
| Non-functional requirement No. | Non-Functional Requirement Description |
| NFR 1 | System Availability |
| NFR 2 | Response Time |
| NFR 3 | Scalability |
| NFR 4 | **Security and Compliance** |
| NFR 5 | **Usability** |
| NFR 6 | **Reliability** |
| NFR 7 | **Data Integrity** |
| NFR 8 | **Accessibility** |
| NFR 9 | **Interoperability** |
| NFR 10 | **Maintainability** |

Table 3.2 Non-Functional Requirements

Acceptance criteria for non-functional requirements

**NFR 1: System Availability**

The system is available 24/7, ensuring that it is always operational and accessible, especially during emergencies.

**NFR 2: Response Time**

The system responds to user interactions within 2 seconds under normal conditions and sends out emergency notifications to all users within 5 seconds of the alert being issued by an administrator.

**NFR 3: Scalability**

It is crucial that the system can handle a growing number of users and data requests at once, particularly during growth scenarios and peak emergency periods, without sacrificing speed.

**NFR 4: Security and Compliance**

The system respects data protection rules. It encrypts user data, stores it securely, and conducts regular security audits to ensure user data is protected.

**NFR 5: Usability**

The user interface (UI) is intuitive and user-friendly, allowing users of all technological proficiencies to navigate easily. The system requires minimal training for both users and administrators.

**NFR6: Reliability**

In addition to a recovery time of minutes in the event of a typical failure, the system's failure rate should be below 1%. Fortunately, its backups data frequently and has a solid disaster recovery strategy.

**NFR7: Data Integrity**

The system ensures that all transactions and data exchanges retain a high level of data correctness and consistency. All data input or retrieval activities guarantee the correct and current reflection of data.

**NFR8: Accessibility**

The system is accessible to all users, students, faculty, and staff.

**NFR9: Interoperability**

The system can interface and exchange data seamlessly with other software systems used for the campus, such as face recognition model, without any data loss or corruption.

**NFR10: Maintainability**

Adding new features or updating current ones does not cause the system to go down for long periods of time, and the system is simple to upgrade and manage.

## 3.3 Target Users and their Needs

The primary target users of the Campus Emergency Alert System are students, faculty, staff, and campus security personnel at the University of Central Asia. The system was created to satisfy the critical needs of these users. Students, faculty, and staff require a reliable and intuitive tool that can promptly inform them of any emergencies, help them navigate through crisis situations safely, and enable them to communicate their status quickly and effectively. They need access to real-time information and resources that can guide them during such events. Meanwhile, campus security personnel and administrators need sophisticated tools for monitoring the situation, coordinating response efforts, and maintaining communication with the entire campus community. The system provides these administrators with detailed real-time data about the location and status of individuals on campus to manage emergency situations effectively. By addressing these needs, the Campus Emergency Alert System ensures a comprehensive approach to campus safety, enhancing preparedness and response capabilities across the university. Furthermore, this system is dynamic and can be used in any other organization to enhance prompt communication and reactiveness during the critical situation, to guarantee the health and safety of users.

# CHAPTER 4. IMPLEMENTATION

I would like to highlight a section that covers the full spectrum of strategies and techniques involved in creating, implementing, and evaluating our Campus Emergency Alert System (CEAS). This part really dives into how we managed the project, our development methods, testing procedures, and the architectural design that all work together to make sure the system is strong, safe, and works smoothly.

The project implemented a hybrid Agile-Waterfall methodology, which combined the methodical phase-based structure of Waterfall with the flexible and iterative advancements inherent in Agile. With the selection of this hybrid model, we maintain the flexibility to address evolving requirements and feedback during the development process, in addition to the advantages of well-defined project timelines and deliverable phases. The subsequent Gantt chart illustrates the project milestones and their respective dates, serving as a visual instrument to monitor advancements and efficiently oversee task dependencies:

