



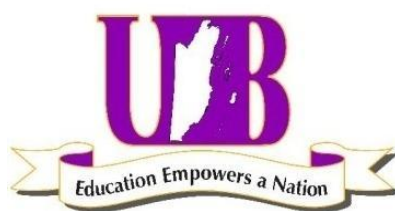
Software Engineering

CMPS4131

Tamika Chen, Raynisha Cornelio, Javier Castellanos, Abner Mencia, Alex Peraza

Report 3

May 17, 2024



# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>INDIVIDUAL CONTRIBUTIONS BREAKDOWN</b>	<b>4</b>
<b>GANTT CHART</b>	<b>6</b>
<b>Customer Statement Requirements</b>	<b>7</b>
Problem Statement	7
<b>Glossary of Terms</b>	<b>10</b>
<b>System Requirements</b>	<b>12</b>
Enumerated Functional Requirements	12
Enumerated Non-Functional Requirements	13
On Screen Appearance Requirements	14
<b>Functional Requirements Specification</b>	<b>15</b>
Stakeholders	15
Actors & Goals	15
Casual Description	17
Use Case Diagram	21
System Sequence Diagram	21
Traceability Matrix	35
Fully Dressed Description	36
<b>Effort Estimation Using Use Case Points</b>	<b>44</b>
<b>Domain Analysis</b>	<b>44</b>
Concept Definitions	44
Association Definitions	47
Attribute Definitions	49
Traceability Matrix	51
System Operation Contracts	52
Mathematical Models / Algorithms	57
Assumptions and Notations	57
Detailed Model	58
<b>Interaction Diagrams</b>	<b>59</b>
<b>Class Diagram and Interface Specification</b>	<b>63</b>
Class Diagram	63
Data Types and Operation Signature	64
Traceability Matrix	64
OCL Contract Specifications	65
<b>System Architecture and System Design</b>	<b>66</b>
Architecture Style	66
Mapping Subsystem to Hardware	67
Connectors and Network Protocols	68

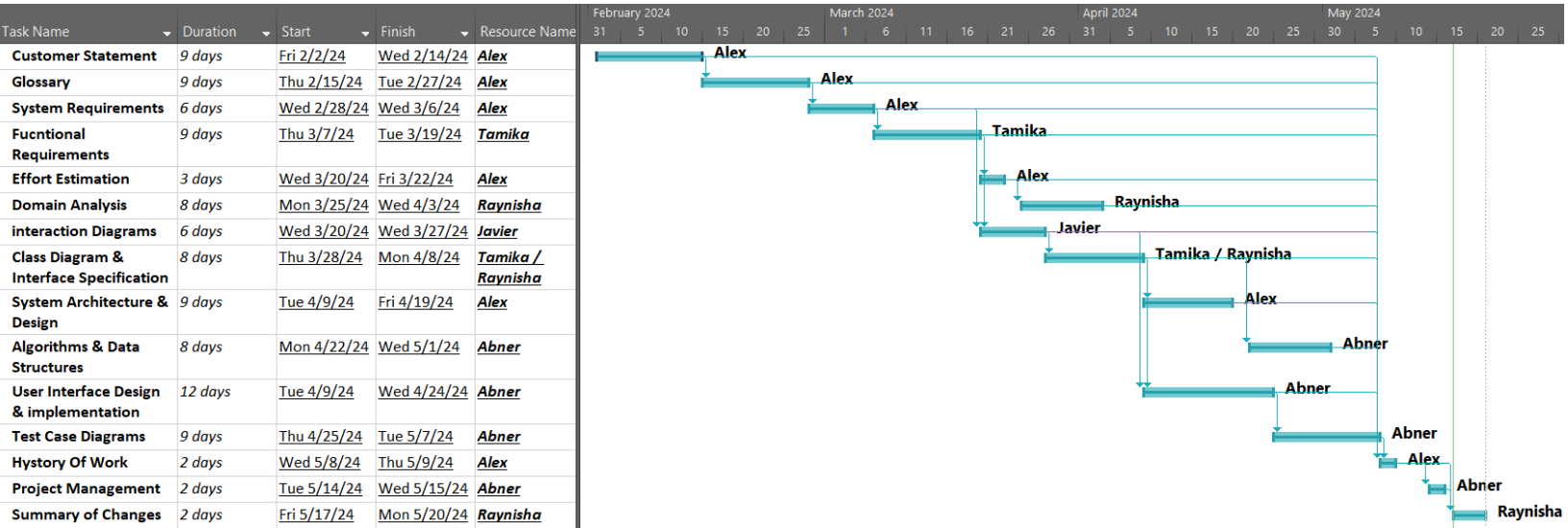
Global Control Flow	69
Hardware Requirements	70
<b>Algorithms and Data Structures</b>	<b>71</b>
<b>User Interface Design and Implementation</b>	<b>72</b>
<b>Design of Tests</b>	<b>75</b>
Overview of Design Tests	75
Strategy to Conduct Unit Testing	80
<b>History of Work</b>	<b>81</b>
History of Work	81
Current Status	81
Future Work	82
<b>Project Management</b>	<b>82</b>
<b>Summary of Changes</b>	<b>82</b>
<b>References</b>	<b>83</b>

## INDIVIDUAL CONTRIBUTIONS BREAKDOWN

Responsibility Level	Team Member Names				
	Tamika	Raynisha	Alex	Javier	Abner
Sec. 1: Customer Statement 6pts			100%%		
Glossary 4pts			100%		
Sec. 2: System Requirements 6pts			100%		
Sec. 3: Functional Requirement Specification 30pts	100%				
Sec. 4: Effort Estimation 4pts			100%		
Sec. 5: Domain Analysis 25pts		100%			
Sec. 6: Interaction Diagrams 40pts				100%	
Sec. 7: Class Diagram & Interface Specifications 20pts	50%	50%			

Sec 8: System Architecture and Design 15pts			100%		
Sec 9: Algorithms and Data Structure 4pts					100%
Sec 10: User Interface Design and Implementation 11pts					100%
Sec 11: Test Case Design 12pts					100%
Sec 12: History of work 5ts			100%		
Sec 13: Project Management 13pts					100%
Sec 13: Summary of Changes 5pts		100%			

GANTT CHART



# Customer Statement Requirements

## Problem Statement

The University of Belize grapples with a multifaceted challenge regarding the safety and security of its faculty, staff, and student body due to the absence of an integrated and user-friendly campus safety application. While the university employs certain safety precautions, they are not optimized to meet its community's dynamic and evolving safety needs. The existing measures lack the seamless ease of use, instantaneous information sharing, and personalized support that a comprehensive safety app could provide. Consequently, there is a growing demand among stakeholders for a centralized platform that enables swift responses to crises, disseminates pertinent safety information, and delivers timely notifications.

The current safety landscape at the University of Belize exhibits several deficiencies. Firstly, the absence of a centralized safety application hinders the efficient dissemination of critical safety information during emergencies. Without a unified platform, the communication channels become disjointed, leading to potential delays and confusion in emergency response efforts. Moreover, the lack of real-time alerts and location-based support further exacerbates the challenges of proactive threat mitigation and timely assistance to individuals in distress.

In response to these challenges, there is an urgent need for the development and implementation of a comprehensive campus safety app tailored to the specific needs of the University of Belize. The envisioned app should serve as a centralized hub for accessing essential safety services, including emergency contact information, crisis management protocols, and campus security resources. It should facilitate real-time communication and collaboration among stakeholders, enabling swift responses to safety threats and emergencies.

The campus safety application must provide real-time alerts and notifications to users concerning various safety incidents, weather advisories, and emergency protocols. This feature is pivotal in ensuring that individuals within the university community are promptly informed about potential threats or hazardous situations, allowing them to take necessary precautions or actions. By delivering timely alerts, the app enhances situational awareness and empowers users to make informed decisions in response to emergencies. Additionally, real-time alerts enable swift dissemination of critical information, facilitating coordinated response efforts and minimizing the impact of adverse events. Whether it's notifying users about safety hazards on campus, impending

severe weather conditions, or outlining emergency procedures, the real-time alerts feature plays a crucial role in enhancing overall safety and security within the university

The application should incorporate location-based support through the utilization of GPS technology. This feature enables users to accurately pinpoint their exact location in the event of emergencies, facilitating swift and effective response from emergency responders. By leveraging GPS technology, the app enhances the ability of users to communicate their precise whereabouts, ensuring that assistance can be promptly dispatched to the relevant location. Moreover, location-based support empowers responders with critical information to assess the situation and provide appropriate assistance in a timely manner. Whether it's guiding individuals to safety during crises or coordinating rescue efforts in emergency scenarios, the integration of location-based support enhances the overall efficacy and responsiveness of the campus safety application, thereby bolstering the safety and security of the university community.

It should encompass comprehensive crisis management tools to aid users in navigating emergency situations effectively. This includes providing intuitive tools and resources such as emergency contact information, evacuation routes, and medical assistance guidelines. By offering accessible and user-friendly interfaces, the app equips individuals with the necessary information and protocols to respond swiftly and appropriately during crises. Emergency contact information ensures that users can quickly reach out to relevant authorities or support services in times of need, facilitating timely assistance and intervention. Additionally, the inclusion of evacuation routes enables users to navigate safely to designated assembly points or shelters during emergencies, minimizing potential risks and hazards. Moreover, the provision of medical assistance guidelines equips users with essential knowledge and procedures for administering first aid or seeking medical attention when faced with medical emergencies. In essence, the integration of crisis management tools within the campus safety application enhances preparedness, responsiveness, and overall safety within the university community.

The campus safety application should incorporate features that allow users to create personalized safety plans and profiles tailored to their individual needs and preferences. This includes the ability for users to input essential medical information, emergency contacts, and specific safety preferences within their profiles. By enabling users to customize their safety plans, the app empowers individuals to proactively prepare for potential emergencies and tailor responses based on their unique circumstances. Including medical information ensures that responders have access to critical health details in the event of medical emergencies, facilitating prompt and informed care. Moreover, the inclusion of emergency contacts allows users to quickly reach out to designated individuals for assistance or support during crises. Additionally,



accommodating specific safety preferences ensures that users can adhere to their preferred safety protocols and procedures in various situations. Overall, the provision of customized safety plans enhances user preparedness, facilitates efficient emergency response, and contributes to a safer and more secure campus environment.

The campus safety application should seamlessly integrate with the existing array of campus security systems, encompassing surveillance cameras, access control points, and emergency call boxes. This integration ensures a cohesive and interconnected approach to campus security, enabling the app to leverage the capabilities of these systems for enhanced monitoring, access control, and emergency response. By synchronizing with surveillance cameras, the app can provide real-time visual monitoring of campus activities, aiding in the detection and prevention of potential security threats. Integration with access control points enables the app to regulate entry and exit points across campus, bolstering security measures and mitigating unauthorized access. Moreover, integration with emergency call boxes facilitates swift and direct communication between users and emergency responders, expediting response times and ensuring timely assistance during emergencies. Overall, the seamless integration of the campus safety application with existing campus resources enhances the overall efficacy and responsiveness of the university's security infrastructure, fostering a safer and more secure environment for all members of the campus community.

In order to cultivate a culture of safety and collaboration within the university community, the campus safety application should incorporate a range of community engagement features. These features include anonymous tip reporting, safety forums, and peer support networks, all aimed at promoting active participation and communication among users. Anonymous tip reporting allows individuals to confidentially report safety concerns or suspicious activities, empowering users to contribute to campus safety without fear of reprisal. Safety forums provide a platform for open dialogue and discussion regarding safety-related topics, facilitating information sharing, awareness-raising, and community-driven initiatives. Additionally, peer support networks foster mutual assistance and solidarity among users, allowing individuals to seek and provide support to one another during times of need. By embracing community engagement features, the app not only strengthens bonds within the university community but also enhances collective efforts towards maintaining a safe and inclusive campus environment.

Ensuring accessibility and usability are paramount considerations for the campus safety application. It should be designed to accommodate all members of the university community, including individuals with disabilities, ensuring equitable access to its features and functionalities. This entails implementing features such as screen reader compatibility, alternative navigation options, and adjustable font sizes to cater to

diverse needs and preferences. Moreover, the app should feature an intuitive interface characterized by clear navigation pathways, logical organization of information, and user-friendly controls. By prioritizing accessibility and usability, the app endeavors to empower all users to effectively utilize its resources, engage in safety protocols, and contribute to a safer campus environment.

By developing and deploying a robust campus safety app, the University of Belize aims to enhance the overall safety and security environment, fostering a conducive atmosphere for learning, working, and personal well-being. The app represents a proactive step towards meeting the evolving safety needs of the university community and ensuring a resilient response to emergent safety challenges.

## Glossary of Terms

Terms	Definition
Emergency Alert System	A system designed to quickly disseminate critical information to users during emergency situations, such as natural disasters, security threats, or medical emergencies.
Location-Based Services (LBS)	Services that utilize geographical location data to provide users with relevant information, such as nearby safety resources, emergency contacts, and real-time alerts based on their current location.
Crisis Communication Tools	Tools and technologies used to facilitate communication and coordination during crisis situations, including messaging platforms, push notifications, and emergency broadcast systems.
User Authentication	The process of verifying the identity of users accessing the safety app through secure authentication mechanisms, such as passwords, biometrics, or two-factor authentication.
Incident Reporting	The process of documenting and

	reporting safety incidents, suspicious activities, or emergencies through the safety app, enabling prompt response and follow-up actions by campus security or authorities.
Accessibility Standards	Guidelines and standards set forth by organizations like the World Wide Web Consortium (W3C) to ensure digital content and applications are accessible to users with disabilities, including requirements for screen readers, keyboard navigation, and alternative text.
Data Encryption	The process of encoding sensitive information transmitted or stored within the safety app to prevent unauthorized access or interception by malicious entities, enhancing data security and privacy.
Push Notifications	Messages or alerts sent to users' devices from the safety app, providing timely updates, reminders, or emergency notifications even when the app is not actively in use.
User Interface (UI) Design	The design and layout of the safety app's graphical user interface (GUI), including elements such as menus, buttons, icons, and navigation bars, aimed at enhancing usability and user experience.
Compliance Regulations	Legal and regulatory requirements governing the collection, storage, and processing of user data within the safety app, ensuring adherence to privacy laws and protection of user rights and information.

# System Requirements

## Enumerated Functional Requirements

ID	PW	REQ-x
REQ1	3	The app will allow users (students, faculty, staff) to register with their university credentials and authenticate their identity securely.
REQ2	4	The app will be equipped with an emergency alert system capable of sending real-time notifications to users in the event of safety threats, natural disasters, or campus emergencies.
REQ3	3	The app will provide location services to users, allowing them to share their current location with campus security or emergency responders during emergencies.
REQ4	2	Users will have access to comprehensive safety resources, including emergency procedures, evacuation routes, and safety guidelines tailored to various scenarios.
REQ5	4	The app will enable users to report safety concerns, incidents, or suspicious activities directly to campus security or relevant authorities, with options for anonymous reporting if desired.
REQ6	4	The app will facilitate two-way communication between users and campus security, allowing users to request assistance, seek clarification, or provide updates during emergencies.
REQ7	4	The app will include a safety check-in feature that allows users to confirm their safety status during campus-wide emergencies, enabling authorities to track and monitor user well-being in real-time.
REQ8	4	Users will receive push notifications and alerts on their mobile devices for important safety updates, campus announcements, and emergency protocols.
REQ9	3	The app will enable users to create personalized safety plans based on their specific needs, including medical conditions, mobility limitations, and emergency contacts.
REQ10	2	The app will integrate with existing campus services, such as campus shuttles, medical facilities, and counseling resources, to provide users with seamless access to support services during

		emergencies.
REQ11	2	The app will feature interactive training modules and educational resources on safety best practices, emergency preparedness, and risk mitigation strategies for users to access at their convenience.

Most Important = 4 | Least Important = 1

## Enumerated Non-Functional Requirements

ID	PW	REQ-x
NONREQ1	3	The application should adhere to industry-standard security protocols to safeguard user data and ensure confidentiality, integrity, and availability. This includes encryption of sensitive information, secure authentication mechanisms, and protection against unauthorized access or data breaches.
NONREQ2	3	The application should have fast response times and be able to handle concurrent users efficiently, especially during emergencies when there may be a surge in usage. Response times for critical functionalities such as alert notifications and location tracking should meet predefined performance benchmarks.
NONREQ3	3	The application should be highly reliable, with minimal downtime or service interruptions. It should be resilient to hardware failures, network outages, and other potential disruptions to ensure continuous availability, especially during emergency situations when reliable communication is crucial.
NONREQ4	2	The application should be scalable to accommodate growth in user base and data volume over time. It should be able to handle increased demand during peak periods without degradation in performance or service quality.
NONREQ5	2	The application should have an intuitive user interface and be easy to use, especially in high-stress situations such as emergencies. User interactions should be straightforward and require minimal training or guidance to navigate through various features and functionalities.
NONREQ6	2	The application should be compatible with a wide range of devices and platforms, including desktops, laptops,

		smartphones, and tablets. It should support multiple operating systems (e.g., iOS, Android, Windows) and web browsers to ensure accessibility across different devices and environments.
NONREQ7	2	The application should be well-documented, with comprehensive user manuals, help guides, and tutorials to assist users in understanding its features and functionalities. Training materials should be provided to educate users on how to effectively utilize the application, especially during emergency preparedness drills or training sessions.
NONREQ8	1	The application should support multiple languages and cultural preferences to cater to a diverse user base.

Most Important = 3 | Least Important = 1

## On Screen Appearance Requirements

ID	PW	REQ-x
AR1	3	The app interface should feature a clean and organized layout with clearly defined sections for different safety features and resources, ensuring ease of navigation and user understanding.
AR2	3	Text displayed on the app interface should be legible and accessible to users of all ages and visual abilities, with adjustable font sizes to accommodate individual preferences.
AR3	3	The app should utilize a color scheme that enhances readability and visual clarity, with appropriate contrast between text and background elements to aid users with visual impairments.
AR4	3	The use of intuitive icons and graphical elements should be incorporated throughout the app interface to visually communicate key safety features, actions, and alerts, enhancing user comprehension and engagement.
AR5	2	The app interface should be designed to adapt seamlessly to different screen sizes and resolutions, ensuring optimal viewing and interaction experiences across a wide range of mobile devices and tablets.

