Design and Implementation of an IoT-based Short Message Relay and Push Notification System Using MQTT Protocol

A Capstone Project Presented to the Faculty of the

School of Computer Studies

University of San Jose-Recoletos

Cebu City, Philippines

In Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in Information Technology

by

Cuico, Kenneth

Libradilla, Joshua

Morales, Jonathan Kent

Salazar, Mary Mackemzie

Salsbury, Shane Kian

May 2023

ABSTRACT

Efficient information dissemination in schools improves the quality of education. However, providing efficient information dissemination is challenging for schools in the Philippines due to geographical location and financial difficulties. Hence, this study develops the Adelante Short Message Relay System (AdelanteSMRS), a mobile and web-based campus short message relay system using MQTT protocol. It uses an Internet of Things (IoT) based system to deliver short message service (SMS) messages from a web or mobile device connected within the range of the school's network via MQTT Protocol. The researchers conducted a series of Data Management, Cellular Network Provider, and Message Sending tests to determine its functional sustainability and performance testing. Results and findings revealed that the system is applicable for school use, and only the issue of signal reception proved to be a problem when using the system. Hence the researchers recommend a better antenna module as a solution.

Keywords: MQTT, SMS, Schools, Internet of Things, IoT, Web App, Mobile App

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Rationale of the Study

In modern centralized information-based societies, information dissemination has become one of the most critical social processes that enable people to develop active learning initiatives (Huang, 2021). As a place where much information is accumulated and handled, schools are the ideal place where information dissemination is practiced and streamlined (Cope & Kalantzis, 2016). Efficient information dissemination in schools is crucial, such as improving education and learning, promoting active participation and engagement with students and parents, enhancing school administration and management, and ensuring safety and emergency communications (Sadiku et al., 2021). However, not all schools can achieve the best effect that effective information dissemination can provide. Schools in the Philippines face several challenges with information dissemination due to their diverse culture and unique geographical and educational landscape. Hence, it is essential to use the current information infrastructure and technology, such as the Internet of Things (IoT), to enhance information dissemination within schools in the Philippines.

Schools utilize multiple applications and social networking sites to disseminate information to their students, parents, and school members. Social media sites like Facebook are the most popular and commonly used. Facebook is a social media platform founded by Mark Zuckerberg in 2004. It enables users to create profiles, share pictures, and connect with families and friends. It also allows users to create pages and groups to get updated with news and content. Sites like Twitter have similar features, albeit rarely used by schools. Some applications allow their users to exchange information, similar to forums. Among them, the most popular one that most schools use is Microsoft Teams. Microsoft Teams is a platform for cooperative communication and collaboration that Microsoft created. It provides communication, file sharing, and collaboration on a single platform for teams and organizations working online, as well as features including video conferencing, document collaboration, and chat-based messaging. Other applications only allow users to create video conference rooms with chat-based messaging features such as Zoom.

However, one aspect that all of these applications and social networking sites have in common can be problematic for some Philippine schools. They all require a reliable internet connection and the appropriate devices to connect with, which provide logistical or financial difficulties for some schools. For example, mountainous areas likely have difficulty catching a good internet reception from cellular towers nearby. Moreover, the Philippines is vulnerable to numerous typhoons yearly, damaging the region's information infrastructure and restricting internet connectivity (Caluza, 2016). Hence, applications and sites that utilize internet connectivity may not be ideal for information dissemination in certain situations, and an alternative method may be needed.

As such, this study proposes an SMS-based system that utilizes Internet-of-things (IoT), giving an alternative method to such situations. The Adelante Short Message Relay System (AdelanteSMRS) is a mobile and web-based campus short message relay system using MQTT protocol with SIM800L and ESP32. It uses an Internet of Things (IoT) based system to deliver short message service (SMS) messages from a web or mobile device connected within the range of the school's network. The system uses an integrated ESP32 microcontroller and SIM800L

GSM/GPRS module connected to a network using Message Queuing Telemetry Transport (MQTT) protocol. This system will enhance the school's effective information dissemination by providing widespread area availability, many reachable recipients, high open rates that recipients can view in seconds, a straightforward communication method for most people, and a complementary method to other communication channels.

Review of Related Literature

According to The Institute of Electrical and Electronics Engineers (IEEE), the Short Message Service (SMS) is a widely known and used communication mechanism for cell phone users (2007). Short Message Service, or SMS, is a communication protocol that utilizes mobile phone networks. It uses standardized communication protocols that enable the exchange of short text messages between mobile phone devices. SMS has been a popular method of communication for many years and is supported by almost all mobile phones and mobile networks worldwide. SMS has an active user base of 3.6 billion users, or 76% of all mobile users, which cements itself as the most widely known communication protocol. It uses standardized communication protocols that enable the exchange of short text messages between mobile phone devices. Although text messaging is usually associated with mobile-to-mobile communication, the inclusion of other mobile technologies has already been taken into action. (Mojares et al., 2013)

Many studies show the success of utilizing SMS in schools. In a study conducted by Dr. James Kadirire entitled The short message service (SMS) for schools/conferences, he developed a program that enables students to send SMS messages to their teachers, who can then choose which messages to display on a big screen and respond to the question or comment in an interactive manner (Kadirire, 2005). He concluded that SMS could be successfully employed in group discussions in businesses and schools because it maintains anonymity, enables people to express their opinions without worrying about backlash, and is comparatively simple.

Another study, entitled Using Short Message Service (SMS) to teach English Idioms to EFL Students, also concluded that students were more enthusiastic and knowledgeable about SMS-based learning than contextual and self-study learning (Hayati et al., 2013). Hayati and his team conducted a test where three test groups (self-study-based, SMS-based, and contextual-based) were given the same idioms but in different manners (pamphlets, SMS messages, and textbooks). A pretest and posttest were given to all three groups, and a post-study survey. Their results suggest that SMS could be considered a practical tool for instruction and learning due to mobile phones' highly user-friendly and cost-effective feature in studying English idioms. However, they highlighted that it should not disregard the role of a teacher as the primary person for teaching. As such, this study assumes that using our SMS-based system would benefit teachers, students, and school administration, whether it is used as an educational aid, emergency alert message, or school-wide announcement.

Review of Related Works

School Event Notification Through SMS (SENT SMS) is a system developed by Ramil G. Lumauag. It is beneficial for parents, instructors, and students to receive up-to-date news from the school directly on their smartphones (Lumauag, 2016). Students will be informed of forthcoming school events, changes to the schedule, and cancellations of classes due to inclement weather via SMS notification. The system involves using a GSM Modem connected to a server computer, which also contains the database, to publish SMS messages to registered numbers in the database. The content message's publishing depends on the admin managing the server computer. This system is similar to the AdelanteSMRS; however, instead of involving the admin only, it also involves teachers, faculty staff, and school administration in managing the content of the SMS sent to students.

Another application Olaleye et al. (2013) created is an SMS-based Notification System that uses JSP and a Tomcat web server to send mass SMS messages to intended recipients. The system is very similar to the AdelanteSMRS as it also has member groups and group-sending functionalities. It also implements a good-to-have feature for the AdelanteSMRS, which is scheduled sending. The application created by the group of researchers also considered the functionalities of the SMS providers at that time, which needed to be improved. Different SMS gateways for the various SMS providers were also implemented into the application. However, the application created by the team is of a different implementation compared to the AdelanteSMRS, which uses PHP and the Laravel framework.

Another study entitled, Final Project Consultation Information System Integrated Notification System Based on SMS Gateway, showcases an application that integrates an SMS notification system to provide information to both lecturers and students of uploads to the system (Satria et al., 2018). Once a student uploads a copy of their final project to the system, the system will notify the lecturers of the recent upload. Likewise, once a lecturer uploads a revised copy of the final project to the system, the students are notified of the new revisions. The researchers also use the same language and database, which are PHP and MySQL but with an additional component of GSM to provide an SMS gateway.

Another study proposed an IoT-based system for monitoring the attendance of the students at the University of Port Harcourt, which is quite similar to this paper for it uses both software (web application) and hardware (ESP32 microcontroller, LED, and fingerprint biometric sensor, among others) to provide a reliable method of managing and storing data of students' attendance performance (Joseph & Moses, 2019). The student's attendance is tracked using fingerprint sensors and an ESP32 microcontroller. This is then sent to the system's online database, where professors and the Head of Department (HOD) can access it when needed. The Wi-Fi connection protocol used by this system allows it to communicate with other smart devices, including personal computers, android smartphones, and the like.

In 2019, Kedare et al. designed another approach to an attendance monitoring system using the SIM800L GSM module to transport the collected data, an ATmega8 microcontroller associated with an Arduino board, and a fingerprint sensor module to register and record fingerprints. This system makes use of biometric scanning, in which the biometrics are initially kept in accordance with the ID numbers of each student. The system will, later on, match the already saved biometrics of a student. A memory module saves the recorded attendance through the microcontroller and the date and time. Additionally, the system will notify the parents via SMS once the student's absence is recorded, preventing students from skipping classes. The researchers of this paper concluded that their proposed system provides a more efficient attendance monitoring system, eliminates errors, and speeds up the process of verifying student attendance.

Some schools still use manual systems and keep their students' records in different files, which increases the possibility of data loss, duplication, and damage. Othoman et al. (2006) proposed a solution to this problem by introducing a database management system in schools around Bandar Jengka, Pahang, that will assist school admins in managing their student records more efficiently and effectively. The proposed database system is a web-based application comprising various technologies such as the hypertext transfer protocol, the Apache web server, and the MySQL database. It can provide users with real-time data to help them make data-driven decisions.

The following applications display the opportunities and potential of an SMS-based notification system. These applications are already being implemented in other parts of the world, but many of these systems still need to be implemented in the Philippines. It is more jarring as it directly opposes the Philippines' reputation as the "texting capital of the world" (Roberts & Hernandez, 2019). The AdelanteSMRS would be part of the many implementations of SMS-based notification systems in our locale, which still need to be made more common.

Significance of the Study

Developing mediums to disseminate information is essential in facilitating the planning, implementation, and assessment of a distinct learning procedure. Particularly in a school setting where face-to-face classes are unavailable, the necessity of a Learning Management System or LMS can be seen in how convenient it is to virtually test and assess students, both on the part of the student and instructors. Students can learn about new lesson modules, assignments, quizzes, and output submissions. At the same time, instructors can publish module lessons and output submissions to all classes without going through the same process for each class.

However, in terms of information dissemination, there are better methods of communication. Although the LMS is designed with features that bring convenience and benefits for students and teachers, the prerequisite for using the LMS requires one to have a stable internet connection and an appropriate device. Hence, there may be times when the LMS may only be available for some to use at all times. For example, a student, who currently has no internet connectivity, may not be able to see any critical announcement made by an instructor. In this situation, the researchers can see that there may be better solutions to this problem than using LMS since it requires one to have internet connectivity. Hence, the researchers need to find another solution to disseminate information without internet connectivity. One of these alternatives can be the Short Message Service or SMS messages.

Short Message Service (SMS), also known as "texting," uses standardized communication protocols to transmit brief text messages via mobile devices and devices that are also SMS-capable. SMS does not need one to access the internet. Instead, it requires one to be within the range of a cellular tower. By setting up a network area that covers the whole school and connecting a device that could send SMS messages to cellular towers, one can receive information from the network and disseminate information without internet connectivity. In addition, pairing it with a management system that can manage the messages being sent can allow instructors to selectively send messages to a particular group of students instantly through SMS. As such, this study proposes a system, AdelanteSMRS, that fulfills the same system that utilizes a network and devices to create a management system for sending SMS messages.

This study, Adelante Short Message Relay System (AdelanteSMRS), a mobile and web-based campus short message relay system using MQTT protocol with SIM800L and ESP32, would benefit the following:

Students – the system can allow students to receive school announcements and information without connecting to the internet. In cases where there is no internet connectivity, students would not need to spend money or find areas with internet connectivity as long as the student is within a cellular tower's range.

School Faculty and Instructors – using the system, the school faculty and instructors can send important information to either a group of students or individual students directly, without the need to connect to the internet. This, in turn, can save time and effort by going through the tedious process of accessing the internet through a computer and manually assigning the information to individual students or groups.

University of San Jose-Recoletos' School of Computer Studies – utilizing the framework of the system, the School of Computer Studies can apply different types of applications, different from the learning management system to the SMS system.

University of San Jose-Recoletos' College Departments – the university's college departments could utilize the system to send important announcements and information to all students under their college department.

University of San Jose-Recoletos – the university can send all students school-wide announcements and information, specifically in notifying university students of essential activities and events. It can also be utilized in emergencies when there is no internet connection.

Future Researchers – the study can be used as a reference for future researchers. Future researchers can use the study as a framework to develop new applications and technologies that utilize the same system or a similar approach to the system.

Project Objectives

A. General Objectives

The researchers' main objective is to develop a mobile-web and IoT-based SMS alert system to assist professors and administrators at the University of San Jose - Recoletos (USJ-R) in disseminating alert messages and important announcements to enrolled students. The goal was to make communication between students and faculty more convenient and in a way that is novel.

B. Specific Objectives

There are various components crucial to the development of this project. Thus, the researchers have the following specific objectives:

1. To integrate the SIM800L module with the ESP32 microcontroller;
2. To implement the IoT MQTT messaging protocol and round-robin algorithm;
3. To create and design the UI for the different features of both the mobile and web applications;
4. To identify and create the different entities involved in the system;
5. To design and set up a database for the system, preferably using MySQL;
6. To identify and create the different use case diagrams and scenarios in the system;
7. To set up and design a Software Architecture laying out the system's different components;
8. To create and set up a web application for the system using PHP with Laravel;
9. To create a mobile application using the Flutter framework;
10. To create a REST API using the Laravel framework; and

11. To debug and test the system extensively using different testing methods appropriate.

Scope and Limitation

This study primarily focuses on implementing the ESP SIM800L GSM (Global Systems for Mobile) / GPRS (General Packet Radio Service) Module using the ESP32 Arduino Micro-controller. The system will use an MQTT messaging protocol and a mix of mobile and web applications to consume, manage and maintain the system. The features that will be used in the module are the following: sending SMS text messages, creating and managing groups, registering, and logging in. The system will broadcast text messages reminding students of the following classes and events in a department and the university. In addition, the framework for creating this system's web and mobile application will be limited to the Laravel Framework (Web) with PHP and Flutter (Mobile) with Dart. Hence, there are some limitations to the number of features the system can implement.

The hardware for the system uses an ESPRESSIF-ESP32 240MHz single-/dual-core 32-bit LX6 microprocessor, which includes Bluetooth and WIFI capabilities and can house a micro SIM for SIM-related features. It has to be noted that certain variables affect its effectiveness. With usual SMS usage considerations, the antenna for connecting to the cellular tower also requires a good signal strength to transmit data from the system to the SMS providers effectively. During the purchase, some modules are of low quality, like its antenna, which is a generic flat-type SIM antenna. This type of antenna does not work well with indoor reception. Hence, using the device indoors may compromise its ability to connect to the cellular tower, prolonging the transmission process or may be lost in some instances. The type of

SIM pack to be used would also affect the signal strength of the devices. Cell towers for Smart and Globe are the most prevalent in Cebu City. As such, these SIM packs, as mentioned earlier, are the ones used in the system. With these considerations in mind, it is preferable to locate the hardware devices with high signal strength, and usage of the SIM packs of the Smart and Globe variety, especially Smart, is prioritized.

The project's development time is nine months, from July 2022 to April 2023. The study will also be limited to only the School of Computer Studies in the University of San Jose - Recoletos, Basak Campus, and courses related to the School of Computer Studies. The study will be done all within the physical confines of the University of San Jose - Recoletos, and students, professors, and faculty administrators will be the system's only users.

During the development of this system, only mobile numbers from students and individuals with their consent to the testing part of this system were used for testing purposes. Any sensitive data found in this study is taken from the school with the consent of the relevant individual with authority. As such, such data would not be presented in this study, but only the results from utilizing such data would be shown.

CHAPTER II

SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATIONS

This chapter contains the project's Architectural Diagram, Use Case Diagram, Use Case Scenarios, Activity Diagram, Entity Relationship Diagram, and Data Dictionary. It shows the system's expected behavior as well as how the different actors involved interact with it.

Application Overview

Architectural Diagram

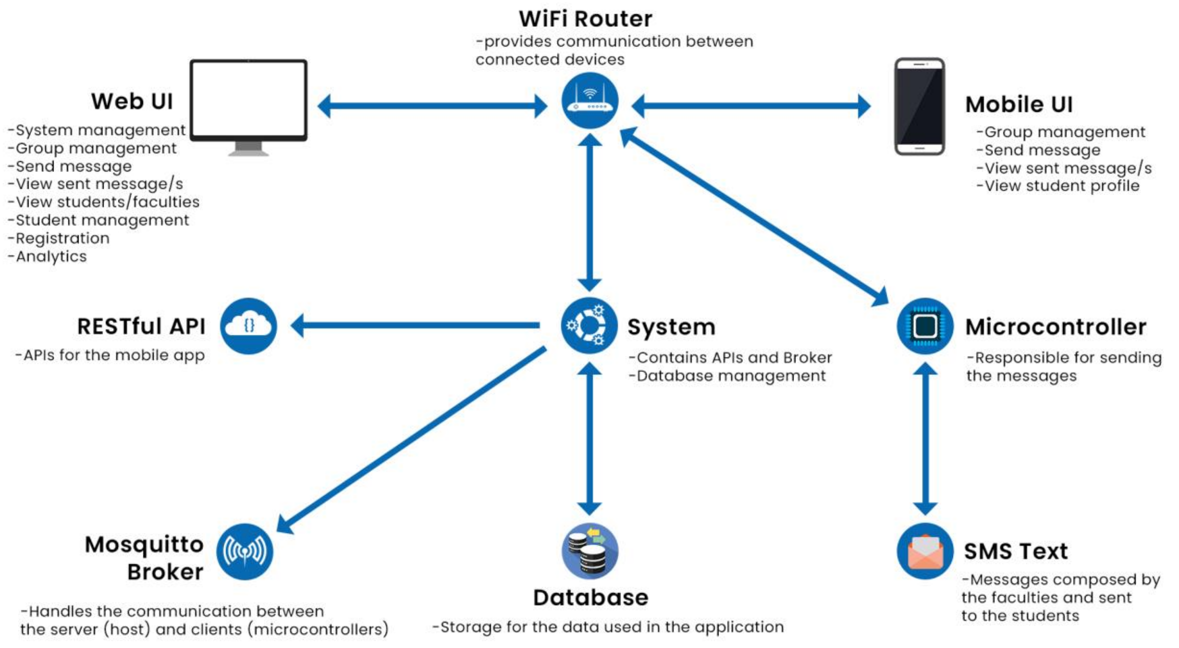


Figure 2.1: ***AdelanteSMRS Architectural Diagram***

Figure 2.1 shows the architectural diagram of the AdelanteSMRS system, laying out the key components, their use, and how they are integrated. The web application serves as an interface that would allow the management of the entire system. In contrast, the mobile application is more compact and portable, focusing primarily on sending alerts and managing groups on the go. It also passes its requests through the system's RESTful web API. Both interfaces interact with a MySQL database to store and retrieve data. The Mosquitto Broker oversees the communication between the server and the microcontrollers. The server processes message queues from the database and passes them to their microcontrollers, determined by a round-robin algorithm. The microcontrollers then process the message and send the relevant SMS messages to the cellular tower.

Use Case Diagram

A Use Case Diagram identifies and lays out the system's different use cases and actors, where use cases are what the actor(s) want to achieve when using the system.

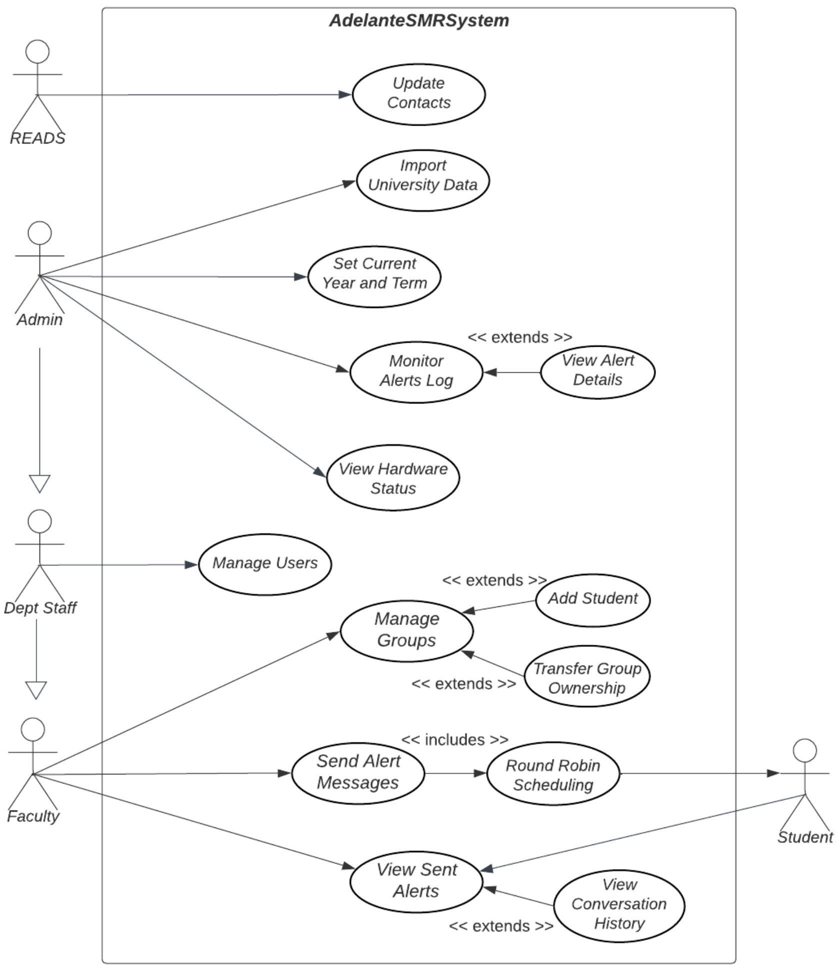


Figure 2.2: ***System Use Case Diagram***

Figure 2.2 shows the system's Use Case Diagram. They identify the actors involved and their actions or goals when using the system. The Admin actor inherits the use cases assigned to the Department Staff, which in turn inherits the use cases assigned to the Faculty. This means that the Admin and Department Staff can do the use cases assigned to the Faculty while having their Actor-specific use cases.

Admins will want to manage the system, which includes importing data related to the university and setting the current term and year. They will also want to monitor the alerts being sent and the status of the ESP32 microcontrollers. Department Staff will want to manage users in their respective departments. Faculty will want to manage courses and groups they are handling, which extends to adding students and transferring ownership of the group. Users will also want to send alert messages to the different groups or courses they are handling and individual students. Admins may view and add students from the entire university. At the same time, Faculty and Department Staff only have access to students under the same college or students they are handling in a group. Round-robin scheduling is also included to account for spam. Users will also be able to view alerts that they have sent or conversation histories. Likewise, Students will be able to view alerts they have received. Finally, READS scholars may use the system to update the contact information of individual students in a particular school or course.

Use Case Scenarios

Use Case Scenarios take each of the use cases identified in the Use Case Diagram and explain its purpose, actors, step-by-step flow of events, and the interaction between the actor and the system.

Use Case Scenario 001: Update Contacts

Use Case 001	Update Contacts
Actor	READS
Purpose	Allow READS to update inaccurate or missing student data, specifically their mobile phone number.
Overview	READS searches for the student’s ID number. Once found, the user will input the student’s mobile phone number to update it within the system.
Precondition(s):	READS is logged in
Postcondition(s):	System displays a notice of the successful operation
Flow of Events	
Actor Action(s)	**System Response**
1. READS types in the ID number of the student whose mobile phone number is to be updated 4. READS inputs the updated mobile phone number of the	2. System retrieves student data from the database according to the READS’ input 3. System displays the information of the fetched student 6. System processes the information and updates the data of the student within the database 7. System displays modal that confirms the successful operation

student 5. READS clicks update	
Exception Flow E1: READS cancels the operation	
1. System closes the window 2. End process	
Exception Flow E2: System fails to fetch student data from the database	
1. System displays a notice to the user 2. End process	
Exception Flow E3: System fails to process the information from the form	
1. System displays an error to the user 2. End process	

Table 2.1: ***Update Contacts Data Use Case Scenario***

Update Contacts is the main and only use case for the READS actor. It allows them to update the mobile phone number of students within the database. It can also only be accessed through the web application of the system.

Use Case Scenario 002: Import University Data

Use Case 002	Import University Data
Actor	Admin
Purpose	Allow Admins to import university data in the system
Overview	Admins upload a .csv file containing various data about the university. These .csvs include: faculty, colleges, programs(degree programs), departments, groups, student list and student load. These .csv files are converted and added to

	the system's database.
Precondition(s):	Admin is logged in
Postcondition(s):	System displays a notice of the successful operation
Flow of Events	
Actor Action(s)	**System Response**
1. Admin selects "Import Data" 3. Admin chooses what type of data to upload. 5. Admin selects specific .csv file 7. Admin selects confirm upload	2. System displays "Import Data" view 4. System displays window for Admin to select a .csv file 6. System receives the file and converts the data within each row to a collection, and displays a table previewing the list of students to be added 8. System converts the data within the collection to database entries, using the columns as a basis
Alternate Flow A1: Data already exists in the database	
1. System updates the existing data if there are any changes	
Exception Flow E1: Admin cancels Select File	
1. System closes the window 2. End process	
Exception Flow E2: Admin cancels upload confirmation	
1. System closes the window 2. End process	
Exception Flow E3: System fails to upload the .csv file	
1. System displays an error to the user 2. End process	

Exception Flow E4: Data in .csv file is not supported/properly formatted, or is corrupted
1. System displays an error to the user 2. End process

Table 2.2: ***Import University Data Use Case Scenario***

Import University Data is one of the primary use cases for the Admin actor. It allows them to import data related to the university into the system using CSV files in bulk. This includes college, department, program, faculty, subjects, students, and student load data. New data is added to the system while existing data is updated. This use case mainly involves the system's web application.

Use Case Scenario 003: Set Current Year and Term

Use Case 003	Set Current Year and Term
Actor	Admin
Purpose	Allow Admins to set the current school year and term for the system.
Overview	Admins select the current school year through a roulette type selection, and term, through the use of a dropdown.
Precondition(s):	Admin is logged in
Postcondition(s):	System displays a success modal to confirm changing of current year and term
Flow of Events	
Actor Action(s)	**System Response**
1. Admin selects	2. System displays a modal showing the current term and year

"Settings" from the top navigation bar. 3. Admin inputs new term and year.	with the input fields. 4. System accepts input and processes the new term and year.
Alternative Flow A1: Admin cancels action	
1. User is directed to dashboard 2. End process	

Table 2.3: ***Set Current Year and Term Use Case Scenario***

Set Current Year and Term allow the Admins to set the current school year and term of the system to match the university's. The student and group data displayed in the system will also reflect that of the selected term and year, automatically archiving data from previous semesters. This use case mainly involves the system's web application.

Use Case Scenario 004: Monitor Alerts Log

Use Case 004	Monitor Alerts Log
Actor	Admin
Purpose	Allow Admins to view and monitor alert messages being sent by other users using the system
Overview	Admins view a table containing real-time data of alerts being sent by users using the system. The data contains the sender, receiver, the date and time the alert was sent, and its status (if it is sent or queued for sending).
Precondition(s):	Admin is logged in
Postcondition(s):	System displays a log of all the alerts sent

Flow of Events	
Actor Action(s)	**System Response**
1. Admin selects “Alerts Log” tab	2. System retrieves alerts data from the database 3. System displays table containing a paginated list of all the alerts sent using the system, sorted from the latest one
Exception Flow E1: System fails to retrieve alerts data from the database	
1. System displays an error to the user 2. End process	

Table 2.4: ***Monitor Alerts Log Use Case Scenario***

Monitor Alerts Log allows the Admins to view a log of all the alerts sent through the system, along with details of the sender, the date/time it was sent, the message's status, and the expected receiver/s. This use case mainly involves the system's web application.

Use Case Scenario 005: View Alert Details

Use Case 005	View Alert Details
Actor	Admin
Purpose	Allow Admins to view more details about a specific alert log
Overview	Admins expand a particular alert log to view more information about it, such as the message ID, recipient count, and the message itself
Precondition(s):	Admin is logged in
Postcondition(s):	System displays an expanded info of the alert log
Flow of Events	

Actor Action(s)	System Response
1. Admin presses "View Details"	2. System expands the alert log row and displays more information about the sent alert

Table 2.5: ***View Alert Details Use Case Scenario***

View Alert Details extends the Monitor Alert Logs use case. It allows Admins to view more details about an alert entry, such as the expected number of recipients and the message's contents.

Use Case Scenario 006: View Hardware Status

Use Case 006	View Hardware Status
Actor	Admin
Purpose	Allow Admins to view the status of the hardware devices which facilitates the sending of the SMS messages.
Overview	Admins view a status of the devices which includes what device number and what its availability is.
Precondition(s):	Admin is in "Alert Logs" view
Postcondition(s):	System displays an expanded info of the devices' status
Flow of Events	
Actor Action(s)	**System Response**
1. Admin presses "Device Status"	2. System retrieves data from a table that houses device data. 3. System displays modal that contains all the devices and their appropriate statuses

Table 2.6: ***View Hardware Status Use Case Scenario***

View Hardware Status extends the Monitor Alert Logs use case. It allows Admins to view the details and status of the hardware devices used in the system.

Use Case Scenario 007: Manage Users

Use Case 007	Manage Users
Actor	Admin Dept Staff
Purpose	Allow users to accept registration requests in the system
Overview	Users access the request route, and may either confirm or deny user’s registration requests
Precondition(s):	User is logged in User is authorized to Manage Users
Postcondition(s):	System approves the requesting user to the system, allowing them to access it
Flow of Events	
Actor Action(s)	**System Response**
1. User selects Registration tab 4. User picks user to approve to the system, and clicks “Confirm”	2. System retrieves user registration requests from the database, according to their role (all users if Admin, users under the same school/college if Department Staff) 3. System displays paginated list of users requesting to be registered within the system 5. System approves the registering user to the system, allowing them to use it
Alternate Flow A1: User clicks “Deny”	
1. System denies the registering user, and removes their credentials and data from	

the database
Exception Flow E1: User cancels Manage Users
1. System closes the window 2. End process
Exception Flow E2: System fails to process User confirmation
1. System displays an error to the user 2. End process

Table 2.7: ***Manage Users Use Case Scenario***

This use case allows users (Admin, Dept Staff, Chairperson, and Dean) to approve registering users to the system, allowing them to use and access it. This usually involves faculty registration to the system and their verification process by their respective department chairpersons.

Use Case Scenario 008: Manage Groups

Use Case 008	Manage Groups
Actor	Admin Faculty Dept Staff
Purpose	Allow Users to add and manage different courses or groups
Overview	User creates a course/group within the system providing its name, and description.
Precondition(s):	Faculty is logged in
Postcondition(s):	System displays the newly-created group
Flow of Events	

Actor Action(s)	System Response
1. User selects "Create Group" 3. User provides Group name and description (optional) 4. User presses "Create"	2. System displays create group form 5. System adds group to the database 6. System displays newly-created group

Table 2.8: ***Manage Groups Use Case Scenario***

This use case allows the users to create and manage the different courses or groups they will use within the system. Providing information such as the name of the course or group and its description, although the latter is optional.

Use Case Scenario 009: Add Student

Use Case 009	Add Student
Actor	Admin Faculty Dept Staff
Purpose	Allow Users to add individual students to their course or group
Overview	Users adds a student to their group by searching for their name, assuming that they already exist in the database
Precondition(s):	User is logged in
Postcondition(s):	System adds the Student to the group
Flow of Events	

Actor Action(s)	System Response
1. User selects “Add Student” 4. User searches for student using their name 6. User clicks on the checkbox on the row of the student 7. User clicks “Add Student” 9. User clicks agree	2. System retrieves paginated list of students who are not a member of the current group, but are a member of one of the user’s created groups from the database 3. System displays add student form 5. System retrieves student data from database according to Faculty user’s input 8. System displays confirmation modal to add student 10. System adds the Student to the group

Table 2.9: ***Add Student Use Case Scenario***

This use case is an extension of the *Manage Groups* use case. It allows users to add a student to a specific group by searching for their name from the database and confirming their addition. Multiple selections of students to be added are also supported.

Use Case Scenario 010: Transfer Group Ownership

Use Case 010	Transfer Group Ownership
Actor	Admin Faculty Dept Staff
Purpose	Allow Users to transfer the ownership of a course or group to another user
Overview	Users choose a group to create a request to transfer its ownership to another user registered in the system

Precondition(s):	User is logged in
Postcondition(s):	System processes the information and creates a new Group Transfer entry in the database
Flow of Events: WEB	
Actor Action(s)	**System Response**
1. User selects "Transfer Group" 3. User types in the username of the to-be owner of their group 6. User selects the to-be owner from the list 7. User clicks "Confirm" 9. User at the receiving end of the requested transfer selects "Approve"	2. System displays transfer group form 4. System retrieves list of users who are under the same College as the currently logged in user from the database according to the User's search query 5. System displays the search results 8. System processes the information and creates a new pending Group Transfer entry in the database 10. System processes the information, sets the Transfer to "confirmed", and accordingly assigns the new owner of the group
Flow of Events: MOB	
1. User selects "Transfer Group" 3. User selects from a dropdown the to-be owner from the list. 5. User clicks "Confirm"	2. System displays transfer group form/dialog 4. System retrieves the selected to-be owner 6. System processes the information and creates a new pending Group Transfer entry in the database 7. System displays successful request dialog 9. System processes the information, sets the Transfer to "confirmed", and accordingly assigns the new owner of the group

7. User clicks “Ok” 8. User at the receiving end of the requested transfer selects “Approve”	
Alternate Flow A1: User selects “Cancel” Group Transfer	
1. System sets the Transfer entry to “canceled” 2. End process	
Exception Flow E1: System fails to create Group Transfer entry to the database	
1. System displays an error to the user 2. End process	
Exception Flow E2: System fails to confirm Group Transfer	
1. System displays an error to the user 2. End process	
Exception Flow E3: System fails to cancel Group Transfer	
1. System displays an error to the user 2. End process	

Table 2.10: ***Transfer Course Use Case Scenario***

This use case is an extension of the *Manage Groups* use case. It allows users to transfer ownership of a group they have created to another user. Thereby removing their access to the group and allowing the other user to take over management and responsibilities.