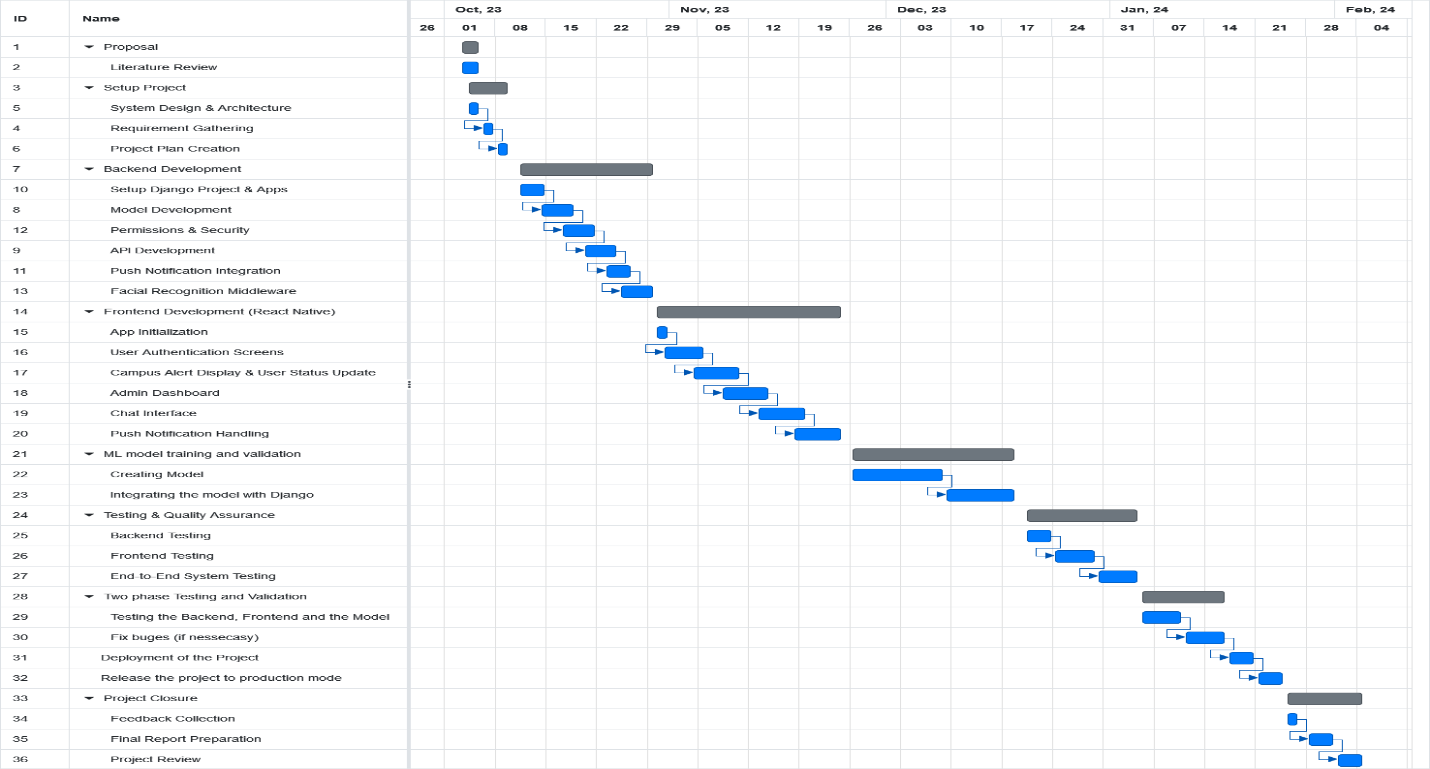


Table 4.1 Giant Chart

Looking at this Giant Chart we can see that there are eleven milestones with sub-section. This table gives the detailed description of the milestones:

|  |  |  |  |
| --- | --- | --- | --- |
| *Milestones* | *Work to be done* | *Outcome* | *Deadline*  dd/mm/yyyy |
| *Milestone 1* | Proposal | Completed Proposal | 20/10/2023 |
| *Milestone 2* | Setup Project | Completed Environment | 20/10/2023 |
| *Milestone 3* | Backend Development | Ready Backend Part | 30/10/2023 |
| *Milestone 4* | Frontend Development | The interface for the app and the admin part will be created | 30/11/2023 |
| *Milestone 5* | ML model training and validation | The model will be trained | 30/12/2023 |
| *Milestone 6* | Integration of Frontend, Backend, and ML model | The frontend and the backend will be connected | 30/01/2023 |
| *Milestone 7* | Connect with the existing security system | The security system will be updated with an additional emergency alert system. | 30/02/2023 |
| *Milestone 8* | Deployment of the Project | Deployed | 30/03/2023 |
| *Milestone 9* | Two phase Testing and Validation | All bugs will be found and fixed | 30/03/2023 |
| *Milestone 10* | Release the project to production mode | The product will be available for production | 30/04/2023 |
| *Milestone 11* | Final Presentation and Report | Completed Report | 30/04/2023 |

Table 4.2 Milestone

During development, milestones and path of development were amended from the original plan. However, the project was finished on schedule and all features were incorporated.