Most Important = 3 | Least Important = 1

# Functional Requirements Specification

## Stakeholders

- University of Belize
- Security Guards
- Students/Faculty/Staff
- Emergency Responders

## Actors & Goals

Actor	Roles	Type	Goals
System	Responsible for:  1.) Promptly disseminating emergency alerts and notifications to users regarding safety threats, natural disasters, or campus emergencies.  2.) Accurately pinpoint users' locations and deliver location-based support during emergencies.  3.) Enable users to report safety concerns, request assistance, and provide updates to authorities, while also allowing security personnel to coordinate responses and communicate instructions to users.  4.) Verify the identity of users accessing the safety app and control access to sensitive features and information based on user roles and permissions.	Participating	1.) Deliver timely alerts to users regarding safety incidents, weather advisories, and emergency protocols.  2.) Utilize GPS technology to provide precise location information during emergencies.  3.) Offer intuitive tools for managing crises, including access to emergency contact information, evacuation routes, and medical assistance guidelines.  4.) Foster a culture of safety through anonymous tip reporting, safety forums, and peer support networks.
		Participating	5.) Ensure the app is accessible to all members of the university community, including





	<p>2.) Students can use the app's location-based features to share their whereabouts during late-night study sessions, off-campus trips, or other situations where safety concerns them.</p> <p>3.) Encourage students to use the app for anonymous tip reporting. Reporting concerns about safety hazards, potential threats, or suspicious behavior contributes to a safer campus.</p>	Initiating	
Emergency Responders	<p>Responsible for:</p> <p>1.) Work closely with the security guards</p> <p>2.) Provide first aid or medical care as needed.</p> <p>3.) Investigations</p>	Initiating	<p>1.) Assess the situation, coordinate with other responders, and reach the scene swiftly.</p> <p>2.) Prioritize actions based on urgency and allocate resources accordingly.</p>

## Casual Description

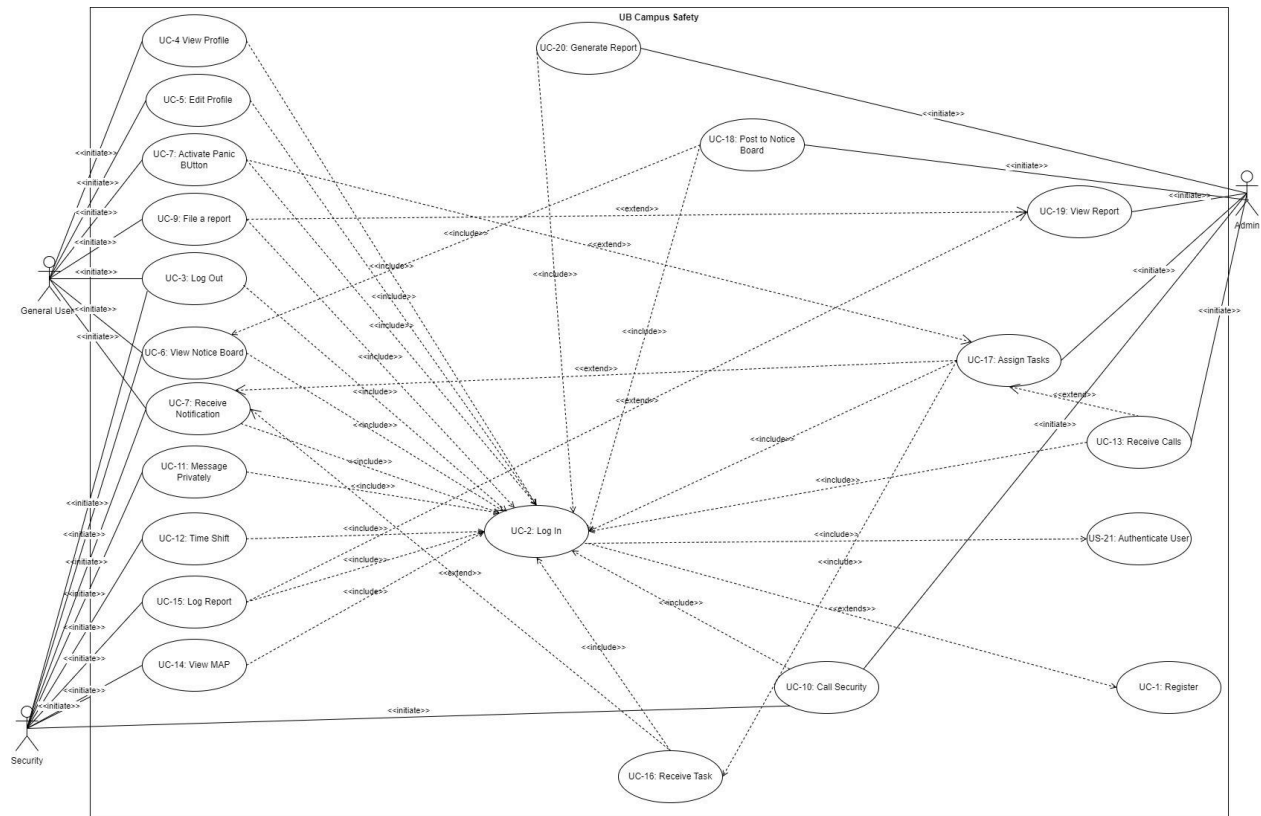
Name	Goals	Requirements Covered
<b><u>GENERAL / ALL Users</u></b>		
UC-1: Register	Allow new users to register for the app using their UB credentials.	Functional Requirements: REQ 1

UC-2:Log In	Registered users use credentials to access application features	Functional Requirements: REQ1
UC-3 Log-out	Users can log out	
UC-4:View Profile	Users have the ability to view their profile information etc...	Functional Requirements:
UC-5:Edit Profile	Users can edit/ update their profile information and set safe status during emergency	Functional Requirements: REQ7
UC-6:View_Notice_Board	Users Get a daily Board section with events of the Day at the Campus	Functional Requirements: REQ2, REQ4
UC-7:Receive Notification	Users get Notification Whenever an emergency/ event etc... is taking place within the UB campus	Functional Requirements: REQ2, REQ8
UC-8:Activate_Panic_Butt on	Enable users to quickly trigger an emergency alert, notifying authorities of their distress and location.	Functional Requirements: REQ3
UC-9:File a report	A form which allows for reports to be made to staff either anonymous or not. This is used to help in investigation ect... when needed	Functional Requirements: REQ3, REQ5
UC-10:Call Security	A feature which can be an alternative to the panic button based on 'user evaluated emergency level' this can provide a sense of security when crossing the campus at night ect...	Functional Requirements: REQ6
<b><u>SECURITY ONLY</u></b>		
UC-11:Message Privately	Feature that allows security staff to stay in	Functional Requirements: REQ6

	touch with each other throughout the day and during their shifts.	
UC-12: Time Shift	This feature is part of a MAP which is planned to be integrated. This is for security staff to be able to see which security boot within the campus have active personnel and which not during their shifts.	Functional Requirements: REQ3
UC-13: Receive Calls	Receive calls from the user side.	Functional Requirements:
UC-14: View MAP	MAP integration such that, whenever a user clicks the panic button, a pushpin appears in the map pinpointing location of the device within the UB campus. This is to allow security staff to move faster and take action more effectively than in a call.	Functional Requirements: REQ2, REQ3
UC-15: Log Report	Security staff log reports on happenings during the day, shift they did, emergencies received, patrols taken ect.. This is to allow for investigations when needed	Functional Requirements: REQ5
UC-16: Receive Task	For Security staff to receive orders from administration when needed.	Functional Requirements: REQ10
<b><u>ADMIN ONLY</u></b>		
UC-17: Assign Tasks	Assign tasks to different security personnel throughout the day when needed. This is for better efficiency throughout the day.	Functional Requirements: REQ10

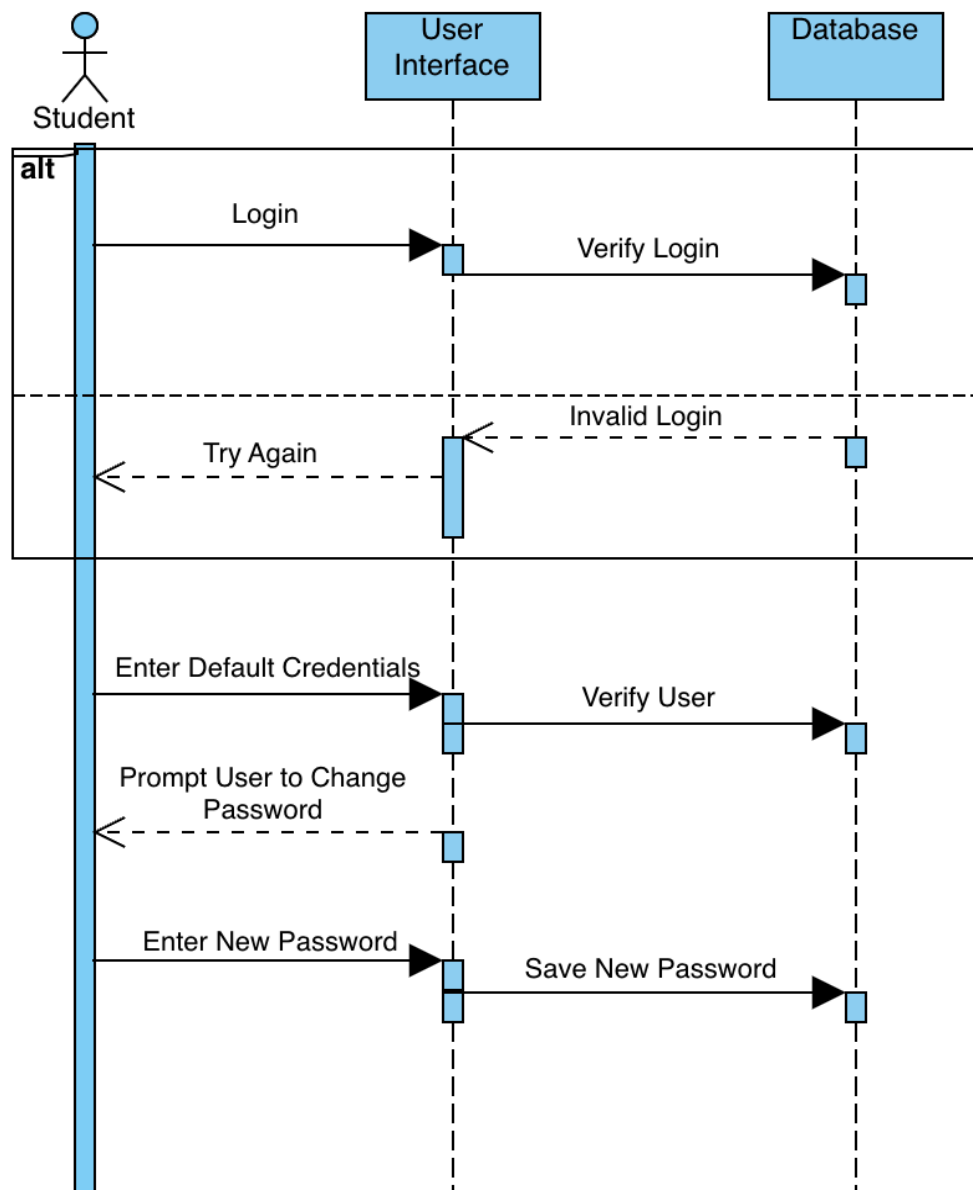
UC-18:Post to Notice Board	This is to add events taking place in the campus/ send emergency alert to campus/ ect... to allow all users to stay updated on what's taking place on campus	Functional Requirements: REQ11, REQ8
UC-19: View Report	Allows for the review of reports made by students for eg. 'investigation' when needed	Functional Requirements:
UC-20:Generate Report	Make a query using filters on the reports table using the GUI provided. This will then allow admin users to filter information which can then be viewed for investigation purposes ect...	Functional Requirements:
UC-21:Authenticate Report	Ensure that submitted reports are verified for authenticity and validity before being processed or acted upon.	Functional Requirements:

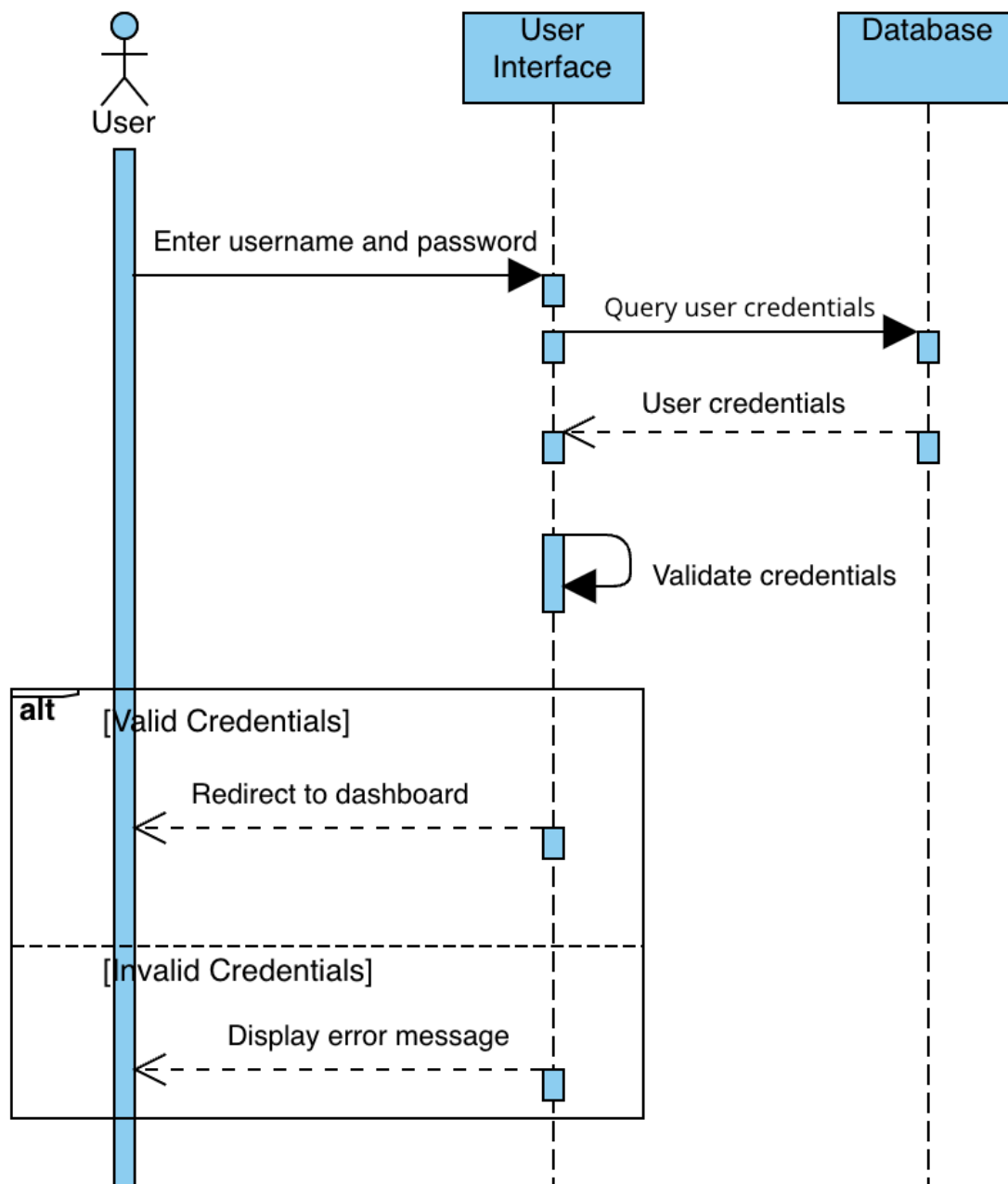
## Use Case Diagram

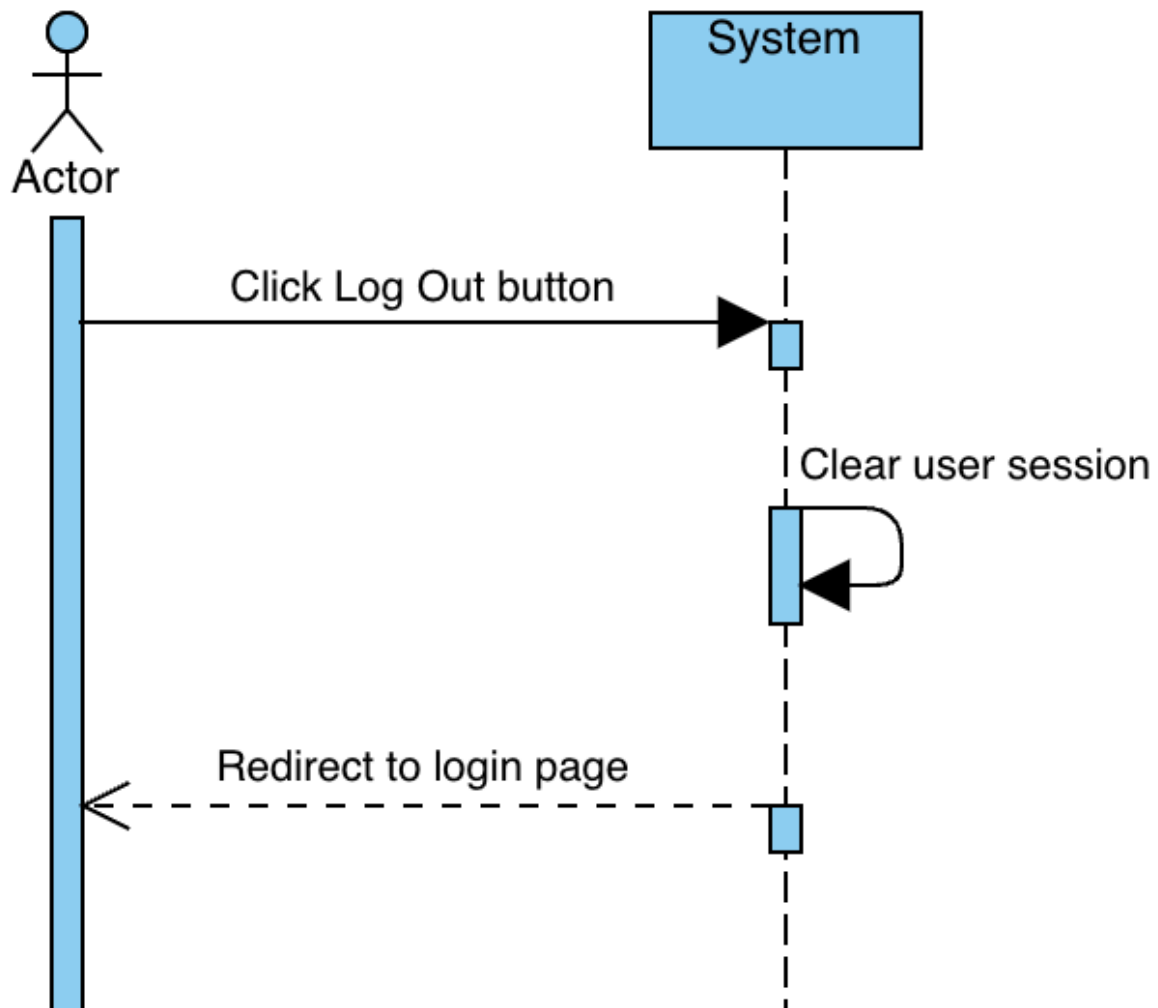


## System Sequence Diagram

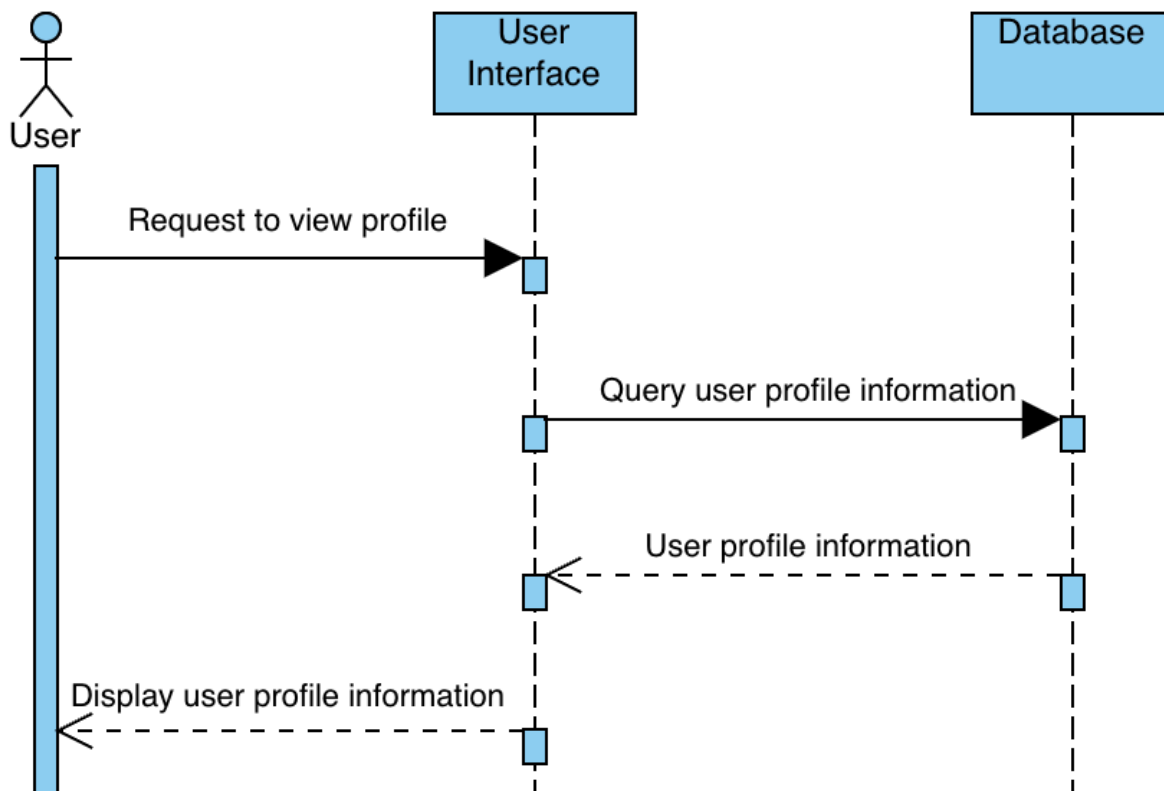
### Use Case 1: Register

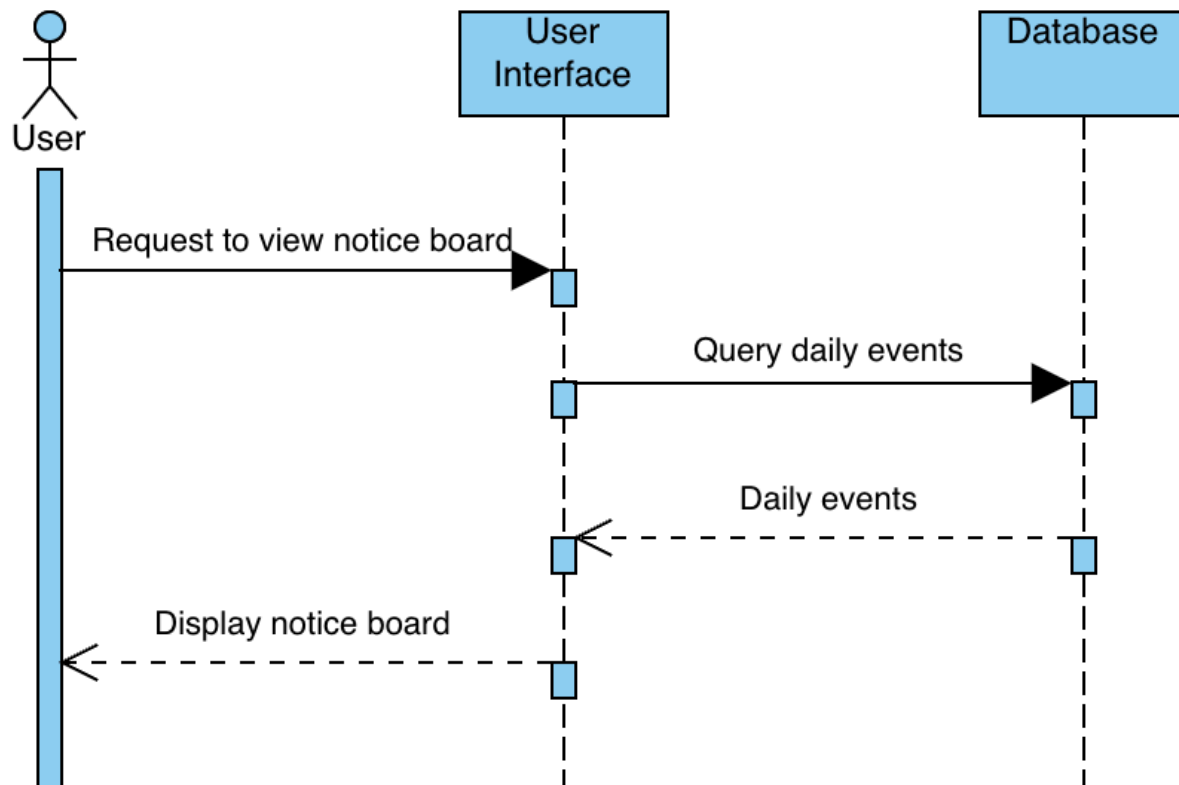


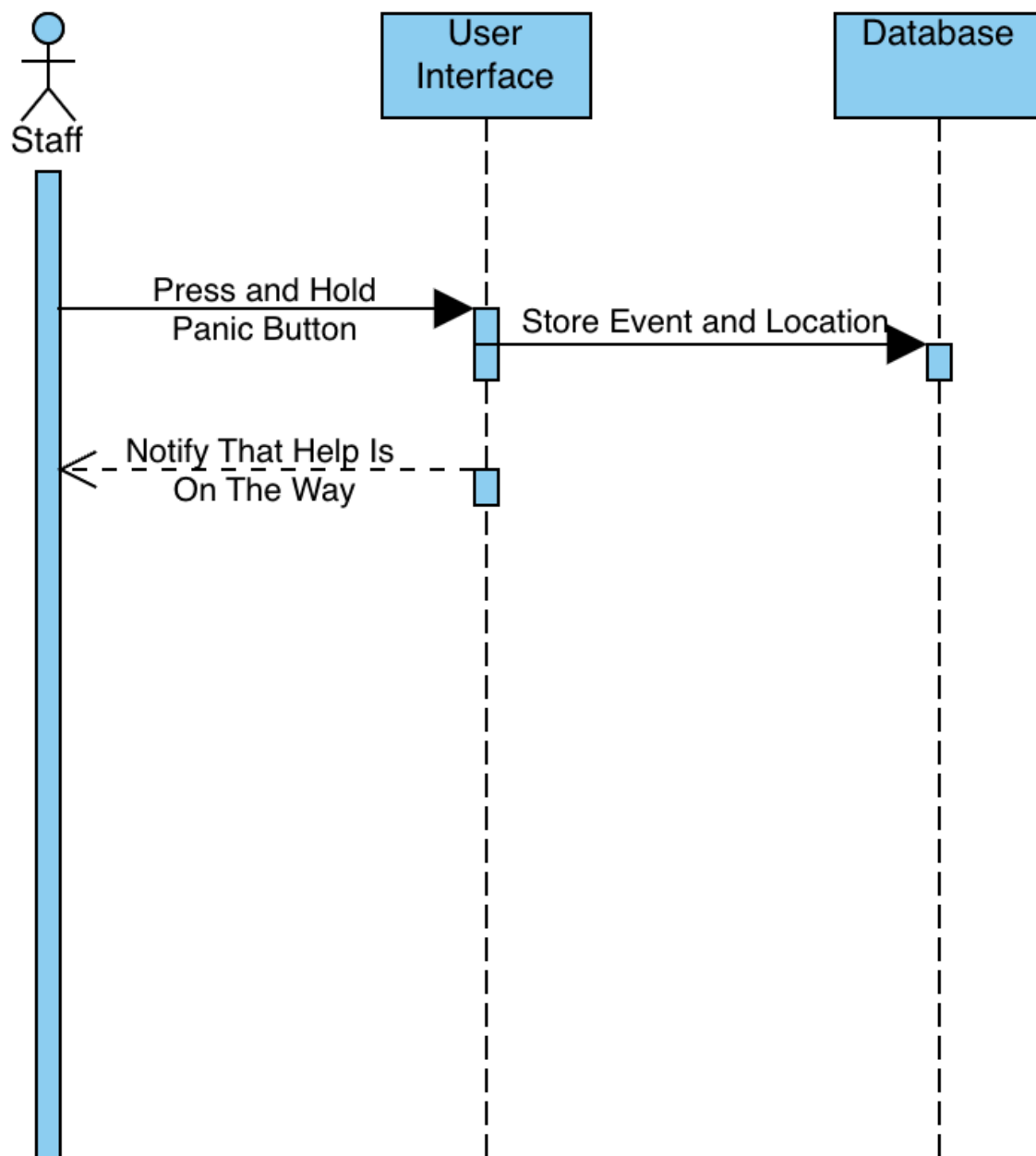
**Use Case 2: Log in**

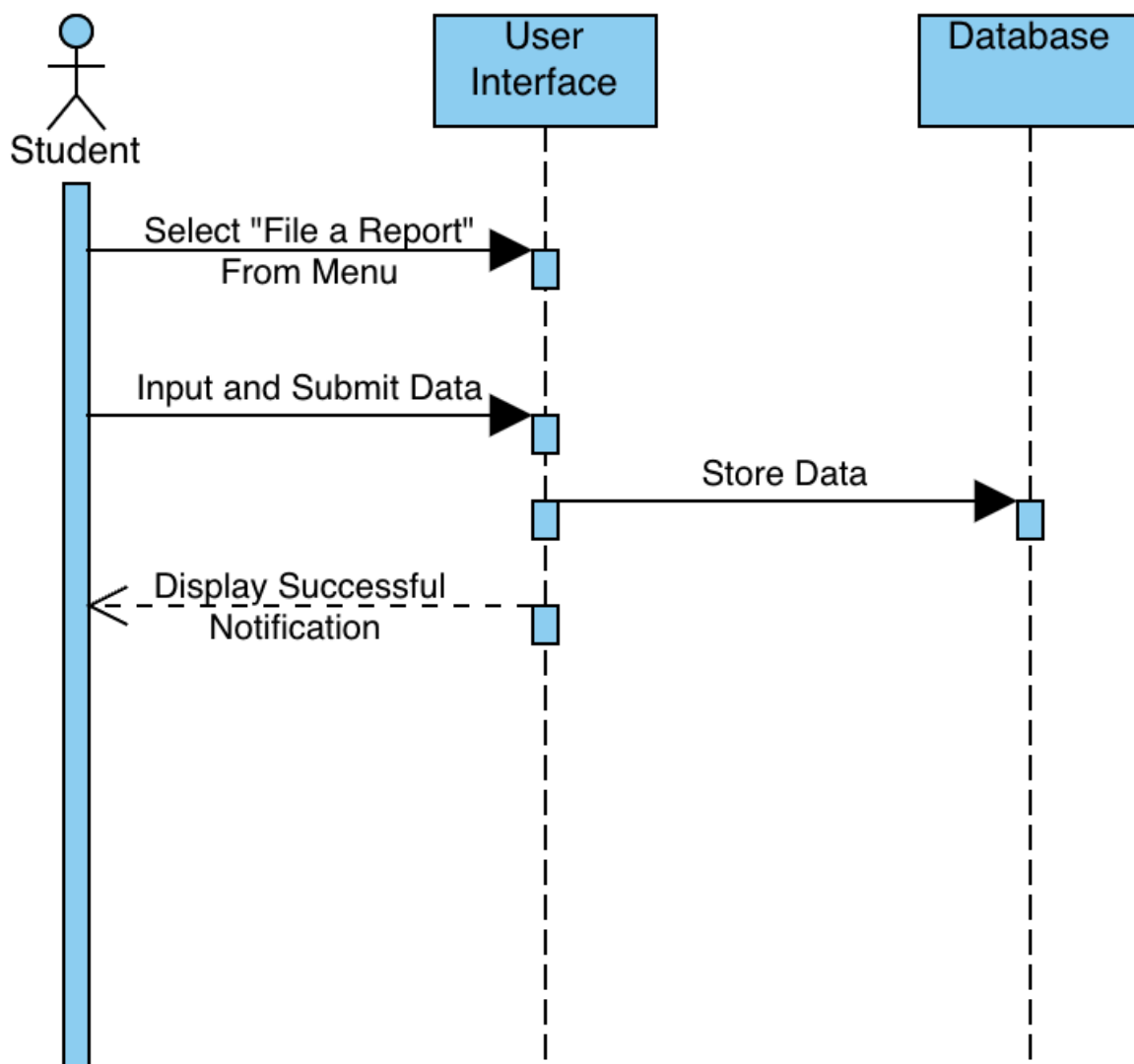
**Use Case 3: Log Out**

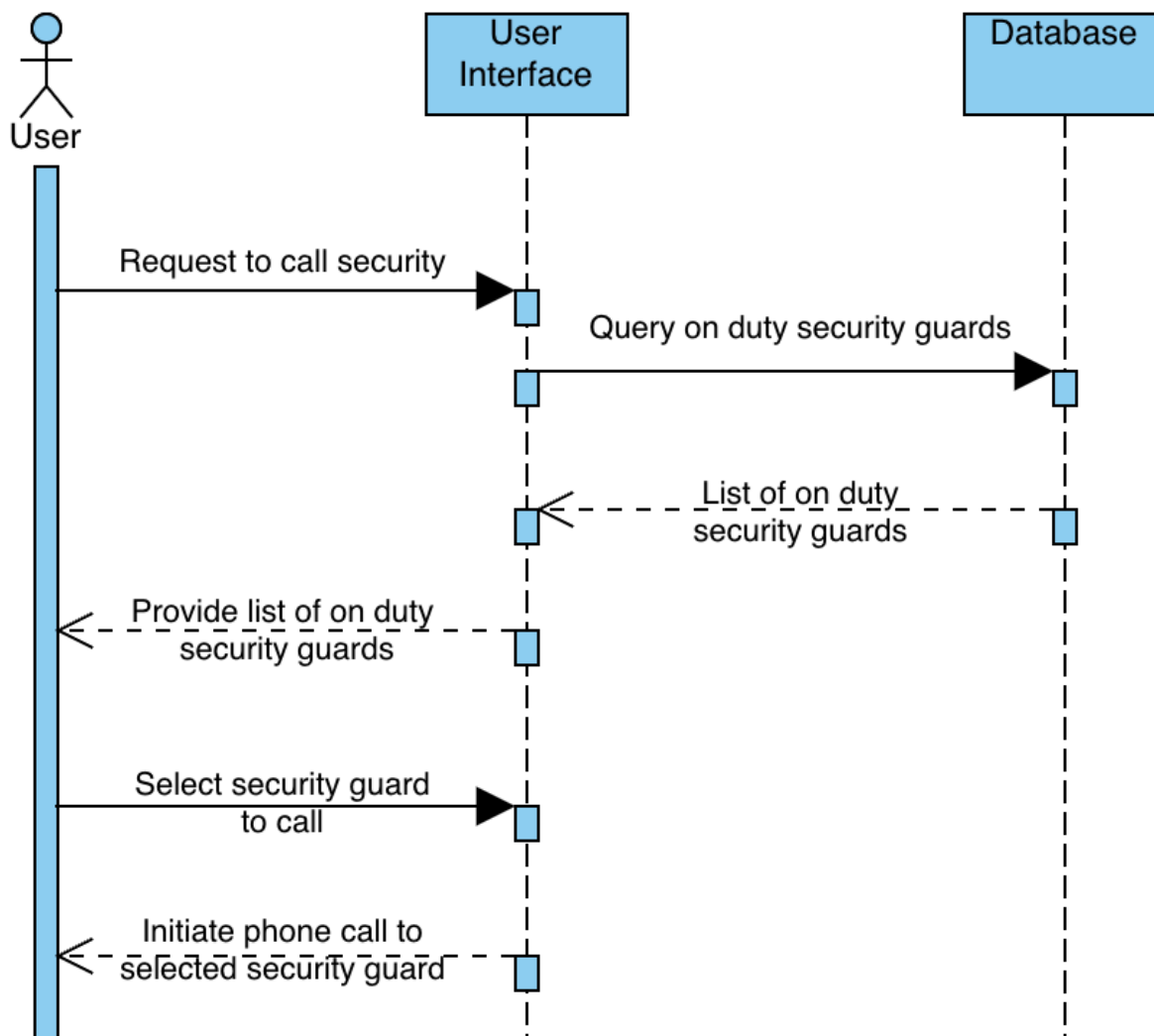


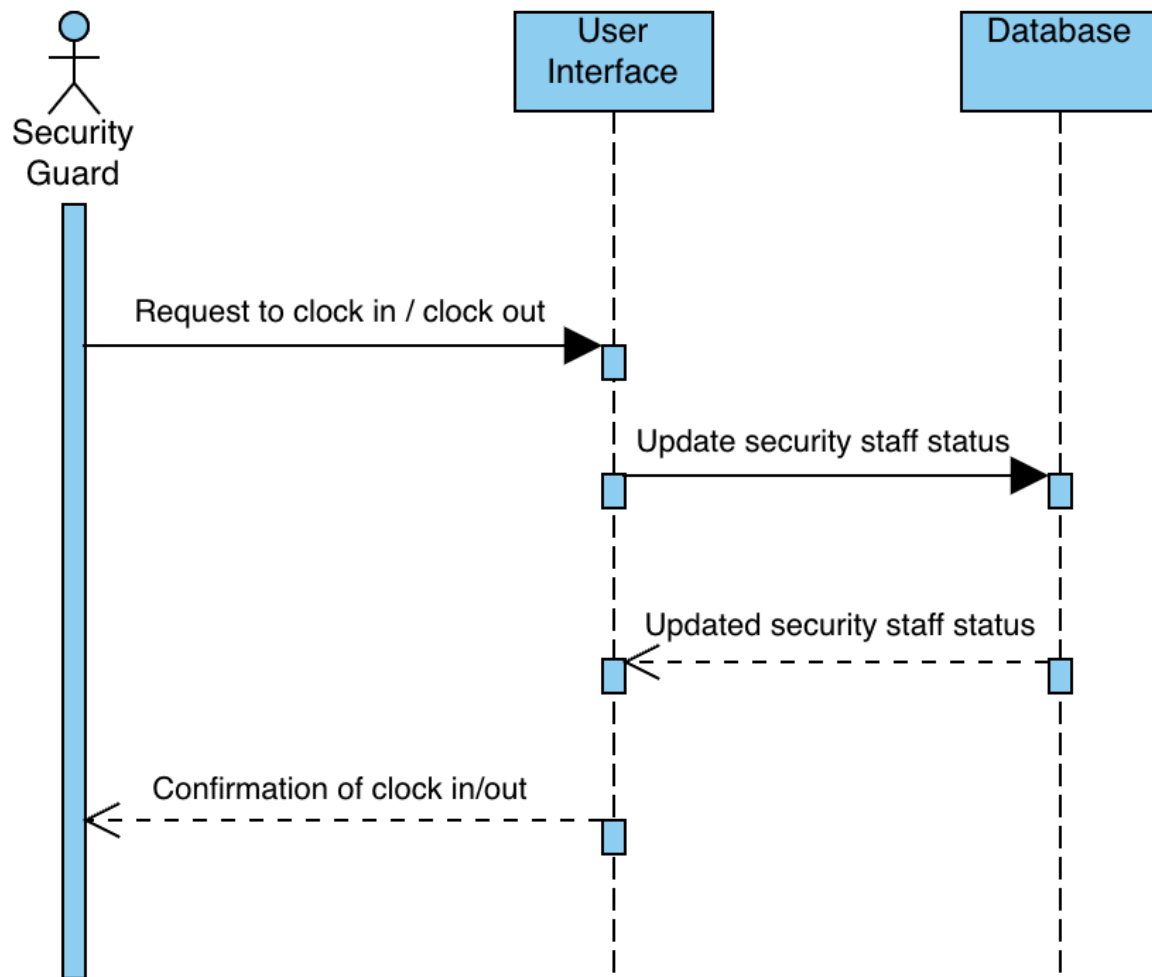
**Use Case 4: View Profile**

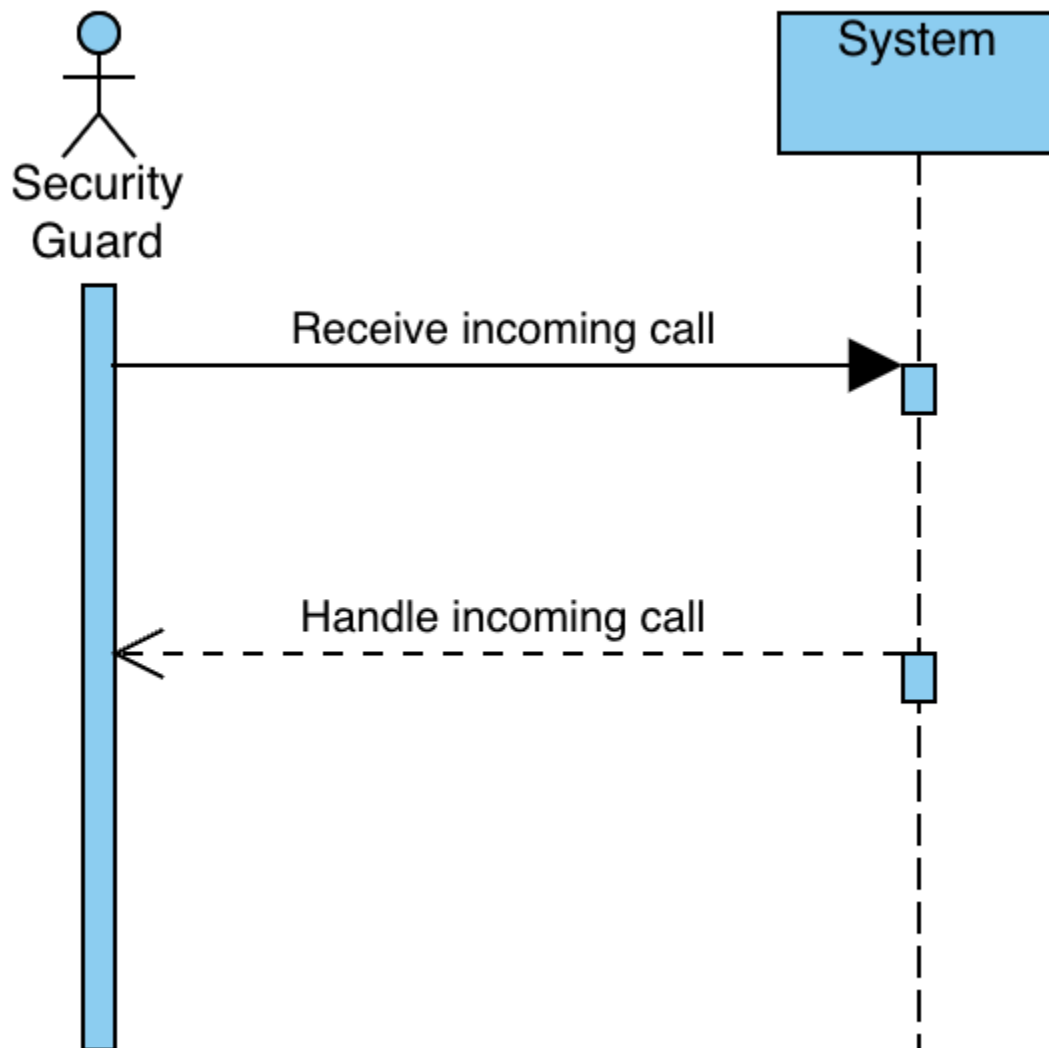
**Use Case 6: View Notice Board**

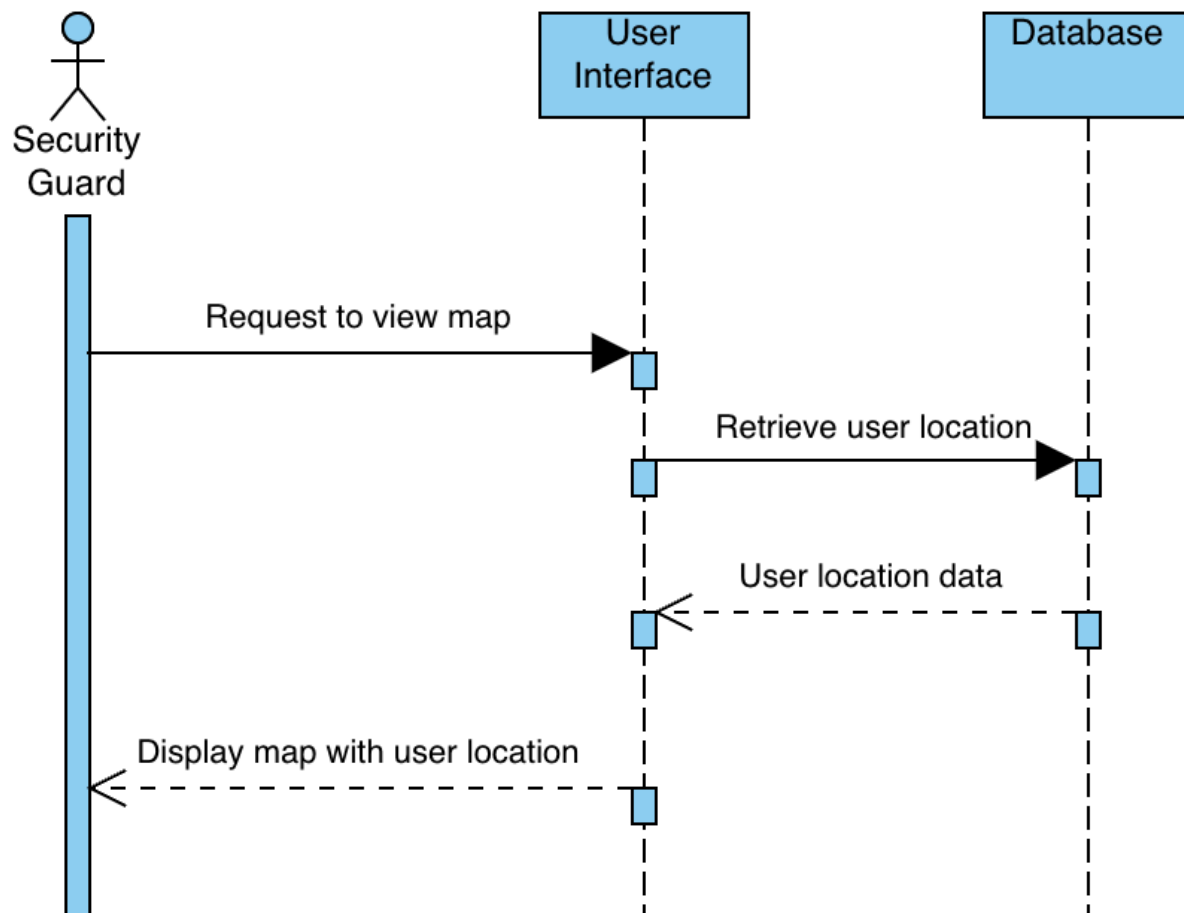
**Use Case 8: Activate the Panic Button**

**Use Case 9: File a Report**

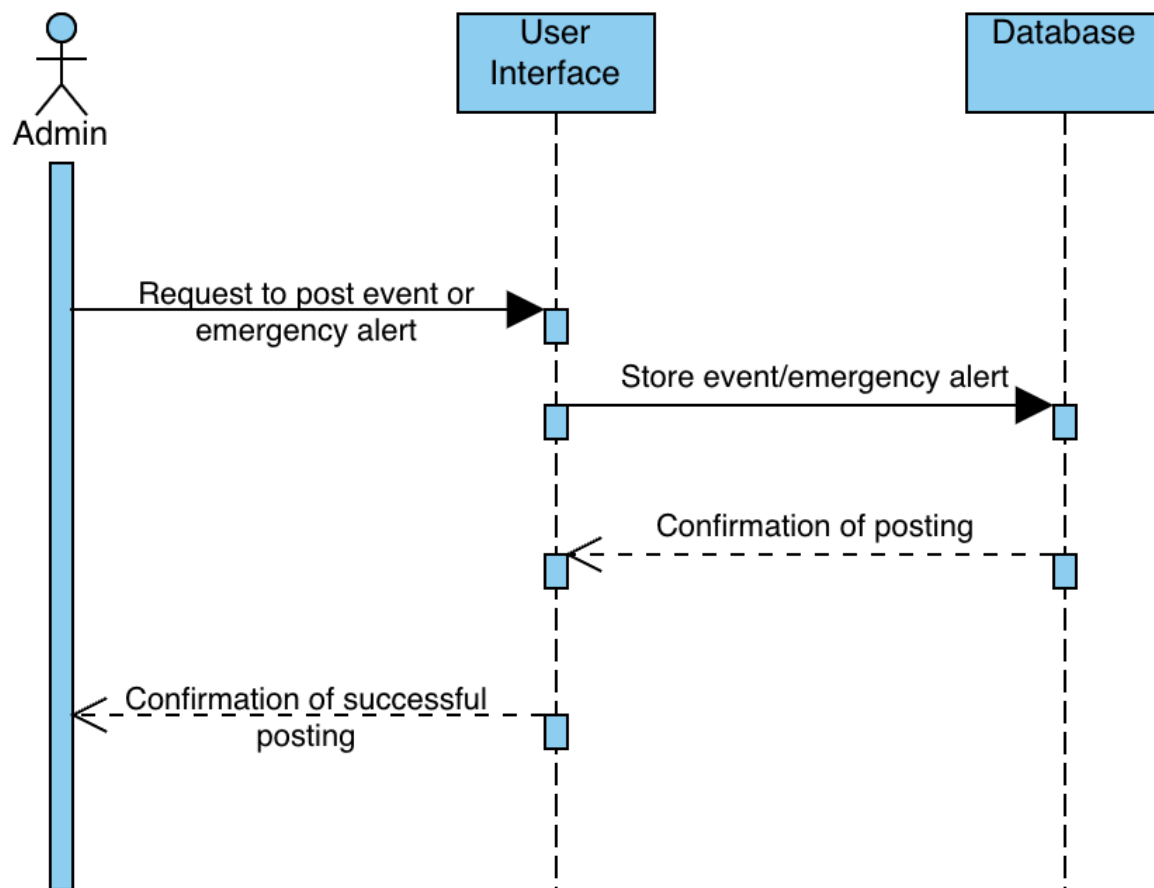
**Use Case 10: Call Security**

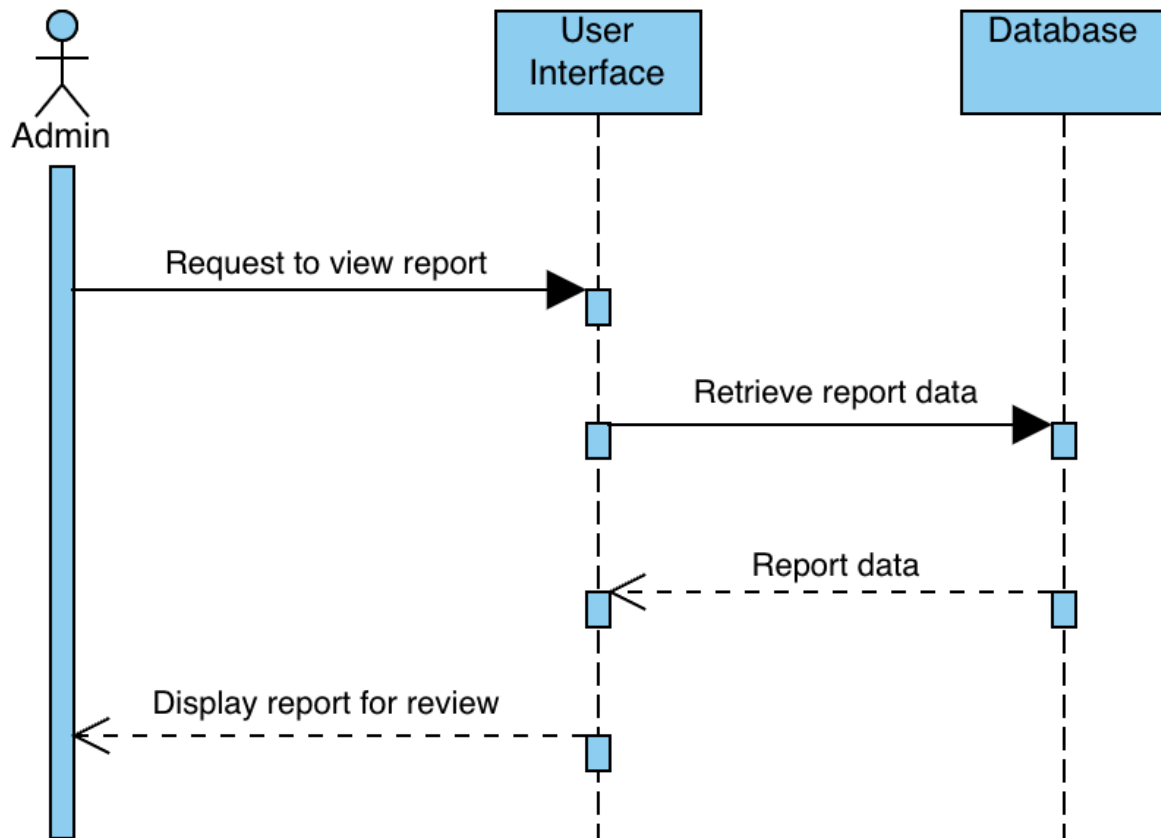
**Use Case 12: Clock in/ Clock out**

**Use Case 13: Receive Calls**

**Use Case 14: View Map**



**Use Case 18: Post to Notice Board**

**Use Case 19: View Report**

[illegible]

## Fully Dressed Description

### Fully Dressed Description of Register

Use Case UC - 1	Register
Related Requirement:	Functional Requirement(s): REQ1
Initiating Actors:	Students, Faculty/Staff, Security Guards
Actor Goal:	<ul style="list-style-type: none"> <li>• Login using their UB default credentials (ID, Email)</li> </ul>
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>• User must use their officially given UB email</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>• Systems stores account in database</li> </ul>
Flow of Events for main Success Scenario:	→ Logs into the systems using default UB credentials ← Prompts the user to change password ← Updates password ← Welcomes user to application
Flow of Events for Extensions:	← Register security guard, students based on their role