Use Case Scenario 011: Send Alert Messages

Use Case 011	Send Alert Messages
Actor	Admin Faculty Dept. Staff
Purpose	Allows Faculty to send an SMS message to students
Overview	Faculty can send a SMS Message to students in his/her assigned courses and assign specific recipients
Precondition(s)	Faculty is logged in
Postcondition(s)	Notification of SMS message sent status
Flow of Events: WEB A (Through “Groups” and messaging entire group)	
Actor Action(s)	**System Response**
1. User presses “Manage Groups” button 3. User presses particular group on “Groups” 5. User presses “Send Message” 7. User will write SMS message 8. User presses “Submit” button	2. System directs user to “Groups” screen 4. System directs user to specific group page 6. System selects recipient as whole course 9. System will process request 10. SMS message queued and sent to students
Flow of Events: WEB B (Messaging specific students)	
Actor Action(s)	**System Response**
1.User presses “Send	2. System directs user to “Send Message” screen

Message to Individuals" button 3. User selects recipients of the message 4. User writes the SMS message 5. User presses "Submit" button	6. System will process request 7. SMS message queued and sent to students
Flow of Events: MOB A (Send to Individual/Custom Group)	
Actor Action(s)	**System Response**
1.User selects "Send a Text" button 3. User taps the recipient field 5. User searches and selects students from the list 8. User taps"Confirm" 9. User inputs the message 10. User taps the "Send" Button	2. System directs user to "Send Message" screen 4. System displays a dialog with a list of students to select from 6. System logs the selected students 7. System displays the selected recipients last names on the recipient field as labels 11. System will create a Pseudo Group based on the list 12. System will process the request passing the newly-created pseudo group's group id and message. 13. SMS message queued and sent to students
Alternate Flow A1: Faculty Member/User Taps "Select"	
1. System displays list of groups the user manages 2. User selects a group 3. System displays the subgroups of that group and/or "All Student" tile 4. User selects a subgroup 5. System displays the students of that subgroup	

6. User selects students from the list	
Flow of Events: MOB B (Send to Group/Subgroup)	
Actor Action(s)	**System Response**
1.User selects "Send a Group Text" button 3. User taps the recipient field 5. User selects a group 7. User selects a subgroup 9. User inputs the message 10. User taps the "Send" Button	2. System directs user to "Send Message" screen 4. System displays list of groups the user manages 6. System displays the subgroups of that group and/or "All Student" tile 8. System displays the selected group/subgroup name and description as label on the recipient field 11. System will process the request passing the selected group's group id and message. 12. SMS message queued and sent to students
Exception Flow E2: Faculty Member does not provide SMS message	
1. System display an error to the user 2. Redirects cursor to unfilled field	

Table 2.11: ***Send Alert Message Use Case Scenario***

Send Alert Message is one of the primary use cases for the Faculty Member. It allows them to select specific students or a group to send a text message. As this is one of the application's core functionality, there are various ways to access it. This use case mainly involves the system's mobile application, which is accessible to all the users of the application.

Use Case Scenario 012: Round Robin Scheduling

Use Case 012	Round Robin Scheduling

Actor	System
Purpose	Avoiding the system being tagged as a spamming bot by the various networks
Overview	The system will implement round-robin algorithm to send alert messages
Pre-condition(s)	System has messages to be sent
Post-condition(s)	System not tagged as spam bot
Flow of Events	
Actor Action(s)	**System Response**
	1. System receives message and number 2. System checks the last device used in message queue 3. System assigns the next device to the message job 4. System queues the message job to the message queue

Table 2.12: ***Round Robin Scheduling Case Scenario***

This use case is an extension of *Send Alert Message* use case. It prevents the protocol of networks from blocking high volumes of SMS traffic by using Round Robin Scheduling.

Use Case Scenario 013: View Sent Alerts

Use Case 013	View Sent Alerts
Actor	Admin Faculty Dept Staff Student
Purpose	Allows Users to view sent alerts to students, courses, groups

Overview	User views the sent SMS alert messages to recipients
Pre-condition(s)	User is logged in User has sent messages to students
Post-condition(s)	System display sent SMS alert messages
Flow of Events	
Actor Action(s)	**System Response**
1. User presses "Outgoing" on Bottom Navigation	2. System directs user to "Outgoing" screen 3. System retrieves and displays recipient data

Table 2.13: ***View Sent Alerts Case Scenario***

This use case allows the faculty to view all sent and outgoing messages to the intended recipients. It shows a list of tiles of the names of the students, groups, or courses. This use case can also only be done on the mobile application.

Use Case Scenario 014: View Conversation History

Use Case 014	View Conversation History
Actor	Admin Faculty Dept Staff Student
Purpose	Allows User to view SMS messages history
Overview	User views history of the sent SMS alert messages to recipients
Pre-condition(s)	User is logged in User has sent messages to students

Post-condition(s)	System display history of sent SMS alert messages
Flow of Events:	
Actor Action(s)	**System Response**
1. User long presses a specific message tile from “Outgoing” screen	2. System retrieves and displays all SMS messages sent to recipient

Table 2.14: ***View Conversation History Case Scenario***

This use case is an extension of *View Sent Alerts*. It displays all the SMS messages sent to the particular recipient.

Activity Diagram

Activity Diagrams visualize the different Use Case Scenarios and their flow of events. It utilizes directional arrows to assist in the order of the processes and communication between actor/s and system/s.

Activity Diagram 1

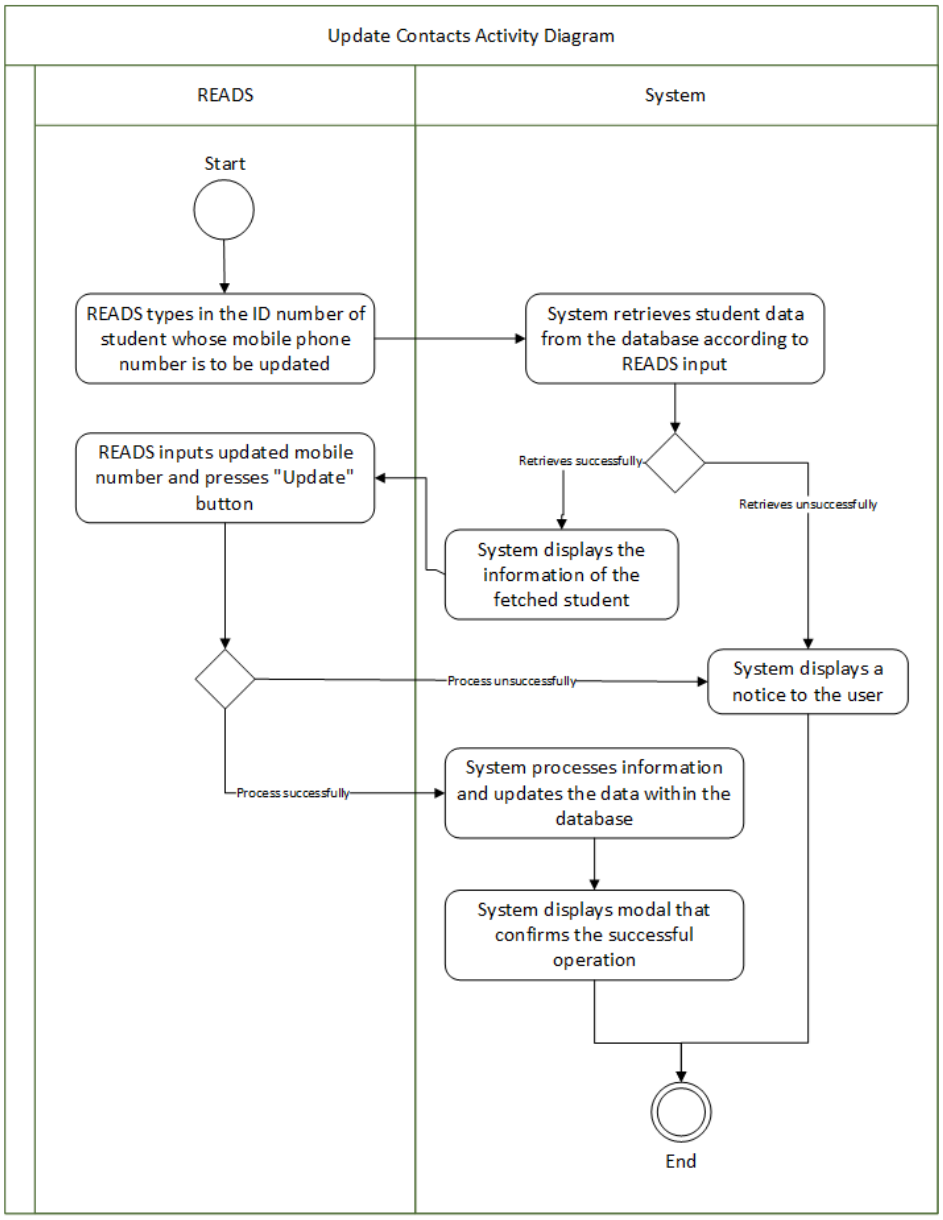

Update Contacts Activity Diagram
READS
System
Start
READS types in the ID number of student whose mobile phone number is to be updated
System retrieves student data from the database according to READS input
Retrieves successfully
Retrieves unsuccessfully
READS inputs updated mobile number and presses "Update" button
System displays the information of the fetched student
Process unsuccessfully
System displays a notice to the user
Process successfully
System processes information and updates the data within the database
System displays modal that confirms the successful operation
End

Figure 2.3: ***Update Contacts Activity Diagram***

Figure 2.3 shows the flow and behavior of the system regarding the changing of contacts of a specific student done by a READS scholar.

Activity Diagram 2

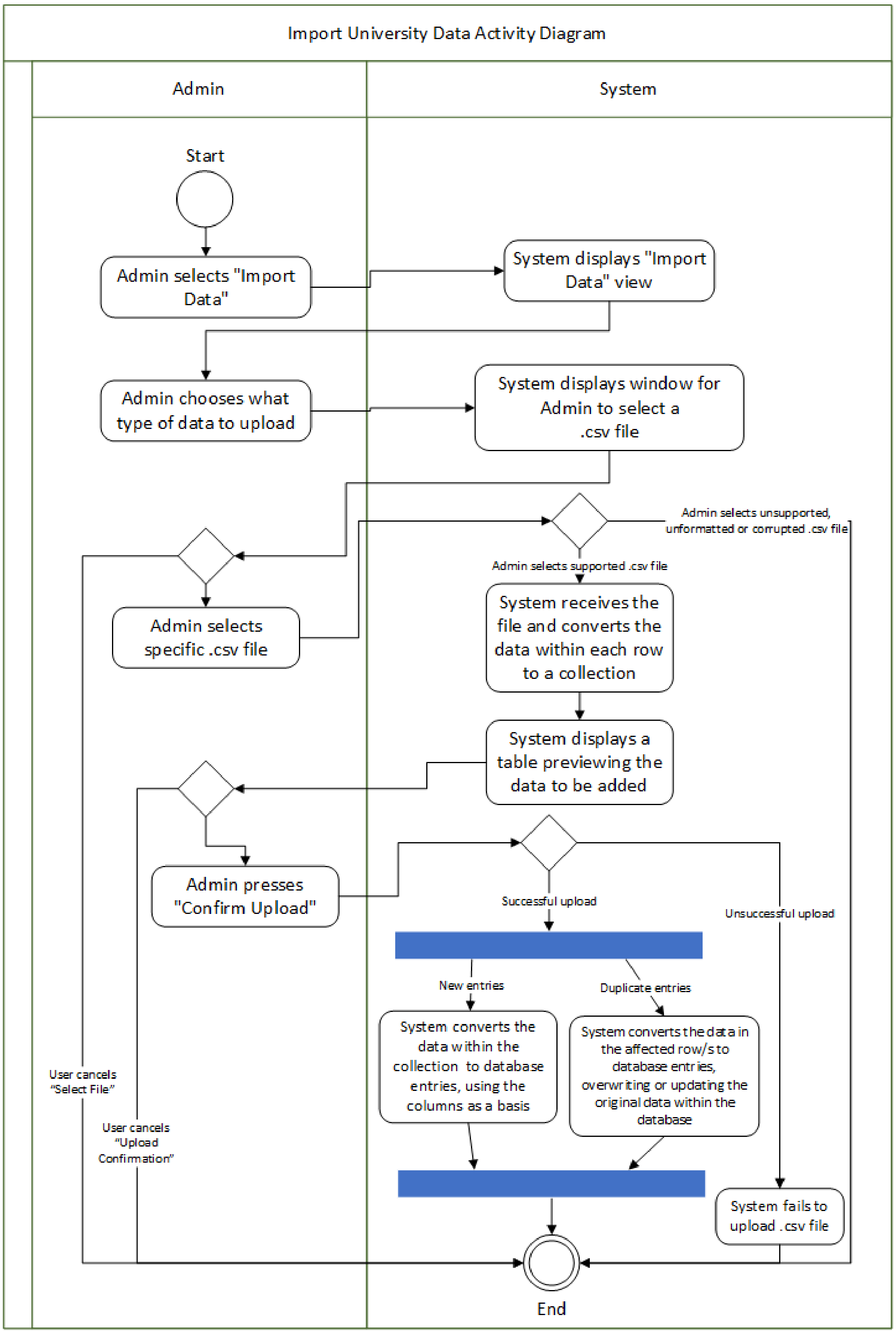

Import University Data Activity Diagram
Admin
System
Start
Admin selects "Import Data"
System displays "Import Data" view
Admin chooses what type of data to upload
System displays window for Admin to select a .csv file
Admin selects unsupported, unformatted or corrupted .csv file
Admin selects supported .csv file
Admin selects specific .csv file
System receives the file and converts the data within each row to a collection
System displays a table previewing the data to be added
Admin presses "Confirm Upload"
Successful upload
Unsuccessful upload
New entries
Duplicate entries
System converts the data within the collection to database entries, using the columns as a basis
System converts the data in the affected row/s to database entries, overwriting or updating the original data within the database
System fails to upload .csv file
User cancels "Select File"
User cancels "Upload Confirmation"
End

Figure 2.4: ***Import University Data Activity Diagram***

Figure 2.4 shows the flow and behavior of the system regarding the *Import University Data* use case, in which the admin uploads a .csv file containing the various data of a university during a given school year while the system imports the data as entries to the database.

Activity Diagram 3

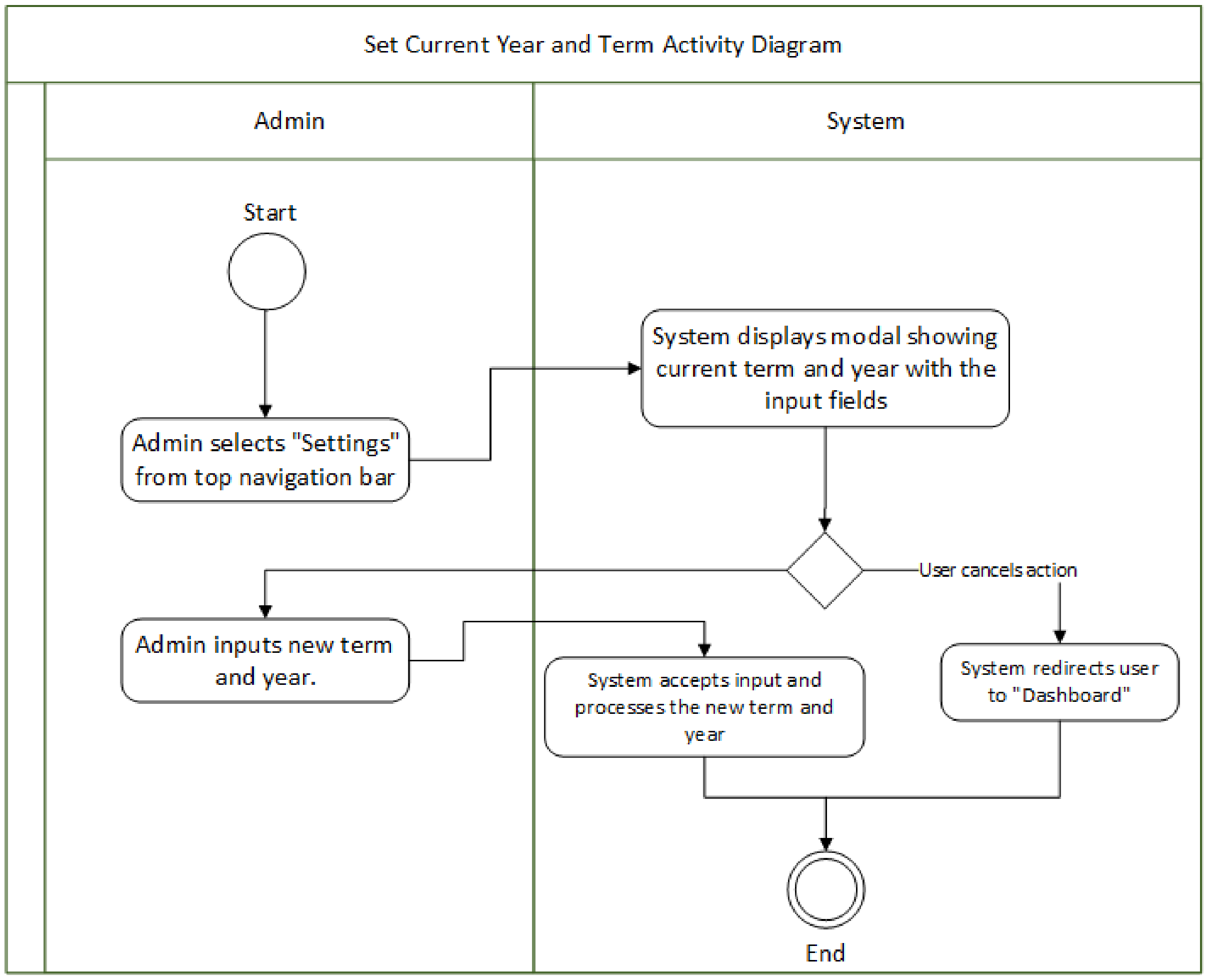

Set Current Year and Term Activity Diagram
Admin
System
Start
Admin selects "Settings" from top navigation bar
System displays modal showing current term and year with the input fields
User cancels action
Admin inputs new term and year.
System accepts input and processes the new term and year
System redirects user to "Dashboard"
End

Figure 2.5: *Set Current Year and Term Activity Diagram*

Figure 2.5 shows the flow and behavior of the system concerning the *Set Current and Year* use case, in which the user sets the term and year of the system to match those of the university's.

Activity Diagram 4

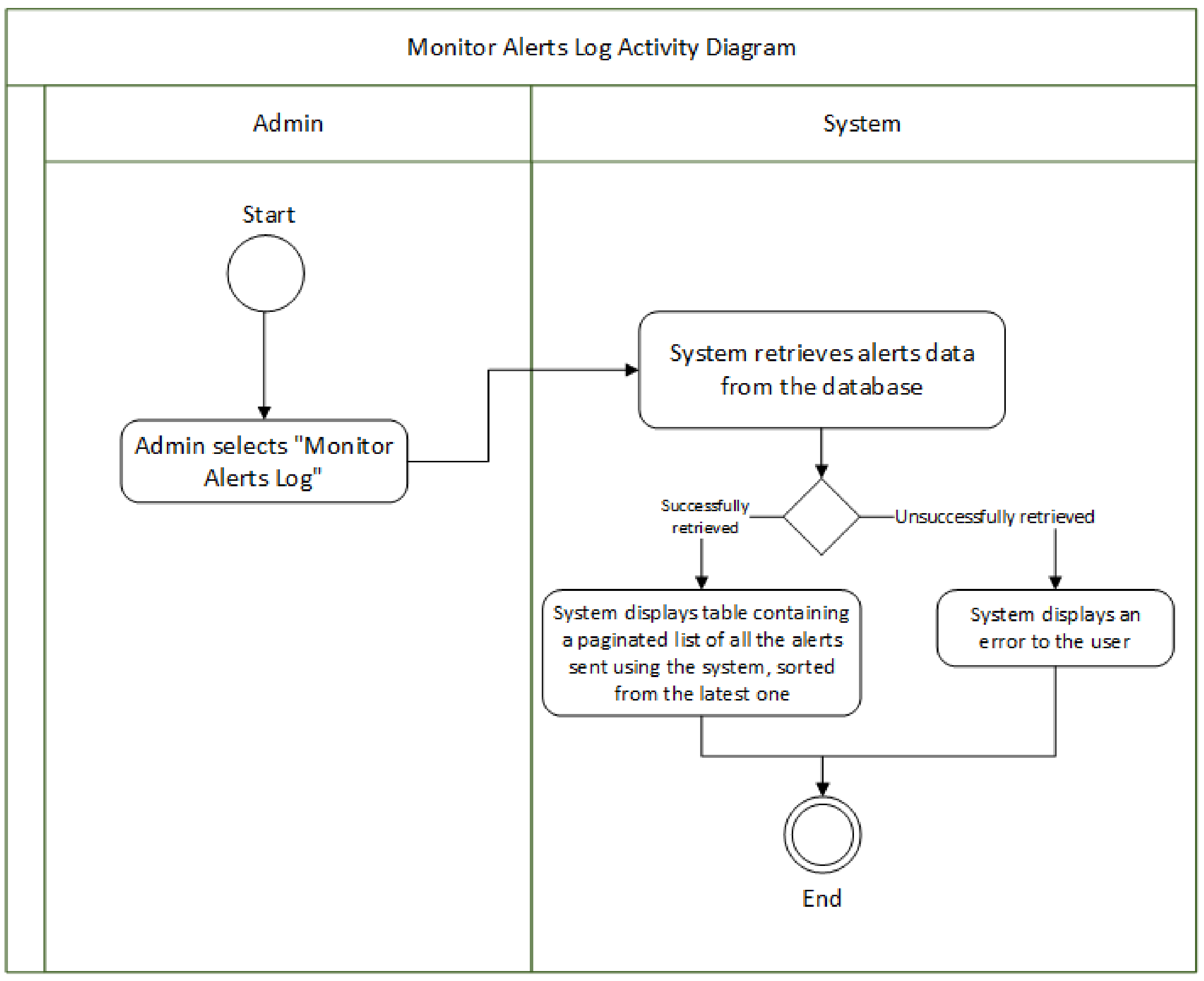

Monitor Alerts Log Activity Diagram
Admin
System
Start
Admin selects "Monitor Alerts Log"
System retrieves alerts data from the database
Successfully retrieved
Unsuccessfully retrieved
System displays table containing a paginated list of all the alerts sent using the system, sorted from the latest one
System displays an error to the user
End

Figure 2.6: ***Monitor Alerts Log Activity Diagram***

Figure 2.6 shows the flow and behavior of the system regarding the *Monitor Alerts Log* use case, in which the system retrieves alert data from the database and displays it on screen for the admin.

Activity Diagram 5

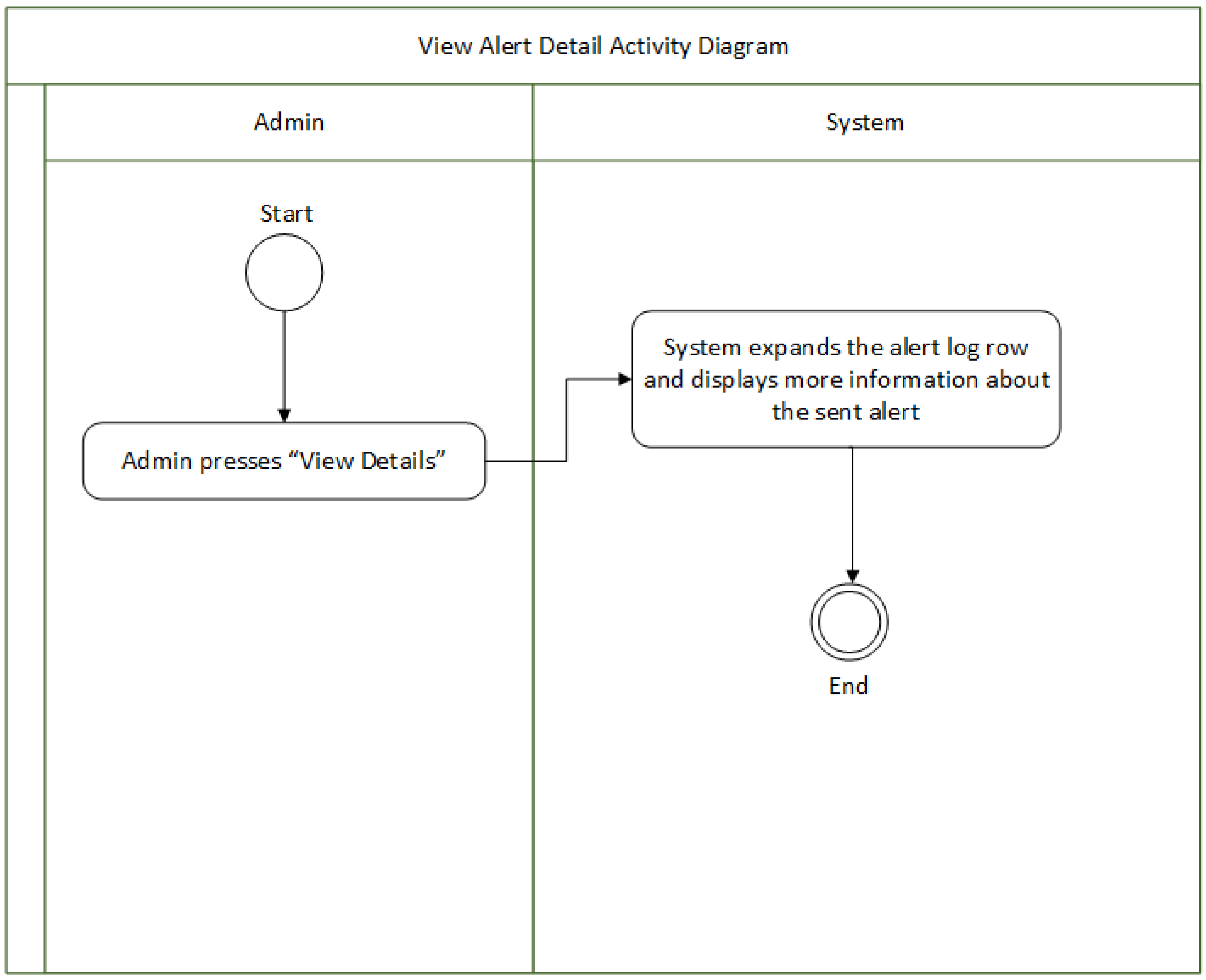

View Alert Detail Activity Diagram
Admin
System
Start
Admin presses "View Details"
System expands the alert log row and displays more information about the sent alert
End

Figure 2.7: *View Alert Detail Activity Diagram*

Figure 2.7 shows the flow and behavior of the system regarding the *View Alert Detail* use case, in which the system displays more information about a specific alert log as requested by the admin.

Activity Diagram 6

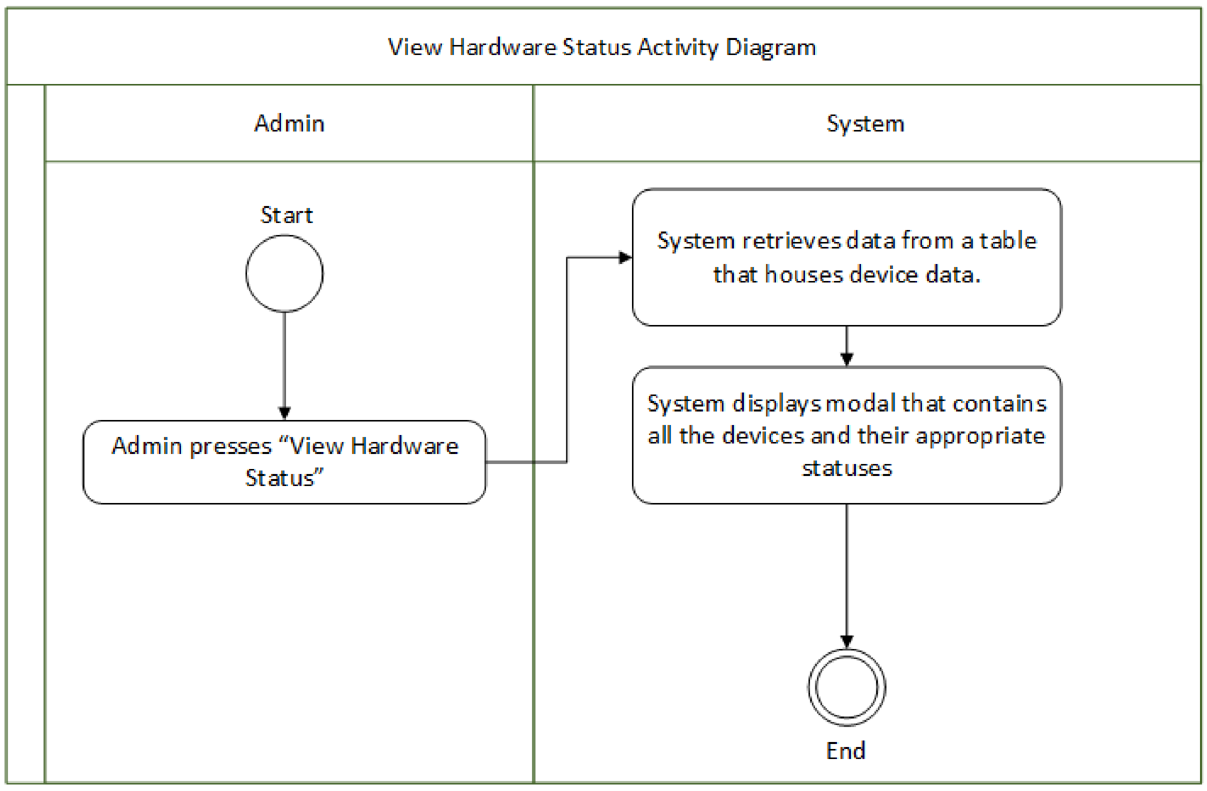

View Hardware Status Activity Diagram
Admin
System
Start
Admin presses "View Hardware Status"
System retrieves data from a table that houses device data.
System displays modal that contains all the devices and their appropriate statuses
End

Figure 2.8: ***View Hardware Status Activity Diagram***

Figure 2.8 shows the flow and behavior of the system concerning the *View Hardware Status* use case, in which the system displays a modal that shows the device statuses.

Activity Diagram 7

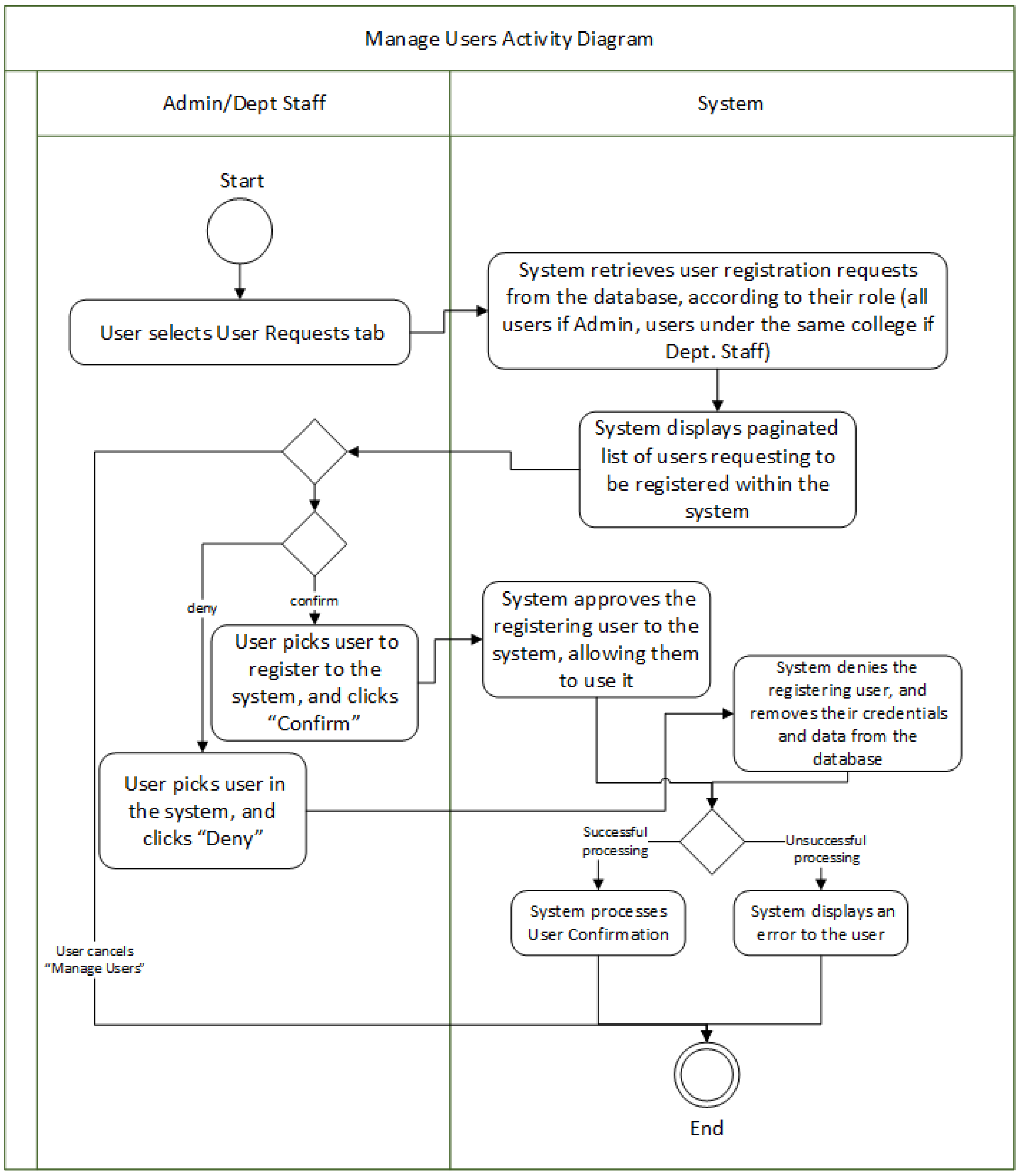

Manage Users Activity Diagram
Admin/Dept Staff
System
Start
User selects User Requests tab
System retrieves user registration requests from the database, according to their role (all users if Admin, users under the same college if Dept. Staff)
System displays paginated list of users requesting to be registered within the system
deny
confirm
User picks user to register to the system, and clicks "Confirm"
User picks user in the system, and clicks "Deny"
System approves the registering user to the system, allowing them to use it
System denies the registering user, and removes their credentials and data from the database
Successful processing
Unsuccessful processing
System processes User Confirmation
System displays an error to the user
User cancels "Manage Users"
End

Figure 2.9: ***Manage Users Activity Diagram***

Figure 2.9 shows the flow and behavior of the system concerning the *Manage Users* use case. In which the administrator or department staff allows or denies access to the registering users to the system.