## 4.1 Development of API

A Django web framework-based Web API is the data interaction layer of the Campus Emergency Alert System. Django, a framework for Python, was chosen as the options are pragmatic designs, fast development capabilities, and solid community support. This is a perfect solution to create a clean, maintainable, and scalable Web API, which is critical for such real-time applications as CEAS. PostgreSQL was selected for database management systems because of its open-source and high-performing capabilities to manage relational data. The fact that it has a lot of features, such as the support for more complex queries, and that it allows us to work with concurrency without read locks, which is exactly what we need for our application, also makes it an ideal choice.

Django's MVT (Model-View-Template) approach simplifies application organization into three primary components. The "Model" represents the database schema and is implemented as Python classes. Each model defines the essential fields and behaviors of the data being stored and guides PostgreSQL in structuring the database tables. For instance, within our system, I've developed models for users, locations, chat, and alerts, each tailored to support our needs. Here's a sample snippet from the alert’s app model:

class EmergencyAlert(models.Model):

    STATUS\_CHOICES = [

        ('stable', 'Stable'),

        ('emergency', 'Emergency'),

    ]

    status = models.CharField(max\_length=50, choices=STATUS\_CHOICES, default='stable')

    triggered\_by = models.ForeignKey('users.UserProfile', on\_delete=models.CASCADE, related\_name='triggered\_alerts', null=True, blank=True)

    timestamp = models.DateTimeField(auto\_now\_add=True)

    notes = models.TextField(blank=True, null=True)

    def \_\_str\_\_(self):

        return f"{self.status} - {self.timestamp.strftime('%Y-%m-%d %H:%M:%S')}"

    def send\_emergency\_notifications(self):

        message\_title = "Emergency Alert"

        message\_body = "The campus status is now 'Emergency'. Please update your status."

        if self.notes:

            message\_body += f" Note: {self.notes}"

        # Getting all device tokens as a list

        device\_tokens = list(DeviceToken.objects.all().values\_list('token', flat=True))

        # Sends notifications through Expo

        for token in device\_tokens:

            self.send\_expo\_notification(token, message\_title, message\_body)

    def send\_expo\_notification(self, token, title, body):

        payload = {

            'to': token,

            'title': title,

            'body': body,

            'data': {

                'screen': 'UpdateStatus'

            },

            'priority': 'high',

            'sound': 'default',

            'channelId': 'default',

        }

        response = requests.post('https://exp.host/--/api/v2/push/send', json=payload)

        if response.status\_code != 200:

            print(f"Failed to send notification: {response.text}")

    @staticmethod

    def update\_user\_status(user\_id, status):

        with transaction.atomic():

            user\_profile = UserProfile.objects.select\_for\_update().get(pk=user\_id)

            user\_profile.status = status

            user\_profile.save()

    def trigger\_emergency(self):

        with transaction.atomic():

            # Updating only users who are not already in danger

            UserProfile.objects.filter(status='stable').update(status='in\_danger')

            self.status = 'emergency'

            self.save()

            self.send\_emergency\_notifications()

This EmergencyAlert model demonstrates an integrated approach within the Campus Emergency Alert System, leveraging Django's ORM capabilities, transaction management, and external service integration (Expo push notifications) to effectively manage and disseminate emergency information to users in real-time. The model encapsulates the essential behaviors and interactions required to handle emergency alerts within the CEAS ecosystem.

Furthermore, view processes requests and provides HTTP replies. Our system relies on views to manage WebSocket data flow and provide real-time emergency push alerts to users.

Template: Our Django-based system leverages a combination of Django templates and APIs to support our mobile app and admin dashboard interfaces effectively. This figure gives the perfect visualization of the MVT:

A diagram of a data flow

Description automatically generated

Figure 4.1 Control flow of MVT

With the API built upon Django Rest Framework (DRF), it offers a range of HTTP utility methods for mobile applications and admin dashboards, ensuring that RESTful principles are followed. A fundamental component of the DRF is serialization, which facilitates the translation of complicated data types into native Python data types, facilitating their easy rendering into a variety of content forms, including JSON, XML, and others. Here is an example of how DRF serializers were implemented:

from rest\_framework import serializers

from .models import UserProfile

from django.conf import settings

class UserProfileSerializer(serializers.ModelSerializer):

    avatar\_url = serializers.SerializerMethodField()

    class Meta:

        model = UserProfile

        fields = '\_\_all\_\_'

    def get\_avatar\_url(self, obj):

        request = self.context.get('request')

        if obj.avatar and hasattr(obj.avatar, 'url'):

            return request.build\_absolute\_uri(obj.avatar.url)

        else:

            return request.build\_absolute\_uri(settings.MEDIA\_URL + 'avatars/default.jpg')

Alongside HTTP-based communication, WebSocket connections are established using Django Channels to provide a persistent, full-duplex communication channel between the client and server. This allows for real-time updates, which are essential for timely user notifications and chat service when a user sends an SOS alert.