### Fully Dressed Description of Login/Logout

Use Case UC - 2	Login
Related Requirement:	Functional Requirement(s): REQ1
Initiating Actors:	Students, Faculty/Staff, Security Guards
Actor Goal:	<ul style="list-style-type: none"> <li>• Login using their UB credentials</li> </ul>
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>• User must have a valid account</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>• System displays the home page</li> </ul>
Flow of Events for main Success Scenario:	→Types username & password ← Welcomes user back

Flow of Events for Extensions:	← Types invalid username & password → outputs a error message
--------------------------------	--

### Fully Dressed Description of Logout

Use Case UC - 2	Logout
Related Requirement:	Functional Requirement(s):
Initiating Actors:	Students, Faculty/Staff, Security Guards
Actor Goal:	<ul style="list-style-type: none"> <li>Log Out</li> </ul>
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>User must have a valid account</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>System displays the home page</li> </ul>
Flow of Events for main Success Scenario:	→ clicks the sidebar → clicks logout ← systems takes back to login page
Flow of Events for Extensions:	← user to logout needs to be authenticated → redirect them to login page

### Fully Dressed Description of View Profile

Use Case UC - 3	View Profile
Related Requirement:	Functional Requirement(s): REQ
Initiating Actors:	Students, Faculty/Staff
Actor Goal:	Uses system to view profile
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>User must have a valid account</li> <li>User must be logged in</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>System displays profile information</li> </ul>
Flow of Events for main	→ Clicks the profile icon on the navigation bar