Activity Diagram 8

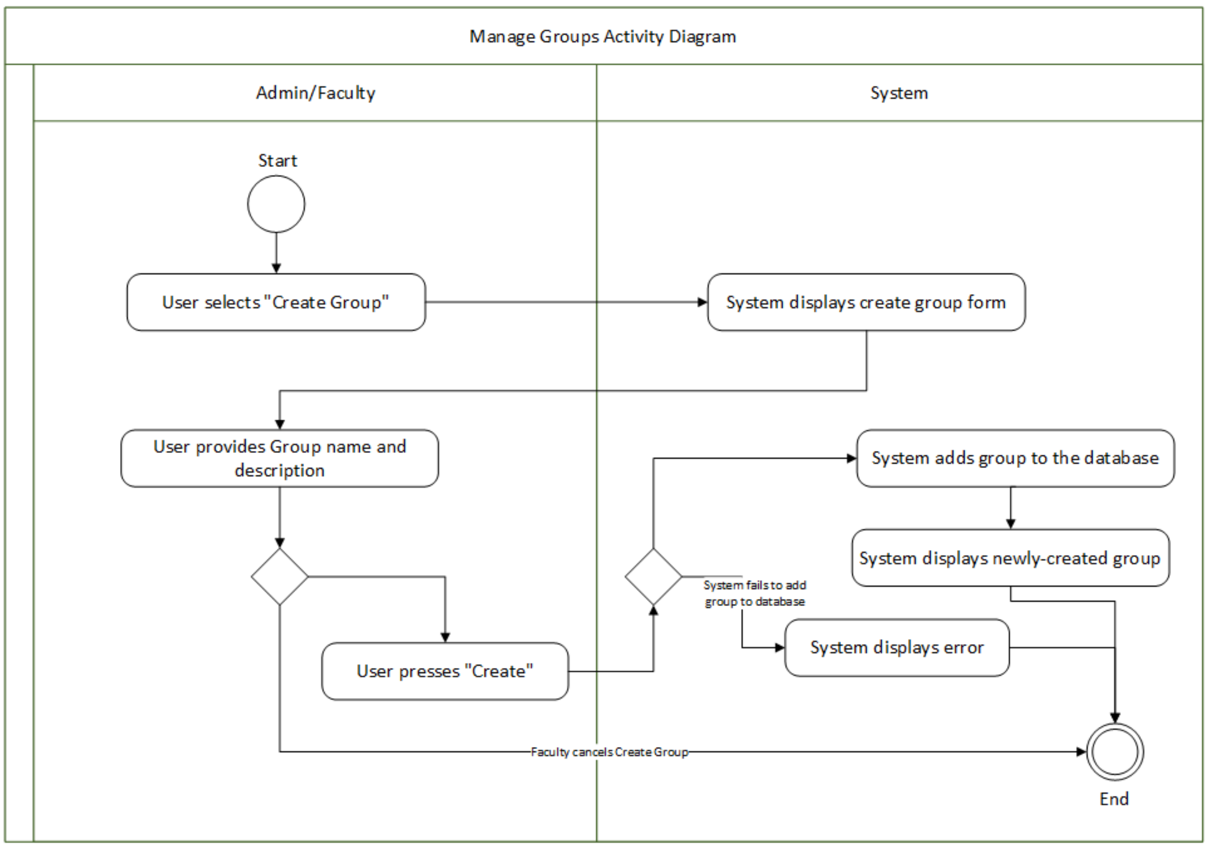

Manage Groups Activity Diagram
Admin/Faculty
System
Start
User selects "Create Group"
System displays create group form
User provides Group name and description
System adds group to the database
System displays newly-created group
System fails to add group to database
System displays error
User presses "Create"
Faculty cancels Create Group
End

Figure 2.10: ***Manage Groups Activity Diagram***

Figure 2.10 shows the flow and behavior of the system regarding the *Manage Groups* use case. A faculty user requests to create a new group, providing the group name and description, and the system adds the created group to the database.

Activity Diagram 9

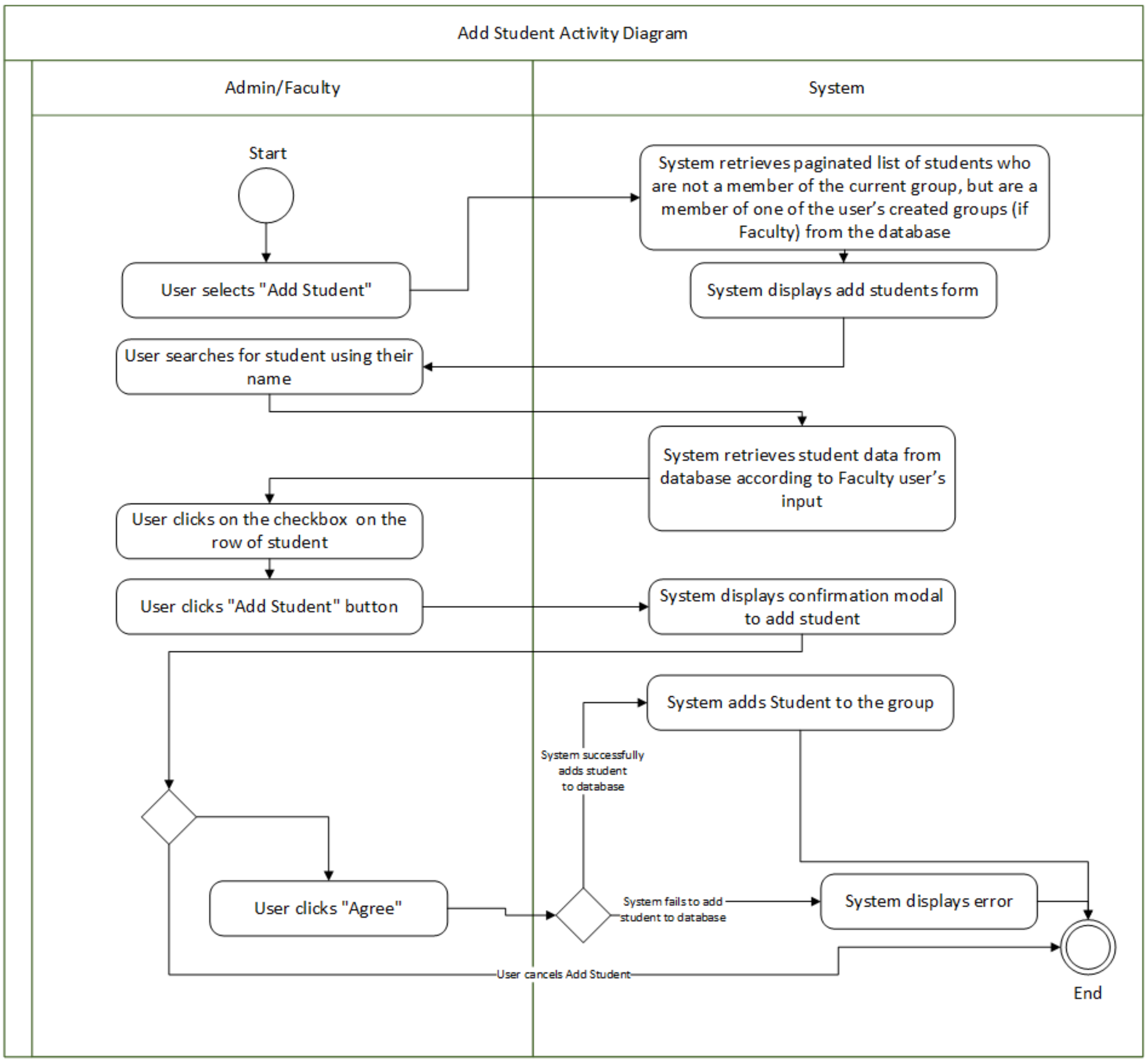

Add Student Activity Diagram
Admin/Faculty
System
Start
User selects "Add Student"
System retrieves paginated list of students who are not a member of the current group, but are a member of one of the user's created groups (if Faculty) from the database
System displays add students form
User searches for student using their name
System retrieves student data from database according to Faculty user's input
User clicks on the checkbox on the row of student
User clicks "Add Student" button
System displays confirmation modal to add student
System adds Student to the group
System successfully adds student to database
User clicks "Agree"
System fails to add student to database
System displays error
User cancels Add Student
End

Figure 2.11: ***Add Students Activity Diagram***

Figure 2.11 shows the flow and behavior of the system concerning the *Add Student* use case. In which a faculty user requests to add a student to their group by searching for and selecting the student.

Activity Diagram 10

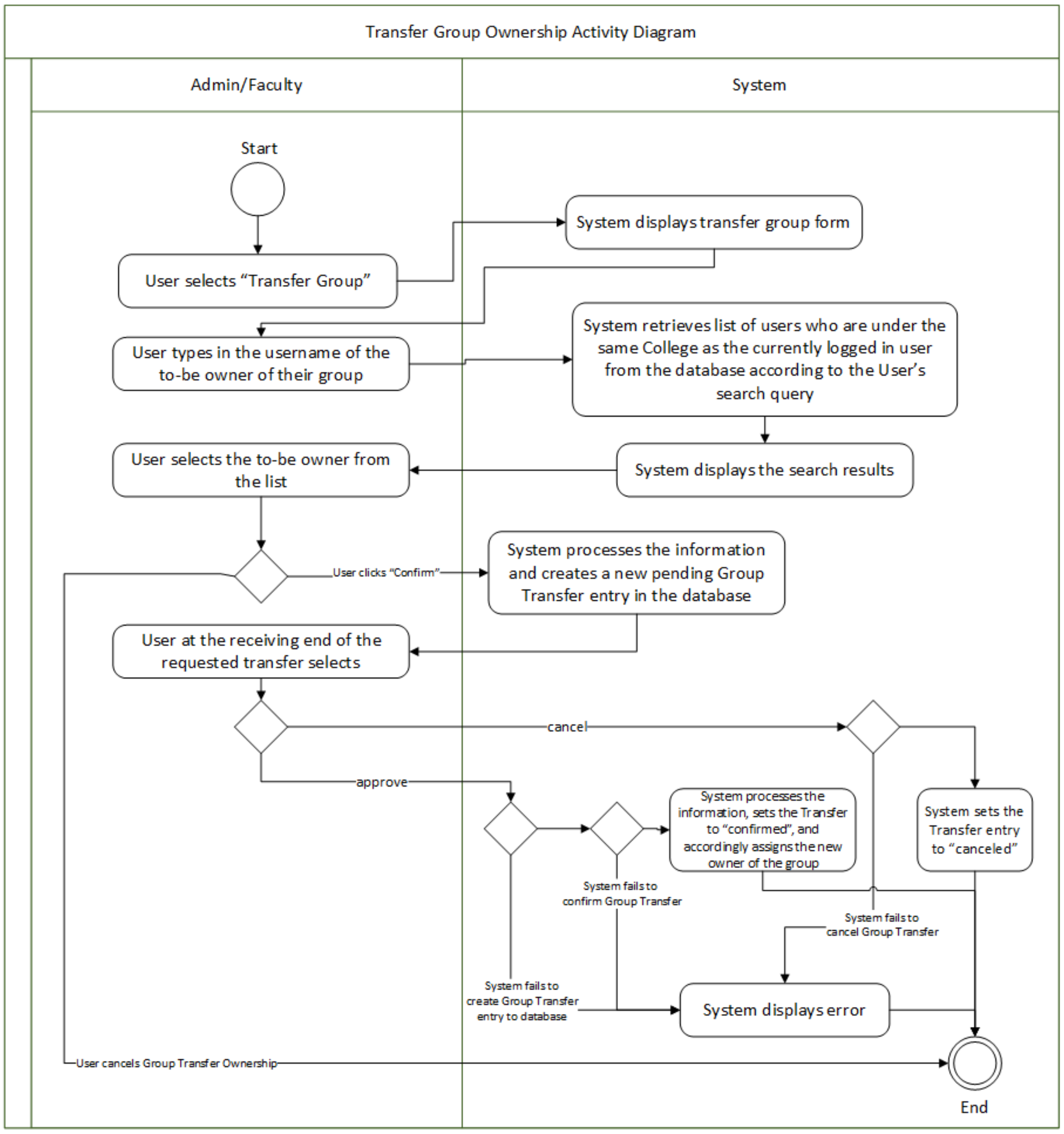

Transfer Group Ownership Activity Diagram
Admin/Faculty
System
Start
User selects "Transfer Group"
System displays transfer group form
User types in the username of the to-be owner of their group
System retrieves list of users who are under the same College as the currently logged in user from the database according to the User's search query
System displays the search results
User selects the to-be owner from the list
User clicks "Confirm"
System processes the information and creates a new pending Group Transfer entry in the database
User at the receiving end of the requested transfer selects
cancel
approve
System processes the information, sets the Transfer to "confirmed", and accordingly assigns the new owner of the group
System sets the Transfer entry to "canceled"
System fails to confirm Group Transfer
System fails to cancel Group Transfer
System fails to create Group Transfer entry to database
System displays error
User cancels Group Transfer Ownership
End

Figure 2.12: ***Transfer Group Ownership Activity Diagram***

Figure 2.12 shows the flow and behavior of the system concerning the *Transfer Group Ownership* use case. A faculty user selects a specific group and transfers the ownership to another user.

Activity Diagram 11

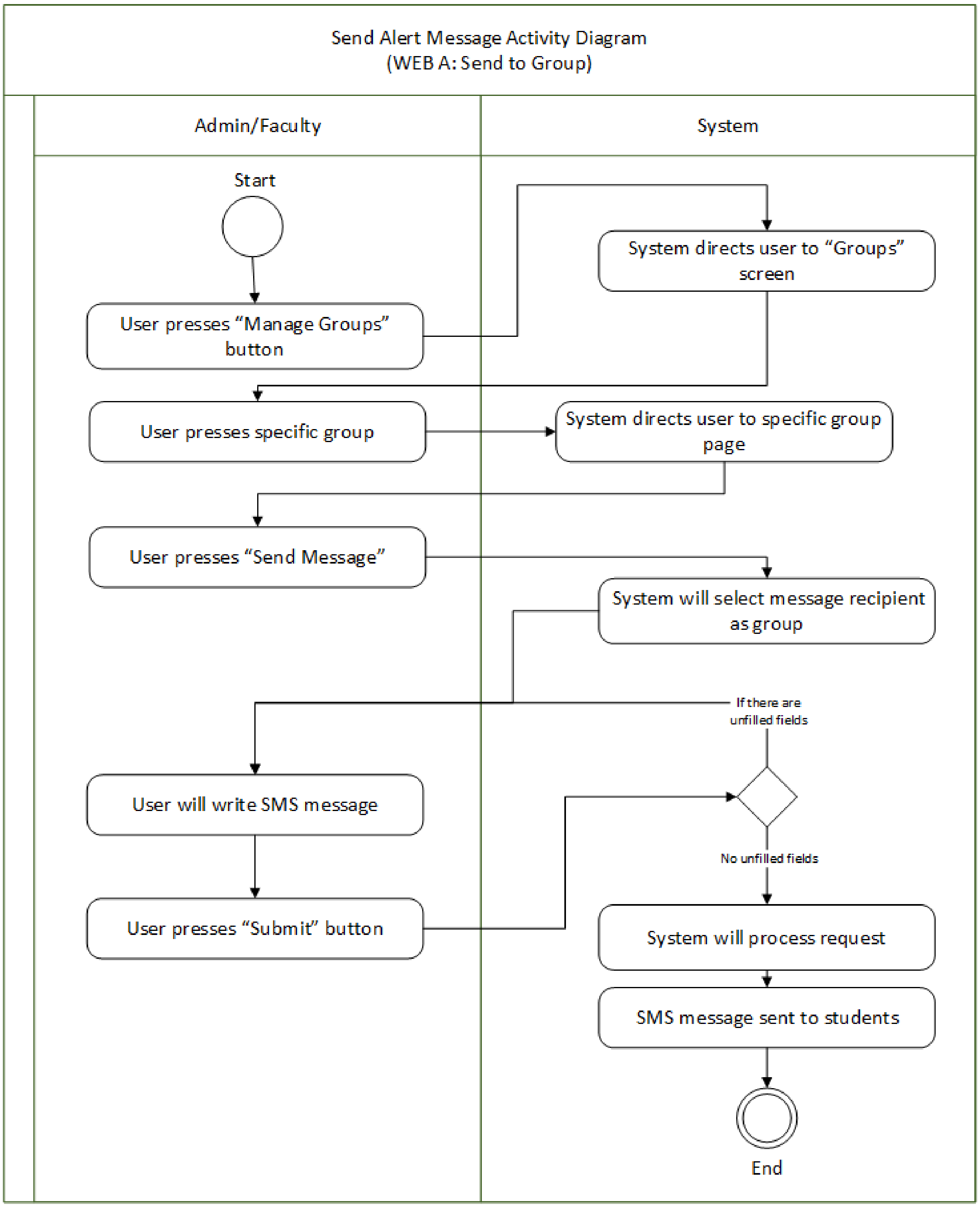

Send Alert Message Activity Diagram
(WEB A: Send to Group)
Admin/Faculty
System
Start
User presses "Manage Groups" button
System directs user to "Groups" screen
User presses specific group
System directs user to specific group page
User presses "Send Message"
System will select message recipient as group
If there are unfilled fields
User will write SMS message
User presses "Submit" button
No unfilled fields
System will process request
SMS message sent to students
End

Figure 2.13.1: *Send Alert Message Activity Diagram (WEB A: Send to Group)*

Figure 2.13.1 shows the flow and behavior of the system concerning the *Send Alert Message* use case. In this figure, the use case is done through the web application groups screen, in which the user requests to send a new alert to all the students in the group, and the system processes the request.

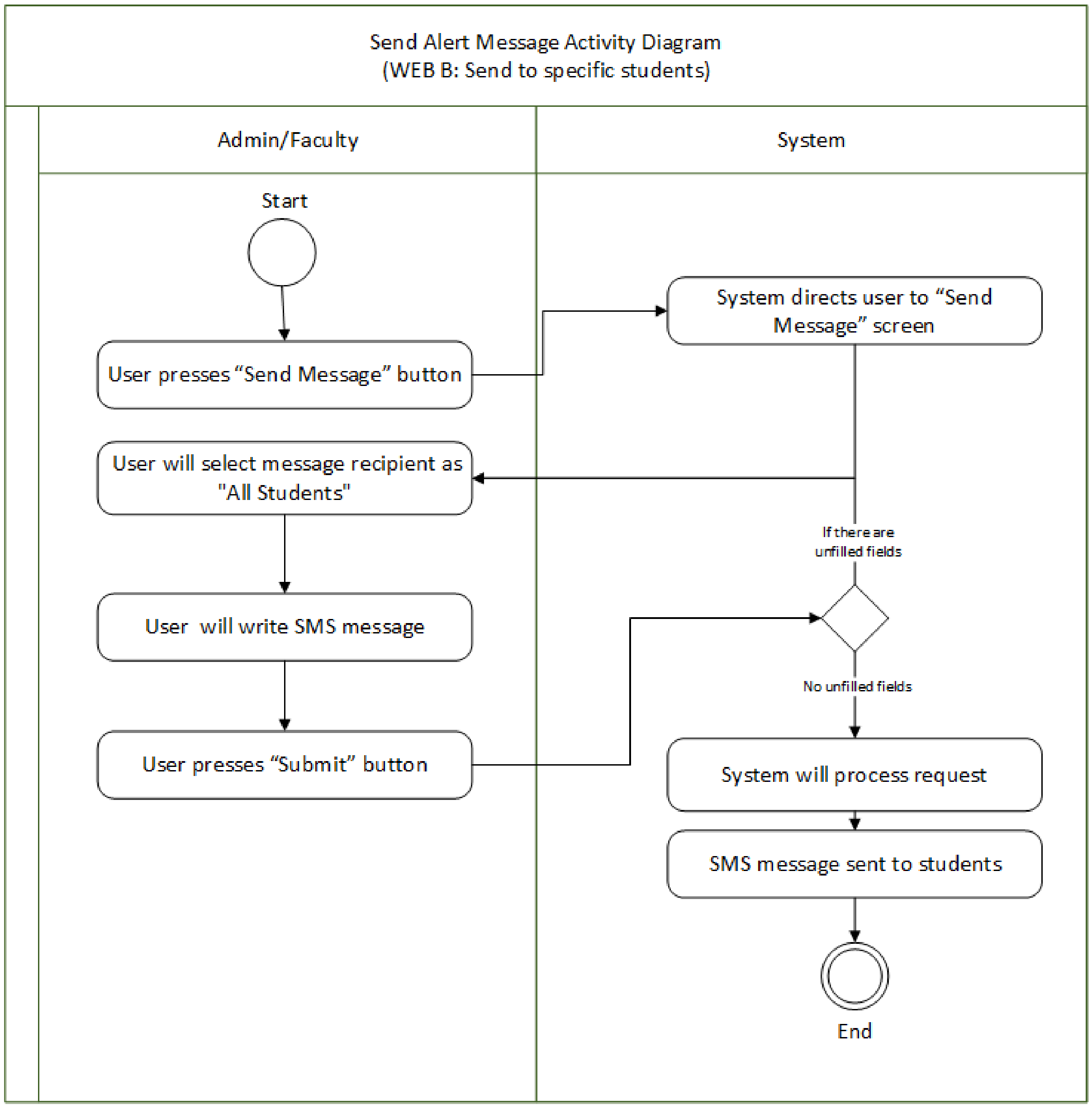

Send Alert Message Activity Diagram
(WEB B: Send to specific students)
Admin/Faculty
System
Start
System directs user to "Send Message" screen
User presses "Send Message" button
User will select message recipient as "All Students"
If there are unfilled fields
User will write SMS message
No unfilled fields
User presses "Submit" button
System will process request
SMS message sent to students
End

Figure 2.13.2: ***Send Alert Message Activity Diagram (WEB B:Send to specific students)***

Figure 2.13.2 shows the flow and behavior of the system concerning the *Send Alert Message* use case. In this figure, the use case is done through the home page screen of the web application using the "Send Message" tile, in which the faculty requests to send a new alert to selected students, and the system processes the request.

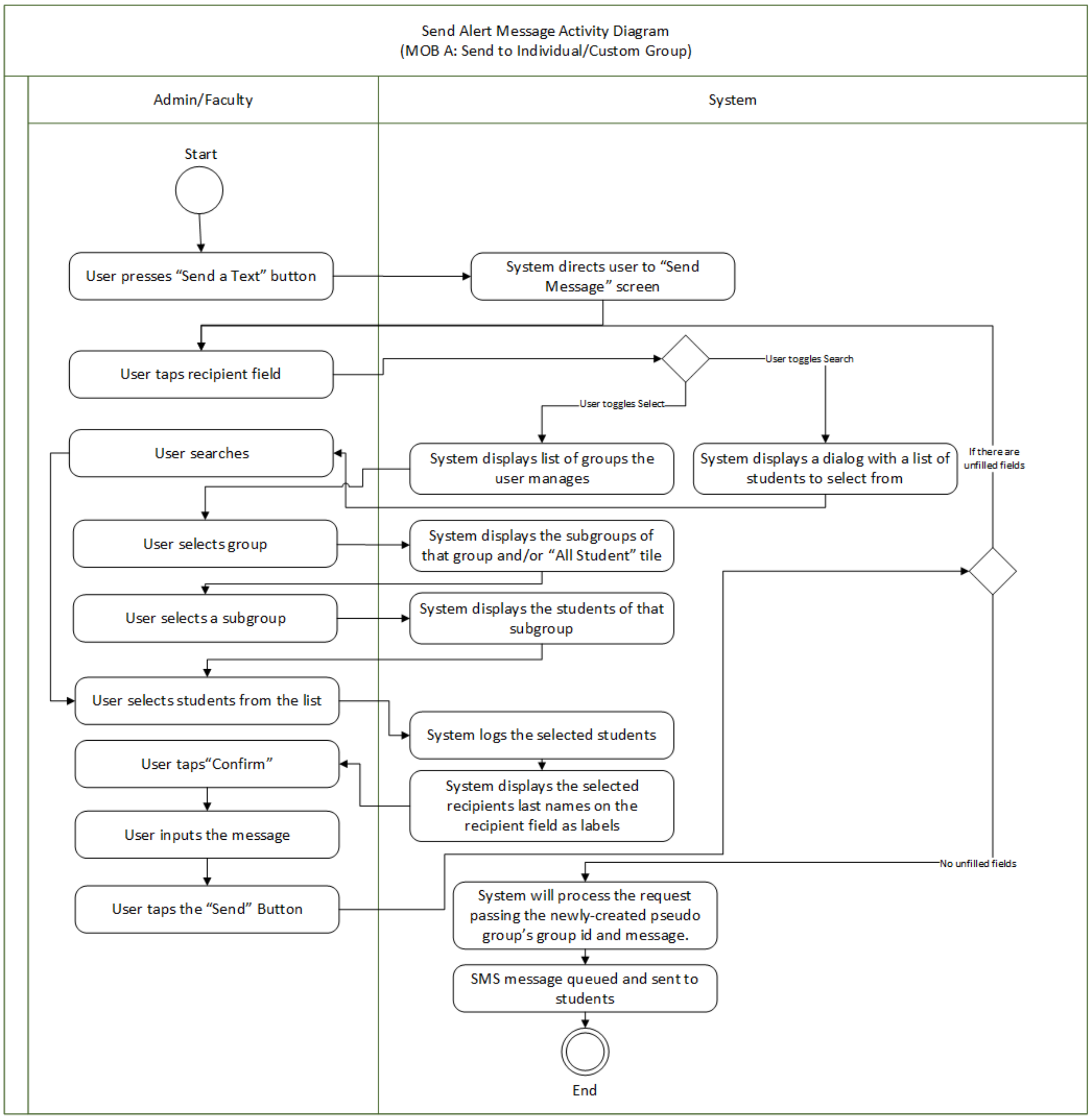

Send Alert Message Activity Diagram
(MOB A: Send to Individual/Custom Group)
Admin/Faculty
System
Start
User presses "Send a Text" button
System directs user to "Send Message" screen
User taps recipient field
User toggles Search
User toggles Select
User searches
System displays list of groups the user manages
System displays a dialog with a list of students to select from
If there are unfilled fields
User selects group
System displays the subgroups of that group and/or "All Student" tile
User selects a subgroup
System displays the students of that subgroup
User selects students from the list
System logs the selected students
User taps"Confirm"
System displays the selected recipients last names on the recipient field as labels
User inputs the message
No unfilled fields
User taps the "Send" Button
System will process the request passing the newly-created pseudo group's group id and message.
SMS message queued and sent to students
End

Figure 2.13.3: ***Send to Individual/Custom Group Activity Diagram (MOB A)***

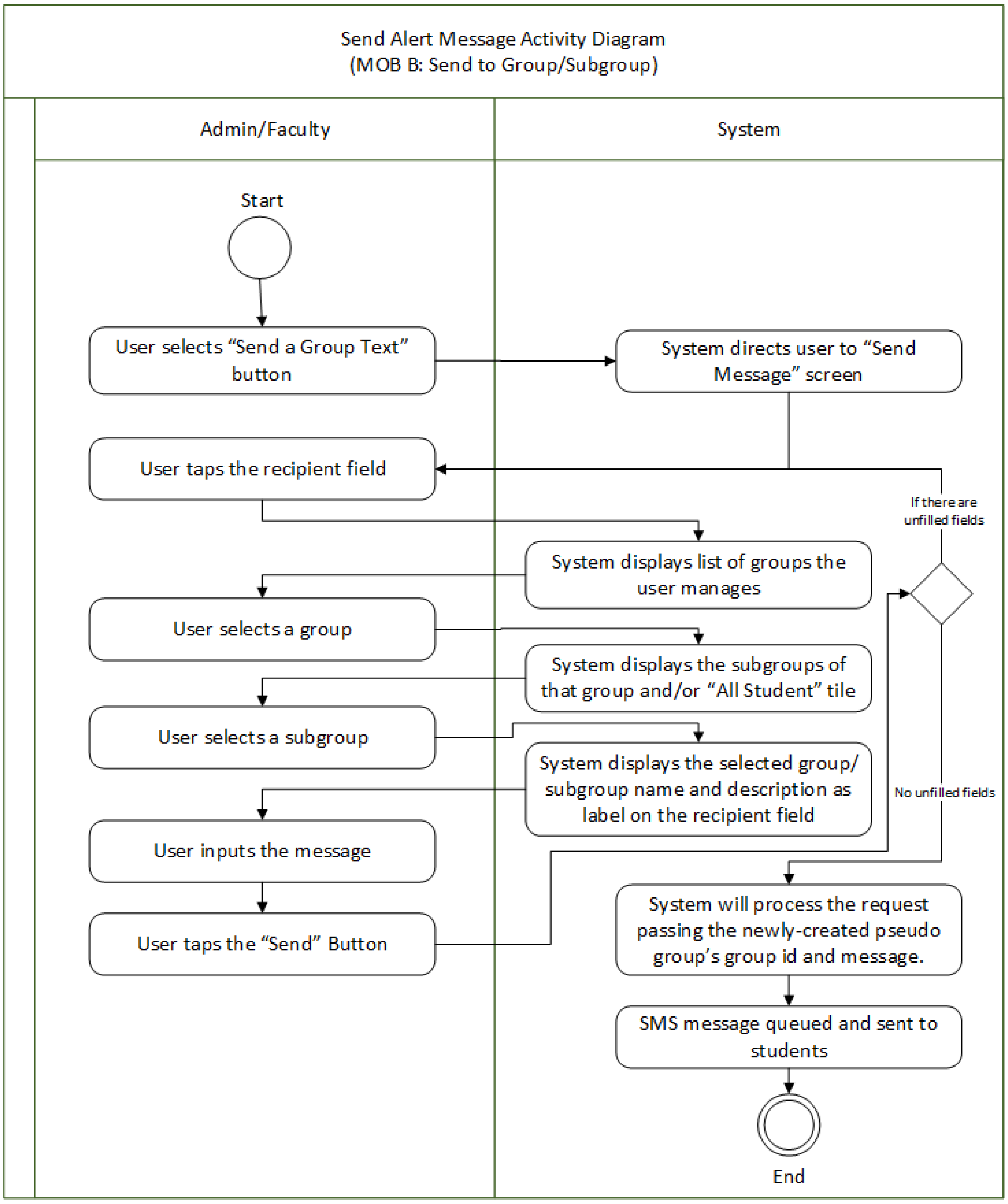

Send Alert Message Activity Diagram
(MOB B: Send to Group/Subgroup)
Admin/Faculty
System
Start
User selects "Send a Group Text" button
System directs user to "Send Message" screen
User taps the recipient field
If there are unfilled fields
System displays list of groups the user manages
User selects a group
System displays the subgroups of that group and/or "All Student" tile
User selects a subgroup
System displays the selected group/ subgroup name and description as label on the recipient field
No unfilled fields
User inputs the message
User taps the "Send" Button
System will process the request passing the newly-created pseudo group's group id and message.
SMS message queued and sent to students
End

Figure 2.13.4: ***Send to Individual/Custom Group Activity Diagram (MOB B)***

Figure 2.13.3 and Figure 2.13.4 shows the flow and behavior of the system concerning the *Send Alert Message* use case. In this figure, the use case is done through the mobile application, in which the faculty requests to send a new alert to recipients, and the system processes the request.