### 4.1.1 Designing the Database

The database is a vital component of the Campus Emergency Alert System (CEAS), structured to support complex operations and ensure data integrity. For development agility, SQLite is used due to its configuration simplicity, while PostgreSQL's robustness and advanced features make it the choice for the production environment. Utilizing Django models, the database is designed to support various important functions: managing users, sending emergency alerts, tracking locations, and enabling direct messaging between users and administrators with careful attention to relationships and normalization to maintain data integrity and performance.

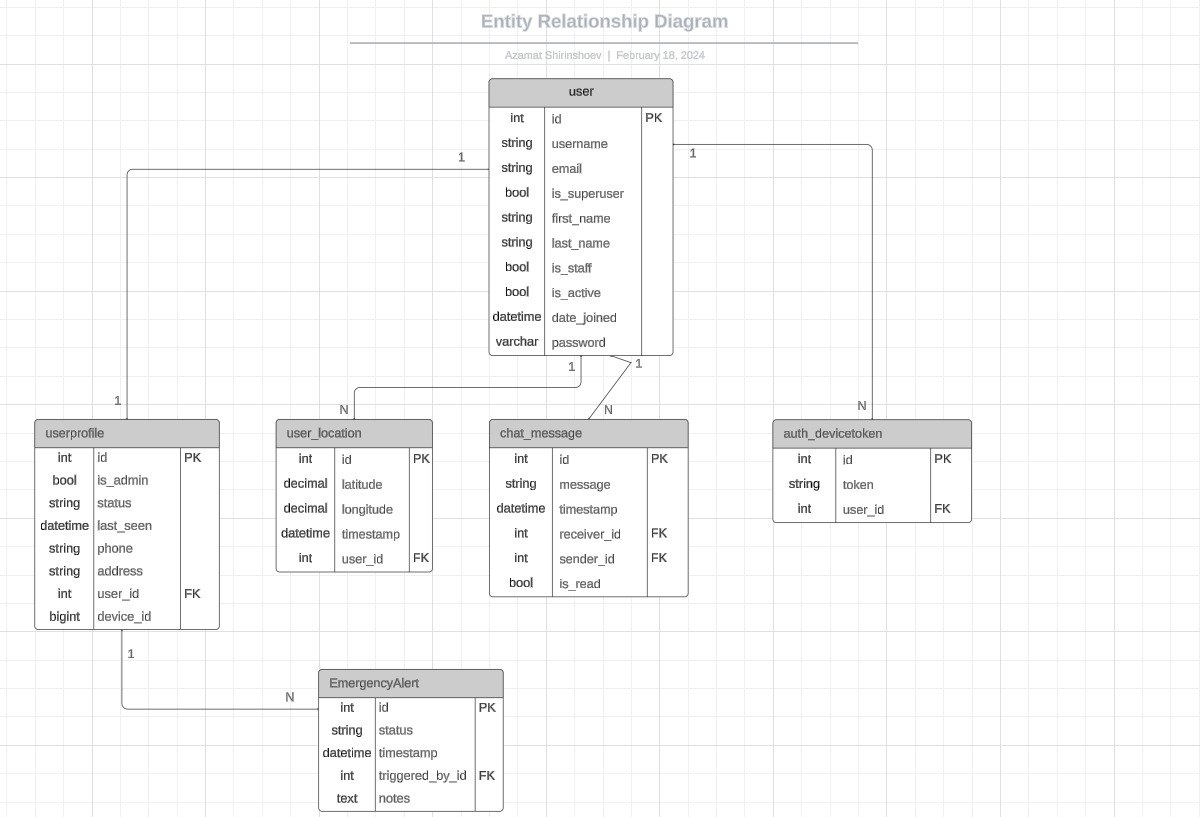


Figure 4.2 Entity Relationship Diagram

This ERD highlights the organization of data entities, their interconnections, and the role of normalization and integrity constraints in maintaining a robust database system for the application.