Success Scenario:	← System shows the users profile
Flow of Events for Extensions:	← user is authenticated ←user click on the profile icon → Display the profile information

#### Fully Dressed Description of View Notice Board

<b>Use Case UC - 5</b>	<b>View Notice Board</b>
Related Requirement:	Functional Requirement(s): REQ 2, REQ4
Initiating Actors:	Students, Faculty/Staff, Security Guards
Actor Goal:	To view any updates posted on the notice board
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>User wants to check if there are any updates on the notice board. User receives notification</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>User gets informed with the up to date information</li> </ul>
Flow of Events for main Success Scenario:	→User selects notice icon from menu ←User gets presented with news from the app →User can click on a specific news headline to view more details
Flow of Events for Extensions:	→Gets Notice posted by Admin →Displays to main Notice Board

#### Fully Dressed Description of Receive Notification

<b>Use Case UC - 6</b>	<b>Receive Notification</b>
Related Requirement:	Functional Requirement(s): REQ 2, REQ8
Initiating Actors:	Students, Faculty/Staff, Security Guards
Actor Goal:	To view receive any notifications
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>User must have notifications turned on for app</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>System displays notification on phone</li> </ul>

Flow of Events for main Success Scenario:	<ul style="list-style-type: none"><li>→ System sends notification to users</li><li>← Notification is displayed on the users phone</li></ul>
Flow of Events for Extensions:	<ul style="list-style-type: none"><li>→ System gets notifications content from database and shown in a notification window (hidden)</li><li>← user clicks on the notification icon</li><li>→ Notification is shown to user</li></ul>

### Fully Dressed Description of Activate Panic Button

<b>Use Case UC - 7</b>	<b>Activate Panic Button</b>
Related Requirement:	Functional Requirement(s): REQ 2, REQ8
Initiating Actors:	Students, Faculty/Staff
Actor Goal:	To help for a serious emergency
Participating Actors:	Security Guards, Admin
Preconditions:	<ul style="list-style-type: none"> <li>User requires immediate assistance in case of an emergency</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>Admin receives alert and sends security guards for assistance</li> </ul>
Flow of Events for main Success Scenario:	→ Enters app ← If already logged in, press and hold panic button → User receives prompt that button has been activated
Flow of Events for Extensions:	When pressed user get option to abort ← If not aborted. Emergency signal is sent to admin/guard on map section →panic page is reloaded after signal is sent

### Fully Dressed Description of File a Report

<b>Use Case UC - 8</b>	<b>File a report</b>
Related Requirement:	Functional Requirement(s): REQ 3, REQ5
Initiating Actors:	Students, Faculty/Staff, Security Guards,
Actor Goal:	To write a report of any suspicious activity or unusual occurrence
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>User can fill out a form to write a report</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>Report is logged and sent to the database where it can be viewed by an admin</li> </ul>
Flow of Events for main Success Scenario:	← If already logged in, User select the File a report item from the menu. → User fill out form



	← UserConfirm submission of form →System stores form in database
Flow of Events for Extensions:	← Student selects the form icon to file a report →Display the form to be filled ←Student fills the form and submits

#### Fully Dressed Description of Call a Security

<b>Use Case UC - 9</b>	<b>Call a Security</b>
Related Requirement:	Functional Requirement(s): REQ6
Initiating Actors:	Students, Faculty/Staff
Actor Goal:	To call security for assistance
Participating Actors:	System, Security
Preconditions:	<ul style="list-style-type: none"> <li>• Security is available</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>• Security is notified of incident</li> <li>• Call date and time is logged</li> </ul>
Flow of Events for main Success Scenario:	← User selects Call icon → User press the Second Call icon to activate a call ← User gets redirected to an available security → System logs time and date of call ← Security accepts call
Flow of Events for Extensions:	← Student selects the contact icon → Display contact added by Admin ←Student can select my contact to add a personal contact → Displays contact added by the student ←Student can select add contact to add a new contact → Displays form to submit the contact information ← Student submits the form

#### Fully Dressed Description of Clock in/ Clock out

<b>Use Case UC - 11</b>	<b>Clock in/ Clock Out</b>
-------------------------	----------------------------

Related Requirement:	Functional Requirement(s): REQ6
Initiating Actors:	Security
Actor Goal:	To notify system that they are available/ present for today
Participating Actors:	System, Admin
Preconditions:	<ul style="list-style-type: none"> <li>Security is available to work</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>Security notifies personnel that they are available to work</li> </ul>
Flow of Events for main Success Scenario:	<b>Clocking In</b> ← Security selects Profile icon → Security presses the clock in icon ← System marks security as available and logs the time stamp
Flow of Events for Extensions:	<b>Clocking Out</b> ← Security selects Profile icon → Security presses the clock in which is shown as clock out if the user to already clocked in icon ← System marks security as unavailable and logs the time stamp

### Fully Dressed Description of UB MAP

<b>Use Case UC - 13</b>	<b>View UB MAP</b>
Related Requirement:	Functional Requirement(s): REQ2, REQ3
Initiating Actors:	Security
Actor Goal:	View where tasks/ assistance is located
Participating Actors:	System
Preconditions:	<ul style="list-style-type: none"> <li>Security has locations turned on</li> <li>Security has tasks to accomplish</li> </ul>
Post Conditions:	<ul style="list-style-type: none"> <li>System display a map with tasks located on it</li> </ul>
Flow of Events for main Success Scenario:	→ Open app ← Select map icon from menu

	→ System display map with tasks
Flow of Events for Extensions:	If there are no tasks for the security → System will still display map

### Fully Dressed Description of Post to Notice Board

<b>Use Case UC - 17</b>	<b>Post to Notice Board</b>
Related Requirement:	Functional Requirement(s): REQ11, REQ8
Initiating Actors:	Admin
Actor Goal:	Post events, emergency alerts that are taking place at campus for students/staff
Participating Actors:	Admin, System
Preconditions:	Admin posts all necessary info on notice board
Post Conditions:	Info is updated on notice board and all users can view
Flow of Events for main Success Scenario:	→ Admin enters the app → Clicks Notice Board Icon ← System shows the Notice Board → Clicks "Add Post" ← System shows a text box → Admin types/adds info → Admin Clicks "Post" ← System then prompts "Post was successfully uploaded"
Flow of Events for Extensions:	← Admin selects Add notification bell icon → Admin then fills in the form with the notice ← Admin submits the form

<b>Use Case UC - 18</b>	<b>View Report</b>
Related Requirement:	Functional Requirement(s):
Initiating Actors:	Admin
Actor Goal:	Admin views report of the users
Participating Actors:	System, Admin

Preconditions:	Admin reviews all the reports made by the users
Post Conditions:	Admin takes info and investigates report
Flow of Events for main Success Scenario:	← Notification is sent to admin → Admin opens notification ← System opens report for admin to view → Admin views report and if checks if investigation is required
Flow of Events for Extensions:	← Admin select the view report icon → Displays all the reports added

## Effort Estimation Using Use Case Points

### Domain Analysis

#### Concept Definitions

##### UC-1: Register(AddUser)

Responsibility Description	Concept Name
Coordinates the actions associated with the user registration	Controller
Render the registration page for users	Page Maker
HTML form specifying the data required for user registration	Interface Page
Verifies and processes the user's registration data	Upload Request
Ensures all the necessary fields are filled out correctly	Upload checker
Manages the database interaction for storing user registration data	Database Connection

*Table 1: Concepts for Register*

##### UC-2: Login

<b>Responsibility Description</b>	<b>Concept Name</b>
Coordinates the actions of all concepts associated with the use case and delegates the work to other concepts.	Controller
Validates user credentials during the login process	Authentication
Renders the page that is specified after a successful login	Page Maker
Manages user sessions and authentication tokens	Sessions management
Establishes a connection with the database to verify user credentials	Database Connection
HTML form specifying the data required for user logging in	Interface Page

*Table 2: Concepts for Login*

UC-3 View Profile:

<b>Responsibility Description</b>	<b>Concept Name</b>
Coordinates the actions of all concepts associated with viewing user profiles	Controller
Renders the profile page for user	Page Maker
HTML document displaying user profile information	Interface Page
Retrieves and displays user-specific profile information	Profile Data
Manages the database interaction for retrieving profile data	Database Connection

*Table 3: Concepts for Viewing Profile*

UC-8 File Report:

<b>Responsibility Description</b>	<b>Concept Name</b>
Coordinates the actions and handles user interactions	Controller
Renders the report form interface	Report Form Renderer
Provides fields for users to input report details	Report Form
Validates the submitted report for completeness and accuracy	Report Validator
HTML document displaying user profile information	Interface Page
Processes and submits the reports to the appropriate authorities	Report Submission
Renders the profile page for user	Page Maker

*Table 4: Concepts for filing a report*

UC14- Log Report:

<b>Responsibility Description</b>	<b>Concept Name</b>
Coordinates the logging of reports and delegates tasks as necessary	Controller
Renders the form for security personnel to input report details	Report Form
Prepares and executes database queries to store the logged reports	Database connection

*Table 5: Concepts for logging a report*

UC17- Post to Notice Board:

<b>Responsibility Description</b>	<b>Concept Name</b>
Coordinates the posting of notices on the notice board and delegates tasks as necessary	Controller

Renders the form for authorized users to input notice details	Notice Board Form
Prepares and executes database queries to store the posted notices	Database connection

*Table 6: Concepts for posting on the notice board*

## Association Definitions

### UC-1: Register(AddUser)

Concept Pair	Association Description	Association Name
Controller ↔ Page Maker	The controller passes the request to Page Maker to render the registration page.	render registration page
Controller ↔ Upload Request	Controller forwards user registration data to Upload Request for verification and processing.	process registration data
Upload Checker ↔ Interface Page	Upload Checker validates the user input specified in the Interface Page.	validate user input
Upload Request ↔ Database Connection	Upload Request interacts with Database Connection to store the user's registration information.	store registration data

*Table 7: Association for Register*

### UC-2: Login

Concept Pair	Association Description	Association Name
User ↔ Authentication System	User provides login credentials to the Authentication System.	provide credentials

Authentication System ↔ User	Authentication System verifies user credentials and grants access.	verify credentials
------------------------------	--	--------------------

*Table 8: Association for Login*

UC-3: View Profile

<b>Concept Pair</b>	<b>Association Description</b>	<b>Association Name</b>
User ↔ Profile Database	User requests profile information from the Profile Database.	Request information
Profile Database ↔ User	Profile Database retrieves and provides profile information to the user.	Provide information

*Table 9: Association for View Profile*

UC-8: File a Report

<b>Concept Pair</b>	<b>Association Description</b>	<b>Association Name</b>
User ↔ Report Form	User submits a report using the Report Form	Submit Report
Report Form ↔ Security	Report Form forwards the report to the Security team	Forward report

*Table 10: Association for File a Report*

UC-14: Log Report

<b>Concept Pair</b>	<b>Association Description</b>	<b>Association Name</b>
User ↔ Report Logging System	User submits a report using the Report Logging System.	submit report



Report Logging System ↔ User	Report Logging System logs the report submitted by the user.	log report
------------------------------	--	------------

*Table 11: Association for Log Report*

UC-17: Post to Notice Board

Concept Pair	Association Description	Association Name
Admin ↔ Notice Board	Admin posts a notice to the Notice Board.	post notice
Notice Board ↔ User	Notice Board displays the posted notice to the user.	display notice

## Attribute Definitions

UC-1: Register(AddUser)

Concept	Attributes	Attribute Description
Upload Request	User Registration Data	User's registration data including first name, last name, email, address, phone number, and password.
Upload Checker	User Input	User input data to be validated.
Database Connection	User Registration Data	User registration data to be stored in the database.

*Table 13: Attribute for Register*

UC-2: Login

Concept	Attributes	Attribute Description
User	Login credentials	User's login credentials (username/password).

*Table 14: Attribute for Login*

UC-3: View Profile

Concept	Attributes	Attribute Description
---------	------------	-----------------------

Profile Database	Profile Information	User's profile information (e.g., name, contact details).
------------------	---------------------	---

*Table 15: Attribute for View Profile*

UC-8: File a Report

Concept	Attributes	Attribute Description
Report Form	Report Data	Data submitted in the report.

*Table 16: Attribute for File a Report*

UC-14: Log Report

Concept	Attributes	Attribute Description
Report Logging System	Report Data	Data submitted in the report.

*Table 17: Attribute for Log Report*

**Traceability Matrix**

		Domain Concepts										
Use Cases	PW	Upload Request	Upload Checker	Database Connection	Authentication	Sessions Management	Profile Data	Report Form Renderer	Report Form	Report Validator	Report Submission	Notice Board Form
UC-1		X	X	X								
UC-2				X	X	X						
UC-3						X						
UC-4				X			X					
UC-5				X	X		X					
UC-6				X								X
UC-7				X								X
UC-8		X	X	X								
UC-9								X	X	X	X	
UC-10				X								
UC-11				X								
UC-12				X								
UC-13				X								
UC-14				X								
UC-15				X				X	X	X	X	
UC-16				X								
UC-17				X								
UC-18				X								X
UC-19				X								
UC-20				X				X	X	X	X	

## System Operation Contracts

Contract Name:	Panic
Operation:	Panic
Cross Reference:	UI Component: Panic Button
Responsibilities:	<ul style="list-style-type: none"> <li>• Trigger an emergency alert</li> <li>• Log the panic event with timestamp and location</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is logged into the app</li> <li>• User has enabled location services</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• An emergency alert is sent</li> <li>• Panic event is recorded in the database</li> </ul>

Contract Name:	Add Report
Operation:	Add Report
Cross Reference:	UI Component: Report Submission Form
Responsibilities:	<ul style="list-style-type: none"> <li>• Allow users to submit detailed reports of incidents</li> <li>• Store report data in the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• Student or a Security user is authenticated</li> <li>• Report form is filled with required information</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Report is saved in the database</li> <li>• Confirmation message is shown to the user</li> </ul>

Contract Name:	Add Contact
Operation:	Add Contact
Cross Reference:	UI Component: Contact Management Form
Responsibilities:	<ul style="list-style-type: none"> <li>• Enable users to add emergency contacts</li> <li>• Store contact information in the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• Admin User is authenticated</li> <li>• Contact form is filled with required information</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Contact is added to the user's contact list by an admin personnel</li> <li>• Contact information is stored in the database</li> </ul>

Contract Name:	View Contact
Operation:	View Contact
Cross Reference:	UI Component: Contact List
Responsibilities:	<ul style="list-style-type: none"> <li>• Display the list of emergency contacts for the user added by an admin personnel</li> <li>• Retrieve contact data from the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Contacts are displayed to the user</li> <li>• Data is fetched from the database and shown in the UI</li> </ul>

Contract Name:	Add Work Log
Operation:	Add Work Log
Cross Reference:	UI Component: Work Log Submission Form
Responsibilities:	<ul style="list-style-type: none"> <li>• Allow security personnel to log their working hours</li> <li>• Store work log data in the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated as security personnel</li> <li>• Work log form is filled with required information</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Work log is saved in the database</li> <li>• Confirmation message is shown to the user</li> </ul>

Contract Name:	View Work Log
Operation:	View Work Log
Cross Reference:	UI Component: Work Log List
Responsibilities:	<ul style="list-style-type: none"> <li>• Display the list of work logs submitted by a security guard personnel</li> <li>• Retrieve work log data from the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated as an admin personnel</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Work logs are displayed to the admin user</li> <li>• Data is fetched from the database and shown in the UI</li> <li>•</li> </ul>

Contract Name:	Map View
Operation:	Map View
Cross Reference:	UI Component: Map Interface
Responsibilities:	<ul style="list-style-type: none"> <li>• Display a map with real-time locations of users that is in danger</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated</li> <li>• Location services are enabled</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Map is displayed with current data</li> <li>• Location updates are shown when the user uses the panic button</li> </ul>

Contract Name:	View Report
Operation:	View Report
Cross Reference:	UI Component: View Report
Responsibilities:	<ul style="list-style-type: none"> <li>• Display a list of incident reports</li> <li>• Retrieve report data from the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Reports are displayed to the admin and guard</li> <li>• Data is fetched from the database and shown in the UI</li> </ul>

Contract Name:	View Profile
Operation:	View Profile
Cross Reference:	UI Component: User Profile Page
Responsibilities:	<ul style="list-style-type: none"> <li>• Display user's profile information</li> <li>• Retrieve profile data from the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• User is authenticated</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Profile information is displayed to the user</li> <li>• Data is fetched from the database and shown in the UI</li> </ul>

Contract Name:	Add User
Operation:	Add User
Cross Reference:	UI Component: User Registration Form
Responsibilities:	<ul style="list-style-type: none"> <li>• Enable new users to register for the app</li> <li>• Store user data in the database</li> </ul>
Preconditions	<ul style="list-style-type: none"> <li>• Registration form is filled with required information</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• New user is added to the database</li> <li>• User can log in with their credentials</li> </ul>



# Mathematical Models / Algorithms

## Assumptions and Notations

### 1. Entities and Attributes:

- **User (U)**: Represents users of the app.
  - $U = \{u_1, u_2, \dots, u_n\}$
- **Panic Alert (P)**: Represents panic alerts triggered by users.
  - $P = \{p_1, p_2, \dots, p_m\}$
- **Report (R)**: Represents reports created by users.
  - $R = \{r_1, r_2, \dots, r_k\}$
- **Contact (C)**: Represents contacts added by users.
  - $C = \{c_1, c_2, \dots, c_l\}$
- **Work Log (W)**: Represents work logs entered by users.
  - $W = \{w_1, w_2, \dots, w_j\}$
- **Map View (M)**: Represents map data viewed by users.
- **Profile (Pf)**: Represents user profiles.
- **User Log (UL)**: Represents user logs.

### 2. Functions and Operations:

- **Panic Alert Function (FPFP)**:
  - $FP: U \rightarrow P$
  - Description:  $FP(ui) = p_x$  triggers a panic alert  $p_x$  by user  $ui$ .
- **Add Report Function (FRFR)**:
  - $FR: U \times \text{Report Data} \rightarrow R$
  - Description:  $FR(ui, data) = r_y$  adds a new report  $r_y$  with given data by user  $ui$ .
- **Add Contact Function (FCFC)**:
  - $FC: U \times \text{Contact Data} \rightarrow C$
  - Description:  $FC(ui, data) = c_z$  adds a new contact  $c_z$  with given data by user  $ui$ .
- **View Contact Function (VCVC)**:
  - $VC: U \rightarrow \{C\}$
  - Description:  $VC(ui) = \{c_1, c_2, \dots, c_m\}$  retrieves all contacts for user  $ui$ .