Activity Diagram 12

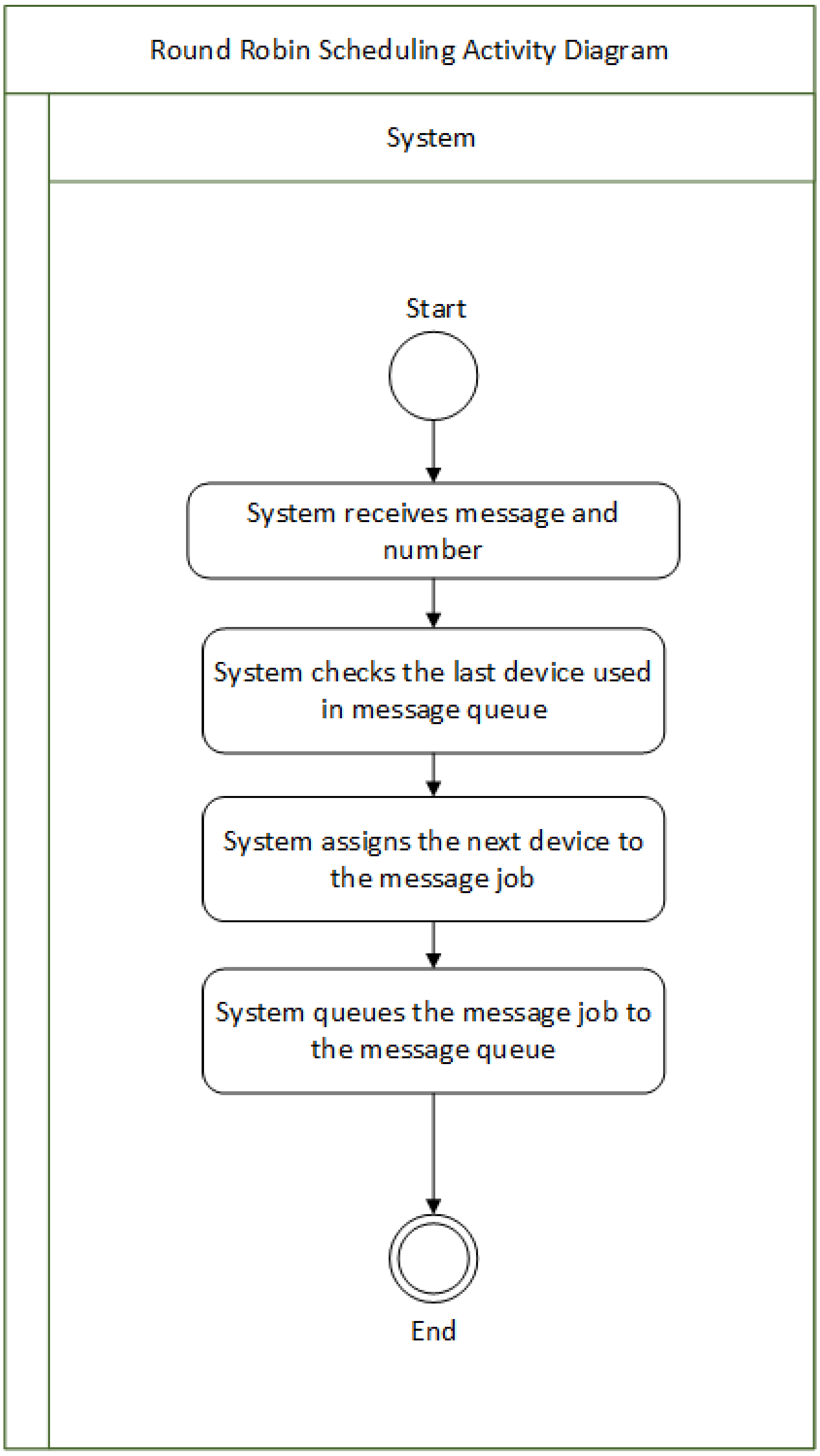

Round Robin Scheduling Activity Diagram
System
Start
System receives message and number
System checks the last device used in message queue
System assigns the next device to the message job
System queues the message job to the message queue
End

Figure 2.14: ***Round Robin Scheduling Activity Diagram***

Figure 2.14 shows the flow and behavior of the system concerning the *Round Robin Scheduling* use case. In which the system implements an algorithm to prevent sending numerous alerts simultaneously.

Activity Diagram 13

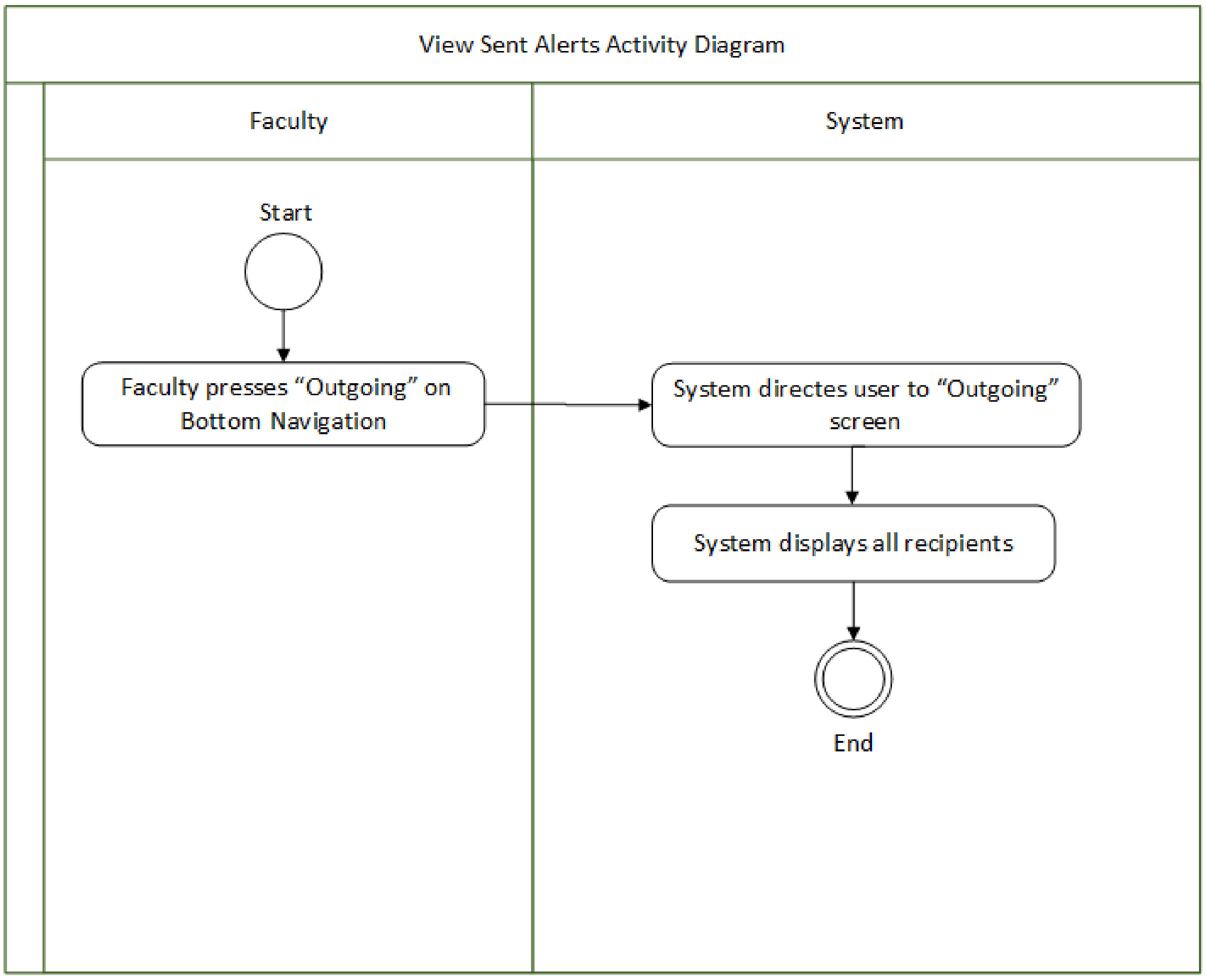

View Sent Alerts Activity Diagram
Faculty
System
Start
Faculty presses "Outgoing" on Bottom Navigation
System directes user to "Outgoing" screen
System displays all recipients
End

Figure 2.15: ***View Sent Alerts Activity Diagram***

Figure 2.15 shows the flow and behavior of the system concerning the *View Sent Alerts* use case. A faculty user requests a record of all the alerts sent to the system.

Activity Diagram 14

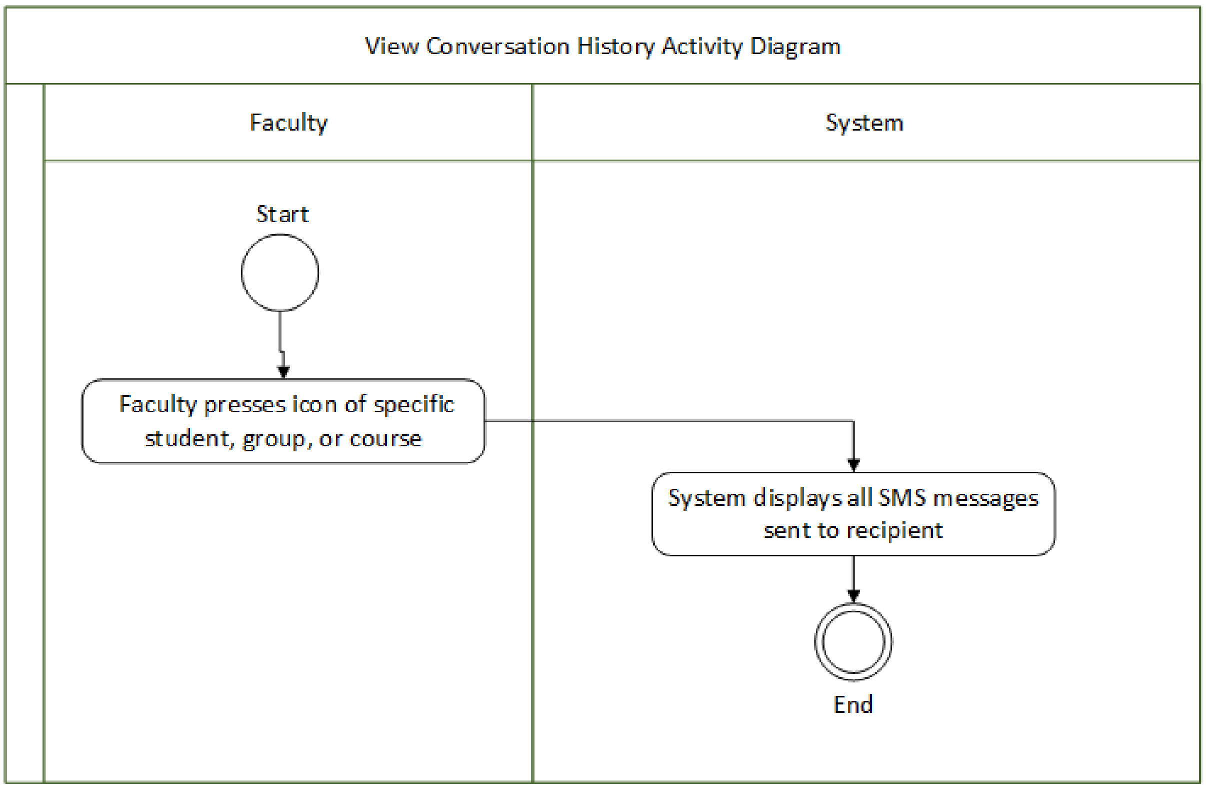

View Conversation History Activity Diagram
Faculty
System
Start
Faculty presses icon of specific student, group, or course
System displays all SMS messages sent to recipient
End

Figure 2.16: ***View Conversation History Activity Diagram***

Figure 2.16 shows the flow and behavior of the system concerning the *View Conversation History* use case. A faculty user requests to get their conversation history with a specific recipient/s and the system retrieves and displays the messages on screen.

Entity Relationship Diagram

The entity relationship diagram displays the different entities used in the system, along with their respective attributes and relationships.

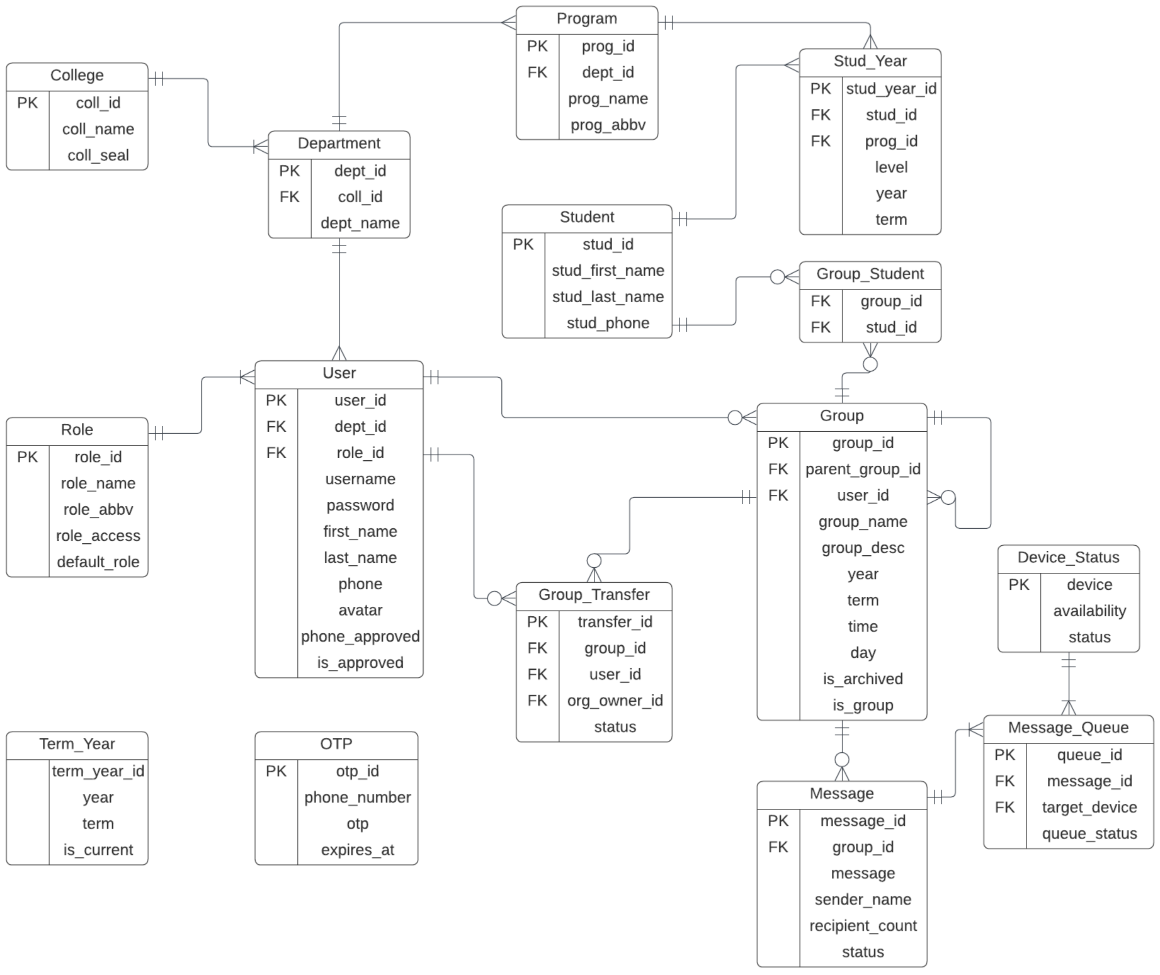


Figure 2.17: ***System Entity Relationship Diagram***

Figure 2.17 shows the numerous entities involved in the system.

Data Dictionary

The data dictionary comprises a more detailed view of the names, definitions, and data attributes of the entities in the system.

Table Name: College

Description: Stores data for the different colleges in the university

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
coll_id	The primary key ID and abbreviation of the college (e.g., SCS)	String		True	False
coll_name	The full college name (e.g. School of Computer Studies)	String	100	True	False
coll_seal	The file name of the college seal image uploaded to the system’s storage	String		False	True

Table 2.15: ***College Data Dictionary***

Table Name: Department

Description: Stores data for the different university departments

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
dept_id	The primary key ID and abbreviation of the department (e.g. ICST)	String		True	False
coll_id	Foreign key of the abbreviation for the college where the department belongs	String		True	False
dept_name	The full department name	String		True	False

Table 2.16: ***Department Data Dictionary***

Table Name: Program

Description: Stores data for the different programs offered by a given department

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
prog_id	The primary key ID of the program entity in integer	Integer		True	False
dept_id	Foreign key of the abbreviation for the department where the program is offered	String		True	False
prog_name	The full department name (e.g., Information Technology)	String	100	True	False
prog_abbv	The abbreviated program name (e.g. BSIT)	String	45	True	False

Table 2.17: ***Program Entity Data Dictionary***

Table Name: Student

Description: Stores data for all the students in the university

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
stud_id	The primary key school ID of the student	String		True	False
stud_first_name	The first name of the student	String		True	False

stud_last_name	The last name of the student	String		True	False
stud_phone	The student’s contact information to be used in sending the messages	String	12	False	True

Table 2.18: ***Student Entity Data Dictionary***

Table Name: Stud_Year

Description: Stores data regarding the student information for a given semester

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
stud_year_id	The primary key ID of the student year entity in integer	Integer		True	False
stud_id	Foreign key of the student’s school ID	String		True	False
prog_id	Foreign key of the program the student is enrolled in for the semester	String		True	False
level	The level of the student for the semester	Integer		True	False
year	The year of the semester	String		True	False
term	The term the student is enrolled in (e.g. 1st Sem.)	String		True	False

Table 2.19: ***Stud_Year Entity Data Dictionary***

Table Name: Group

Description: Stores data regarding the different groups in the system, which may be based on the offerings handled by a faculty for a given semester or

custom groups such as a group containing students under a specific college, degree program, or year level

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
group_id	The primary key ID of the group entity in integer	Integer		True	True
parent_group_id	Nullable ID which serves as an indication that a group exists within another group as a subgroup	Integer		False	True
user_id	Foreign key of the user handling the group	Integer		True	False
group_name	The name of the group (e.g., 11011, SCS, BSIT-4)	String	65	True	False
group_desc	The description of the group (e.g., Capstone 2, Programming 1)	String	240	False	True
year	The year of the semester the group was created	String		True	False
term	The term the group was created (e.g., 1st Sem.)	String		False	False
time	The scheduled time for the group or offering (e.g., 1:00 pm to 3:00 pm)	String		False	True
day	The scheduled day for the group or offering (e.g., MWF)	String		False	True
is_archived	Indication if the group is hidden or not	Boolean		False	False
is_group	Indication if the group is created through the	Boolean		False	False

	individual sending feature				

Table 2.20: ***Group Entity Data Dictionary***

Table Name: Group_Transfer

Description: Stores data for requests to transfer the ownership and access of a given group to another user

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
transfer_id	The primary key ID of the group transfer entity in integer	Integer		True	False
group_id	Foreign key ID of the group to be transferred	Integer		True	False
user_id	Foreign key ID of the new owner of the group	Integer		True	False
org_owner_id	Foreign key ID of the original owner of the group	Integer		True	False
status	The status of the transfer (i.e., Pending, Approved, Canceled)	String		True	False

Table 2.21: ***Group_Transfer Entity Data Dictionary***

Table Name: Term_Year

Description: Stores data of the term and year of the university

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
term_year_id	The primary key ID of the term year entity in	Integer		True	False

	integer				
year	The year of the semester	String		True	False
term	The term data itself	String		True	False
is_current	Indication for the current term and year in the university and the system	Boolean		True	False

Table 2.22: ***Term_Year Entity Data Dictionary***

Table Name: Message

Description: Stores data of the messages sent using the system

Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
message_id	The primary key ID of the message entity in integer	Integer		True	False
group_id	Foreign key ID of the group the message is sent to	Integer		True	False
message	The contents of the message	String	240	True	False
sender_name	The first and last name of the user who sent the message	String	50	True	False
recipient_count	The expected recipient count of the message	Integer		True	False
status	The status of the message (i.e., Queued, Sent)	String	12	True	False

Table 2.23: ***Message Entity Data Dictionary***

Table Name: Message_Queue

Description: Stores data for the messages queued through jobs

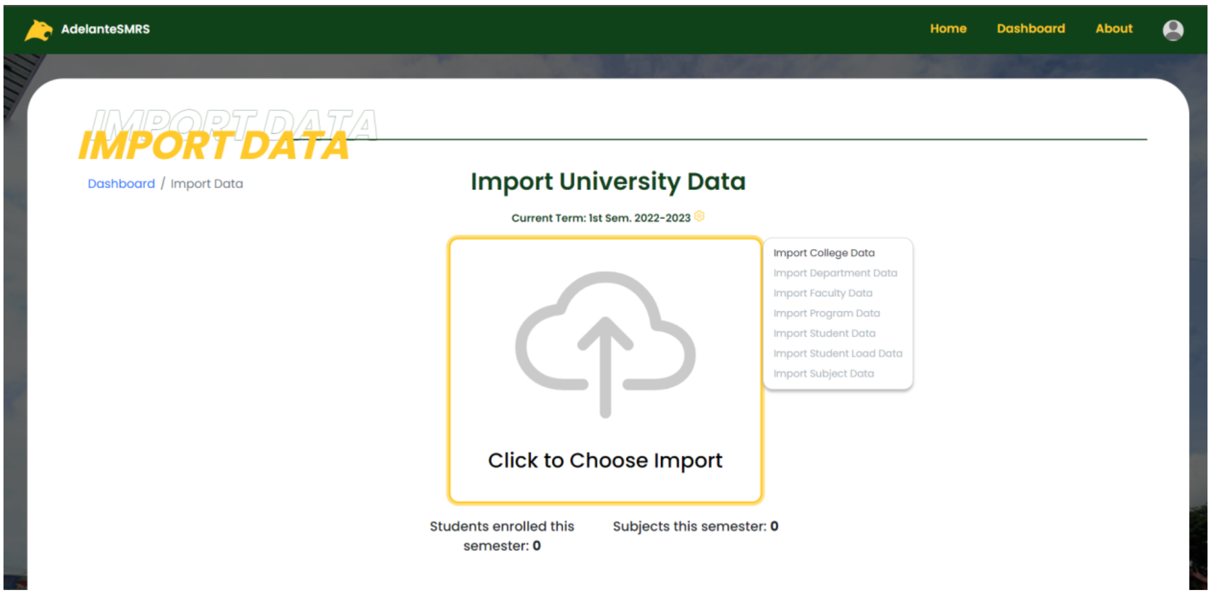
Field Name	**Description**	**Data Type**	**Field Size**	**Required**	**Nullable**
queue_id	The primary key of the message_queue entity in integer	Integer		True	False
message_id	Foreign key ID of the message queued	Integer		True	False
queue_status	The status of the message	String	12	True	False
target_device	The name of the device assigned to send the message	String		True	False

Table 2.24: ***Message_Queue Entity Data Dictionary***

User Interface Design

The user interface design represents the "look and feel" of the system, providing a graphic output that envelops and accompanies the different features or use cases.

Admin



Import University Data

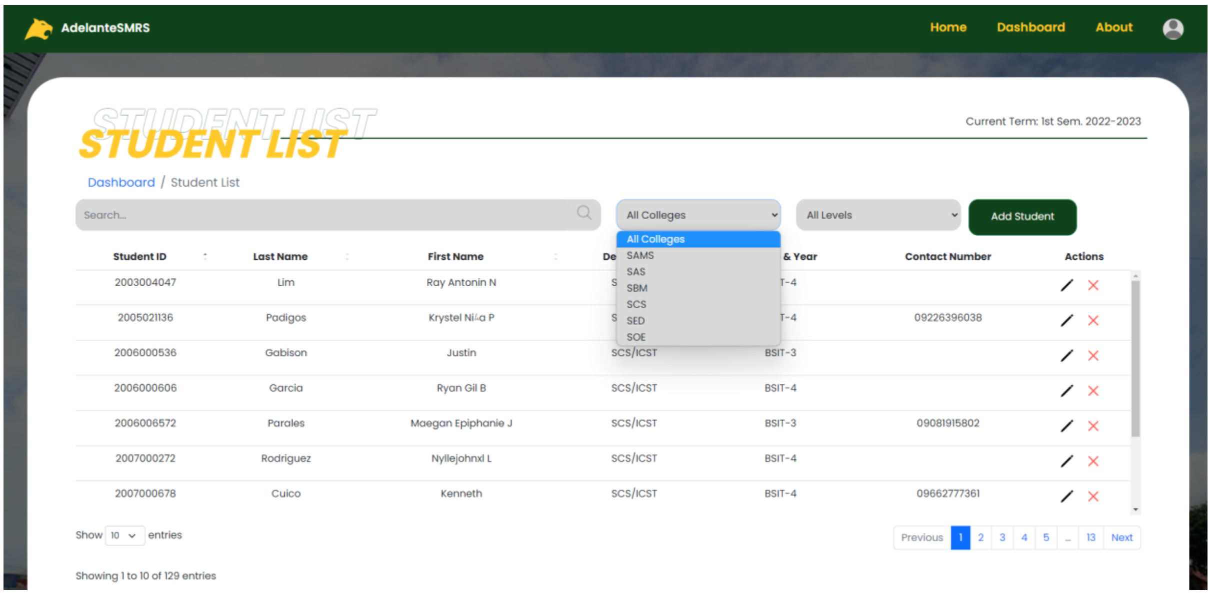
Current Term: 1st Sem. 2022-2023

Students enrolled this semester: 0

Subjects this semester: 0

Figure 2.18: ***Import University Data***

This screen shows how an Admin can import several data into the system, such as College, Department, Faculty, Program/Courses, Students, Student Load, and Subject Data. This functionality allows the automatic creation of courses belonging to a particular faculty and the students enrolled in that course.



Current Term: 1st Sem. 2022-2023

Student ID	Last Name	First Name	De	& Year	Contact Number	Actions
2003004047	Lim	Ray Antonin N	S	T-4		
2005021136	Padigos	Krystel Ni¼a P	S	T-4	09226396038	
2006000536	Gabison	Justin	SCS/ICST	BSIT-3		
2006000606	Garcia	Ryan Gil B	SCS/ICST	BSIT-4		
2006006572	Parales	Maegan Epiphanie J	SCS/ICST	BSIT-3	09081915802	
2007000272	Rodriguez	Nyllejohnxl L	SCS/ICST	BSIT-4		
2007000678	Cuico	Kenneth	SCS/ICST	BSIT-4	09662777361	

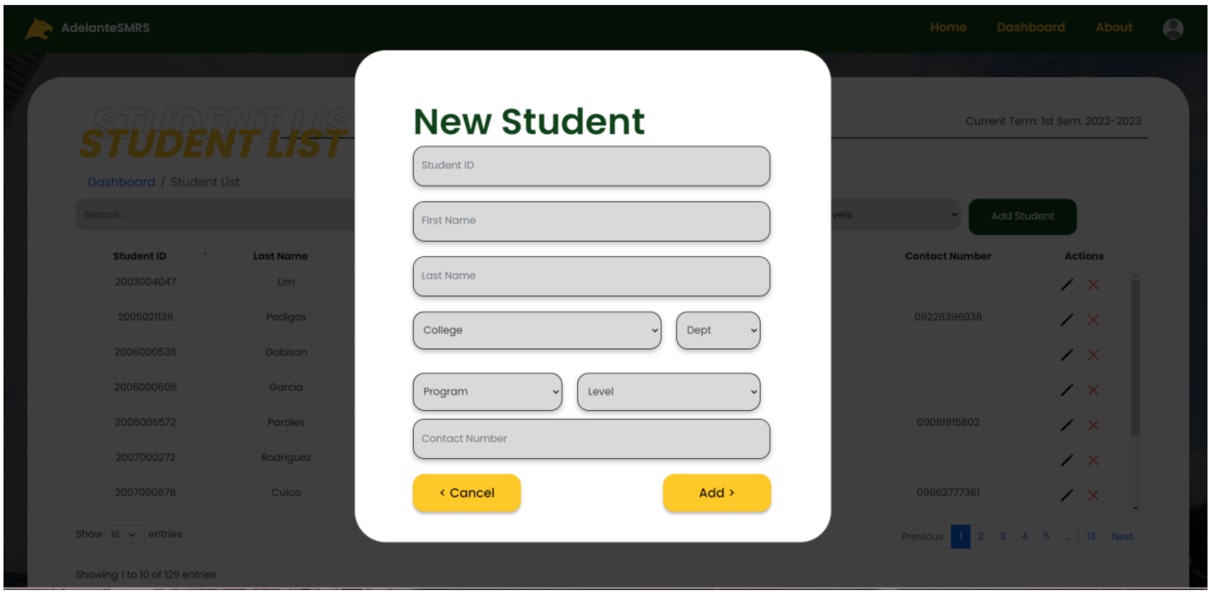
Show 10 entries

Previous 1 2 3 4 5 … 13 Next

Showing 1 to 10 of 129 entries

Figure 2.19: ***Manage Students - Student List***

This screen displays all the students of all colleges enrolled in the university for a particular term and year. The admin can filter the displayed students in this screen using the dropdown menu for 'All Colleges' and 'All Levels.'

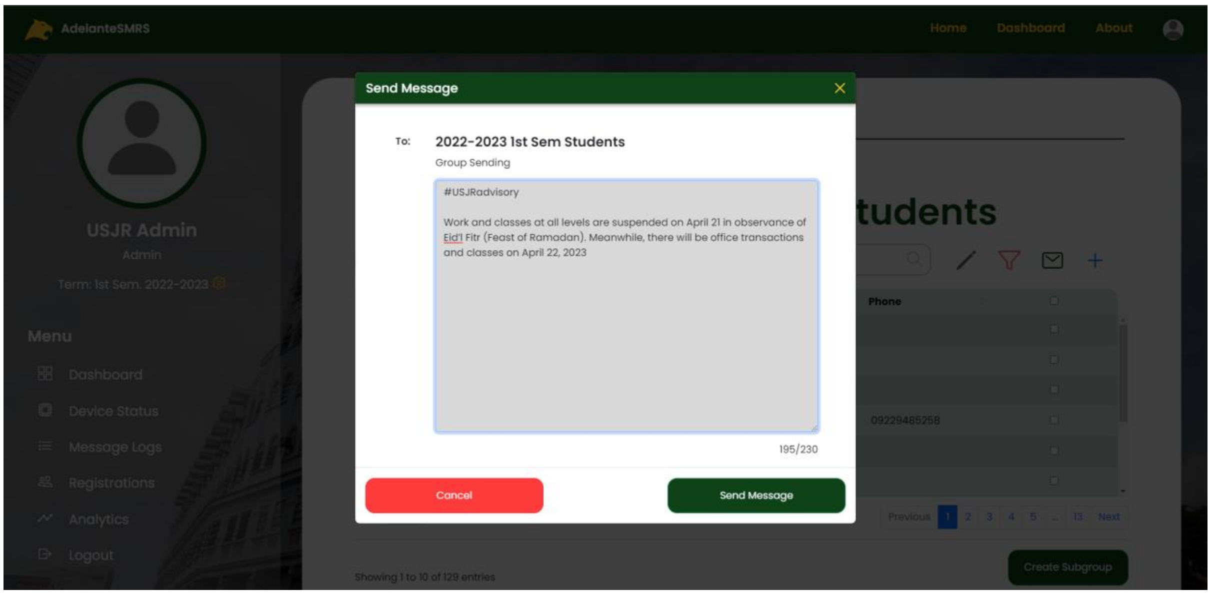


Student ID	Last Name	Contact Number
2003004047	Lim	
2005021136	Padigos	09226396038
2006000536	Gabison	
2006000606	Garcia	
2006006572	Parales	09081915802
2007000272	Rodriguez	
2007000678	Cuico	09662777361

Showing 1 to 10 of 129 entries

Figure 2.20: ***Manage Student - Add New Student***

This screen shows a modal whenever the admin clicks on the 'Add Student' button. This functionality allows the admin to add a new student to the database if there are late enrollees in the school.



To: 2022-2023 1st Sem Students
Group Sending

#USJRadvisory

Work and classes at all levels are suspended on April 21 in observance of Eid'l Fitr (Feast of Ramadan). Meanwhile, there will be office transactions and classes on April 22, 2023

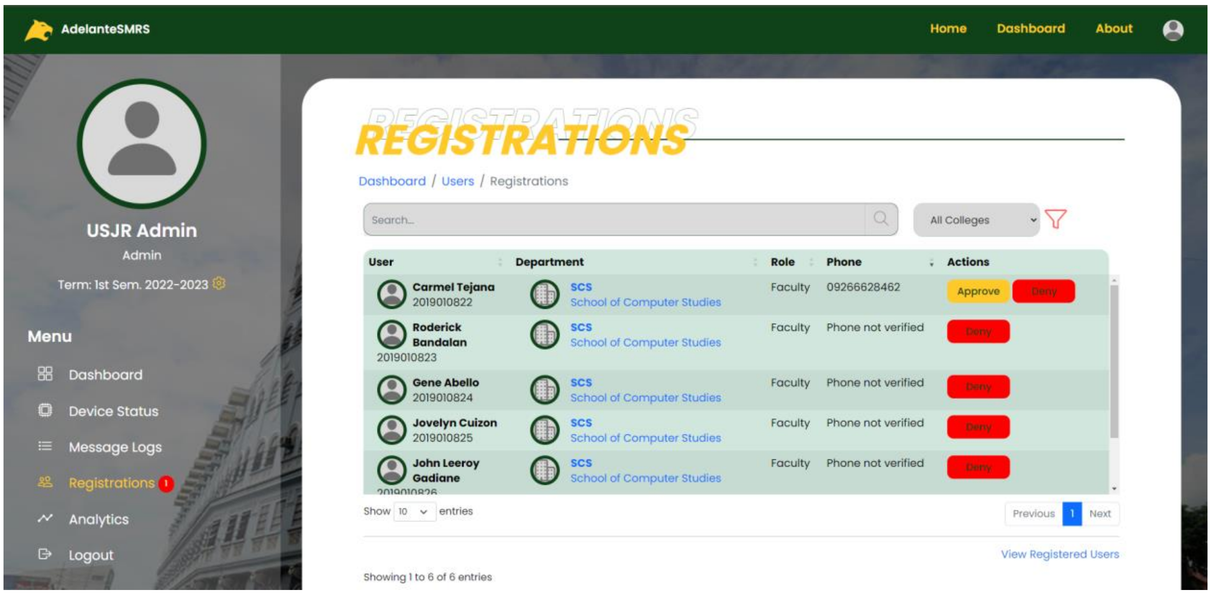
195/230

09229485258

Showing 1 to 10 of 129 entries

Figure 2.21: ***University-wide sending***

This screen shows how an admin, department staff, and faculty can send messages to a group or an individual student enrolled in the current year and term. The screenshot above shows an example of how an admin can send a university-wide announcement. Faculty and department staff can only send messages to students from their specific groups and departments, respectively.