### 4.1.2 Implementing the API

In the Django implementation, the API heavily relies on serializers. They play a crucial role in converting a model instance to JSON format and vice versa. In section 4.1, we discussed how DRF was utilized for serializations. Serialization was utilized for user registration and emergency alerts, as the system heavily depends on the API. For instance:

Registration Serialization: This serialization enables users to securely transmit their private credentials by checking the uniqueness of email and autogenerates a unique username based on user’s email address.

class RegistrationSerializer(serializers.ModelSerializer):

    password2 = serializers.CharField(style={'input\_type': 'password'}, write\_only=True)

    phone = serializers.CharField(max\_length=15, write\_only=True)

    address = serializers.CharField(write\_only=True)

    class Meta:

        model = User

        fields = ['email', 'password', 'password2', 'phone', 'address']

        extra\_kwargs = {

            'password': {'write\_only': True}

        }

    def save(self):

        password = self.validated\_data['password']

        password2 = self.validated\_data['password2']

        if password != password2:

            raise serializers.ValidationError({'password': 'Passwords must match.'})

        if User.objects.filter(email=self.validated\_data['email']).exists():

            raise serializers.ValidationError({'email': 'A user with that email already exists.'})

        username = self.generate\_unique\_username(self.validated\_data['email'])

        user = User(

            email=self.validated\_data['email'],

            username=username

        )

        user.set\_password(password)

        user.save()

        UserProfile.objects.create(

            user=user,

            phone=self.validated\_data.get('phone', ''),

            address=self.validated\_data.get('address', '')

        )

        return user

   @staticmethod

    def generate\_unique\_username(email):

        base\_username = email.split('@')[0]

        username = base\_username

        num = 1

        while User.objects.filter(username=username).exists():

            username = f"{base\_username}{num}"

            num += 1

        return username

The API view of this serializer is responsible for managing user registration. It handles the POST request that contains the registration data. Upon receiving the POST request, the view creates an instance of the 'RegistrationSerializer' with the data from the request. Following successful registration, the user's email must be verified to activate the account. This two-step verification process adds an additional layer of security, ensuring that only legitimate users gain access. Unless the user's email is verified, their account will be deactivated, and they won't be able to log in. This API is seamlessly integrated with the React Native Expo-based mobile application, providing a smooth registration experience for users. The integration ensures that users can register, log in, and interact with the system through a secure and intuitive interface. Similarly, serializers for emergency alerts allow for the structured transmission of critical information from the server to the users. The system ensures real-time data flow, enabling users to receive instant notifications about emergencies and allowing administrators to get immediate status updates from users in distress, using additional expo-notification service. This figure shows how the notification is sent from backend to the devices.