- **Add Work Log Function (FWFW):**
  - $FW:U \times \text{Work Log Data} \rightarrow WFW:U \times \text{Work Log Data} \rightarrow W$
  - Description:  $FW(ui, data) = waFW(ui, data) = wa$  adds a new work log  $wawa$  with given data by user  $uiui$ .
- **View Work Log Function (VWVW):**
  - $VW:U \rightarrow \{W\} VW:U \rightarrow \{W\}$
  - Description:  $VW(ui) = \{w1, w2, \dots, wb\} VW(ui) = \{w1, w2, \dots, wb\}$  retrieves all work logs for user  $uiui$ .
- **Map View Function (FMFM):**
  - $FM:U \rightarrow MFM:U \rightarrow M$
  - Description:  $FM(ui) = mFM(ui) = m$  retrieves map data  $mm$  for user  $uiui$ .
- **View Report Function (VRVR):**
  - $VR:U \rightarrow \{R\} VR:U \rightarrow \{R\}$
  - Description:  $VR(ui) = \{r1, r2, \dots, rn\} VR(ui) = \{r1, r2, \dots, rn\}$  retrieves all reports for user  $uiui$ .
- **View Profile Function (VPfVPf):**
  - $VPf:U \rightarrow PfVPf:U \rightarrow Pf$
  - Description:  $VPf(ui) = pfVPf(ui) = pf$  retrieves profile data  $pfpf$  for user  $uiui$ .
- **Add User Function (FUFU):**
  - $FU:\text{User Data} \rightarrow UFU:\text{User Data} \rightarrow U$
  - Description:  $FU(data) = un+1FU(data) = un+1$  adds a new user  $un+1un+1$  with given data.

## Detailed Model

### 1. Panic Alert Model:

- Trigger: When a user  $uiui$  presses the panic button.
- Outcome: Generate a panic alert  $pxpx$  and notify security personnel.
- Mathematical Representation:  
 $px = FP(ui)px = FP(ui)$
- Constraints: A user can trigger multiple panic alerts.

### 2. Report Management:

- Adding a Report:  
 $ry = FR(ui, data)ry = FR(ui, data)$

- Viewing Reports:  
 $\{r1, r2, \dots, rn\} = VR(ui) \{r1, r2, \dots, rn\} = VR(ui)$
- 3. **Contact Management:**
  - Adding a Contact:  
 $cz = FC(ui, data) cz = FC(ui, data)$
  - Viewing Contacts:  
 $\{c1, c2, \dots, cm\} = VC(ui) \{c1, c2, \dots, cm\} = VC(ui)$
- 4. **Work Log Management:**
  - Adding a Work Log:  
 $wa = FW(ui, data) wa = FW(ui, data)$
  - Viewing Work Logs:  
 $\{w1, w2, \dots, wb\} = VW(ui) \{w1, w2, \dots, wb\} = VW(ui)$
- 5. **Map View:**
  - Accessing Map View:  
 $m = FM(ui) m = FM(ui)$
- 6. **Profile Management:**
  - Viewing Profile:  
 $pf = VPf(ui) pf = VPf(ui)$
- 7. **User Management:**
  - Adding a New User:  
 $un+1 = FU(data) un+1 = FU(data)$

## Interaction Diagrams

*Figure 1: Sequence Diagram for Register(UC-1)*

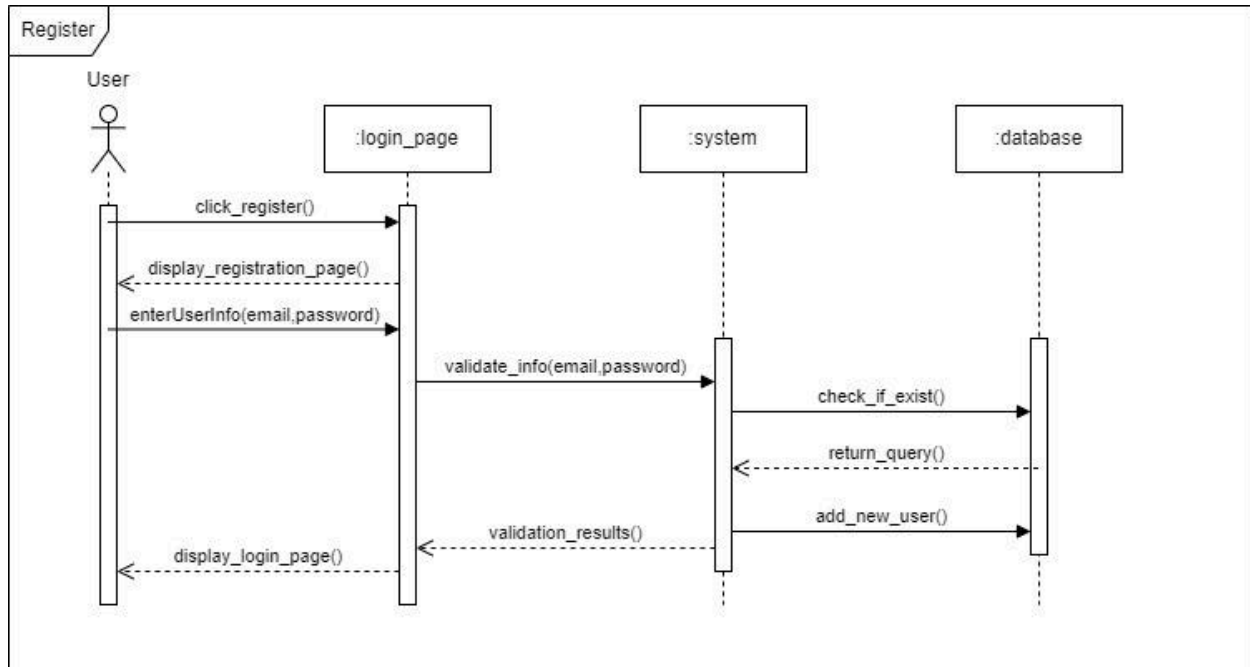


Figure 2: Sequence Diagram for Login(UC-2)

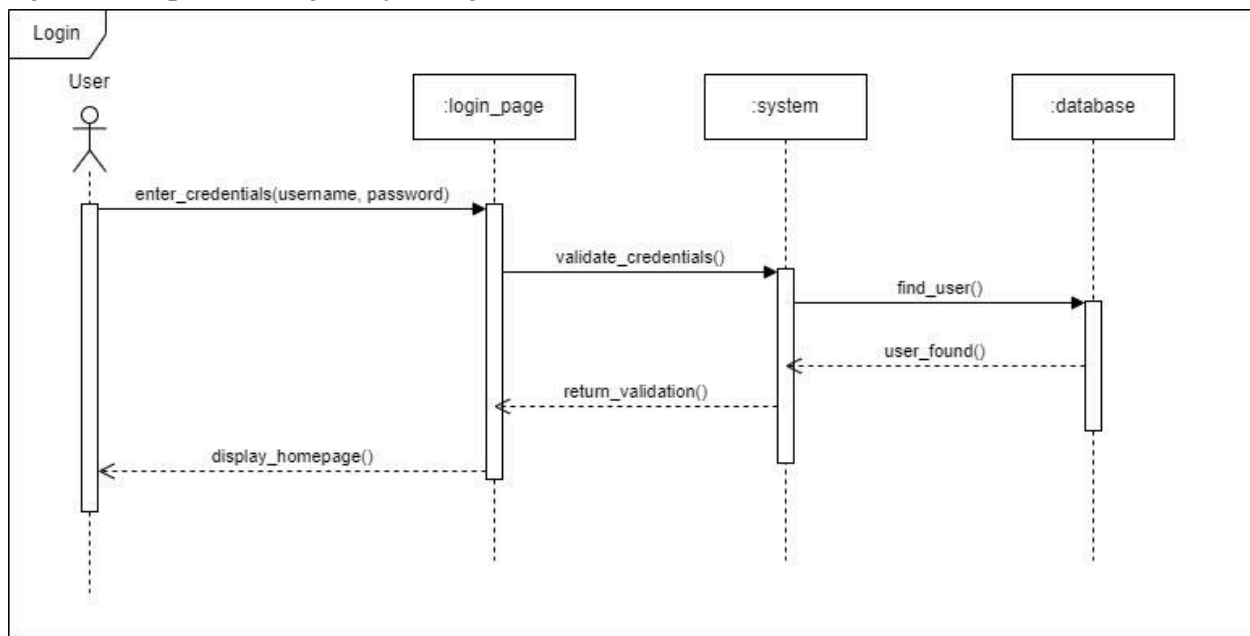


Figure 3: Sequence Diagram for View Profile(UC-3)

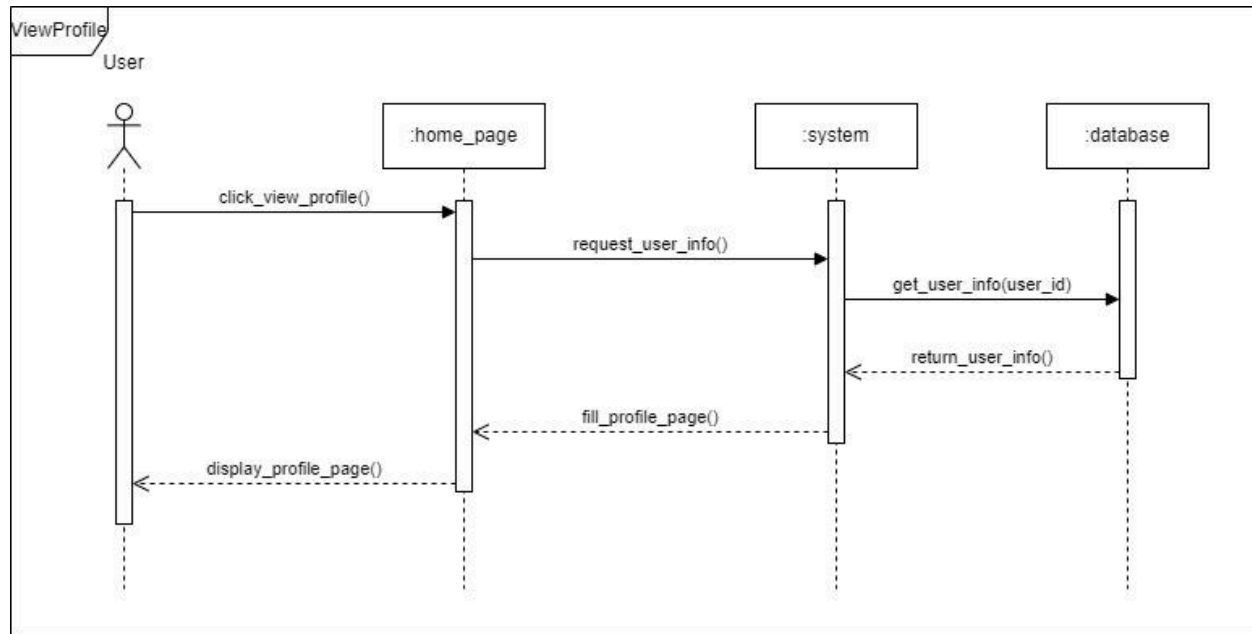


Figure 4: Sequence Diagram for File a Report(UC-8)

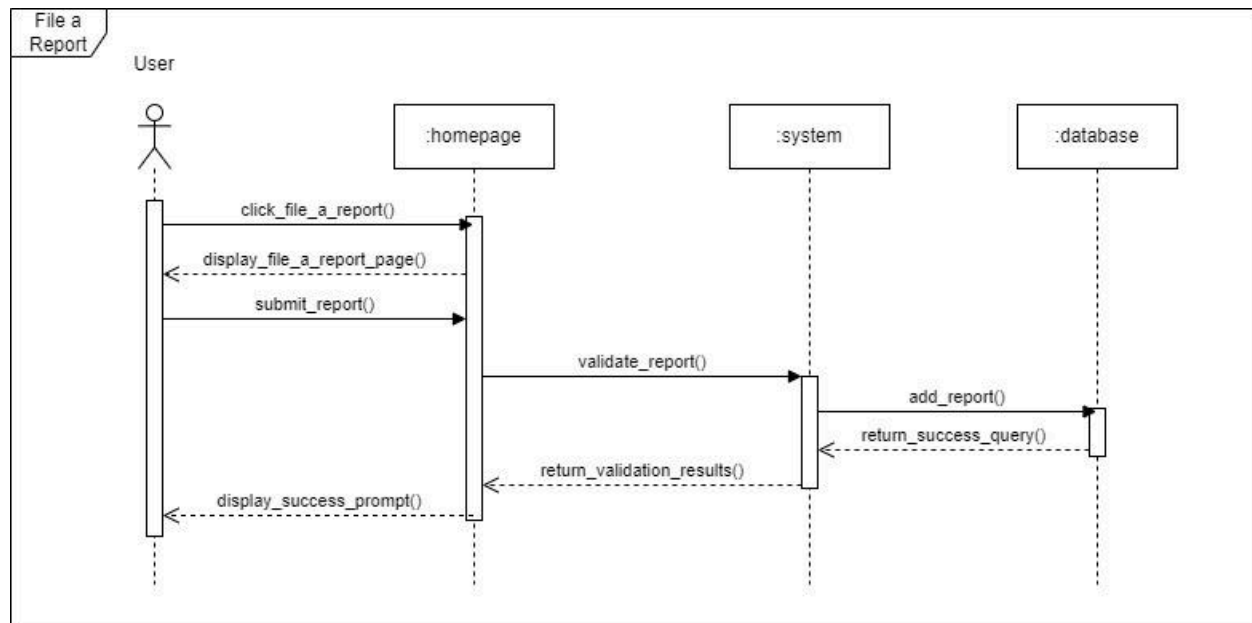


Figure 5: Sequence Diagram for Log Report(UC-14)

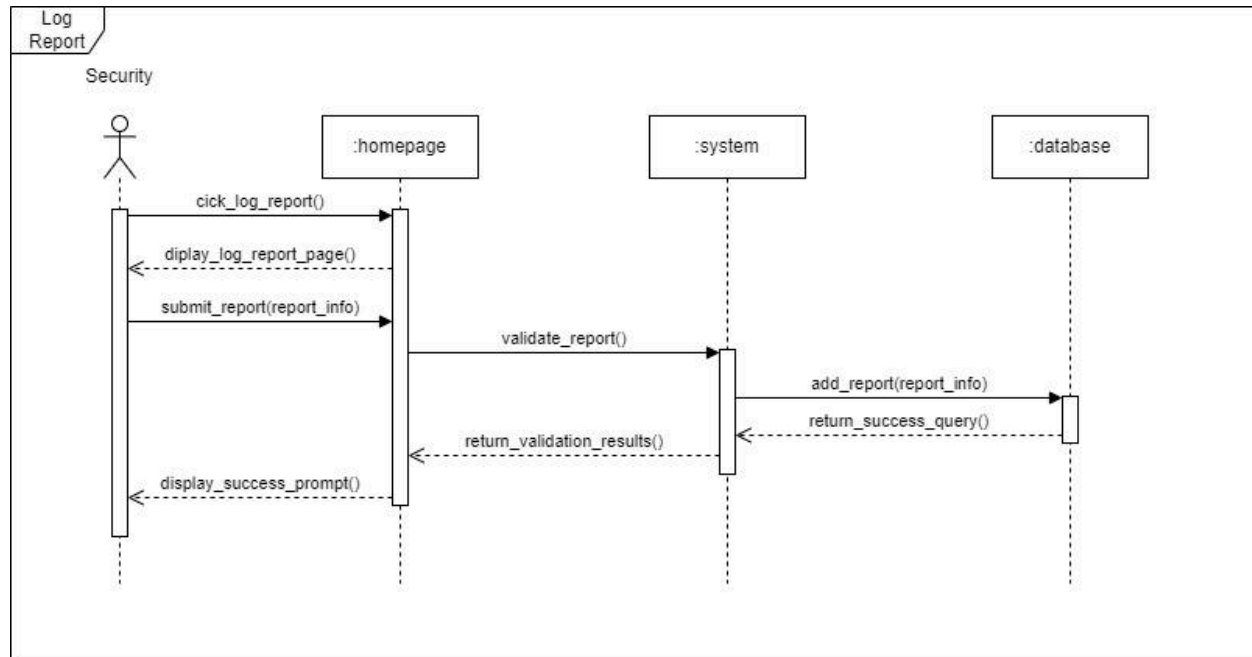
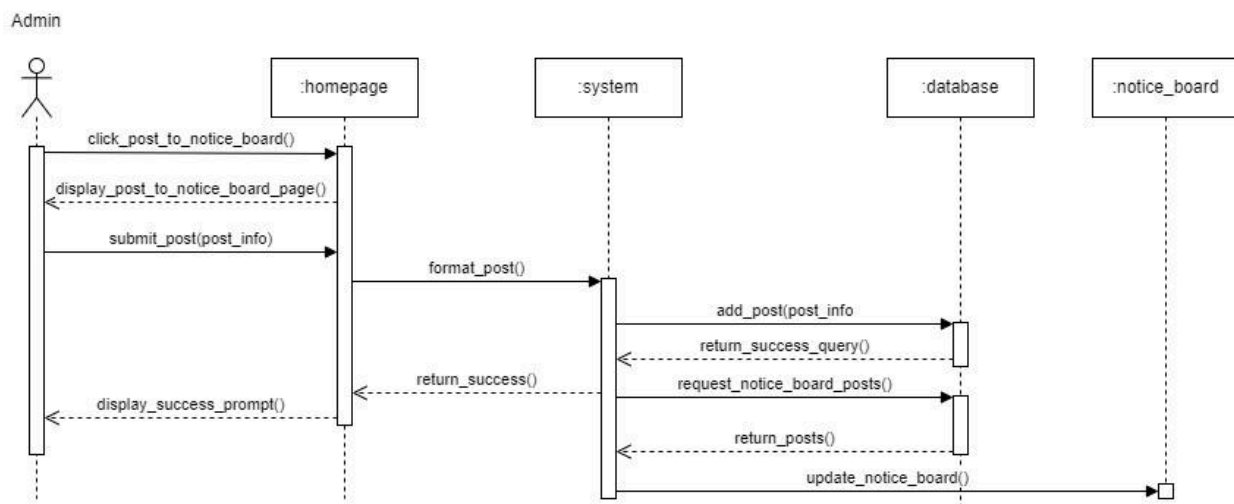
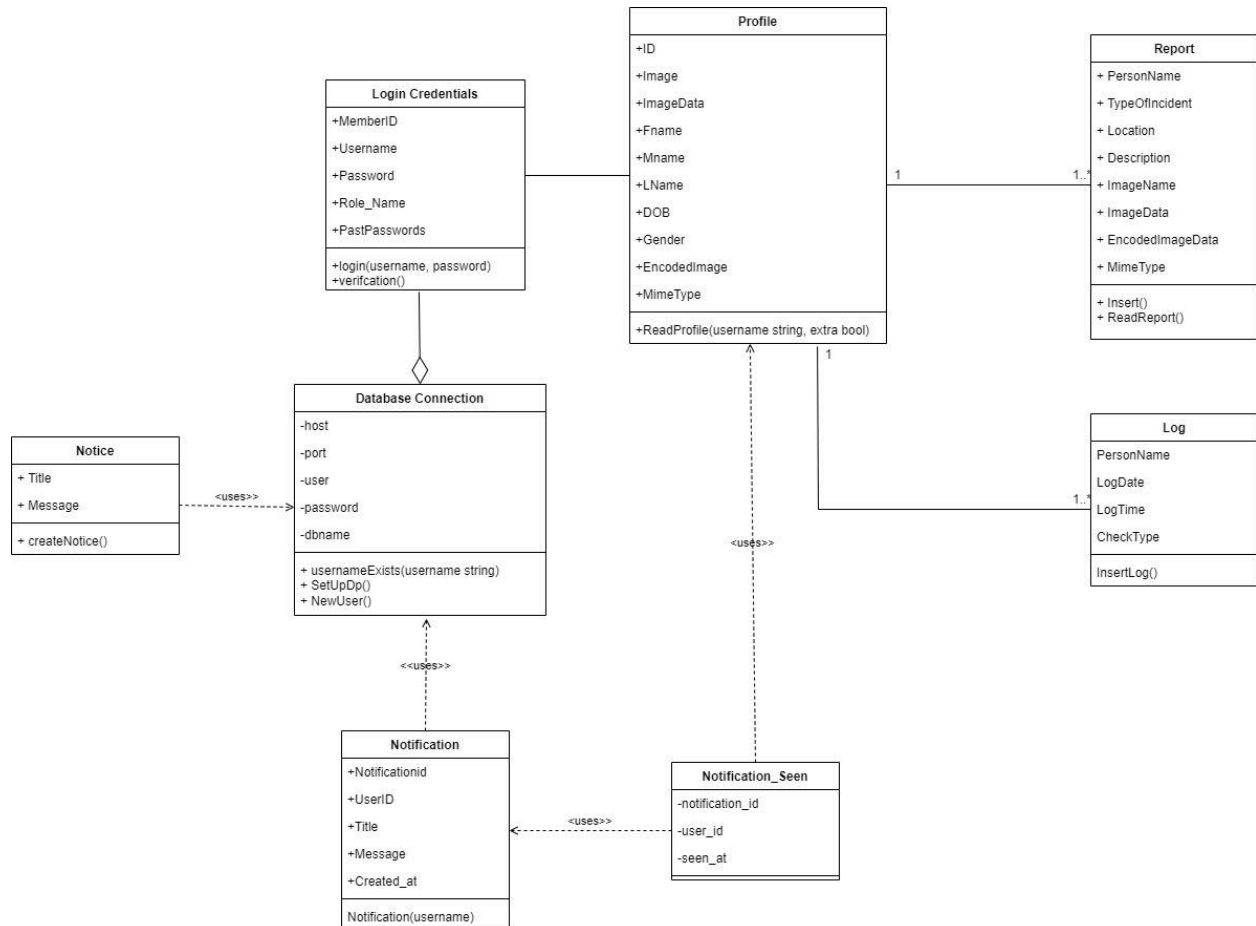