User	Department	Role	Phone	Actions
Carmel Tejana 2019010822	SCS School of Computer Studies	Faculty	09266628462	Approve Deny
Roderick Bandalan 2019010823	SCS School of Computer Studies	Faculty	Phone not verified	Deny
Gene Abello 2019010824	SCS School of Computer Studies	Faculty	Phone not verified	Deny
Jovelyn Cuizon 2019010825	SCS School of Computer Studies	Faculty	Phone not verified	Deny
John Leeroy Gadiane 2019010826	SCS School of Computer Studies	Faculty	Phone not verified	Deny

Showing 1 to 6 of 6 entries

Figure 2.22: ***View User Registration List***

This screen allows the administrator and department staff to view users registered in the system, their colleges or schools, and their info. It is also where they regulate registration requests within the system, approving or denying users who registered.

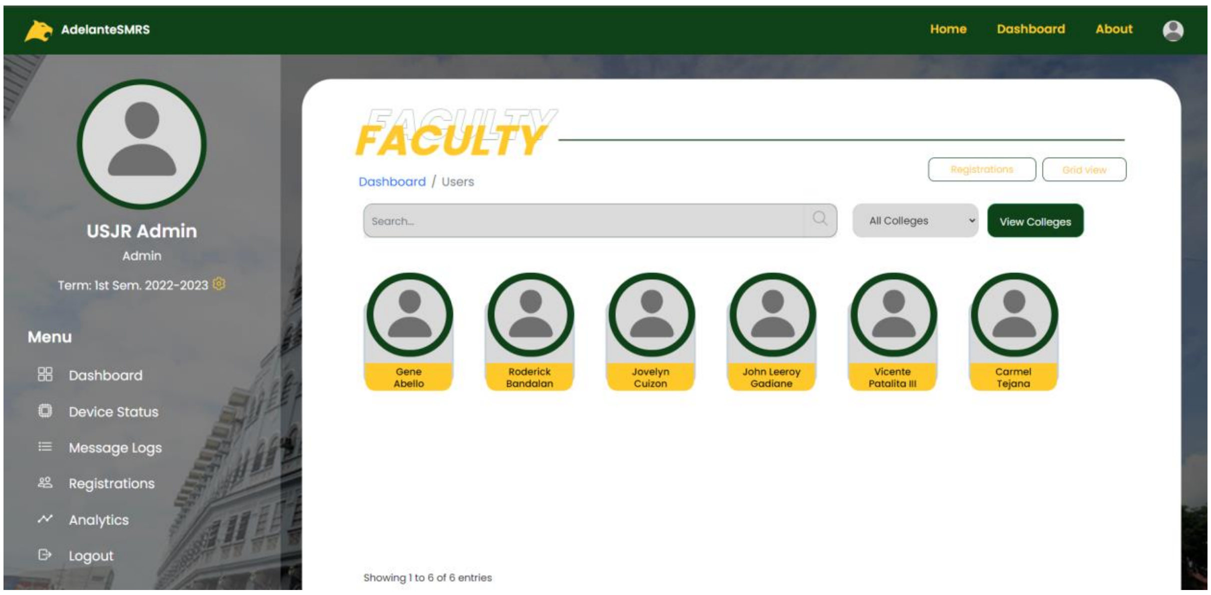


Figure 2.23: ***Faculty List***

This screen displays a comprehensive list of all the faculties currently employed at the university. The administrator can filter the list to view the faculties belonging to a specific college. Likewise, department staff members can also view the complete list of faculties, but their access is restricted only to those affiliated with them.

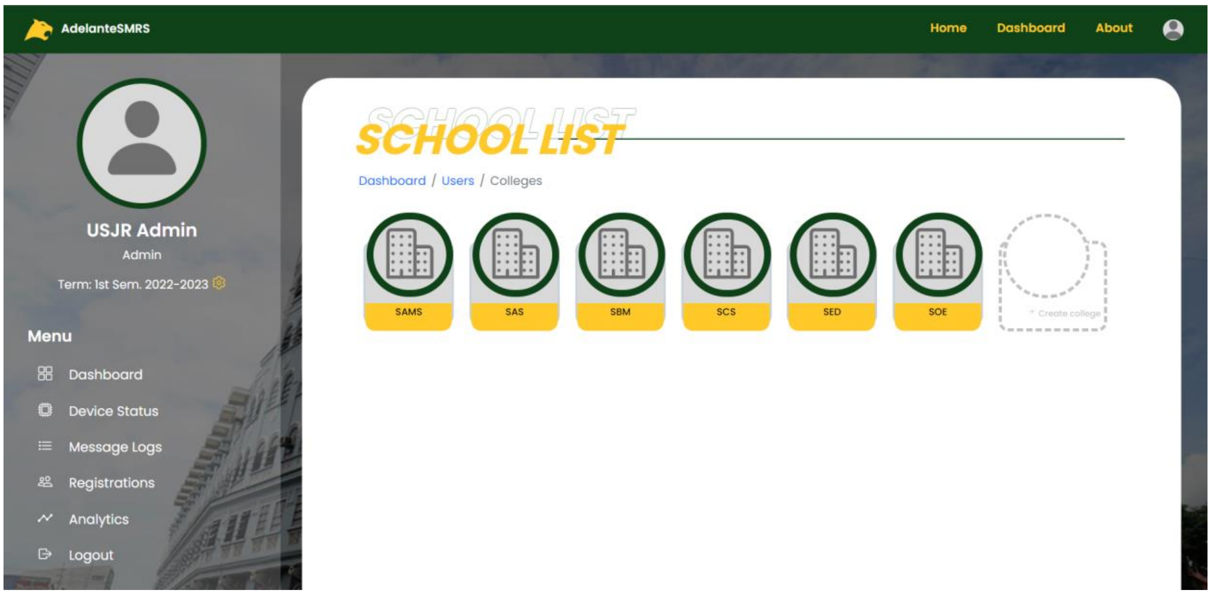
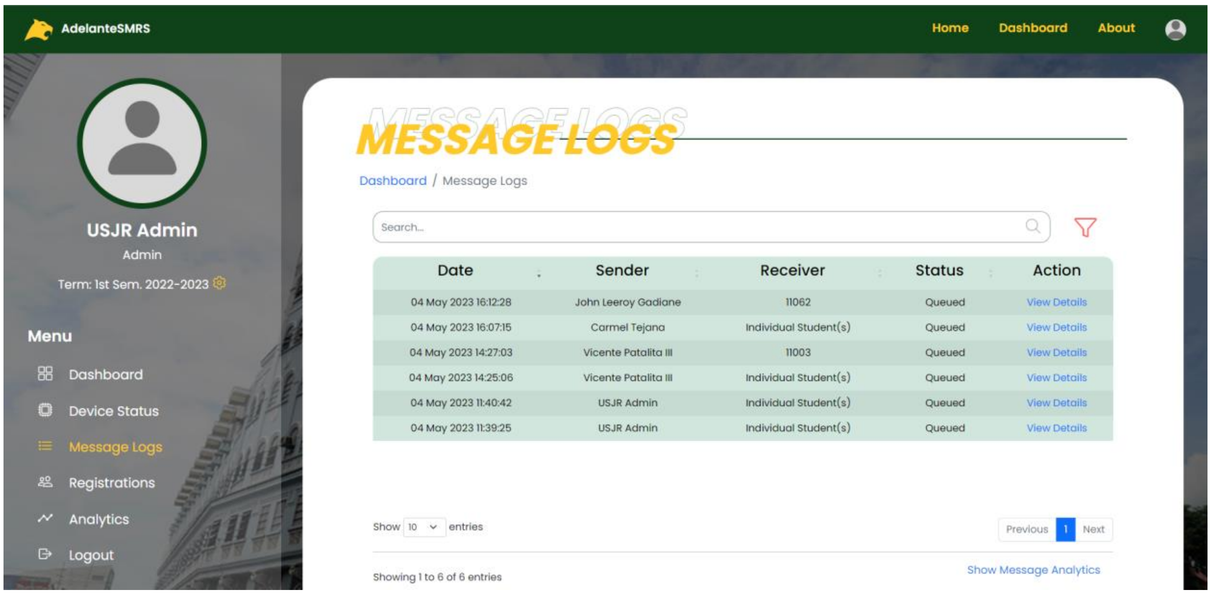


Figure 2.24: ***School List***

This screen shows all the schools or colleges currently in the university. The administrator has exclusive permission to create a new college only if the university establishes a new school or college.



Date	Sender	Receiver	Status	Action
04 May 2023 16:12:28	John Leeroy Gadiane	11062	Queued	View Details
04 May 2023 16:07:15	Carmel Tejana	Individual Student(s)	Queued	View Details
04 May 2023 14:27:03	Vicente Patalita III	11003	Queued	View Details
04 May 2023 14:25:06	Vicente Patalita III	Individual Student(s)	Queued	View Details
04 May 2023 11:40:42	USJR Admin	Individual Student(s)	Queued	View Details
04 May 2023 11:39:25	USJR Admin	Individual Student(s)	Queued	View Details

Show 10 entries

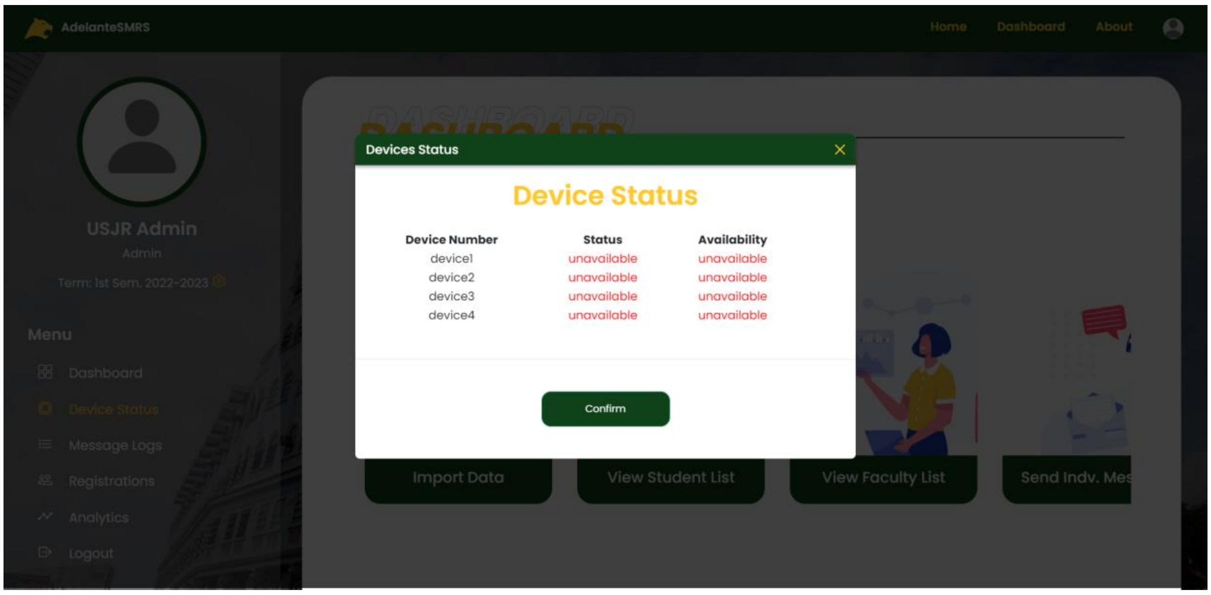
Previous 1 Next

Showing 1 to 6 of 6 entries

Show Message Analytics

Figure 2.25: ***Message Logs***

This screen shows all the messages sent by the system users, providing the administrator with an overview of message details such as the date sent, message recipients, and the message's status - either 'Queued' or 'Sent .'This helps the administrator keep track of user messages effectively. Additionally, the administrator can obtain a detailed summary of a specific message by clicking on the 'View Details' action.



Device Number	Status	Availability
device1	unavailable	unavailable
device2	unavailable	unavailable
device3	unavailable	unavailable
device4	unavailable	unavailable

Figure 2.26: ***Device Status***

This screen shows the status of the devices used in the system. This allows the admin to monitor the performance and availability of the devices. The status column indicates the status of the device.

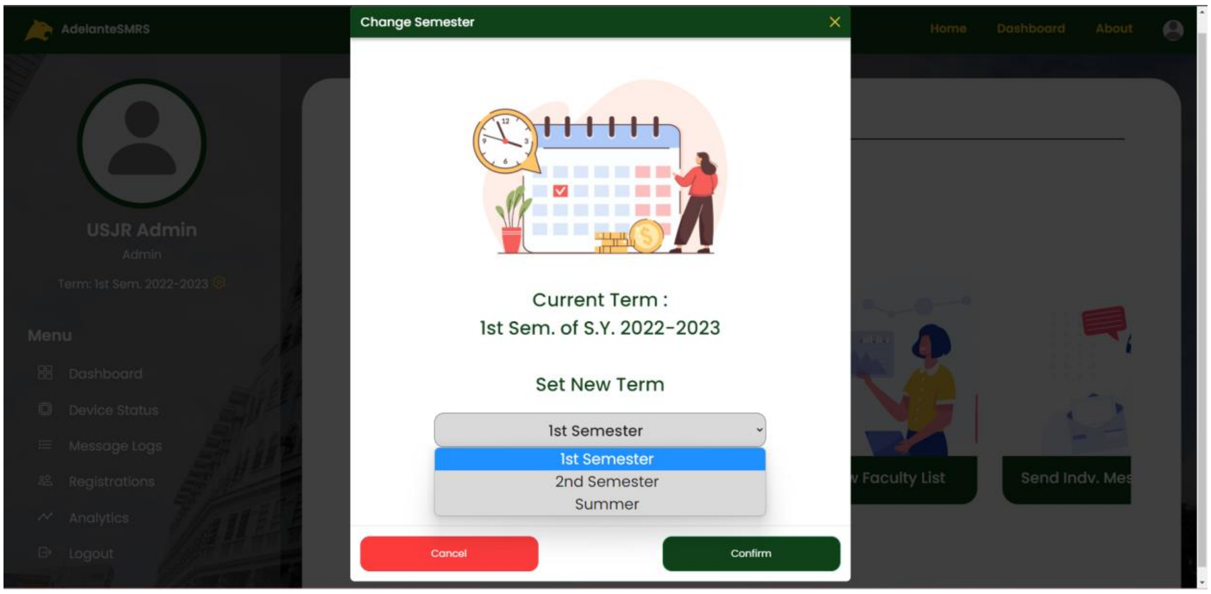


Figure 2.27: ***Set New School Term***

This screen shows how the administrator can select a new school term from a dropdown menu, automatically archiving any groups or courses offered during the previous semester. Consequently, users can only view the groups or courses that are associated with the specific term and year that has been set.

Department Staff

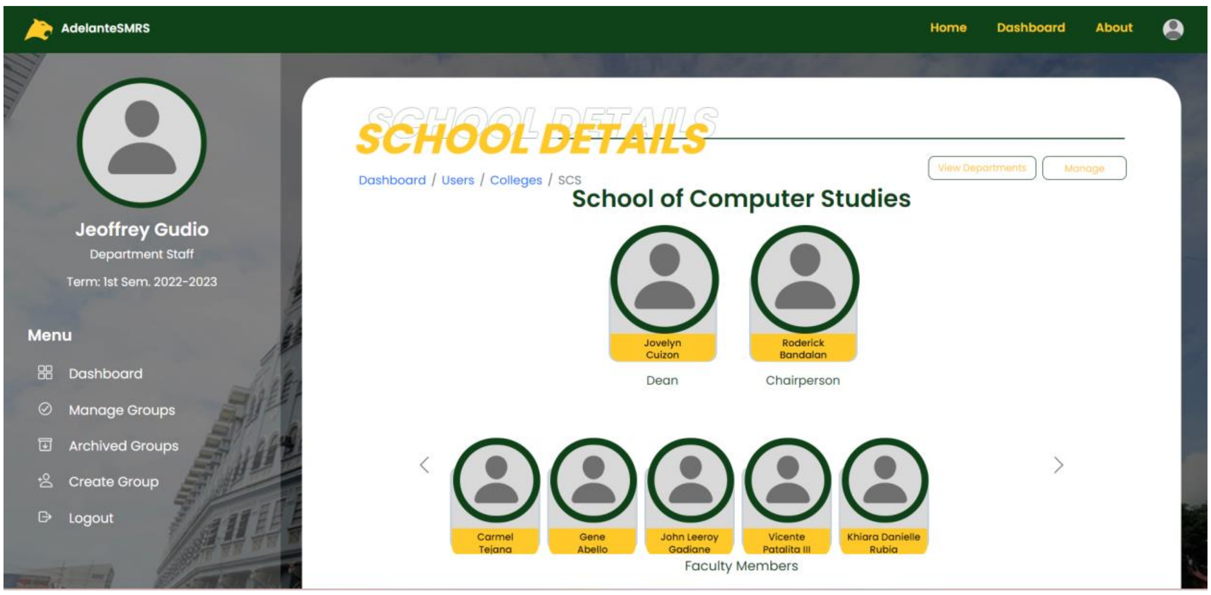
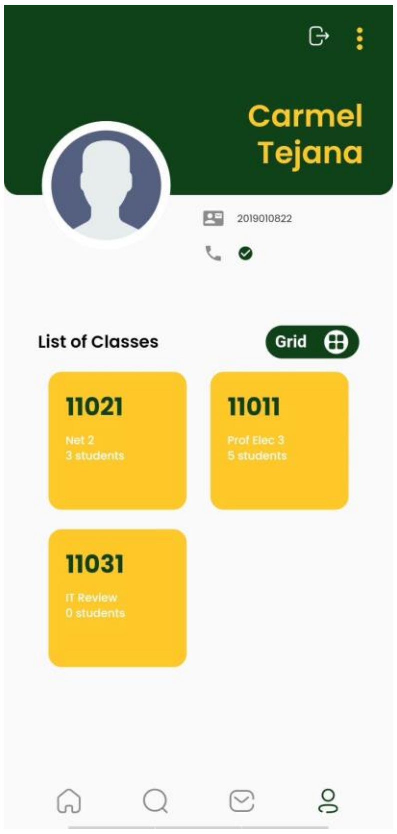


Figure 2.28: ***View School Details***

In addition to managing user registrations and viewing faculty lists within their department, department staff can manage their department by appointing a new dean or chairperson.



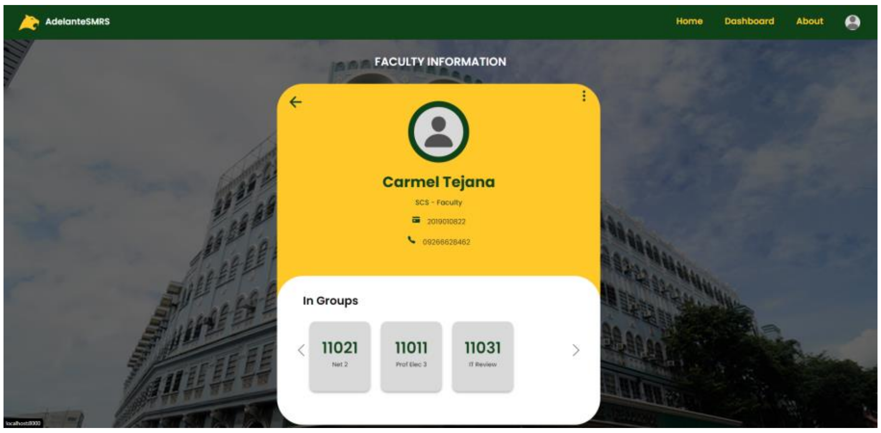


Figure 2.29: ***View User Information (Mobile and Web)***

The screen on the left is the User Profile section which displays the personal information of the logged-in user, including their name, username, and contact information status (indicated by a check icon). The user can also view the classes they teach for a specific term and year. The screen on the right is the web page for viewing user information.

Faculty

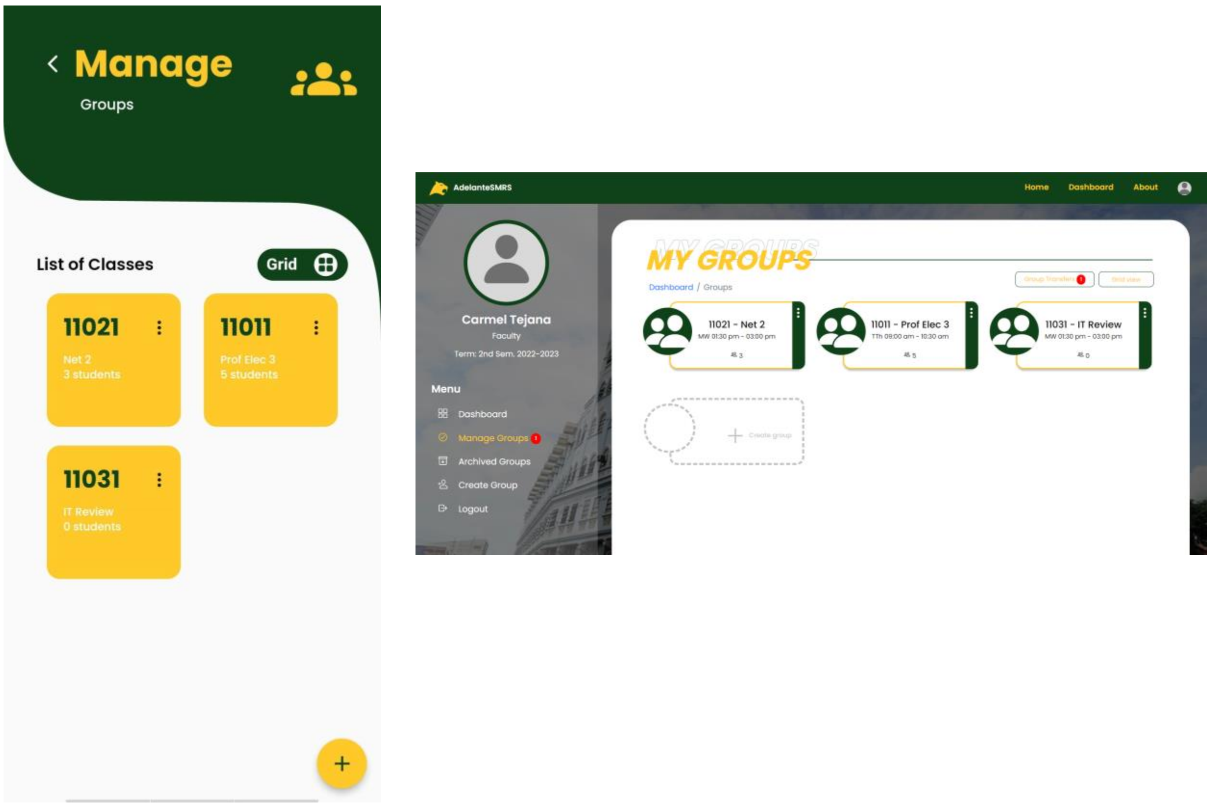
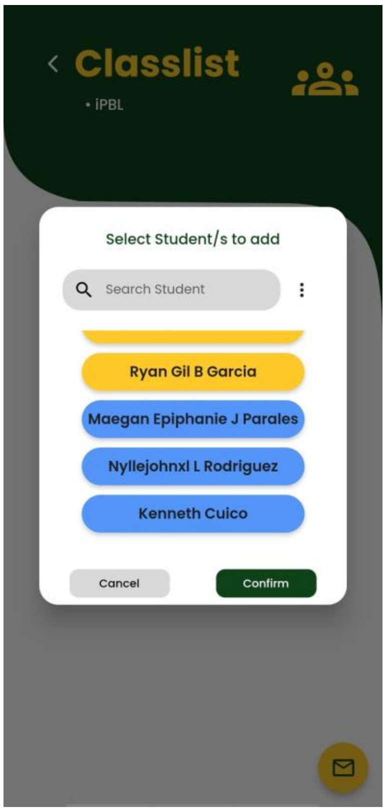
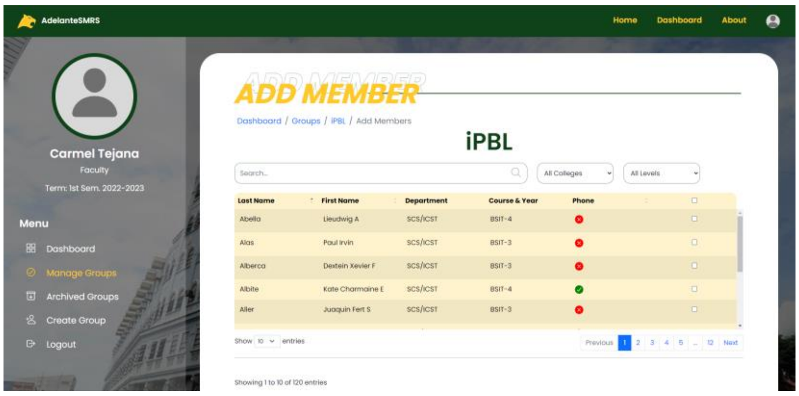


Figure 2.30: ***Manage Groups (Mobile and Web)***

Both screens show a list of courses a user currently handles for a specific term and year. Users can also create custom groups for situations where a group that is not related to a particular course needs to be established. This page also allows users to manage their groups by editing the group name (for custom groups only), archiving the group, and transferring group ownership.



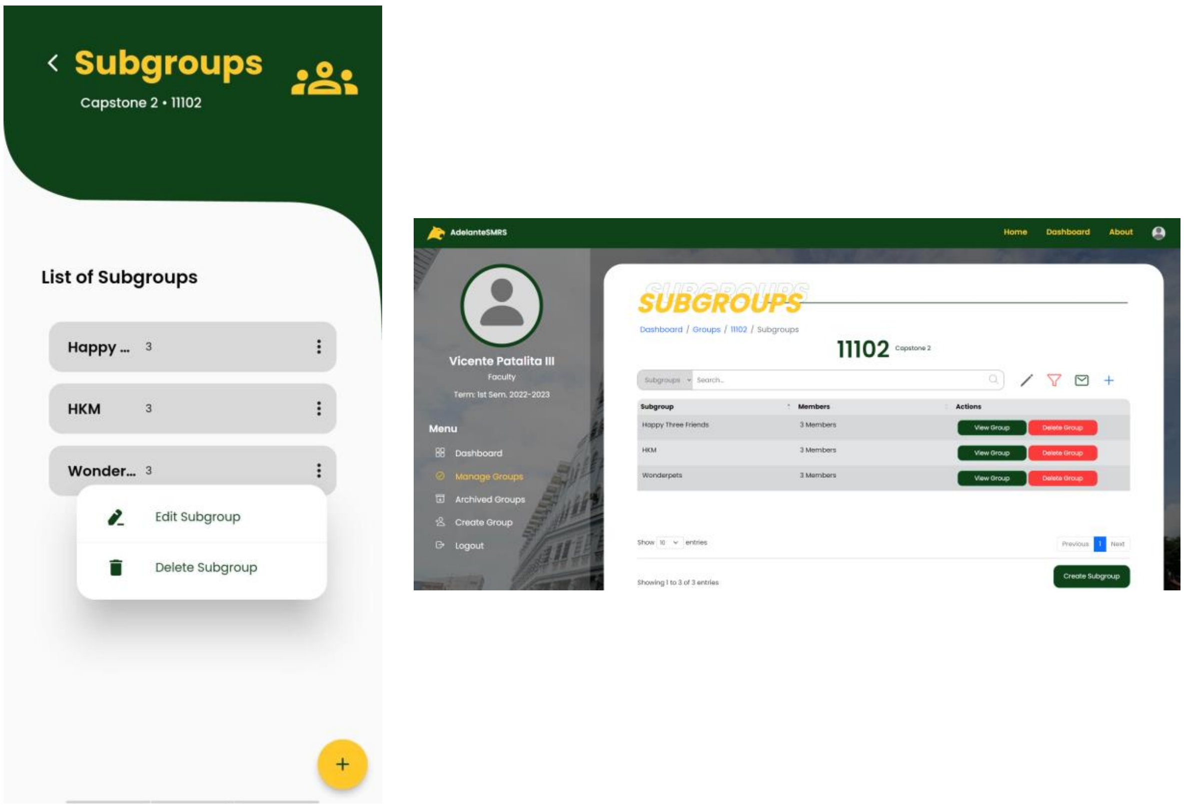


Last Name	First Name	Department	Course & Year	Phone	☐
Abella	Lieudwig A	SCS/ICST	BSIT-4	✗	☐
Alas	Paul Irvin	SCS/ICST	BSIT-3	✗	☐
Alberca	Dextein Xevier F	SCS/ICST	BSIT-3	✗	☐
Albite	Kate Charmaine E	SCS/ICST	BSIT-4	✓	☐
Aller	Juaquin Fert S	SCS/ICST	BSIT-3	✗	☐

Showing 1 to 10 of 120 entries

Figure 2.31: ***Add Member to a Group***

The screens demonstrate how users can add students to a particular group. The modal (mobile application) and data table (web application) display a list of students not currently group members. On the mobile application, users can select the student name they wish to add (blue indicates that the student is selected) or search for the student's name. Users can search for a student's name on the web version or use the filter feature to select the student by checking the provided checkbox.

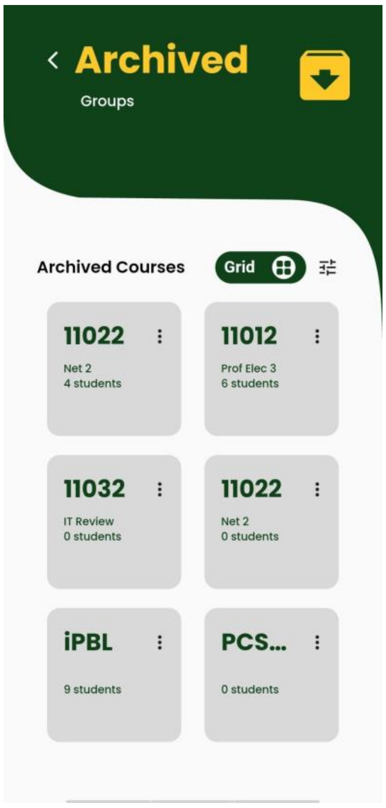


Happy ...	3
HKM	3
Wonder...	3

Subgroup	Members	Actions
Happy Three Friends	3 Members	View Group Delete Group
HKM	3 Members	View Group Delete Group
Wonderpets	3 Members	View Group Delete Group

Figure 2.32: ***View Subgroup List***

Both screens display all subgroups that have been created within a specific group. Users can create subgroups by tapping the plus button (mobile) or clicking the 'Create Subgroup' button (web), then selecting the students to include. Users can tap on the list tile (mobile) to view the members in a subgroup or click 'View Group' (web). Moreover, users can edit the subgroup name and delete the subgroup.



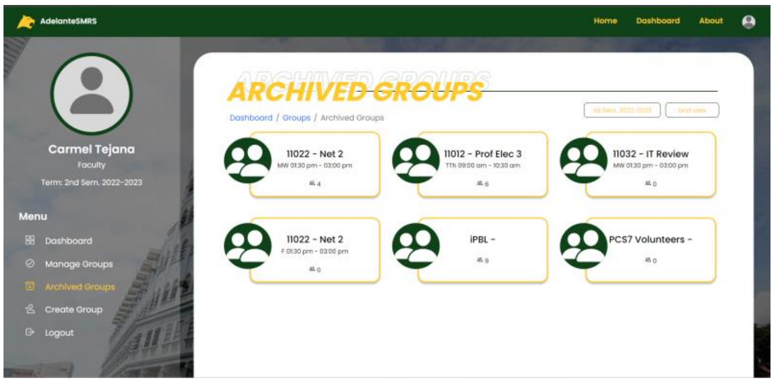
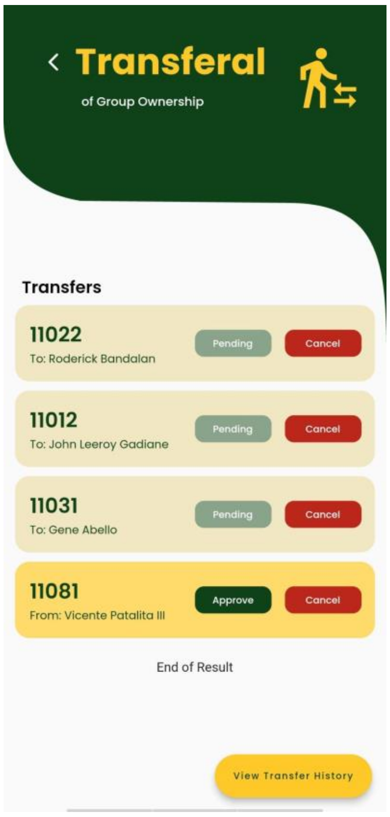


Figure 2.33: ***View Archived Groups***

Once the term changes, courses offered during the previous semester will be automatically archived. Both screens allow users to view all the archived courses from previous or current terms. Additionally, users can archive custom groups and choose to restore or permanently delete them.