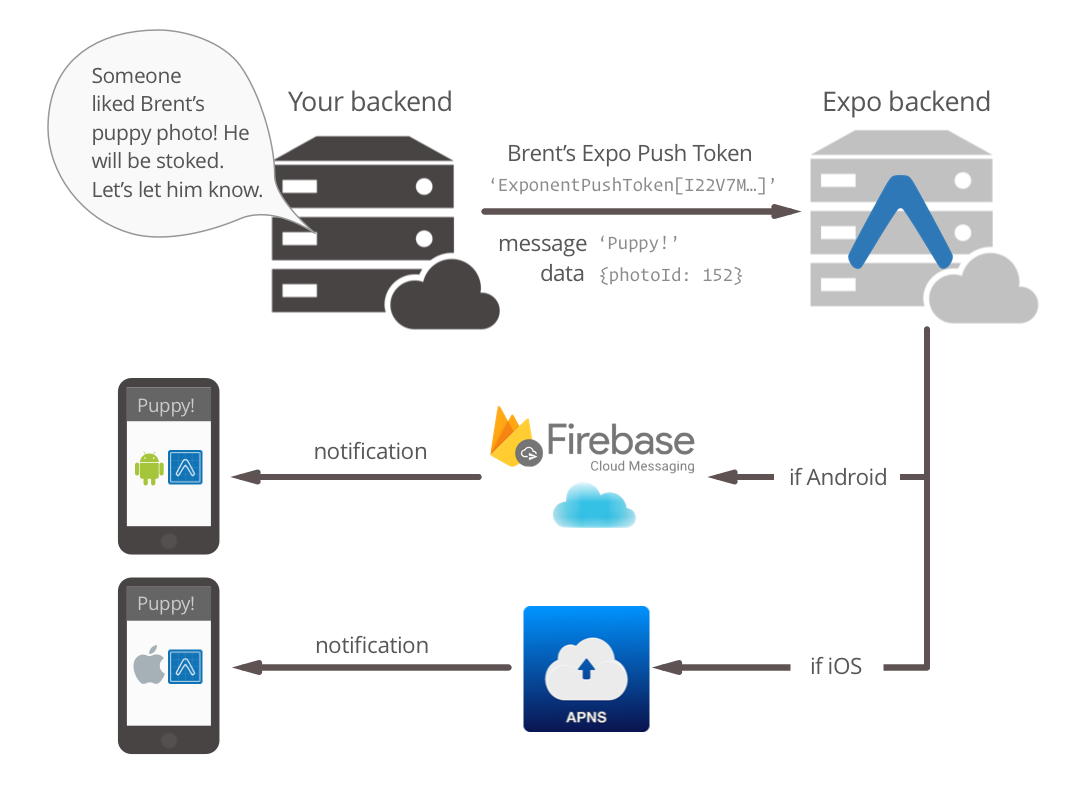


Figure 4.3 Sending notifications with Expo's Push API

Following the successful integration of real-time data transmission through the RESTful API, the need for even more immediate and interactive communication led to the incorporation of WebSockets in the CEAS. This transition is a natural progression from the use of Django REST Framework for asynchronous communications to leveraging Django Channels for real-time, bidirectional communication between the server and clients.

### 4.1.3 WebSocket Usage for Chat Service and Notifications

WebSockets provide a persistent connection between the user's device and the server, which is essential for features that require instant data exchanges, such as our chat service and immediate emergency notifications. This allows instant message delivery between the campus security and users. With this setup, when a user sends an SOS alert, the system not only notifies security personnel but also allows for immediate two-way communication. This can be critical when guiding users during emergencies or when security needs more information to assess the situation. Here is how an SOS Alert is managed by the WebSocket consumer:

class SOSConsumer(AsyncWebsocketConsumer):

async def connect(self):

await self.channel\_layer.group\_add("sos\_alerts", self.channel\_name)

await self.accept()

After connecting, the ‘SOSConsumer’ joins the "sos\_alerts" group to send SOS notifications to all connected clients.

async def disconnect(self, close\_code):

await self.channel\_layer.group\_discard("sos\_alerts", self.channel\_name)

When a client disconnects, the consumer leaves the "sos\_alerts" group.

async def receive(self, text\_data):

text\_data\_json = json.loads(text\_data)

The receive method handles the incoming message that are received via the WebSocket connection.

async def send\_sos\_alert(self, event):

message = event['message']

await self.send(text\_data=json.dumps(message))

When messages need to be broadcast to "sos\_alerts" clients, send\_sos\_alert is called. WebSocket client gets the SOS alert, which is in our case is the admin.

Building upon the enhanced real-time capabilities provided by WebSockets, another vital component of our system's responsiveness and intelligence is the integration of middleware for face recognition. This technology is pivotal in automating and enhancing the security features of the CEAS by enabling advanced user identification and status updates during emergencies.

### 4.1.4 Middleware for Face Recognition Model

Face Recognition Implementation: The face recognition middleware is meant to automate the process of identifying people within the campus during an emergency and employs advanced machine learning algorithms to improve situation awareness during the emergency. The face\_recognition & OpenCV library is used by the system to process images from live video feeds, to detect the faces and to compare them with a pre-trained dataset of known faces.

* Training the Model: The system starts by fetching images from the training directory, where each image is linked to a user. The images are then used to derive the face encodings using the face\_recognition library, which are then saved along with the user labels.

Code Integration and Flow: The face recognition middleware will start only if the admin triggers an emergency. This view demonstrates this logic perfectly:

@csrf\_exempt

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def trigger\_emergency\_alert(request):

try:

title = request.GET.get('title', 'Emergency Alert')

message = request.GET.get('message', 'The campus status is now "Emergency". Please update your status.')

triggered\_by = None

if request.user.is\_authenticated:

triggered\_by = request.user.userprofile

# Creating an EmergencyAlert

alert = EmergencyAlert(

status='emergency',

triggered\_by=triggered\_by,

notes=message

)

alert.save()

UserProfile.objects.update(status='in\_danger')

# Send notifications if the user is authenticated

if request.user.is\_authenticated:

alert.send\_emergency\_notifications()

# Starting the FRM to capture

FRM\_URL = f"{os.environ.get('FACE\_RECOGNITION\_URL')}/start\_face\_recognition/"

response = requests.get(FRM\_URL)

if response.status\_code == 200:

print("FRM face recognition started successfully.")

else:

print("Failed to start FRM face recognition.")

return JsonResponse({"message": "Emergency alert triggered successfully. FRM notified."}, status=status.HTTP\_200\_OK)

except Exception as e:

return JsonResponse({"error": str(e)}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