Figure 6: Sequence Diagram for Post to Notice Board(UC-17)



# Class Diagram and Interface Specification

## Class Diagram



## Data Types and Operation Signature

Login Credentials	
MemberID	int
Username	string
Password	string
Role_Name	int
PastPasswords	string
login(username, password) void verification() void	

Report	
PersonName	string
TypeOfIncident	string
Location	string
Description	string
ImageName	string
ImageData	[]byte
EncodedImageData	string
MimeType	string
Insert() int, error ReadReport() void	

Profile	
ID	int
Image	string
ImageData	[]byte
Fname	string
Mname	string
LName	string
DOB	time.Time
Gender	string
EncodedImage	string
MimeType	string
+ReadProfile(username string, extra bool) []struct, error	

Log	
PersonName	string
LogDate	time.Time
LogTime	time.Time
CheckType	string
InsertLog() int, error	

Notification	
Notificationid	int
UserID	int
Title	string
Message	string
Created_at	time.Time
Notification(username) string, error	

Notification_Seen	
notification_id	int
user_id	int
seen_at	time.Time

Notice	
Title	string
Message	string
InsertNotice() int, error	

Database Connection	
host	string
port	int
user	string
password	string
dbname	string
+ usernameExists(username string) bool, error + SetUpDp() void + NewUser() int, error	

## Traceability Matrix

	Login Credentials	Report	Profile	Log	Notification	Notification_Seen	Notice	Database Controller
Controller								
Page Maker								
Interface Page								
Upload Request								
Upload Checker								
Database Connection								
Authentication								
Sessions Management								



Profile Data								
Report Form Renderer								
Report Form								
Report Validator								
Report Submission								
Notice Board Form								

**Login Credentials:**

- Deals with login data from the login page.
- Successful login creates session
- Requires profile Data and database connection.
- Required page to be rendered

**Reports:**

- Evolved from Report Validator as it deals with validating and formatting reports
- Requires page to be rendered and a successful database connection
- Concerned with primary actor student

**Profile:**

- Evolved from Profile Data as it deals with displaying data for user
- Concerned with all actors
- Requires database connection

**Log:**

- Concerned with primary actor Security
- Deals with daily report submitted by actor
- Unlike the report from the student actors, does not use upload checker

**Notifications:**

- Deals with managing and formatting notifications created by system admins

**Notification\_Seen:**

- Used to manage users who have seen the notification

**Notice:**

- A very simple class used to manage the title and message from notices created by the system admin to output to the notice board

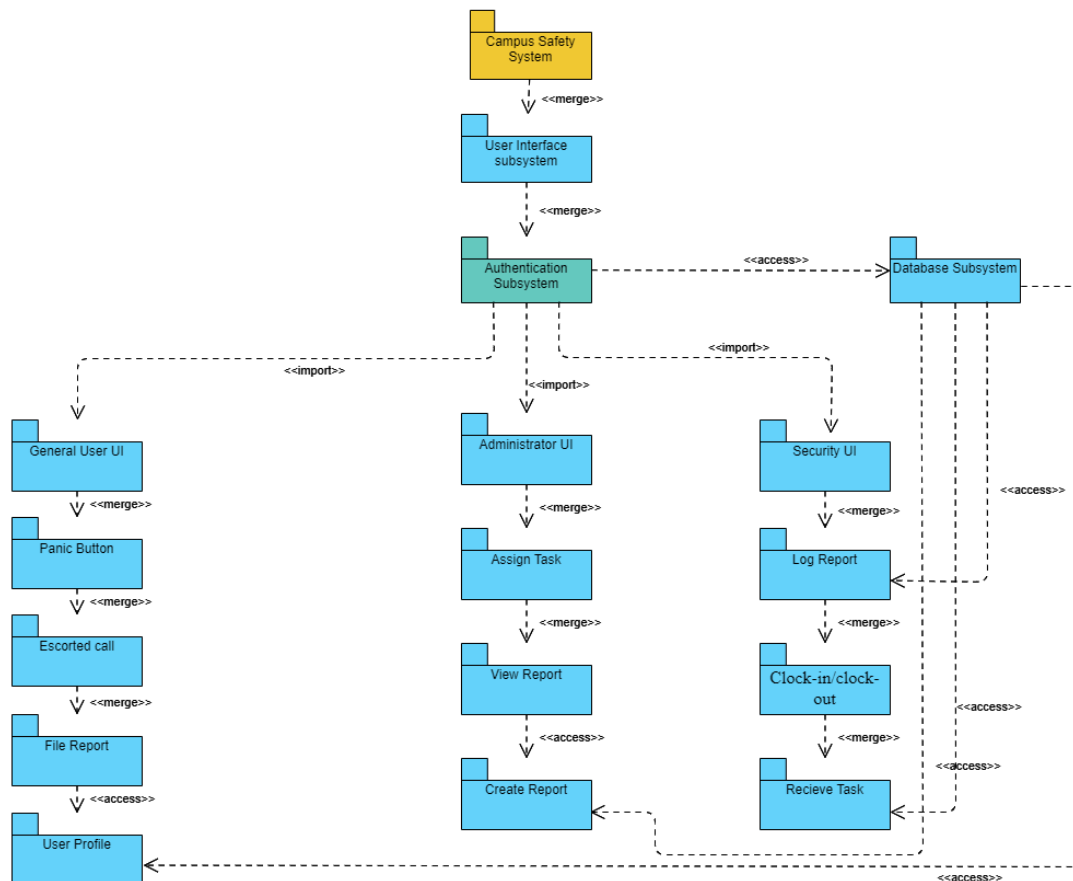
**Database Controller:**

- Main database controller, used to serve as a controller for all queries. Evolved from Database Connection

**OCL Contract Specifications**

# System Architecture and System Design

## Identifying Subsystems



## Architecture Style

**This campus safety system will implement three architecture styles: Client Server Architecture, Microservices Architecture, and Event-Driven Architecture.**

1. Client-Server Architecture: With this design, the application will be split into two main parts: the server, which handles the business logic and stores and manages data, and the client, which runs on the user's device. When the client asks the server for information or commands, the server answers appropriately. The program is simpler to update and maintain because of this architecture, which enables the concentration of logic and data. By adding new servers as the user base expands, it also permits scalability. The UB campus safety app's client exposes this data to users in an intuitive

interface, while the server stores details about the campus's amenities, activities, academic resources, and user profiles.

2. **Microservices Architecture:** A microservices architecture consists of building an application as a group of loosely linked services, each of which is in charge of handling a particular function. Greater flexibility and agility are made possible by the ability for each service to be designed, implemented, and scaled independently. Different microservices can manage features like course registration, campus maps, event scheduling, library resources, and so on for a university campus safety app. Complex applications, such as university campus safety apps, might benefit from microservices architecture's ability to promote modularity and accelerate development cycles.

3. **Event-Driven Architecture:** In an event-driven architecture, components of the system communicate with each other by generating and responding to events. Events can be triggered by user actions, system events, or external inputs, and they are handled asynchronously. This architecture promotes loose coupling and scalability, as components can react to events without direct dependencies on each other. For a university campus safety app, event-driven architecture can be utilized for functionalities such as real-time notifications for upcoming events, panic alerts, or changes in campus services.

## **Mapping Subsystem to Hardware**

### **1. Server Infrastructure:**

- **Hardware:** High-performance servers with multi-core processors and sufficient RAM to handle concurrent user requests.
- **Functionality:** Responsible for storing and managing campus maps, user data, and real-time location tracking information.
- **Requirements:** Reliable network connectivity, fault-tolerant architecture, and scalable storage solutions.
- **Components:** Load balancers, database servers, and backup systems.

### **2. Geolocation Hardware:**

- **Smartphones:** Utilize GPS, Wi-Fi, and cellular networks for accurate location tracking of users.
- **Beacons:** Install Bluetooth Low Energy (BLE) beacons at key locations across the campus for precise indoor positioning.
- **Requirements:** Compatibility with the campus safety app, low power consumption, and minimal maintenance.
- **- Integration:** APIs to interface with the mapping subsystem and provide real-time location data.

### **3. User Devices:**

- Smartphones/Tablets: Platforms for accessing the campus safety app and receiving real-time alerts and notifications.
- Requirements: Compatibility with major operating systems (iOS, Android), sufficient processing power, and network connectivity.

#### 4. Networking Equipment:

- Wi-Fi Access Points: Ensure ubiquitous coverage across the campus for seamless connectivity.
- Ethernet Switches/Routers: Backend infrastructure for routing data between servers, devices, and external networks.
- Requirements: High bandwidth, low latency, and support for secure communication protocols (e.g., HTTPS, SSL/TLS).
- Security: Implementation of firewalls, intrusion detection/prevention systems, and network segmentation to safeguard data.

#### 5. Backend APIs and Services:

- API Gateway: Centralized interface for accessing backend services, ensuring secure and efficient communication.
- Microservices Architecture: Modular design for scalability, allowing independent development and deployment of services.
- Authentication/Authorization: Implement robust authentication mechanisms to safeguard user data and access control policies.

#### 6. Integration with External Systems:

- Emergency Services: Interface with local emergency services (police, fire department, medical services) for seamless coordination during emergencies.
- Campus Facilities Management: Integration with facilities management systems for real-time updates on building access, maintenance activities, and infrastructure status.

## Connectors and Network Protocols

### Connectors:

1. Wi-Fi: Allows devices to connect to the university's wireless network for communication.
2. Bluetooth: Enables short-range communication between devices, useful for features like proximity detection and device pairing.
3. GPS (Global Positioning System): Provides location information for devices, allowing for features like real-time tracking and geofencing.
4. Cellular network: Enables communication via mobile networks, ensuring connectivity even when Wi-Fi is unavailable.

5. Ethernet: For wired connections, typically used for more stable and secure connections in specific areas like offices or security stations.

### **Network Protocols:**

1. HTTP/HTTPS (Hypertext Transfer Protocol/Secure): For communication between the app and web servers, used for accessing campus safety information, updates, and other online resources securely.
2. TCP/IP (Transmission Control Protocol/Internet Protocol): Fundamental protocols for transmitting data across networks, ensuring reliable communication between devices.
3. UDP (User Datagram Protocol): Provides faster communication suitable for real-time data transmission, such as live video streaming or voice calls.
4. MQTT (Message Queuing Telemetry Transport): Lightweight messaging protocol suitable for IoT devices and sensors, allowing for efficient communication between devices on the network.
5. SNMP (Simple Network Management Protocol): Used for monitoring and managing network devices, helpful for ensuring the reliability and security of network infrastructure supporting the safety app.
6. SIP (Session Initiation Protocol): Used for initiating and terminating multimedia communication sessions, potentially useful for features like emergency calls or video conferencing within the app.
7. LDAP (Lightweight Directory Access Protocol): For accessing directory information, such as user authentication and authorization data, which could be relevant for user management within the safety app.

### **Global Control Flow**

1. Initialization and Authentication: Upon launching the university campus safety app, the user is prompted to authenticate their identity, typically through a login screen. Authentication may involve entering a username and password or utilizing biometric authentication methods such as fingerprint or facial recognition.
2. Main Menu Navigation: After successful authentication, the user is directed to the main menu interface of the app. Here, they are presented with various options and functionalities tailored to campus safety, such as reporting incidents, accessing emergency contacts, requesting assistance, or exploring safety tips and resources.
3. Incident Reporting: One of the core functionalities of the app is the ability for users to report incidents or concerns regarding safety on campus. This can include incidents such as accidents, suspicious activities, or emergencies. Upon selecting the incident reporting feature, the user is guided through a form where they can provide details such

as the type of incident, location, and any additional information necessary for responders.

4. **Emergency Assistance Request:** In case of emergencies, users can quickly request assistance through the app. This feature typically involves a prominent emergency button on the main menu, which, when activated, immediately alerts campus security or emergency services to the user's location and situation.

5. **Notification and Alert System:** The app includes a notification and alert system to disseminate important safety information to users in real-time. This can include alerts about weather emergencies, campus closures, or security threats. Users may receive these notifications through push notifications, SMS messages, or email, depending on their preferences and the severity of the situation.

6. **Safety Resources and Information:** Additionally, the app serves as a hub for safety resources and information, providing users with access to safety tips, campus maps highlighting safe zones and emergency exits, contact information for campus security personnel and support services, as well as educational materials on topics such as personal safety and emergency preparedness.

7. **Logging Out and Session Management:** Finally, users can log out of the app to end their session securely, especially if accessing the app from shared or public devices. Proper session management is crucial to protect user privacy and ensure that sensitive information is not accessible to unauthorized individuals.

## **Hardware Requirements**

### **Mobile Requirements:**

- OS: Android 8.0 or later; iOS 12.0 or later
- Storage: Minimum of 2GB available space
- Screen Resolution: Minimum screen resolution of 640 x 480
- Network Bandwidth: Stable internet connection with a minimum bandwidth of 60 kbps

### **PC Requirements:**

- OS: Any operating system compatible with the following web browsers:
- Chrome: Version 70 or later
- Firefox: Version 70 or later
- Safari: Version 12 or later
- Opera: Version 60 or later
- Network: Broadband internet connection is required or Physical LAN connection