Transferal
of Group Ownership

Transfers

11022
To: Roderick Bandalan
Pending Cancel

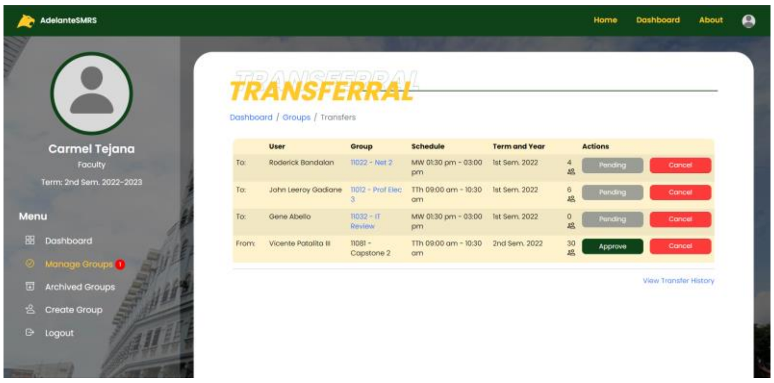
11012
To: John Leeroy Gadiane
Pending Cancel

11031
To: Gene Abello
Pending Cancel

11081
From: Vicente Patalita III
Approve Cancel

End of Result

View Transfer History



Carmel Tejana
Faculty
Term: 2nd Sem. 2022-2023

TRANSFERRAL

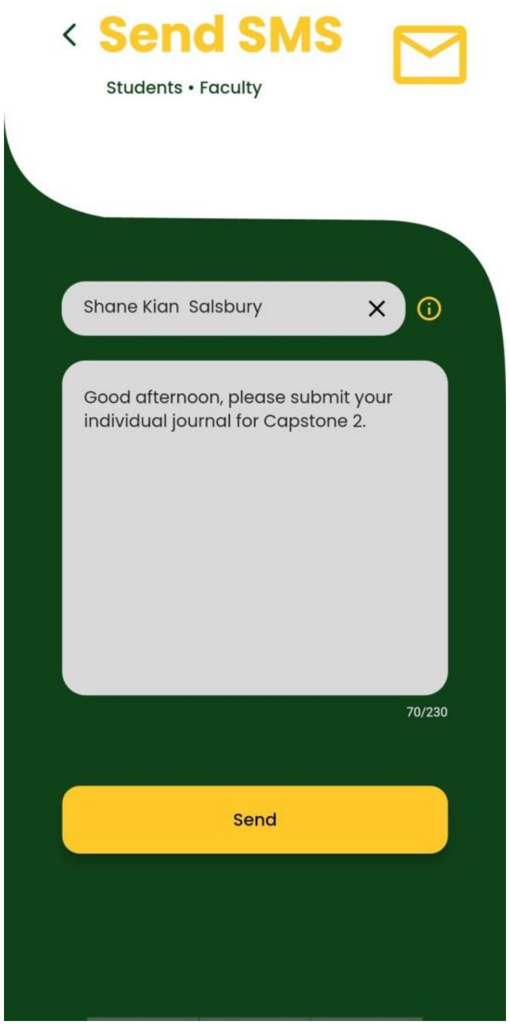
Dashboard / Groups / Transfers

	User	Group	Schedule	Term and Year		Actions	
To:	Roderick Bandalan	11022 - Net 2	MW 01:30 pm - 03:00 pm	1st Sem. 2022	4	Pending	Cancel
To:	John Leeroy Gadiane	11012 - Prof Elec 3	TTh 09:00 am - 10:30 am	1st Sem. 2022	6	Pending	Cancel
To:	Gene Abello	11032 - IT Review	MW 01:30 pm - 03:00 pm	1st Sem. 2022	0	Pending	Cancel
From:	Vicente Patalita III	11081 - Capstone 2	TTh 09:00 am - 10:30 am	2nd Sem. 2022	30	Approve	Cancel

View Transfer History

Figure 2.34: ***View Group Transfers***

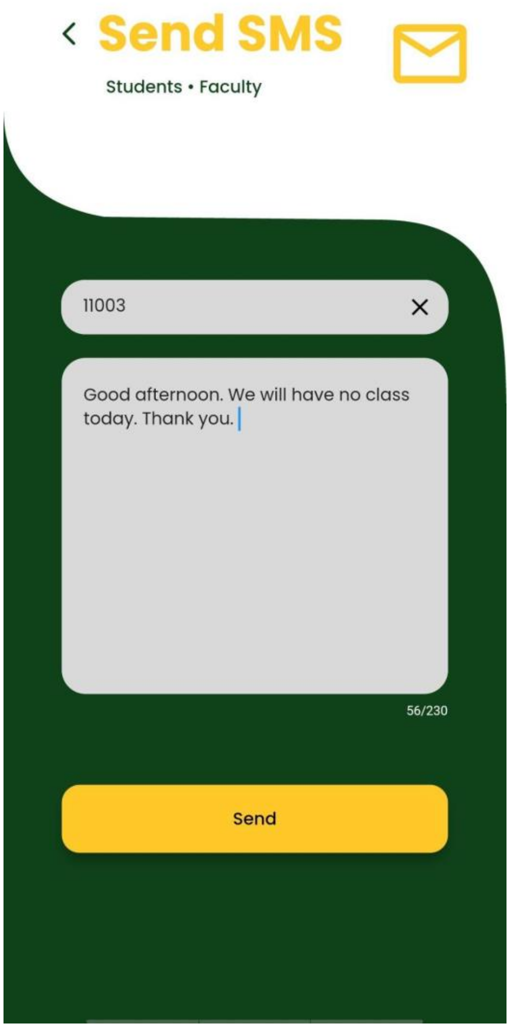
These screens show group transfers created by the currently logged-in user and the groups being transferred to them. When the user clicks the "Approve" button, group ownership will be transferred to them. Users can also access the transfer group history to view confirmed or canceled transfer requests.



Shane Kian Salsbury

Good afternoon, please submit your individual journal for Capstone 2.

70/230



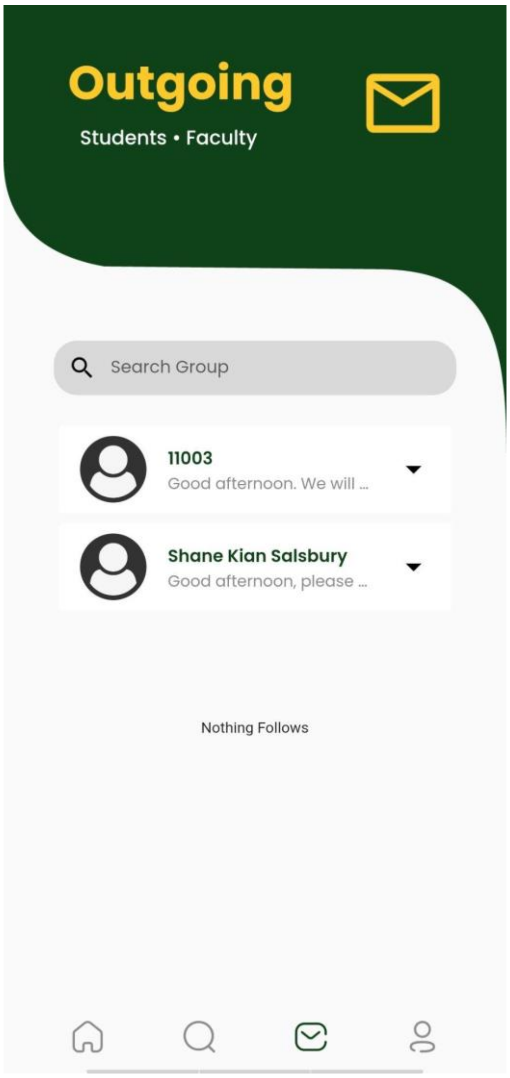
11003

Good afternoon. We will have no class today. Thank you.

56/230

Figure 2.35.1 and Figure 2.35.2: ***Send Individual Message & Send Group Message***

These screens demonstrate how users can compose messages to individual students or those in a specific group. The small text box at the top allows users to select the recipients. It can be a group, a subgroup, or specific individuals from a group, depending on the selected option. The larger text box is where the user can type their message, with a limit of 280 characters.



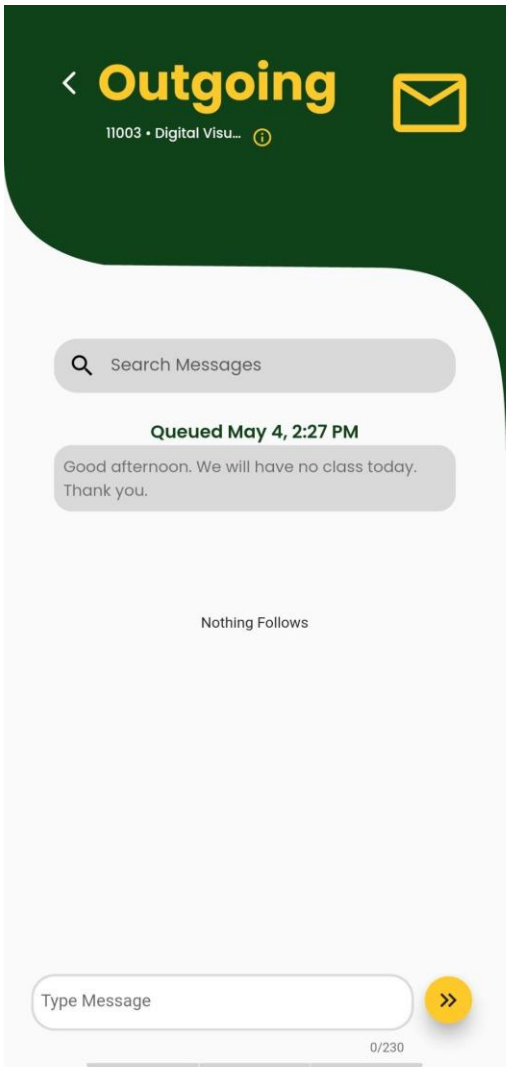
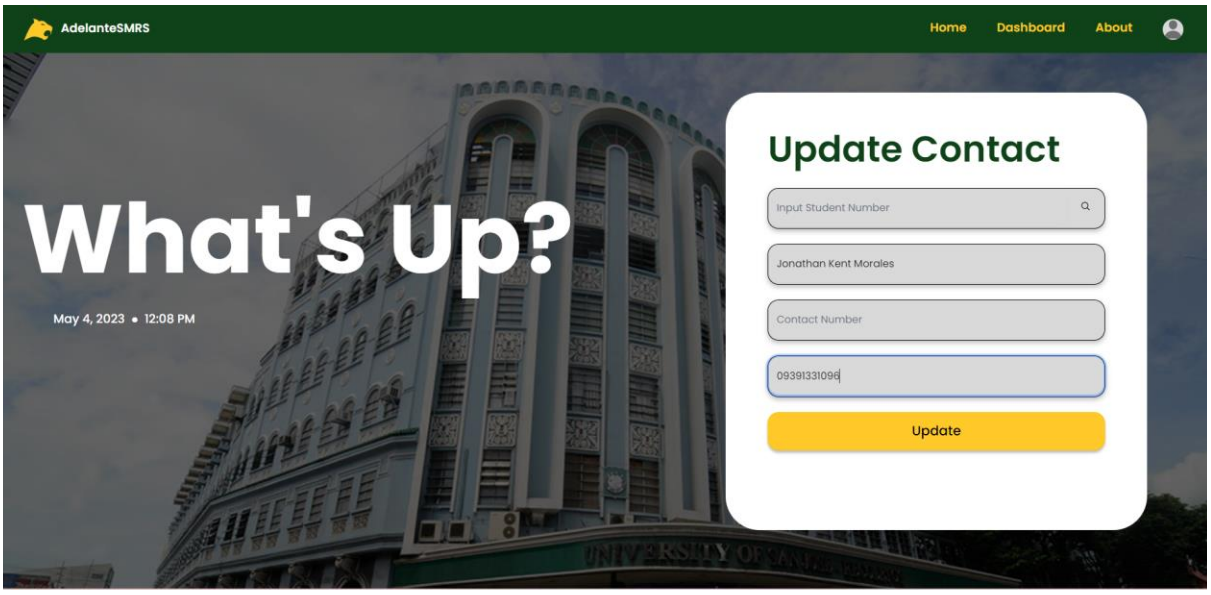


Figure 2.36.1: Outgoing Screen **Figure 2.36.2: Outgoing Alert Info**

The left screen displays a list of all the user's sent alerts. When the user long presses on a specific alert, they will be redirected to the alert info screen (right screen), where they can view all the messages they have sent to that group. The details of the message, such as the status (Queued or Sent) and the date and time it was sent, are also shown. In addition, users can directly send a message to the group by typing in the text box at the bottom of the screen.

READS Scholar



Input Student Number

Jonathan Kent Morales

Contact Number

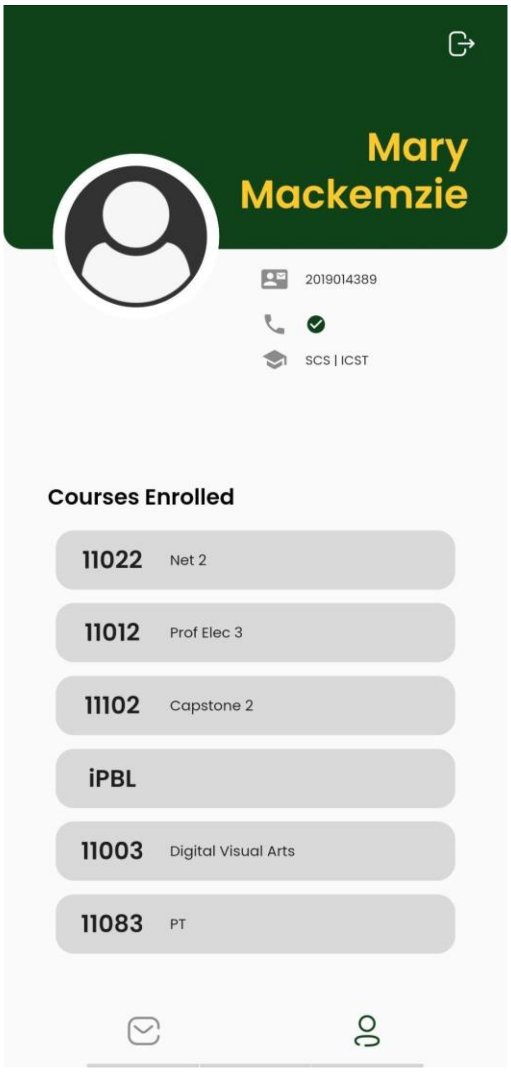
09391331096

Figure 2.37: ***Change Contact Number***

This screen shows the use case exclusive to the READS Scholar, *Change Contact Number*. It gives the ability of the READS Scholar to search for the student's student number and retrieve the student's name and previous contact number.

Changing the contact number follows with the user inputting the student's new contact number and pressing update.

Student (Mobile-use only)

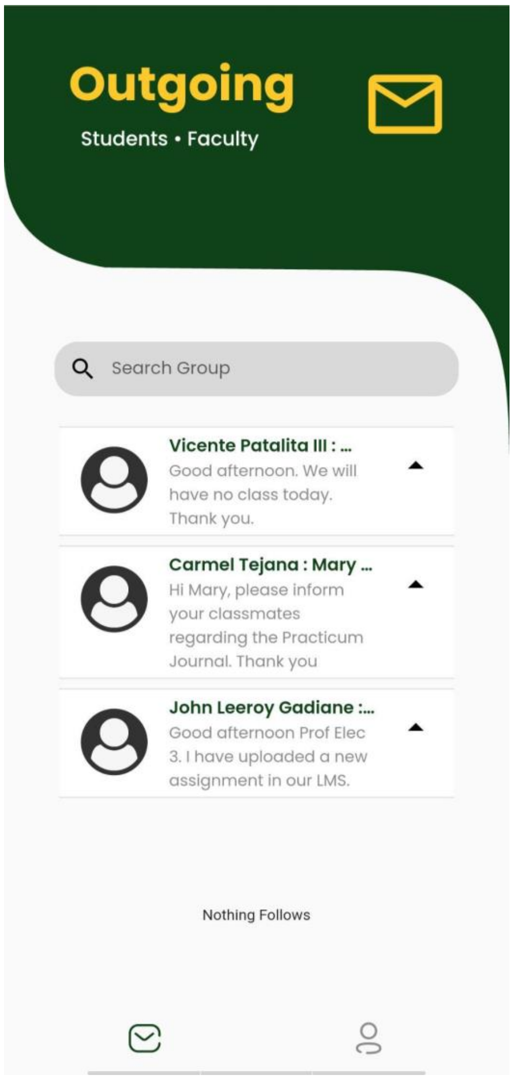

Mary Mackemzie

2019014389

SCS | ICST

Courses Enrolled

11022	Net 2
11012	Prof Elec 3
11102	Capstone 2
iPBL	
11003	Digital Visual Arts
11083	PT


Outgoing

Students • Faculty

Vicente Patalita III : ... Good afternoon. We will have no class today. Thank you.

Carmel Tejana : Mary ... Hi Mary, please inform your classmates regarding the Practicum Journal. Thank you

John Leeroy Gadiane :... Good afternoon Prof Elec 3. I have uploaded a new assignment in our LMS.

Nothing Follows

Figure 2.38: ***Student Profile and View Sent Alerts***

The screens show what the students can view once they log in to the mobile application. A profile screen displays their name, school ID, department, and the groups they are a member of. The other screen that the students can view is the outgoing screen which displays all the sent messages to the student. Additionally, they can see the conversation's complete history by holding the sender's tile.

CHAPTER III

SOFTWARE DEVELOPMENT AND TESTING

This chapter details the development of the system, along with the different tools used and integrated, as well as the tests the researchers conducted on the SIM800L and ESP32 modules during development.

Development and Testing Process

The system and its development are divided into three major components: the mobile application, the web application, the REST API backend, and the hardware component consisting of the SIM800L and ESP32 microcontroller. Furthermore, the incremental model is used. Requirements are identified and divided into smaller modules, where each module is designed, developed, and tested either sequentially or in parallel to the others before being merged once they are finished. Each increment introduces new functionality to the previous build, which repeats until all the requirements are implemented and fully working. Functionality and performance testing is performed on each module during development to test if the requirements are functioning as expected.

Tools used in the development of the system’s mobile application:

- Android Studio 2021.2.1 – the official IDE for Android software development, with platform-specific features such as formatting, emulators, and debugging capabilities. Android Studio also supports community-made plugins to customize and further improve user experience.

- Flutter 3.3.0 – an open-source front-end software development kit for cross-platform applications (Android, iOS, Windows). It utilizes Dart as its primary programming language, making UI development much more straightforward and effective with just a single codebase.
- Http 0.13.5 – a Dart API package that enables support for HTTP functionalities and requests within the program. This package allows the mobile application to interact with the web REST API service.
- Get 4.6.5 (GetX) – a Dart package that streamlines state management, navigation, and dependency injection within Flutter applications.

Tools used in the development of the system's web application and API:

- Visual Studio Code 1.7.31 – a code editor that supports various programming languages. Its features include formatting, debugging, code refactoring, and Git version control. It also supports community-made plugins, which include extensions for specific languages to improve user experience. Visual Studio Code was also used in the development of the mobile application.
- Laravel 9 – a web backend framework utilizing PHP and used to develop web applications and even REST APIs. It uses the model-view-controller (MVC) architecture, making integration with relational databases much more effortless.
- MySQL Workbench 8.0 – an interactive visual tool for database design in MySQL that allows users to create, insert, view, and manipulate data within a MySQL database.
- MySQL Community Server 8.0.27 – MySQL's database server, which allows querying and connectivity within integrated applications.

- PHP 8.1.7 Development Server – PHP's built-in web server used to develop and test web applications written in the language.
- Composer 2.4.4 – dependency manager for PHP. It allows easier management and installation of libraries supported by the language.
- Maatwebsite Excel 3.1.44 – Laravel package that makes working with CSV files easier, such as importing and exporting data from and to CSV files
- Yajra DataTables 10.2.0 – Laravel package that makes displaying and managing HTML tables and their data easier. It is built on the jQuery DataTables plugin and utilizes server-side querying.
- php-mqtt/laravel-client – It is a Laravel wrapper for the php-mqtt/client package. It allows connecting to an MQTT broker to publish messages and subscribe to topics with a Laravel application.

Other tools used in the development of the system:

- Figma - is an online collaborative web application for wireframing and designing the user interface.
- ESP32 LILYGO TTGO T-Call V1.4 - Espressif ESP32-based microcontroller with Wireless and SIM800L modules, which provides capability on the device to connect to the Wi-Fi and integrate GSM modem functions. The said functionality enables the ability to send SMS, connect to the internet via GPRS, and many more.
- Arduino IDE - is a software/IDE for Arduino-related programming projects and communicates with the Arduino hardware to upload programs or codes. It also supports community-made plugins, which include both board-specific and general board managers and libraries,

to widen the scope and abilities given the existence and capability of the hardware.

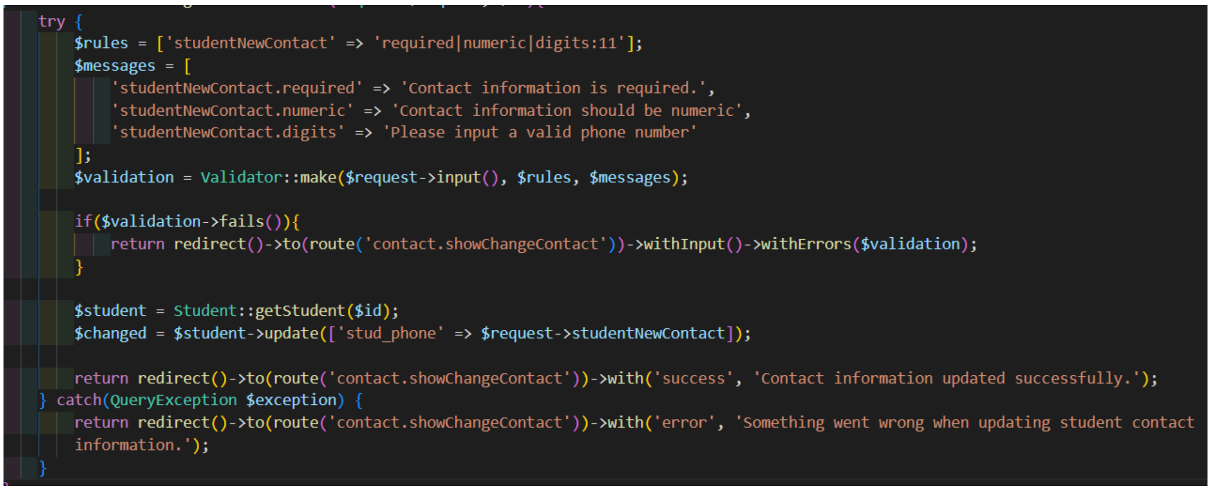
- PubSubClient - a library by Nick O'Leary for Arduino programming projects that integrates MQTT protocol. This enables the device to send or receive messages from a broker supporting the protocol.
- WiFi Arduino Library - a built-in library of Arduino enables network communication that sends/receives UDP packets through Wi-Fi. It also supports static or dynamic (DHCP) IP address configuration.
- Mosquitto Broker - is a lightweight, powerful, open-source message broker that implements the MQTT protocol.

Development Process

This section covers the technical details of the development process for all three major components of the system: the web application, the mobile application, and the hardware. It shows the code snippets of the various use cases defined in the system.

I. Web Application

Update Contacts

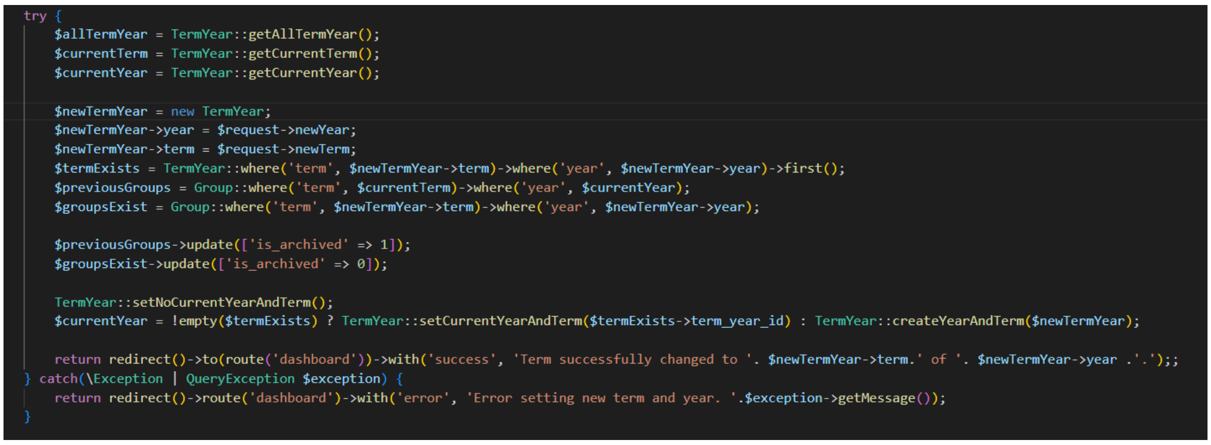


```
try {
    $rules = ['studentNewContact' => 'required|numeric|digits:11'];
    $messages = [
        'studentNewContact.required' => 'Contact information is required.',
        'studentNewContact.numeric' => 'Contact information should be numeric',
        'studentNewContact.digits' => 'Please input a valid phone number'
    ];
    $validation = Validator::make($request->input(), $rules, $messages);

    if($validation->fails()){
        return redirect()->to(route('contact.showChangeContact'))->withInput()->withErrors($validation);
    }

    $student = Student::getStudent($id);
    $changed = $student->update(['stud_phone' => $request->studentNewContact]);

    return redirect()->to(route('contact.showChangeContact'))->with('success', 'Contact information updated successfully.');
} catch(QueryException $exception) {
    return redirect()->to(route('contact.showChangeContact'))->with('error', 'Something went wrong when updating student contact
    information.');
}
```

Figure 3.1: ***Update Contacts Code Snippet***

The following shows the code snippet for the READS scholar-exclusive use case *Update Contacts*. It retrieves two inputs from a form, the first being *studentNewContact*, which is a student's updated contact information input. This input is validated before anything is done in the database. It ensures that the input is not null, is in numeric format, and has 11 digits. These are represented by the required, numeric, and digits validation types, respectively. If it passes the validation, the function retrieves the student's information using the second input called id. The phone number of the student data that was retrieved is then updated with the new contact information input.

Set Current Year and Term

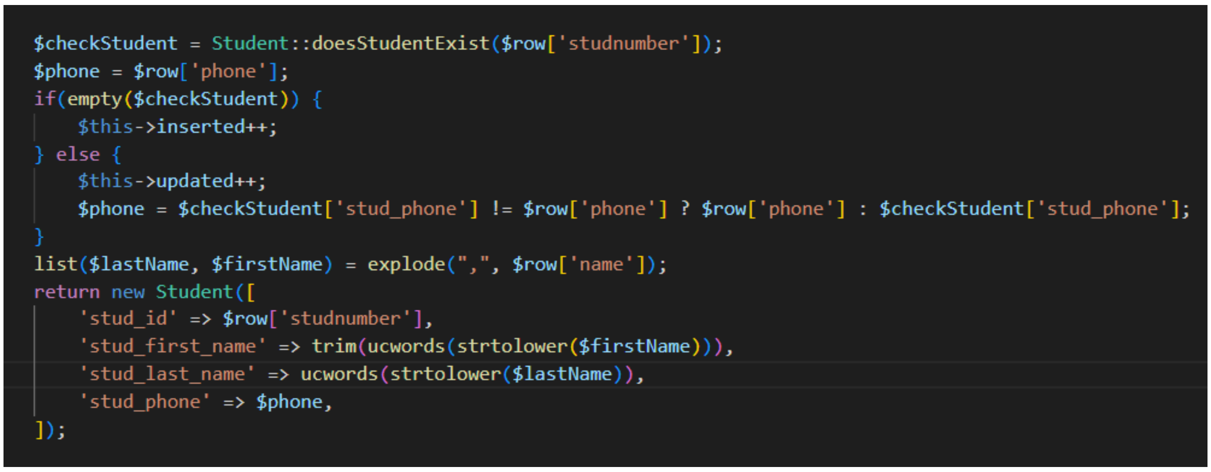


```php
try {
    $allTermYear = TermYear::getAllTermYear();
    $currentTerm = TermYear::getCurrentTerm();
    $currentYear = TermYear::getCurrentYear();

    $newTermYear = new TermYear;
    $newTermYear->year = $request->newYear;
    $newTermYear->term = $request->newTerm;
    $termExists = TermYear::where('term', $newTermYear->term)->where('year', $newTermYear->year)->first();
    $previousGroups = Group::where('term', $currentTerm)->where('year', $currentYear);
    $groupsExist = Group::where('term', $newTermYear->term)->where('year', $newTermYear->year);

    $previousGroups->update(['is_archived' => 1]);
    $groupsExist->update(['is_archived' => 0]);

    TermYear::setNoCurrentYearAndTerm();
    $currentYear = !empty($termExists) ? TermYear::setCurrentYearAndTerm($termExists->term_year_id) : TermYear::createYearAndTerm($newTermYear);

    return redirect()->to(route('dashboard'))->with('success', 'Term successfully changed to '. $newTermYear->term.' of '. $newTermYear->year .'.');;
} catch(\Exception | QueryException $exception) {
    return redirect()->route('dashboard')->with('error', 'Error setting new term and year. '.$exception->getMessage());
}
```

Figure 3.2: *Set Current Year and Term Code Snippet*

The following shows the code snippet for the *Set Current Year and Term* use case the Admin actor utilizes. First, this function checks if the selected year and term exist in the *term_year* table. If it does not exist in the table, it would set the flag "isCurrent" to 0 of all *term_year* rows and create a row for that specific term and year, which already has its flag set to 1. If it does exist, similarly, it would set the flag to 0 for all rows in the table and look for the specific term and year inputted in the table. It would then set the "isCurrent" flag to 1 once it finds the specific row.

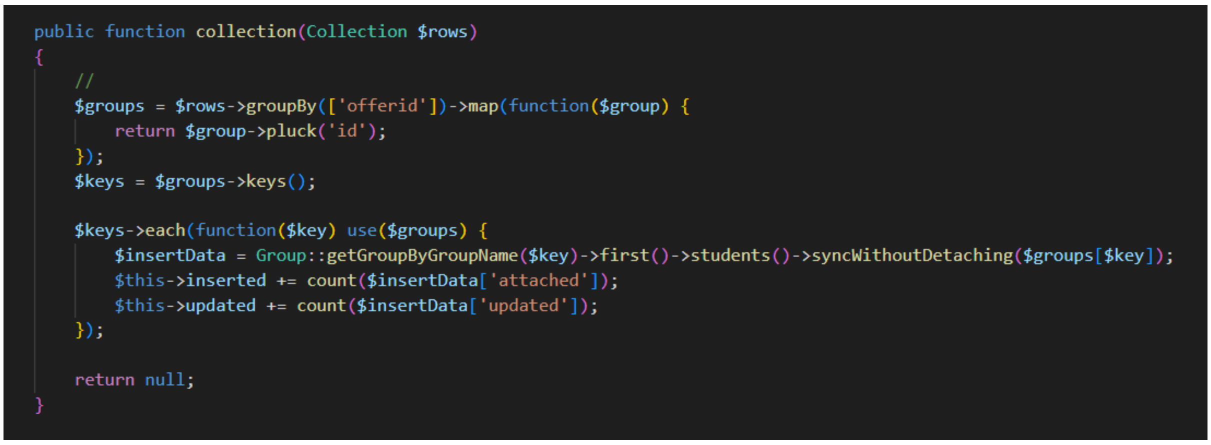
Import University Data Code Snippet



```php
$checkStudent = Student::doesStudentExist($row['studnumber']);
$phone = $row['phone'];
if(empty($checkStudent)) {
    $this->inserted++;
} else {
    $this->updated++;
    $phone = $checkStudent['stud_phone'] != $row['phone'] ? $row['phone'] : $checkStudent['stud_phone'];
}
list($lastName, $firstName) = explode(",", $row['name']);
return new Student([
    'stud_id' => $row['studnumber'],
    'stud_first_name' => trim(ucwords(strtolower($firstName))),
    'stud_last_name' => ucwords(strtolower($lastName)),
    'stud_phone' => $phone,
]);
```

Figure 3.3.1: ***Import University Data Code Snippet 1***

The following shows the code snippets for the *Import University Data* use case the Admin actor utilizes. This function uses the Import class from the Maatwebsite Excel package, where it receives each row of the CSV file and converts the data to a new model in the system, which in the case of this code snippet, is a Student. Students already existing in the database are skipped and only updated if their phone numbers do not match. The first and last names are retrieved by dividing the name string using the comma character pattern. This is implemented in response to the provided CSV format from the EDP, where the students' first and last names are contained under one name column.

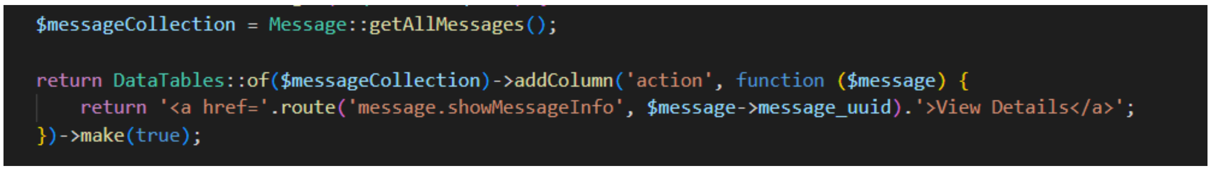


```
public function collection(Collection $rows)
{
    //
    $groups = $rows->groupBy(['offerid'])->map(function($group) {
        return $group->pluck('id');
    });
    $keys = $groups->keys();

    $keys->each(function($key) use($groups) {
        $insertData = Group::getGroupByGroupName($key)->first()->students()->syncWithoutDetaching($groups[$key]);
        $this->inserted += count($insertData['attached']);
        $this->updated += count($insertData['updated']);
    });

    return null;
}
```

Figure 3.3.2: ***Import University Data Code Snippet 2***

The following shows another code snippet for the *Import University Data* use case involving the Student Load information, where rows are grouped according to the OfferID column before iterating through each one and syncing it with the Student ID to the *Group_Student* pivot table.