After the face recognition model started it detects faces and recognizes them by comparing with known encodings. When the user is captured, it will send back request to update the status of the user to ‘safe.’ Here is the code that is responsible for detecting the faces and recognizing.  
def get\_frame(self):

names = set()

frame = self.vs.read()

frame = cv2.flip(frame, 1)

frame = imutils.resize(frame, width=900)

imgS = cv2.cvtColor(cv2.resize(frame, (0, 0), None, 0.25, 0.25), cv2.COLOR\_BGR2RGB)

# Detecting faces and recognize

facesCurFrame = face\_recognition.face\_locations(imgS)

encodesCurFrame = face\_recognition.face\_encodings(imgS, facesCurFrame)

for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):

matches = face\_recognition.compare\_faces(self.encodeListKnown, encodeFace)

faceDis = face\_recognition.face\_distance(self.encodeListKnown, encodeFace)

matchIndex = np.argmin(faceDis)

if matches[matchIndex]:

name = self.classNames[matchIndex].upper()

names.add(name)

self.draw\_face(frame, name, faceLoc, True)

self.names1 = names

ret, jpeg = cv2.imencode('.jpg', frame)

return jpeg.tobytes()

When the emergency is resolved and the security staff is certain that everyone is secure, they can send a stable request notice to alert users that the campus is stable. Meanwhile, the facial recognition will stop to function. Having established the sophisticated middleware for face recognition and its seamless operation with the backend to enhance real-time responsiveness, it is crucial to discuss how the mobile application integrates with these backend services to maximize user engagement and safety during emergencies.

### 4.1.5 Integration with Mobile Application

**Emergency Notifications and User Status Updates:** The mobile app is designed to provide immediate notifications during emergencies. When the security team triggers an alert, the following occurs:

* **Notification Handling:** Users receive push notifications, which was sent using the integrated push notification service facilitated by the backend. These notifications alert users about the emergency, ensuring they are informed regardless of their current app activity.
* **Status Update Interface:** Upon interacting with the notification, users are directed to a status update screen within the app. Here, they can mark themselves as safe, in danger, or out, enabling a dynamic flow of crucial information back to the security team.

**SOS Alert Functionality:**

* **Proactive Safety Measures:** Users have the capability to initiate an SOS alert directly from the app. This alert is immediately transmitted to the backend and then relayed to administrators and security personnel via the WebSocket connection, ensuring instant communication of the user's distress and location.
* **Real-Time Location Sharing:** The SOS functionality is designed to share the user’s real-time location with the backend, aiding swift response and appropriate action from the emergency team.

**Real-Time Communication via WebSocket:**

* **Chat Service Integration:** The mobile app features a real-time chat service, powered by WebSockets, which allows direct and immediate communication between users and administrators. This is especially vital during emergencies when rapid exchange of information can significantly impact the outcomes.
* **WebSocket Connectivity:** The continuous open connection provided by WebSockets ensures that messages are exchanged in real time, offering an efficient way to maintain communication without delays typical of traditional HTTP requests.

**Mobile Application Development:**

* **Cross-Platform Accessibility:** The app is created using React Native and controlled by Expo. It offers a smooth user experience across many device ecosystems. Due to the cross-platform capabilities, all users, regardless of their device type, will be able to access these crucial functionalities.
* **Security and Privacy:** All communications between the mobile app and the backend, including SOS signals and chat messages, are encrypted to ensure user data privacy and security compliance.

# **CHAPTER 5. WORKING MECHANISM**

In this chapter, the operational functionality of the Campus Emergency Alert System (CEAS) is described in detail with the aid of screenshots for illustration. This illustration offers an evident explanation of the ways in which every element of the system contributes to the overarching safety and security mechanism implemented at the University of Central Asia.

## 5.1 System Workflow Overview

As we have explored the main components of the system, the following diagram will provide a comprehensive visualization of how these components interact and function together.

A diagram with colorful text

Description automatically generated with medium confidence

Figure 5.1 System Architecture

## 5.2 Mobile Application Functionality

SOS Alert: This feature allows users to notify the security team about potential threats. The user-friendly interface helps the user easily access the SOS Alert button, which is demonstrated in figure 5.2. And figure 5.3 illustrates how users receive and interact with real-time alerts on their devices.

A screenshot of a phone

Description automatically generated A screenshot of a screen

Description automatically generated

(a) (b)

Figure 5.2. a. App Dashboard, and b. Real-time Alert and Update Status Screen

## 5.3 Admin Dashboard Functionality

To monitor the users status the admin dashboard provides statistical, and visual representation of users status and location. Furthermore, admins are able to send the emergency alert, view upcoming messages from users in the chatbot, and view individual user profile to check their location and information. Figure 5.4 shows the admin dashboard with tools for monitoring and managing emergencies in real-time.