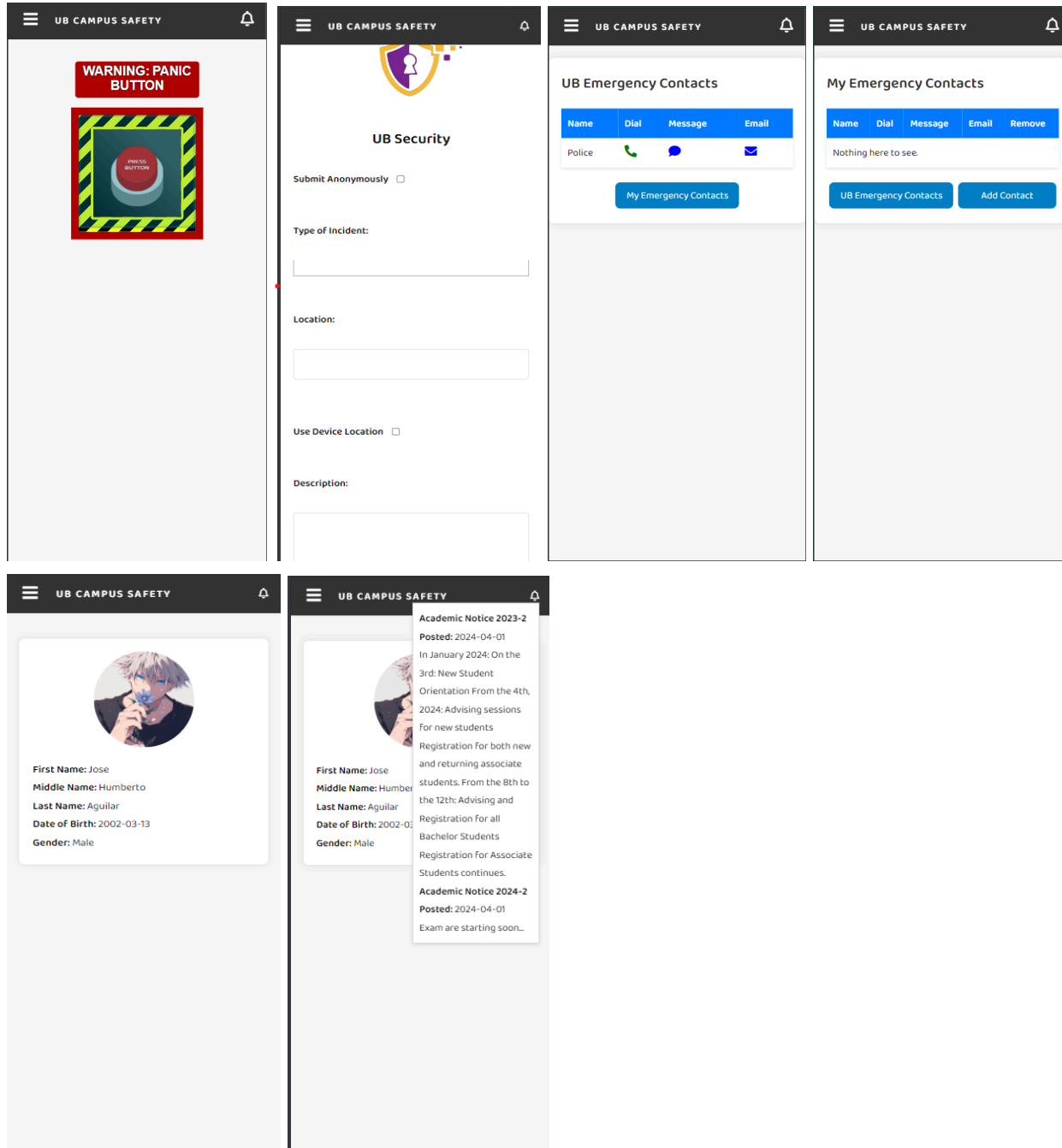
# Algorithms and Data Structures

## The UB Campus Security application will utilize:

1. **Hash Tables:** In the UB campus safety application, the concept of hash tables can be applied through database indexing. For instance, indexing on critical columns like username in the LOGIN table facilitates swift retrieval of user login details. This indexing mechanism enhances the efficiency of data retrieval processes within the application. Similar indexing techniques can be employed across other tables, such as notification and Report, optimizing data retrieval performance. The utilization of indexing serves a comparable purpose by enabling rapid data lookup based on specific keys, thereby enhancing the overall efficiency and responsiveness of the application.
2. **Arrays:** In the JavaScript files for the UB campus safety application, arrays play a crucial role in managing various elements and event listeners within the user interface. Specifically, arrays are utilized to store references to DOM elements representing list items (li\_items) and facilitate the handling of mouse hover and mouse leave events for each list item. Additionally, arrays are employed to store references to essential HTML elements such as the hamburger menu and the wrapper element. By utilizing arrays, the application efficiently manages and manipulates multiple elements of the same type, enhancing the overall user experience and interactivity of the interface.
3. **Criteria used to choose the Data Structure:**
  - Efficiency and speed of data retrieval: Arrays are chosen for fast and easy traversal.
  - Security: Hash tables are used to store sensitive information securely.
  - Scalability: Arrays and hash tables can be easily scaled up to handle a large amount of data and users.
  - Flexibility: JS is used for data exchange due to its flexibility and lightweight nature.

# User Interface Design and Implementation

## Student



These diagrams demonstrate the Home look of the Student's interface then when the student clicks on the hamburger menu icon it will show the option to select either profile or to add a report and also if the user clicks on the notification bell, it will display the notice that the administrator has added.



## Security Guard

### Work Log Form

Select Date:

05/16/2024

Select Time:

01:45 PM

Select Check Type:

Check In

Submit

### UB Security

Submit Anonymously ☐

Type of Incident:

Location:

Use Device Location ☐

Description:

### Emergency Call in Request

Name: Alex Peraza

Location: 17501359, -88.214837

PHASE-2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

Name: Raynisha Cornelio

Location: 17501366, -88.214823

PHASE-2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

Name: Raynisha Cornelio

Location: 17501420, -88.214763

PHASE-2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

### Reports

Person Name: Anonymous

Type of Incident: Vandalism

Location: 175045661, -88.1962133

Description: not sure

Image:

Person Name: Jose Aguilar

Type of Incident: Vandalism

Location: 175045661, -88.1962133

Description: The graffiti, a kaleidoscope of colors and shapes, seemed to scream for attention, demanding recognition in a world often blinded by routine. From the bold, sweeping curves of vibrant red to the intricate, labyrinthine patterns of deep indigo, each mark carried with it a message—a cry for expression, a rebellion against the mundane.

Image: No image available

Person Name: Anonymous

Type of Incident: Assault

### Security Guard Profile

First Name: John

Middle Name: Aguilar

Last Name: Sanchez

Date of Birth: 1987-06-16

Gender: Male

The following diagrams show the Security guard panel. When the user logs in, they will be greeted with the check-in / check-out form so that they can submit. When the user has done that, they can access the other panels by clicking on the hamburger menu and selecting which panel they want to visit.

## Administrator

UB CAMPUS SAFETY

Add Notice

Title:

Message:

Submit

UB CAMPUS SAFETY

Add Emergency Contacts

Emergency Contact Name:

Phone Number:

Email:

Submit

UB CAMPUS SAFETY

Emergency Call in Request

Name: Alex Peraza

Location: 17.501359,-88.214837

PHASE: 2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

Name: Raynisha Cornelio

Location: 17.501366,-88.214823

PHASE: 2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

Name: Raynisha Cornelio

Location: 17.501420,-88.214763

PHASE: 2 BELAMA

17°30'04.9"N 88°12'...

Ampliar el mapa

UB CAMPUS SAFETY


Reports

Person Name: Anonymous

Type of Incident: Vandalism

Location: 17.5045661,-88.1962133

Description: not sure

Image: 

Person Name: Jose Aguilar

Type of Incident: Vandalism

Location: 17.5045661,-88.1962133

Description: The graffiti, a kaleidoscope of colors and shapes, seemed to scream for attention, demanding recognition in a world often blinded by routine. From the bold, sweeping curves of vibrant red to the intricate, labyrinthine patterns of deep indigo, each mark carried with it a message—a cry for expression, a rebellion against the mundane.

Image: No image available

Person Name: Anonymous

Type of Incident: Assault

UB CAMPUS SAFETY

Check in / check out Log's

Name: John Sanchez

Date: 2024-03-31

Time: 6:08 PM

Status: checkIn


Name: John Sanchez

Date: 2024-03-31

Time: 6:11 PM

Status: checkOut

UB CAMPUS SAFETY



First Name: Alex

Middle Name: Humberto

Last Name: Peraza

Date of Birth: 2003-03-12

Gender: Male

The following diagrams show the admin panel. When the user logs in, they will be greeted with the notice form, if the administrator has something important to say or remind the user about any events the administrator can submit a notice form.. If the user chooses not to submit a notice, they can access the other panels by clicking on the hamburger menu and selecting which

panel they want to visit from what they wish to do, like registering a new user, viewing a report or log and checking their profile.

## Design of Tests

### Overview of Design Tests

The functionalities used in the test cases were essential elements that were directly related to the overall functioning of the system and our priority needs. It's crucial to remember that it is not possible to automate regression testing for all aspects of a project. It is very difficult, if not impossible, to build an automated script due to its intricacy and the vast number of alternative circumstances. Also, creating such a script would probably take more time and effort than it would save. For now, humans are still better suited for this activity because they are mentally capable to evaluate, react to, and calculate elements that robots are not yet able to handle. We are planning to evaluate our six primary use cases through testing.

#### UC-#1: Register(AddUser)

<b>Test Case</b>	<b>TC-1</b>
<b>Use Case being tested</b>	<b>Register</b>
<b>Criteria for Success/Fail</b>	<b>Tests that a user can be successfully added to the Database and create default Credentials</b>
<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>
<b>Step 2: Use data models with mock Database</b>	<b>Create an instance of the our data models with mock database connection and define sample input data</b>
<b>Step 3: create mock of helper function (usernameExist)</b>	<b>Create a mock which will return false(username does not exist) so that there are no repetition on usernames</b>

<b>Step 4: Define Expected results</b>	<b>Create definitions for Expected queries to our mock database and definitions of their return values to both tables, user and login credentials tables.</b>
<b>Step 5: Call Function</b>	<b>We call the NewUser Function With Sample data which was previously defined and print errors in any Otherwise move to next step</b>
<b>Step 6: Check Return Value</b>	<b>Check if Function Returned previously defined Expected ID (1), if not, fail test</b>
<b>Step 7: Check for other database Expectation</b>	<b>Verify that Database did not had more Queries/Inserts than previously defined</b>

### UC-2: Login

<b>Test Case</b>	<b>TC-2</b>
<b>Use Case being tested</b>	<b>Login</b>
<b>Criteria for Success/Fail</b>	<b>Test that a registered user can login with their correct credentials</b>
<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>
<b>Step 2: Create mock http &amp; call Function</b>	<b>Create Mock request to simulate method POST And a response Recorder and pass to verification function</b>
<b>Step 3: redirect</b>	<b>If step 2 Successful, pass test</b>

### UC-3: View Profile

<b>Test Case</b>	<b>TC-3</b>
------------------	-------------

<b>Use Case being tested</b>	<b>ViewProfile</b>
<b>Criteria for Success/Fail</b>	<b>Test that a user can view their profile details</b>
<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>
<b>Step 2: Use data models with mock Database</b>	<b>Create an instance of the our data models with mock database connection and define sample input data</b>
<b>Step 3: Define Expected Queries</b>	<b>Create definitions for Expected queries to our mock database</b>
<b>Step 4: Call Function</b>	<b>We call the ReadProfile Function With Sample data which was previously defined and print errors in any Otherwise move to next step</b>
<b>Step 5: Define Expected Return Value</b>	<b>Define Expected Return Value on our models</b>
<b>Step 6: Compare Results</b>	<b>Compare to see if Expected results are the same as the actual returned results</b>
<b>Step 7: Check for other database Expectation</b>	<b>Verify that Database did not had more Queries/Inserts than previously defined</b>

#### UC-8 File a Report

<b>Test Case</b>	<b>TC-4</b>
<b>Use Case being tested</b>	<b>FileAReport</b>
<b>Criteria for Success/Fail</b>	<b>Test if a user can generate and submit a report and if it was posted to the database</b>

<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>
<b>Step 2: Use data models with mock Database</b>	<b>Create an instance of the our data models with mock database connection and define sample input data</b>
<b>Step 3: Define Expected results</b>	<b>Create definitions for Expected queries to our mock database</b>
<b>Step 4: Call Function</b>	<b>We call the Insert Function With Sample data which was previously defined and print errors in any Otherwise move to next step</b>
<b>Step 5: Check Return Value</b>	<b>Check if Function Returned previously defined Expected ID (1), if not, fail test</b>
<b>Step 6: Check for other database Expectation</b>	<b>Verify that Database did not had more Queries/Inserts than previously defined</b>

#### UC-14 Log Report

<b>Test Case</b>	<b>TC-5</b>
<b>Use Case being tested</b>	<b>LogReport</b>
<b>Criteria for Success/Fail</b>	<b>Test that security guards can log information and if it was posted to the database</b>
<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>

<b>Step 2: Use data models with mock Database</b>	<b>Create an instance of the our data models with mock database connection and define sample input data</b>
<b>Step 3: Define sample input data</b>	<b>Create definitions for Input data to our mock database and Define Expected Query</b>
<b>Step 4: Call Function</b>	<b>We call the InsertLog Function With Sample data which was previously defined and print errors in any Otherwise move to next step</b>
<b>Step 5: Check Return Value</b>	<b>Check if Function Returned previously defined Expected ID , if not, fail test</b>
<b>Step 6: Check for other database Expectation</b>	<b>Verify that Database did not had more Queries/Inserts than previously defined</b>

#### UC-17 Post to Notice Board

<b>Test Case</b>	<b>TC-6</b>
<b>Use Case being tested</b>	<b>PostToNoticeBoard</b>
<b>Criteria for Success/Fail</b>	<b>Test that admins can post information to notice boards</b>
<b>Input Data:</b>	<b>Alphanumeric</b>
<b>Test Procedure:</b>	
<b>Step 1: Create a mock Database connection</b>	<b>Use Mock library to create a fake database connection which simulates the table we are going to query.</b>
<b>Step 2: Use data models with mock Database</b>	<b>Create an instance of the our data models with mock database connection and define sample input data</b>

<b>Step 3: Define sample Data &amp; Query</b>	<b>Define sample data And Query that is to be inserted to mock database</b>
<b>Step 4: Call Function</b>	<b>Call Function With sample Data</b>
<b>Step 5: Verify Return Value</b>	<b>Verify if returned ID matched Expected ID which was previously defined</b>
<b>Step 6: Check for other database Expectation</b>	<b>Verify that Database did not had more Queries/Inserts than previously defined</b>

## Strategy to Conduct Unit Testing

The strategies used for conducting integration testing would involve the following steps:

1. **Isolate Components:** Ensure that each unit test is focused on a single function or module. Isolate the component being tested by mocking dependencies such as the database or external APIs.
2. **Test Database Interactions:** Use a test database or an in-memory database like SQLite for testing database interactions. Ensure the test database schema mirrors the production schema. Mock database queries and transactions using tools like sqlmock for Go.
3. **Automate Test Execution:** Use continuous integration tools like GitHub Actions to automate the running of unit tests on every commit or pull request. Ensure the CI pipeline includes steps to set up the test environment, run tests, and report results.
4. **Mock External Dependencies:** Mock external services and APIs that the web app interacts with. For example, mock email services or third-party APIs to ensure tests run independently of external systems.
5. **Test HTTP Handlers:** Write unit tests for HTTP handlers in Go to verify they return the correct status codes, headers, and responses. Use packages like `httptest` to simulate HTTP requests and responses.
6. **Validate Input Data:** Write unit tests to validate user input for forms and APIs. Ensure that input validation logic correctly handles edge cases and invalid data.
7. **Use Test Fixtures:** Set up test fixtures to provide a known state for tests. Use fixtures to populate the test database with initial data needed for testing.



8. Cover Edge Cases: Ensure unit tests cover edge cases and boundary conditions. Test for scenarios like empty inputs, maximum length inputs, and unusual user behaviors.
9. Ensure Code Coverage: Use tools like ``go test -cover`` to measure code coverage in Go. Aim for high coverage, but focus on covering critical paths and logic branches.
10. Mock User Sessions and Authentication: Write tests for authentication and authorization logic. Mock user sessions and tokens to test access control and user-specific functionality.

## History of Work

### History of Work

As a team, we collaborated closely throughout the entire process of creating our application. We relied on GitHub as our main tool for managing our project. We regularly shared our code updates using GitHub Repository and uploaded essential files, reports, and finalized documents. This helped us keep track of our progress and ensured that everyone had access to the latest version of our work. Our teamwork extended beyond just sharing files and code. We held frequent meetings where we worked together to solve any problems or bugs in our code. These sessions were valuable opportunities for us to discuss our project's direction, review our reports, and provide updates on our individual tasks.

In addition to formal meetings, we also maintained constant communication through a group chat on WhatsApp. This allowed us to stay connected even when we couldn't meet in calls or physically. We used this platform to share quick updates, ask questions, and discuss any issues that arose during our work. Despite our effective collaboration, we did encounter a minor issue with GitHub. Occasionally, there were instances where changes made by team members didn't appear after they were uploaded. This meant that we sometimes had to spend extra time troubleshooting and ensuring that all updates were properly reflected in our repository.

### Current Status

We have successfully implemented several crucial use cases within our application to enhance user experience, ensure security, and facilitate effective communication. For instance, users can seamlessly register using their organizational credentials, granting them access to various features. Once logged in, they can effortlessly view their profile information and stay updated on campus events through the notice board feature. Additionally, in case of emergencies, users can activate the panic button, triggering an immediate notification to security personnel with their precise location pinned on a map. Moreover, security staff members benefit from functionalities like clocking in/out to track active personnel, receiving calls from users for swift assistance, and viewing the map to respond promptly to emergencies. Admins can post important alerts to the notice board and review reports submitted by students for further investigation when necessary. These implemented use cases reflect our commitment to delivering a comprehensive and secure

application that caters to the diverse needs of our users while ensuring efficient management and response capabilities.

## **Future Work**

Moving forward, there are several areas of future work to enhance our application's functionality and user experience. One aspect to focus on is the integration of additional features to further streamline user interactions and improve safety measures. For instance, implementing advanced location tracking functionalities could enhance the accuracy and efficiency of emergency response systems. Moreover, expanding the communication capabilities by integrating real-time messaging or chat functionalities could facilitate faster and more effective communication between users and security personnel. Additionally, enhancing the reporting system with advanced analytics capabilities could provide valuable insights for better decision-making and proactive management of campus safety. Furthermore, continuous refinement of existing features based on user feedback and emerging technological advancements will be essential to ensure that our application remains relevant and effective in meeting the evolving needs of our users and maintaining a safe and secure campus environment.

## **Project Management**

Throughout the process of creating this application, all group members actively contributed to its development and management. We faced several challenges during the project, with one major issue being the establishment of effective communication within our team. Since most of our meetings were online, coordinating schedules and keeping everyone updated was difficult. However, through collaborative efforts and regular check-ins, we overcame this obstacle and maintained our momentum. Learning different diagrams, such as use case and class diagrams, was challenging at first. We spent time understanding and creating these diagrams, as shown in our report. By working together and supporting each other, we managed to sharpen these essential skills. Despite the difficulties, we successfully navigated the project by leveraging teamwork and consistent communication. This experience has not only improved our technical skills but also our ability to work effectively as a team in a virtual environment.

## **Summary of Changes**

This is the final report which has been through many phases and many revisions. The product that is currently being displayed is the final version of all previous reports in a revised way. The first and most significant change was the addition and reviewed versions of the Use Cases. In the previous report, some use cases may appear to have been missing, while some seem to not make much sense. The use cases can be considered the most important part of the project, so due to these changes made, many other parts of the report have been modified to fit these changes. These changes include, the use case diagram, majority of the traceability matrices, graphs, additions to the sequence diagrams and descriptions of the relevant use cases.

## References

AltexSoft. (2023, November 30). *Functional and Nonfunctional Requirements Specification*.

AltexSoft. Retrieved April 24, 2024, from

<https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/>

Booch, G., & Jain, S. (2024, January 3). *Domain Modeling - Software Engineering*.

GeeksforGeeks. Retrieved March 20, 2024, from

<https://www.geeksforgeeks.org/software-engineering-domain-modeling/>

Tuck, K. (2020, February 21). *Regression testing can never be fully automated*. Kevin Tuck.

Retrieved March 31, 2024, from

<https://kevintuck.co.uk/regression-testing-can-never-be-fully-automated/>

Fotso Omer. (2023, June 29). *Mathematical modeling in software engineering*. Medium.

Retrieved May 15, 2024, from

<https://medium.com/@wafomer912/mathematical-modeling-in-software-engineering-eabe7ac6f8f3>