Monitor Alerts Log

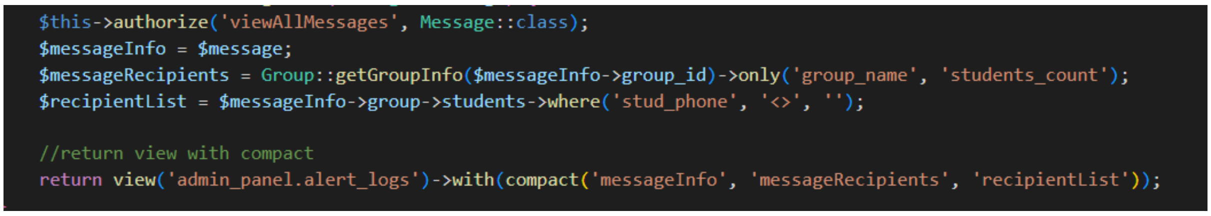


```
$messageCollection = Message::getAllMessages();

return DataTables::of($messageCollection)->addColumn('action', function ($message) {
    return '<a href='.route('message.showMessageInfo', $message->message_uuid).'>View Details</a>';
})->make(true);
```

Figure 3.4: ***Monitor Alerts Log Code Snippet***

The following shows the code snippet for the *Monitor Alerts Log* use case the Admin actor utilizes. The function uses the Yajra DataTables package based on JQuery DataTables. It retrieves all messages in the system in a paginated list and passes it to the DataTable to be formatted and displayed on the page as a searchable table. A *View Details* link is also appended to the DataTable's final column, which routes users to the *Show Message Info* screen showing a modal containing more information about the alert selected.

View Alert Details

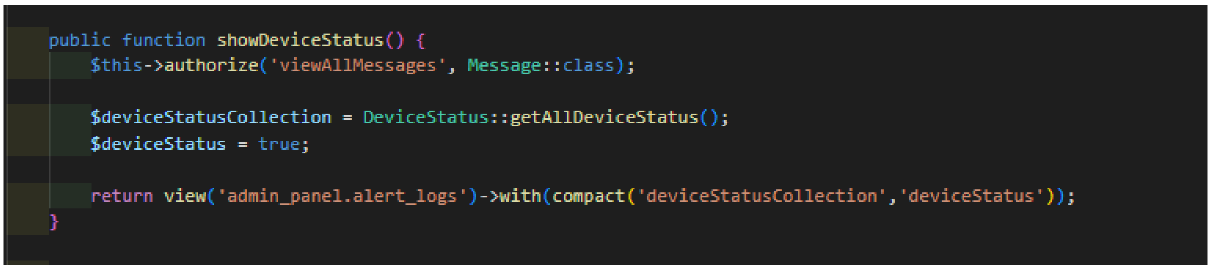


```
$this->authorize('viewAllMessages', Message::class);
$messageInfo = $message;
$messageRecipients = Group::getGroupInfo($messageInfo->group_id)->only('group_name', 'students_count');
$recipientList = $messageInfo->group->students->where('stud_phone', '<>', '');

//return view with compact
return view('admin_panel.alert_logs')->with(compact('messageInfo', 'messageRecipients', 'recipientList'));
```

Figure 3.5: ***View Alert Details Code Snippet***

The following shows the code snippet for *View Alert Details* use case utilized by the Admin actor. The function is called when the user clicks on the *View Details* of a row in the Alert Logs table. It retrieves the details of that particular message: Date and Time, Sender, Group, Recipient Count, Expected Recipients, Message, and Status. The function then throws the details back to the page to be displayed to the user. It also makes sure to retrieve only the recipients with contact information expected to have received the alert. Lastly, the function ensures that the currently logged-in user is authorized to view the messages, which means only the Admin should be able to access this page.

View Hardware Status

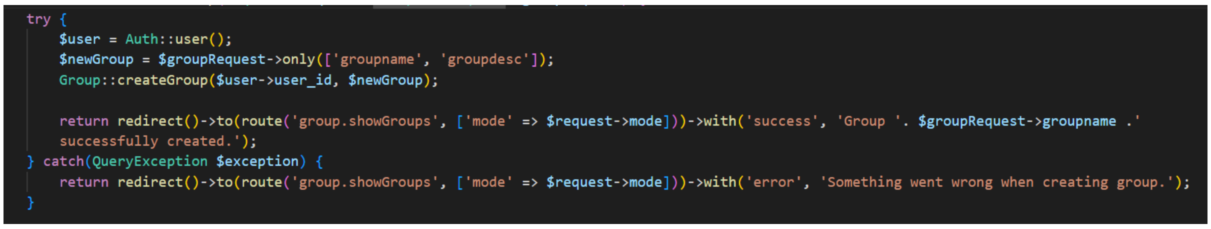


```php
public function showDeviceStatus() {
    $this->authorize('viewAllMessages', Message::class);

    $deviceStatusCollection = DeviceStatus::getAllDeviceStatus();
    $deviceStatus = true;

    return view('admin_panel.alert_logs')->with(compact('deviceStatusCollection','deviceStatus'));
}
```

Figure 3.6: ***View Hardware Status Code Snippet***

The following shows the code snippet for *View Hardware Status* use case utilized by the Admin actor. The function is called when the user clicks the "View Device Status" on the Alert Logs Page. It retrieves all device status data from the database and stores it in a collection. Each device in the collection contains the device name, availability, and status. Once they have been retrieved, the function returns to the Alert Logs view with the device status collection variable and a trigger variable to trigger the Device Status Modal, which would show each device's status in the system.

Manage Groups



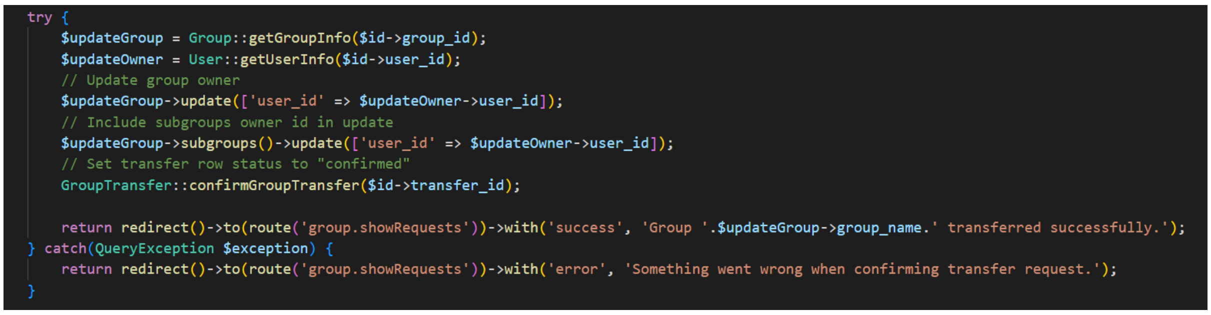
```php
try {
    $user = Auth::user();
    $newGroup = $groupRequest->only(['groupname', 'groupdesc']);
    Group::createGroup($user->user_id, $newGroup);

    return redirect()->to(route('group.showGroups', ['mode' => $request->mode]))->with('success', 'Group '. $groupRequest->groupname .'
    successfully created.');
} catch(QueryException $exception) {
    return redirect()->to(route('group.showGroups', ['mode' => $request->mode]))->with('error', 'Something went wrong when creating group.');
}
```

Figure 3.7: ***Manage Groups Code Snippet***

The following shows the code snippet for the *Manage Groups* use case. The function receives the group name and description input requests from the Manage

Group view. The function then creates a group based on the values received with the additional information of the group owner. It redirects the user back to the previous page, along with data regarding the outcome of the operation.

Transfer Group Ownership

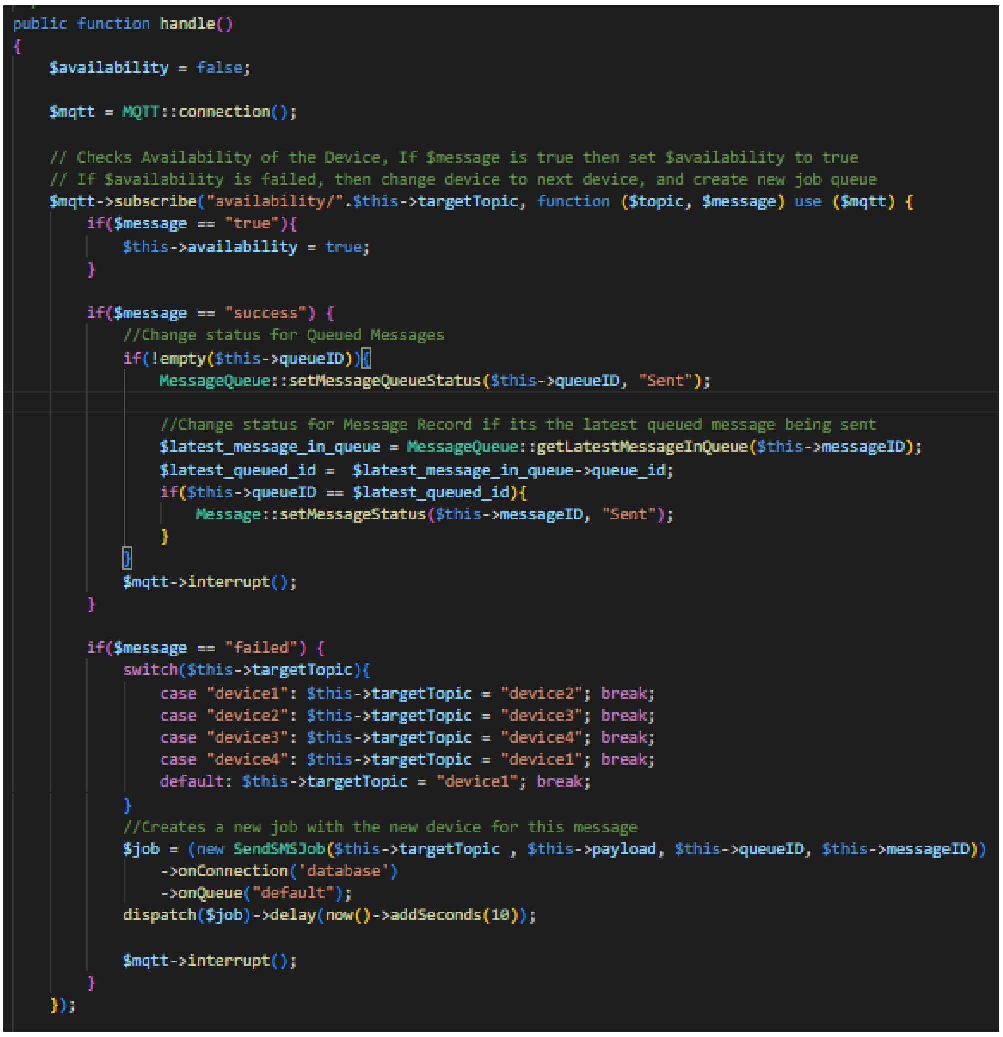


```
try {
    $updateGroup = Group::getGroupInfo($id->group_id);
    $updateOwner = User::getUserInfo($id->user_id);
    // Update group owner
    $updateGroup->update(['user_id' => $updateOwner->user_id]);
    // Include subgroups owner id in update
    $updateGroup->subgroups()->update(['user_id' => $updateOwner->user_id]);
    // Set transfer row status to "confirmed"
    GroupTransfer::confirmGroupTransfer($id->transfer_id);

    return redirect()->to(route('group.showRequests'))->with('success', 'Group '.$updateGroup->group_name.' transferred successfully.');
} catch(QueryException $exception) {
    return redirect()->to(route('group.showRequests'))->with('error', 'Something went wrong when confirming transfer request.');
}
```

Figure 3.8: ***Transfer Group Ownership Code Snippet***

The following shows the code snippet for the *Transfer Group Ownership* use case for Groups. The function retrieves data about the group to be transferred and its supposed new owner. The function works by changing the group owner ID to the ID of the selected new owner. If there are subgroups, their owner IDs would also be changed to match that of the new group owner. It then redirects the user either back to the previous page with a success or an error to display in the view depending on the outcome of the operation.

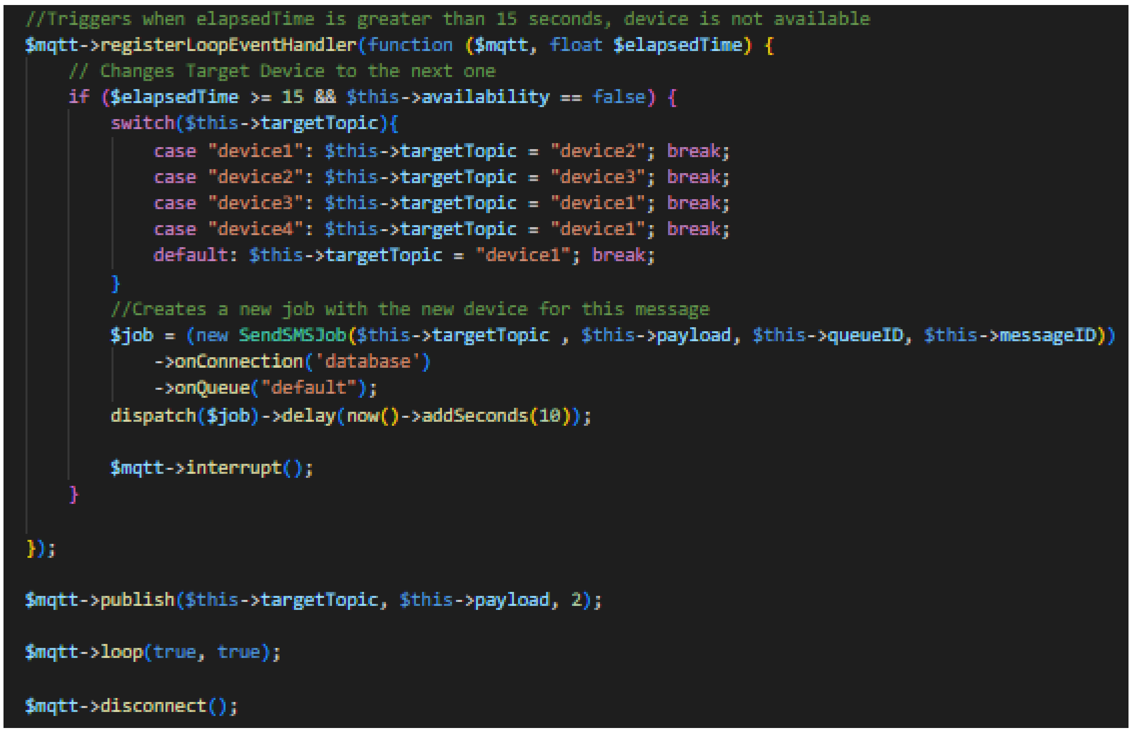
Send Alert Messages



```php
public function handle()
{
    $availability = false;

    $mqtt = MQTT::connection();

    // Checks Availability of the Device, If $message is true then set $availability to true
    // If $availability is failed, then change device to next device, and create new job queue
    $mqtt->subscribe("availability/".$this->targetTopic, function ($topic, $message) use ($mqtt) {
        if($message == "true"){
            $this->availability = true;
        }

        if($message == "success") {
            //Change status for Queued Messages
            if(!empty($this->queueID)){
                MessageQueue::setMessageQueueStatus($this->queueID, "Sent");

                //Change status for Message Record if its the latest queued message being sent
                $latest_message_in_queue = MessageQueue::getLatestMessageInQueue($this->messageID);
                $latest_queued_id =  $latest_message_in_queue->queue_id;
                if($this->queueID == $latest_queued_id){
                    Message::setMessageStatus($this->messageID, "Sent");
                }
            }
            $mqtt->interrupt();
        }

        if($message == "failed") {
            switch($this->targetTopic){
                case "device1": $this->targetTopic = "device2"; break;
                case "device2": $this->targetTopic = "device3"; break;
                case "device3": $this->targetTopic = "device4"; break;
                case "device4": $this->targetTopic = "device1"; break;
                default: $this->targetTopic = "device1"; break;
            }
            //Creates a new job with the new device for this message
            $job = (new SendSMSJob($this->targetTopic , $this->payload, $this->queueID, $this->messageID))
                ->onConnection('database')
                ->onQueue("default");
            dispatch($job)->delay(now()->addSeconds(10));

            $mqtt->interrupt();
        }
    });
```

Figure 3.9.1: ***Send Alert Message Job Code Snippet***

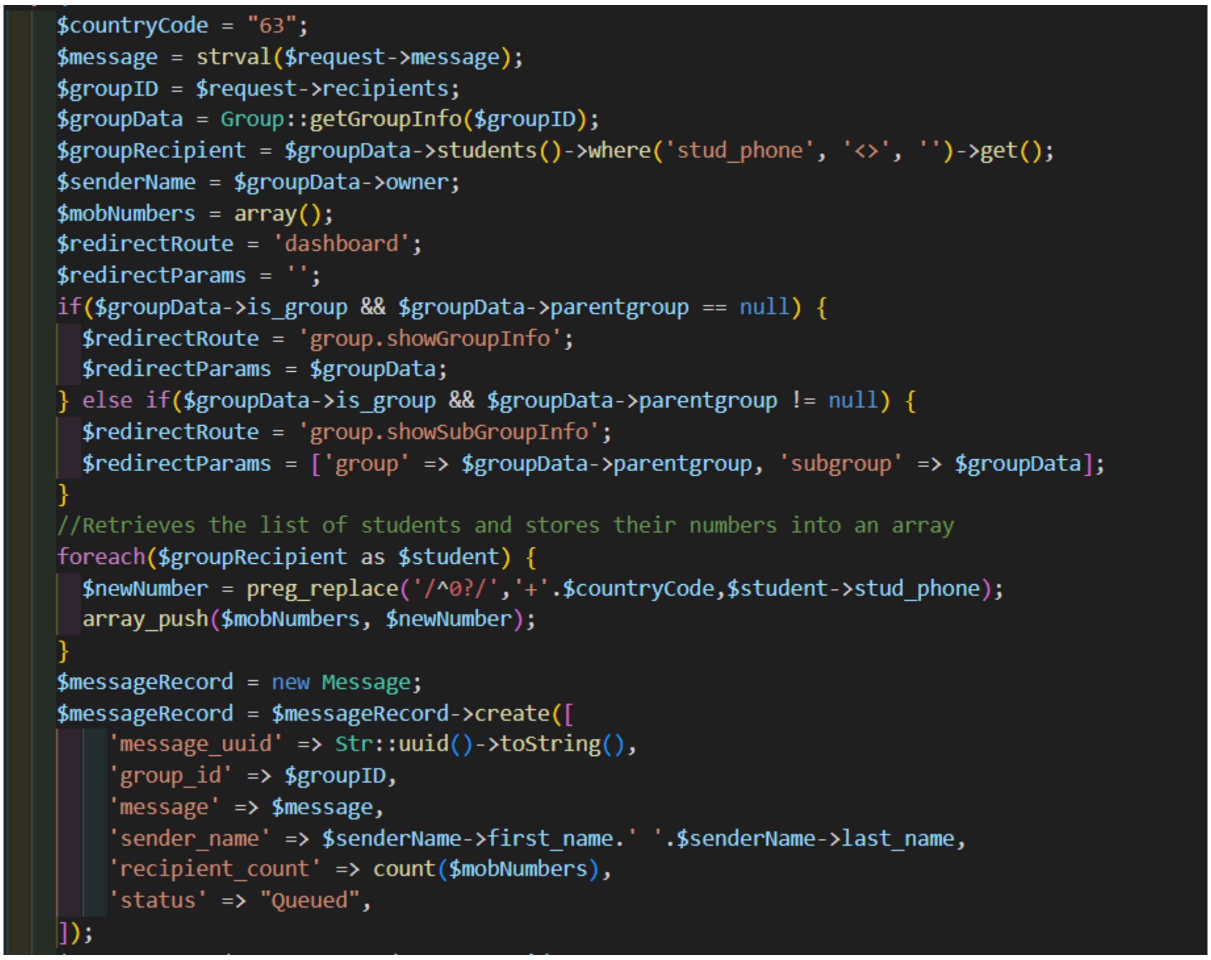
The following code snippet shows the first part of the *Send Alert Message Job*, part of the *Send Alert Message* use case functionality. The handle function first sets the variable *$availability* to false and initiates a connection with the MQTT server. When it connects, it triggers a subscribe function loop to listen to the target device, in this case, the message job's availability/target topic, to check its availability. On the other hand, when idle, the device would publish a message "true" to the MQTT server if it is available. If the subscribe function receives "true," it changes *$availability* to true. As for the "failed" condition, this is sent when the device fails to send a message. It changes the targetTopic to the next device and then creates and dispatches a new Job with the same parameters aside from the new targetTopic, which is then queued for processing. Once done, it triggers an interrupt to stop the subscribe function from looping. It is important to note that this "failed" condition is part of the round-robin algorithm that the system implemented in case the device fails to send the message.



```php
//Triggers when elapsedTime is greater than 15 seconds, device is not available
$mqtt->registerLoopEventHandler(function ($mqtt, float $elapsedTime) {
    // Changes Target Device to the next one
    if ($elapsedTime >= 15 && $this->availability == false) {
        switch($this->targetTopic){
            case "device1": $this->targetTopic = "device2"; break;
            case "device2": $this->targetTopic = "device3"; break;
            case "device3": $this->targetTopic = "device1"; break;
            case "device4": $this->targetTopic = "device1"; break;
            default: $this->targetTopic = "device1"; break;
        }
        //Creates a new job with the new device for this message
        $job = (new SendSMSJob($this->targetTopic , $this->payload, $this->queueID, $this->messageID))
            ->onConnection('database')
            ->onQueue("default");
        dispatch($job)->delay(now()->addSeconds(10));

        $mqtt->interrupt();
    }

});

$mqtt->publish($this->targetTopic, $this->payload, 2);

$mqtt->loop(true, true);

$mqtt->disconnect();
```

Figure 3.9.2: ***Send Alert Message Job Code Snippet (contd.)***

The following code snippet shows the continuation of the *Send Alert Message Job*, which is part of the functionality for the Send Alert Message use case. The *registerLoopEventHandler* is the function responsible for triggering processes in case the server receives no response from the device. Suppose the function does not hear from the device within 15 seconds, and the variable *$availability* is still set to "false." In that case, the condition within the *registeredLoopEventHandler* will execute to change the *targetTopic* to the next device, then create and dispatch another Job to the queue, similar to how the "failed" condition does it. The publish function is the one responsible for publishing the payload to the device in the MQTT server, the device. Since publishing and subscribing require constant listening to the MQTT server, we must make a loop. Hence, this loop function is responsible for that task. The second parameter of the loop is placed in case the publish function successfully sends the message to the device, hence, ending the loop. Lastly, the disconnect function would disconnect the connection to the MQTT server, a standard practice for responsible coding.

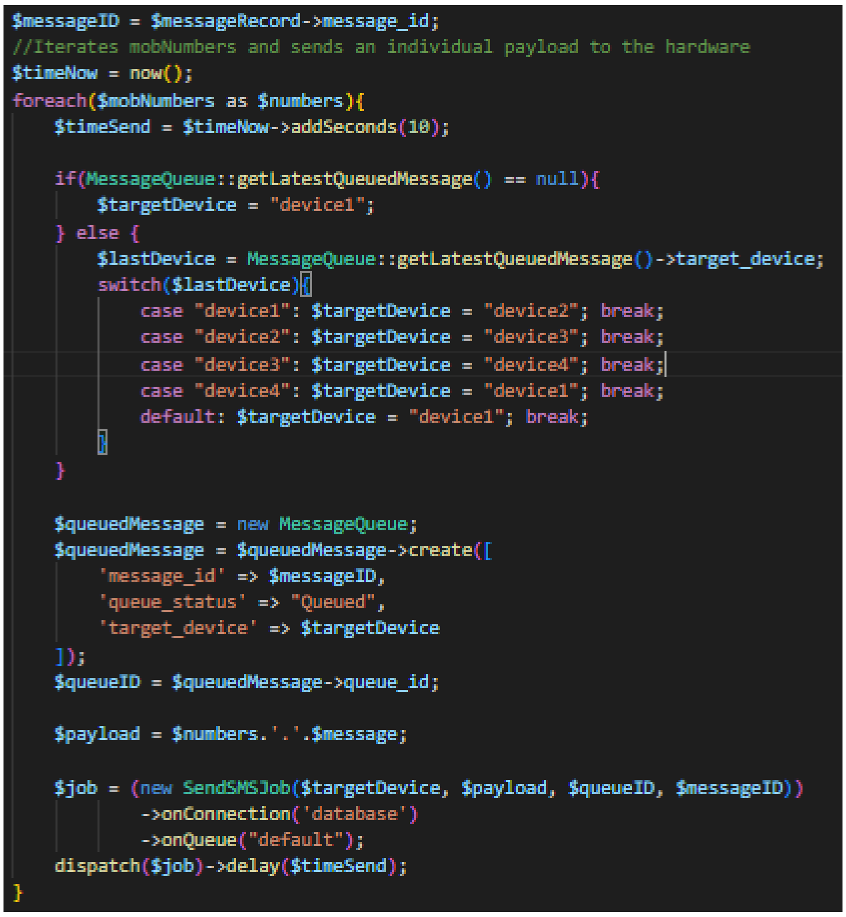


```php
$countryCode = "63";
$message = strval($request->message);
$groupID = $request->recipients;
$groupData = Group::getGroupInfo($groupID);
$groupRecipient = $groupData->students()->where('stud_phone', '<>', '')->get();
$senderName = $groupData->owner;
$mobNumbers = array();
$redirectRoute = 'dashboard';
$redirectParams = '';
if($groupData->is_group && $groupData->parentgroup == null) {
  $redirectRoute = 'group.showGroupInfo';
  $redirectParams = $groupData;
} else if($groupData->is_group && $groupData->parentgroup != null) {
  $redirectRoute = 'group.showSubGroupInfo';
  $redirectParams = ['group' => $groupData->parentgroup, 'subgroup' => $groupData];
}
//Retrieves the list of students and stores their numbers into an array
foreach($groupRecipient as $student) {
  $newNumber = preg_replace('/^0?/','+'.$countryCode,$student->stud_phone);
  array_push($mobNumbers, $newNumber);
}
$messageRecord = new Message;
$messageRecord = $messageRecord->create([
    'message_uuid' => Str::uuid()->toString(),
    'group_id' => $groupID,
    'message' => $message,
    'sender_name' => $senderName->first_name.' '.$senderName->last_name,
    'recipient_count' => count($mobNumbers),
    'status' => "Queued",
]);
```

Figure 3.9.3: ***Send Alert Message Controller Code Snippet***

The following shows the code snippets for *Send Alert Message* use case in the *MessageController*. This code snippet is responsible for creating and dispatching the job requests queued for processing by the queue worker. Firstly, it extracts the list of recipients for the messages and then places the mobile numbers of each recipient into the *$mobNumbers* array. Next, it creates an instance of the Message class containing the necessary data for recording purposes.

Round Robin Scheduling



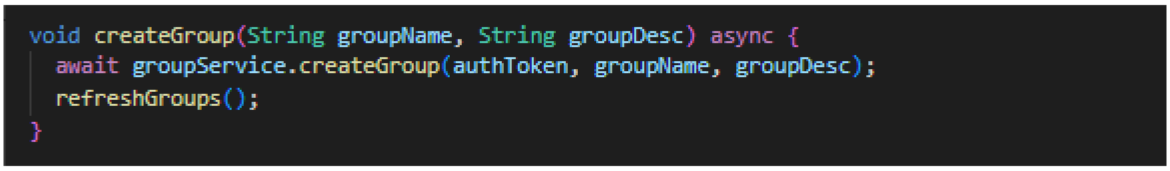
```
$messageID = $messageRecord->message_id;
//Iterates mobNumbers and sends an individual payload to the hardware
$timeNow = now();
foreach($mobNumbers as $numbers){
    $timeSend = $timeNow->addSeconds(10);

    if(MessageQueue::getLatestQueuedMessage() == null){
        $targetDevice = "device1";
    } else {
        $lastDevice = MessageQueue::getLatestQueuedMessage()->target_device;
        switch($lastDevice){
            case "device1": $targetDevice = "device2"; break;
            case "device2": $targetDevice = "device3"; break;
            case "device3": $targetDevice = "device4"; break;
            case "device4": $targetDevice = "device1"; break;
            default: $targetDevice = "device1"; break;
        }
    }

    $queuedMessage = new MessageQueue;
    $queuedMessage = $queuedMessage->create([
        'message_id' => $messageID,
        'queue_status' => "Queued",
        'target_device' => $targetDevice
    ]);
    $queueID = $queuedMessage->queue_id;

    $payload = $numbers.'.'.$message;

    $job = (new SendSMSJob($targetDevice, $payload, $queueID, $messageID))
            ->onConnection('database')
            ->onQueue("default");
    dispatch($job)->delay($timeSend);
}
```

Figure 3.10: ***Round Robin Scheduling Code Snippet***

The following shows the code snippet for *Round-Robin Scheduling* use case for Messages. This is a continuation of Figure 3.9.3, where a new message record has been created. Once a new message record has been created, it proceeds to iterate over each mobile number in the *$mobNumbers* array. Every iteration determines which target device this mobile number is set by checking the device of the latest message queued in the database, doing a round-robin process. Once a target device has been determined, it then creates a new instance of a message queue, then creates and dispatches a new job that the queue worker processes. The handling process can be seen back at Figure 3.9.1.

II. Mobile Application

Manage Group

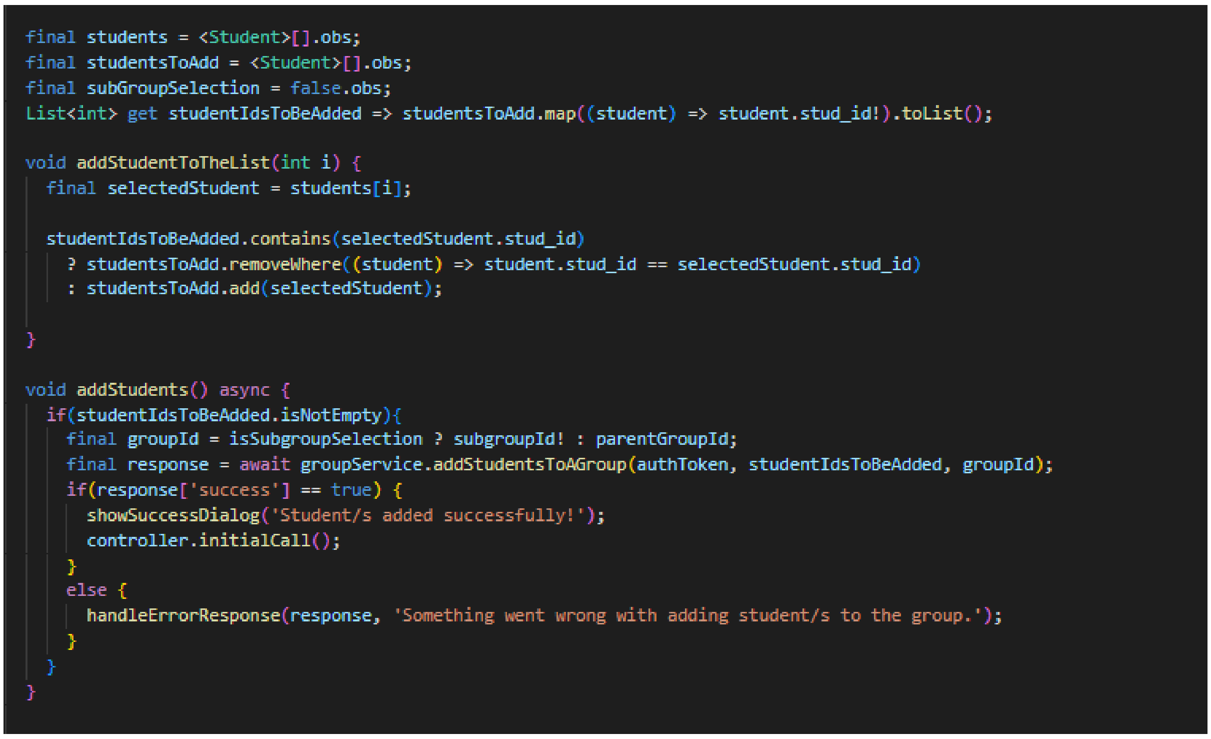


```
void createGroup(String groupName, String groupDesc) async {
  await groupService.createGroup(authToken, groupName, groupDesc);
  refreshGroups();
}
```

Figure 3.11: ***Create Groups Code Snippet***

The following shows the code snippet for *Manage Groups* use case for Groups. The snippet shows the creation of a group taking in previous variables as data parameters for the *createGroup* function in the service.

Add Student

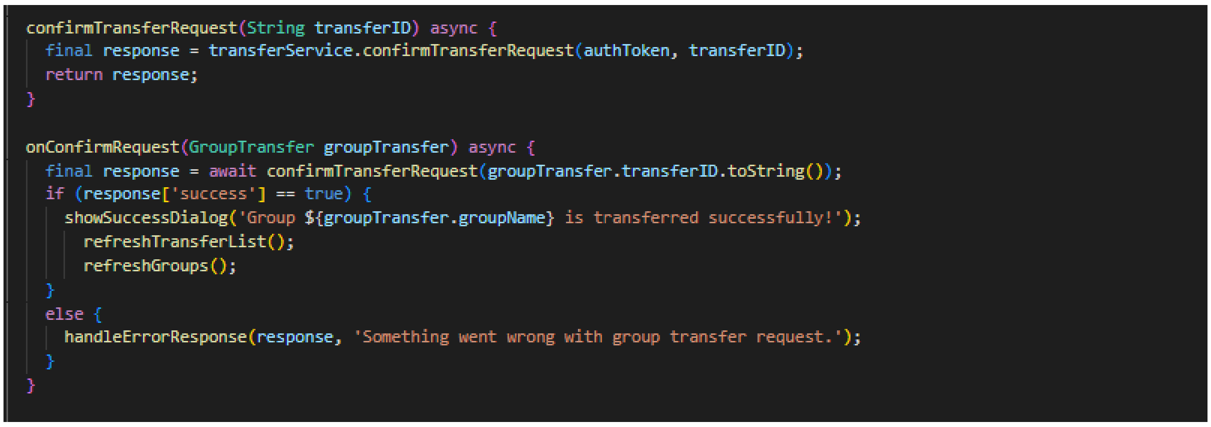


```
final students = <Student>[].obs;
final studentsToAdd = <Student>[].obs;
final subGroupSelection = false.obs;
List<int> get studentIdsToBeAdded => studentsToAdd.map((student) => student.stud_id!).toList();

void addStudentToTheList(int i) {
  final selectedStudent = students[i];

  studentIdsToBeAdded.contains(selectedStudent.stud_id)
    ? studentsToAdd.removeWhere((student) => student.stud_id == selectedStudent.stud_id)
    : studentsToAdd.add(selectedStudent);

}

void addStudents() async {
  if(studentIdsToBeAdded.isNotEmpty){
    final groupId = isSubgroupSelection ? subgroupId! : parentGroupId;
    final response = await groupService.addStudentsToAGroup(authToken, studentIdsToBeAdded, groupId);
    if(response['success'] == true) {
      showSuccessDialog('Student/s added successfully!');
      controller.initialCall();
    }
    else {
      handleErrorResponse(response, 'Something went wrong with adding student/s to the group.');
    }
  }
}
```

Figure 3.12: ***Add Student Code Snippet***

The following shows the code snippet for *Add Student* use case for Groups. The function *addStudentToTheList* checks if the student already exists in the list and either adds or removes them from the *studentsToAdd* list. The lower function, named *addStudents*, then determines if the group is a subgroup and then proceeds to add the students to the group. Depending on the outcome, it then shows a success dialog box or an error response.