A screenshot of a computer

Description automatically generated

Triggering an Emergency will send alert notification.

ChatBot

Figure 5.4 Admin dashboard

Additionally, when the admin triggers an emergency, the face recognition model will start to work to capture the users and update their status automatically.

# **CHAPTER 7. CONCLUSION**

This thesis describes the design, implementation, and assessment of the Campus Emergency Alert System (CEAS) at the University of Central Asia. The CEAS has been shown to improve the university's safety and security infrastructure via extensive study, systematic development, and rigorous testing.

**Integration of Advanced Technologies:** The CEAS combines face recognition, real-time alarm systems, and immediate communication channels into a unified architecture. This integration is set to revolutionise UCA's existing emergency response procedures by enabling accurate and immediate responses to possible threats.

**Enhancement of Safety Measures:** With the introduction of the CEAS, the UCA is better prepared to manage emergency circumstances. The system's architecture is intended to promote rapid distribution of information, which is critical in emergency situations. The mobile app and admin dashboard work together to keep all campus members, from students to security professionals, informed and ready to respond, as necessary.

**Operational Efficiency:** The system's ability to automate critical responses and streamline communication channels minimizes response times and maximizes the efficacy of safety protocols.

**Data Security and Privacy:** Data security and privacy were given a lot of thought when the CEAS was being made. The system is designed to keep personal information safe by using encryption and strict data protection rules.

**Implications for Institutions:** The implications of this work extend beyond UCA. The system is not limited, and it can serve other educational and non-educational institutions or corporations seeking to upgrade their safety infrastructure. It stands as an exemplar of how technology can be harnessed to create safer learning spaces.

**Recommendations for Future Enhancements:** Future improvements could involve expanding the system's deployment, adding more analytics for predictive threat assessment, and investigating modern technology advances to further enhance system capabilities. This thesis has established the framework for a complex emergency alert system.

**Final Thoughts:** In conclusion, the CEAS offers a promising solution to the challenge of campus security. It embodies a forward-thinking approach to the application of technology in safeguarding educational spaces and contributes to the ongoing evolution of campus safety strategies. As institutions worldwide continue to seek improved security measures, the CEAS developed for UCA stands as a beacon of innovation and practical application in this critical field.

# **REFERENCES**

[1] Communication technology, emergency alerts, and campus safety. (2009, January 1). IEEE Journals & Magazine | IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/5072380>.

[2] Campus Emergency Notification Systems on JSTOR. (n.d.). <https://www.jstor.org/stable/26628657>.

[3] Abdullahi, A. M., & Orukpe, P. E. (2016). DEVELOPMENT OF AN INTEGRATED CAMPUS SECURITY ALERTING SYSTEM. Nigerian Journal of Technology. Retrieved from: <https://doi.org/10.4314/njt.v35i4.26>.

[4] Smart Mobile-Based Emergency Management and Notification System. (2016, August 1). IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/7592738>.

[5] Elliott, Madelyn, "CAMPUS SAFETY AND THE IMPACT OF EMERGENCY ALERT SYSTEMS"

(2023). Theses and Dissertations. 3117: <https://rdw.rowan.edu/etd/3117>.

[6] Statistics, N. C. (2022, June 28). *Safety and Security Practices at Public Schools.* Retrieved from

National Center for Education Statistics: <https://nces.ed.gov/programs/coe/indicator/a19>.

[7] Yan Liu, W. Z. (2020, January 15). Retrieved from ScienceDirect:

<https://www.sciencedirect.com/science/article/abs/pii/S0140366419312149>.

[8] Gordon Gow, T. K. (2009). Communications Technology, Emergency Alerts, and Campus Safety.

*Communications Technology and Campus Saferty: Critical Sociotechnical Concerns for Emergency*

*Messaging at Canadian Universities*, 6.

[9] Lee, J., & Cho, Y. (2018). Geolocation in safety applications. *Safety Science*, 30(2), 120-127.

[10] Martin, L., & Rice, J. (2019). Real-time location-based services in emergencies. *TechTrends*, 26(3), 49-56.

[11] Mindy Menn, C. P.-P. (2021). When minutes matter: A university emergency notification system

dataset. 9.

[12] Han, W., Ada, S., Sharman, R., & Rao, H.R. (2015). Campus Emergency Notification Systems: An

Examination of Factors Affecting Compliance with Alerts. MIS Q., 39, 909-929.

[13] What is application deployment? | vmware glossary. (n.d.). https://www.vmware.com/topics/glossary/content/application-deployment.html