Transfer Group Ownership

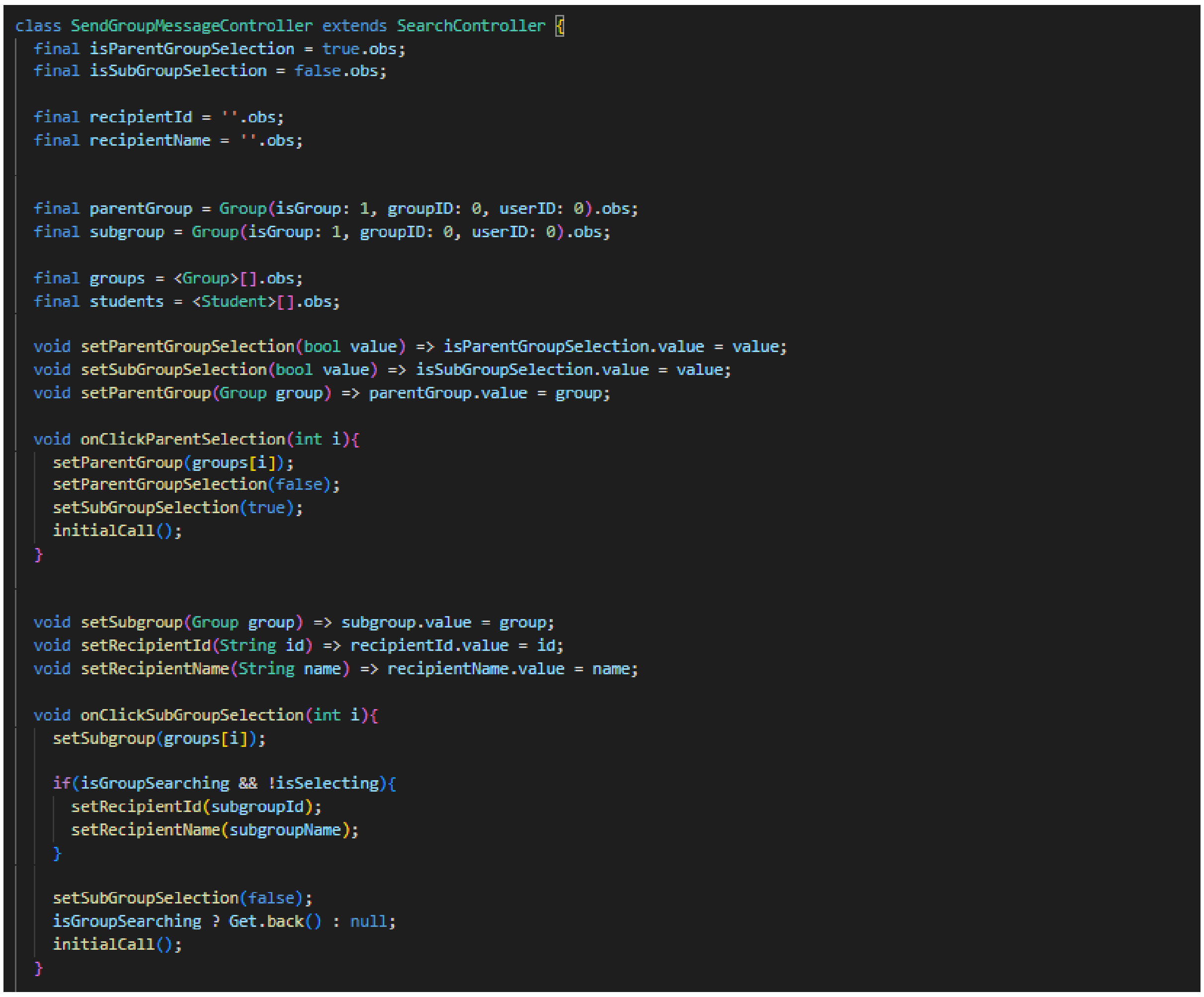


```
confirmTransferRequest(String transferID) async {
  final response = transferService.confirmTransferRequest(authToken, transferID);
  return response;
}

onConfirmRequest(GroupTransfer groupTransfer) async {
  final response = await confirmTransferRequest(groupTransfer.transferID.toString());
  if (response['success'] == true) {
    showSuccessDialog('Group ${groupTransfer.groupName} is transferred successfully!');
      refreshTransferList();
      refreshGroups();
  }
  else {
    handleErrorResponse(response, 'Something went wrong with group transfer request.');
  }
}
```

Figure 3.13: ***Transfer Group Ownership Code Snippet***

The following shows the code snippet for *Transfer Group Ownership* use case for Groups. The *onConfirmRequest* function is executed whenever the user clicks on the confirm button. This function waits for the response from *confirmTransferRequest*, which processes the transfer group use case and changes the owner id of the group. It then concurrently shows a success dialog or an error response depending on the result of the operation.

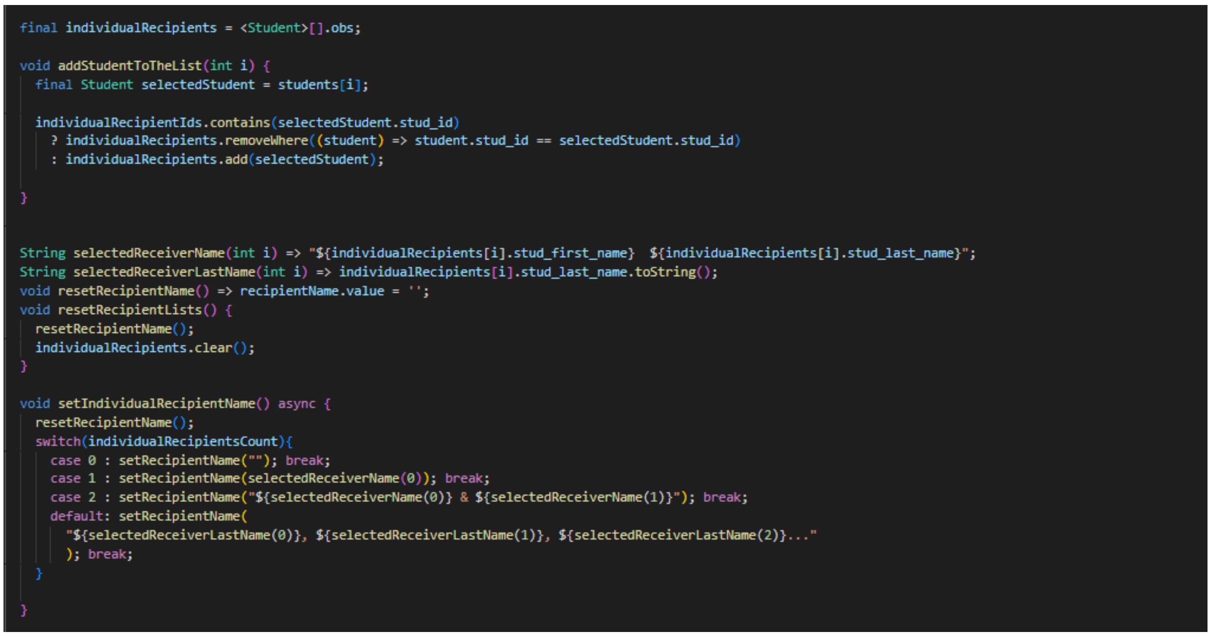
Send Alert Messages



```
class SendGroupMessageController extends SearchController {
  final isParentGroupSelection = true.obs;
  final isSubGroupSelection = false.obs;

  final recipientId = ''.obs;
  final recipientName = ''.obs;


  final parentGroup = Group(isGroup: 1, groupID: 0, userID: 0).obs;
  final subgroup = Group(isGroup: 1, groupID: 0, userID: 0).obs;

  final groups = <Group>[].obs;
  final students = <Student>[].obs;

  void setParentGroupSelection(bool value) => isParentGroupSelection.value = value;
  void setSubGroupSelection(bool value) => isSubGroupSelection.value = value;
  void setParentGroup(Group group) => parentGroup.value = group;

  void onClickParentSelection(int i){
    setParentGroup(groups[i]);
    setParentGroupSelection(false);
    setSubGroupSelection(true);
    initialCall();
  }


  void setSubgroup(Group group) => subgroup.value = group;
  void setRecipientId(String id) => recipientId.value = id;
  void setRecipientName(String name) => recipientName.value = name;

  void onClickSubGroupSelection(int i){
    setSubgroup(groups[i]);

    if(isGroupSearching && !isSelecting){
      setRecipientId(subgroupId);
      setRecipientName(subgroupName);
    }

    setSubGroupSelection(false);
    isGroupSearching ? Get.back() : null;
    initialCall();
  }
```

Figure 3.14.1: ***Send Alert Messages, Group Recipient Selection Code Snippet***

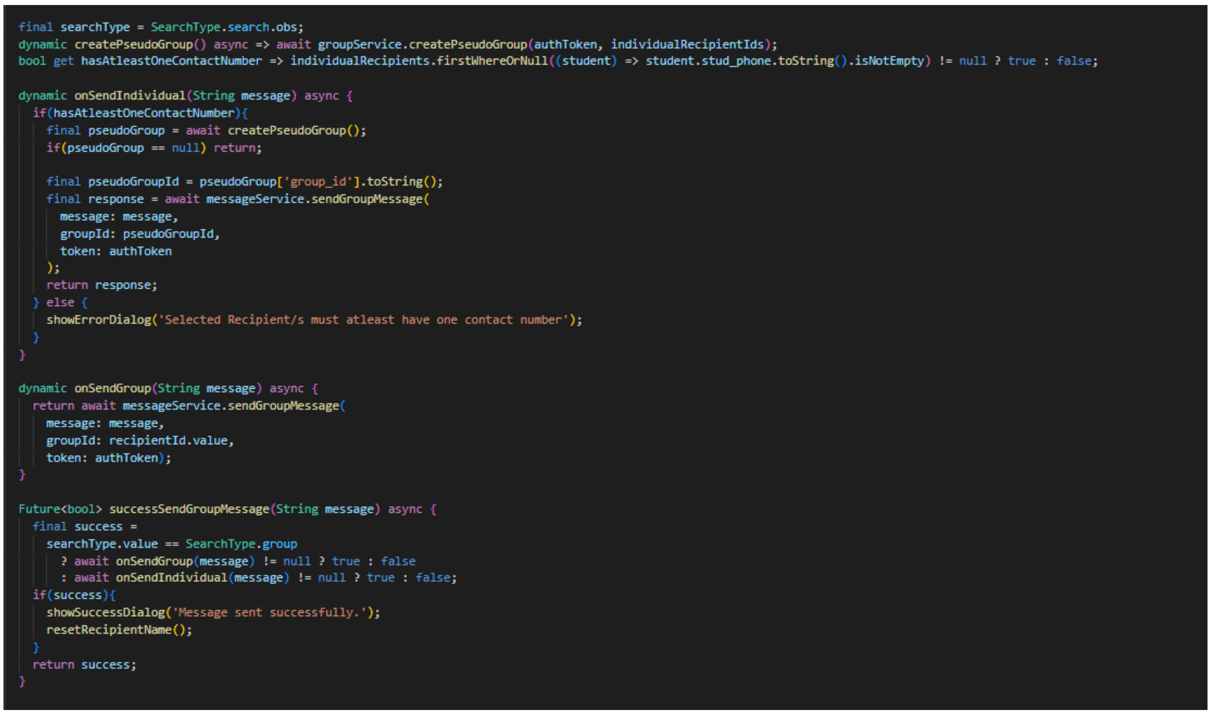
The following shows the code snippet for Recipient Selection for the *Send Alert Messages* use case. This is the first phase of *Send Alert Messages*, where it will determine the recipients by first identifying which parent group the message would be sent to. After selecting the parent group, the variables *isParentGroupSelection* and *isSubGroupSelection* would be reversed to show the screen for selecting which subgroup to send the messages to. The subgroup choices would also include the "All Students" option.



```
final individualRecipients = <Student>[].obs;

void addStudentToTheList(int i) {
  final Student selectedStudent = students[i];

  individualRecipientIds.contains(selectedStudent.stud_id)
    ? individualRecipients.removeWhere((student) => student.stud_id == selectedStudent.stud_id)
    : individualRecipients.add(selectedStudent);

}


String selectedReceiverName(int i) => "${individualRecipients[i].stud_first_name}  ${individualRecipients[i].stud_last_name}";
String selectedReceiverLastName(int i) => individualRecipients[i].stud_last_name.toString();
void resetRecipientName() => recipientName.value = '';
void resetRecipientLists() {
  resetRecipientName();
  individualRecipients.clear();
}

void setIndividualRecipientName() async {
  resetRecipientName();
  switch(individualRecipientsCount){
    case 0 : setRecipientName(""); break;
    case 1 : setRecipientName(selectedReceiverName(0)); break;
    case 2 : setRecipientName("${selectedReceiverName(0)} & ${selectedReceiverName(1)}"); break;
    default: setRecipientName(
      "${selectedReceiverLastName(0)}, ${selectedReceiverLastName(1)}, ${selectedReceiverLastName(2)}..."
      ); break;
  }

}
```

Figure 3.14.2: ***Send Alert Messages, Individual Recipient Selection Code Snippet***

The following shows the code snippet for Individual Recipient Selection for the *Send Alert Messages* use case. This is an alternative first phase of Send Alert Messages wherein the user selects specific individuals not in a group for a message. The *addStudentToTheList* function selects students to be part of the recipients to be sent the messages. The snippet then shows how the group name for the pseudo-group is created based on the algorithm shown in the function *setIndividualRecipientName*.

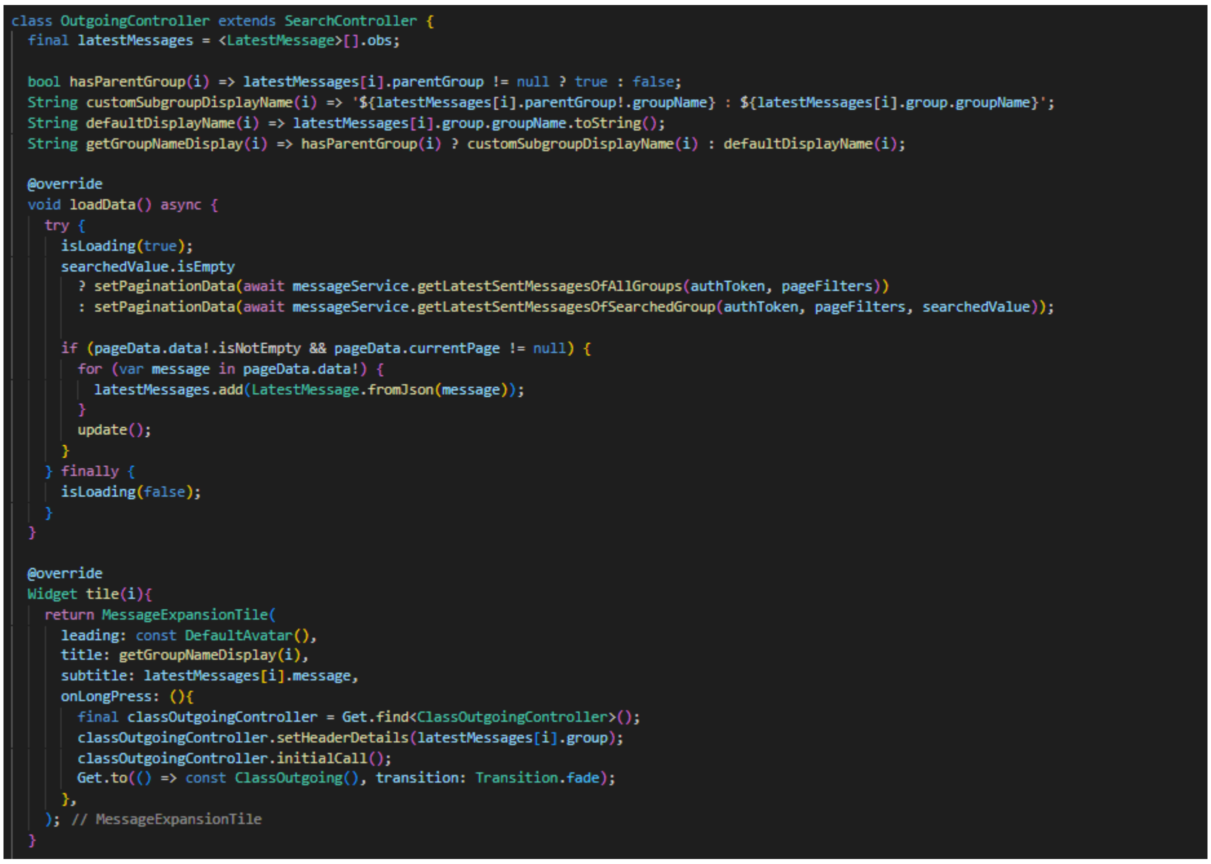


```
final searchType = SearchType.search.obs;
dynamic createPseudoGroup() async => await groupService.createPseudoGroup(authToken, individualRecipientIds);
bool get hasAtleastOneContactNumber => individualRecipients.firstWhereOrNull((student) => student.stud_phone.toString().isNotEmpty) != null ? true : false;

dynamic onSendIndividual(String message) async {
  if(hasAtleastOneContactNumber){
    final pseudoGroup = await createPseudoGroup();
    if(pseudoGroup == null) return;

    final pseudoGroupId = pseudoGroup['group_id'].toString();
    final response = await messageService.sendGroupMessage(
      message: message,
      groupId: pseudoGroupId,
      token: authToken
    );
    return response;
  } else {
    showErrorDialog('Selected Recipient/s must atleast have one contact number');
  }
}

dynamic onSendGroup(String message) async {
  return await messageService.sendGroupMessage(
    message: message,
    groupId: recipientId.value,
    token: authToken);
}

Future<bool> successSendGroupMessage(String message) async {
  final success =
    searchType.value == SearchType.group
      ? await onSendGroup(message) != null ? true : false
      : await onSendIndividual(message) != null ? true : false;
  if(success){
    showSuccessDialog('Message sent successfully.');
    resetRecipientName();
  }
  return success;
}
```

Figure 3.14.3: ***Send Alert Messages, Sending to Recipients Code Snippet***

The following shows the code snippet for Sending to Recipients of the *Send Alert Messages* use case. If the sending is for individual students, it will call the function *onSendToIndividual*, which creates a pseudo-group to house selected students. After that, it will call the *MessageService* and the *sendGroupMessage* function with the parameters. The same thing would happen for group sending as it will call the function *onSendToGroup*. The function will immediately call the MessageService and the *sendGroupMessage* function with its parameters. After which, may it be Individual or Group Sending, it will then show a success dialog or an error response depending on the result of the operation.

View Sent Alerts

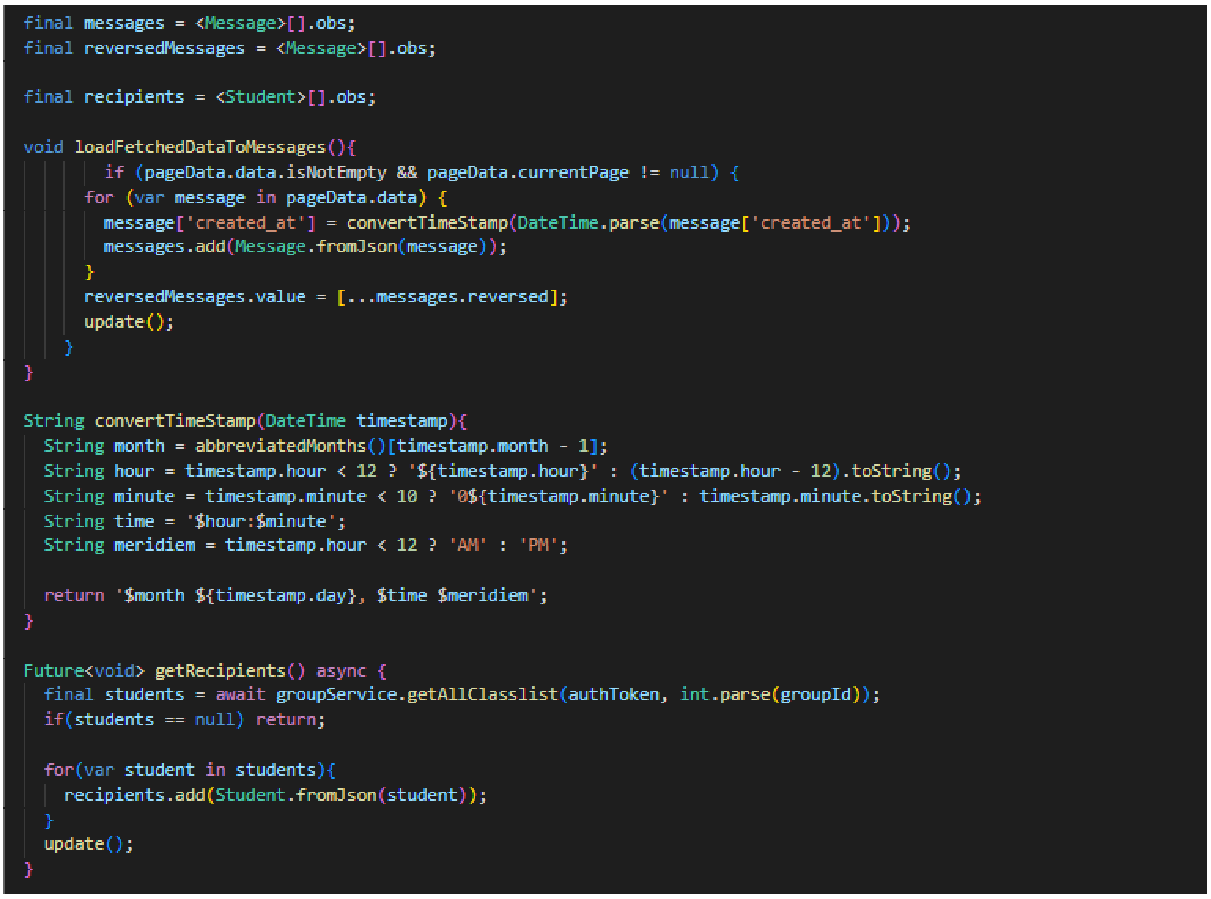


```
class OutgoingController extends SearchController {
  final latestMessages = <LatestMessage>[].obs;

  bool hasParentGroup(i) => latestMessages[i].parentGroup != null ? true : false;
  String customSubgroupDisplayName(i) => '${latestMessages[i].parentGroup!.groupName} : ${latestMessages[i].group.groupName}';
  String defaultDisplayName(i) => latestMessages[i].group.groupName.toString();
  String getGroupNameDisplay(i) => hasParentGroup(i) ? customSubgroupDisplayName(i) : defaultDisplayName(i);

  @override
  void loadData() async {
    try {
      isLoading(true);
      searchedValue.isEmpty
        ? setPaginationData(await messageService.getLatestSentMessagesOfAllGroups(authToken, pageFilters))
        : setPaginationData(await messageService.getLatestSentMessagesOfSearchedGroup(authToken, pageFilters, searchedValue));

      if (pageData.data!.isNotEmpty && pageData.currentPage != null) {
        for (var message in pageData.data!) {
          latestMessages.add(LatestMessage.fromJson(message));
        }
        update();
      }
    } finally {
      isLoading(false);
    }
  }

  @override
  Widget tile(i){
    return MessageExpansionTile(
      leading: const DefaultAvatar(),
      title: getGroupNameDisplay(i),
      subtitle: latestMessages[i].message,
      onLongPress: (){
        final classOutgoingController = Get.find<ClassOutgoingController>();
        classOutgoingController.setHeaderDetails(latestMessages[i].group);
        classOutgoingController.initialCall();
        Get.to(() => const ClassOutgoing(), transition: Transition.fade);
      },
    ); // MessageExpansionTile
  }
```

Figure 3.15: ***View Sent Alerts Code Snippet***

The following displays the code snippet for *View Sent Alerts* use case. The use case functions by extracting all necessary data to be placed in the tiles for the Outgoing screen. The extracted variables are then placed in the tile Widget to form the many tiles to be rendered once their respective pages are rendered. The connecting action toward the following use case, *View Conversation History*, can also be seen on the long press. This action would bind the *classOutgoingController* to the selected message group and pass this data to the *View Conversation History* screen.

View Conversation History



```
final messages = <Message>[].obs;
final reversedMessages = <Message>[].obs;

final recipients = <Student>[].obs;

void loadFetchedDataToMessages(){
        if (pageData.data.isNotEmpty && pageData.currentPage != null) {
      for (var message in pageData.data) {
        message['created_at'] = convertTimeStamp(DateTime.parse(message['created_at']));
        messages.add(Message.fromJson(message));
      }
      reversedMessages.value = [...messages.reversed];
      update();
    }
}

String convertTimeStamp(DateTime timestamp){
  String month = abbreviatedMonths()[timestamp.month - 1];
  String hour = timestamp.hour < 12 ? '${timestamp.hour}' : (timestamp.hour - 12).toString();
  String minute = timestamp.minute < 10 ? '0${timestamp.minute}' : timestamp.minute.toString();
  String time = '$hour:$minute';
  String meridiem = timestamp.hour < 12 ? 'AM' : 'PM';

  return '$month ${timestamp.day}, $time $meridiem';
}

Future<void> getRecipients() async {
  final students = await groupService.getAllClasslist(authToken, int.parse(groupId));
  if(students == null) return;

  for(var student in students){
    recipients.add(Student.fromJson(student));
  }
  update();
}
```

Figure 3.16: ***View Conversation History Code Snippet***

The following shows the code snippet for *View Conversation History* use case for Messages. The snippet shows the retrieval of all the messages in the message group selected. After the messages are retrieved, it is reversed to reproduce the feel of an actual messaging application. The conversion of the DateTime from the database to the frontend in the function *convertTimeStamp* and the retrieval of the recipients to be shown on the page can also be seen.

III. Hardware

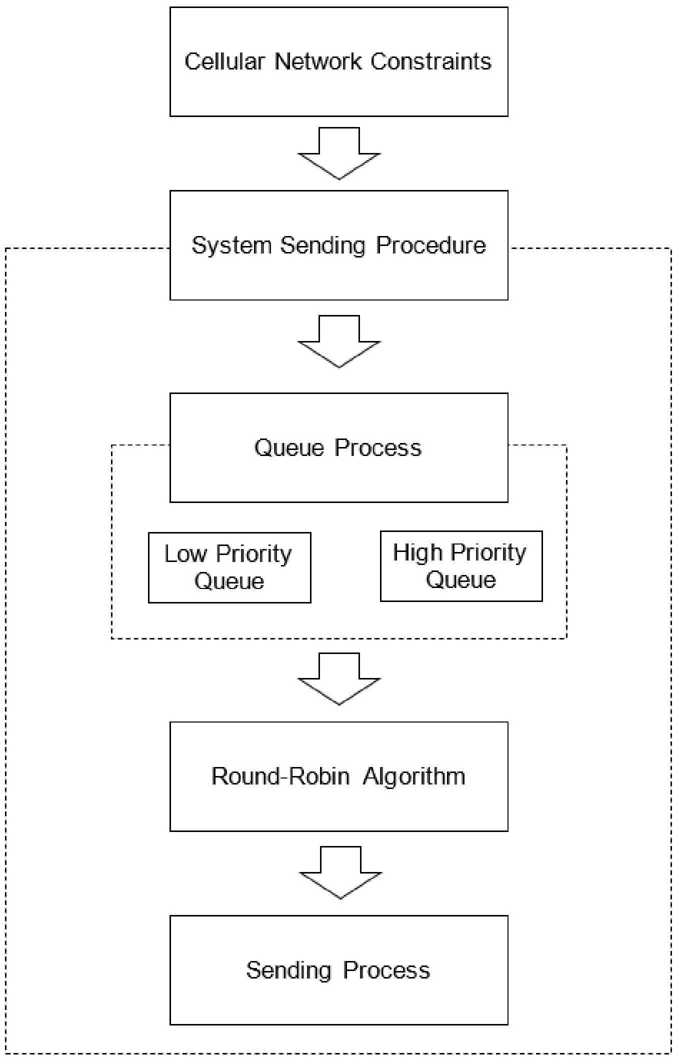
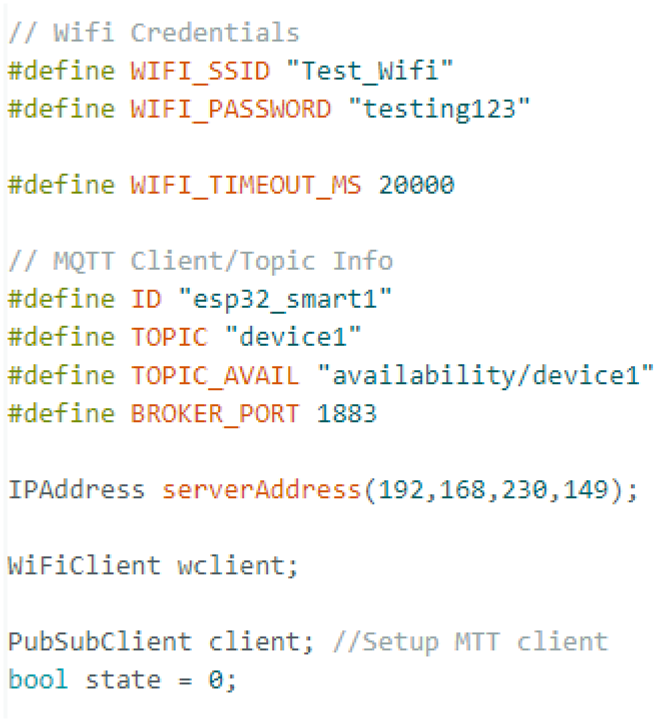


Figure 3.17: ***System Overview and Development Process***

This section will go over the technical steps involved during the development of the system. As a system primarily focuses on sending Short Message Service (SMS) messages, it is crucial to determine how the system should implement the SMS message process to ensure it arrives at its intended recipients. Figure 3.17 illustrates the System Overview and Development Process for the Send SMS functionality of the System, highlighting the Cellular Network Constraints and the System Sending Procedure. Primarily, before sending the SMS messages, it would undergo the System Sending Procedure, which is dependent on the Cellular Network Constraints, to determine the process in which the messages would be distributed to the different hardware that initiates the sending procedure.

Hardware Credentials and Parameters

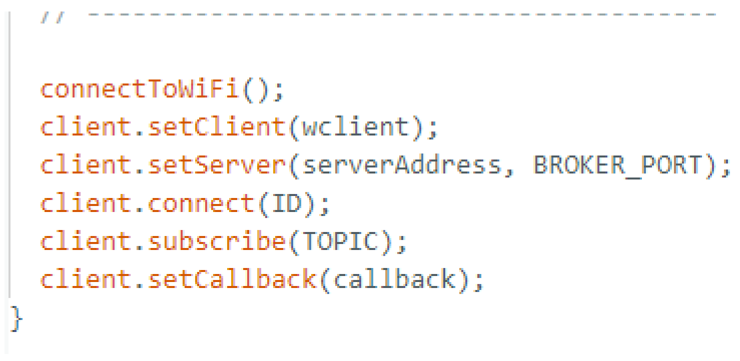


```
// Wifi Credentials
#define WIFI_SSID "Test_Wifi"
#define WIFI_PASSWORD "testing123"

#define WIFI_TIMEOUT_MS 20000

// MQTT Client/Topic Info
#define ID "esp32_smart1"
#define TOPIC "device1"
#define TOPIC_AVAIL "availability/device1"
#define BROKER_PORT 1883

IPAddress serverAddress(192,168,230,149);

WiFiClient wclient;

PubSubClient client; //Setup MTT client
bool state = 0;
```

Figure 3.18: ***Hardware Credentials and Parameters Code Snippet***

Figure 3.18 shows the credentials and parameters needed on the hardware side of the system. The Wi-Fi credentials only require the SSID and Password of the network the MQTT server is located. The MQTT Client side requires a unique ID that would be reflected in the MQTT server. TOPIC "device1" is the topic required for the device to subscribe in order to receive any messages published by the server in that topic. It changes from "device2" to "device4" for other devices. TOPIC_AVAIL, on the other hand, is the topic that the device publishes a "true" message when it is idle or available. The server would subscribe to this topic to check whether the device is available. BROKER_PORT is the port where the MQTT Server was assigned. Lastly, serverAddress requires the IP address of the server where the MQTT Server is running.

Hardware Setup Function

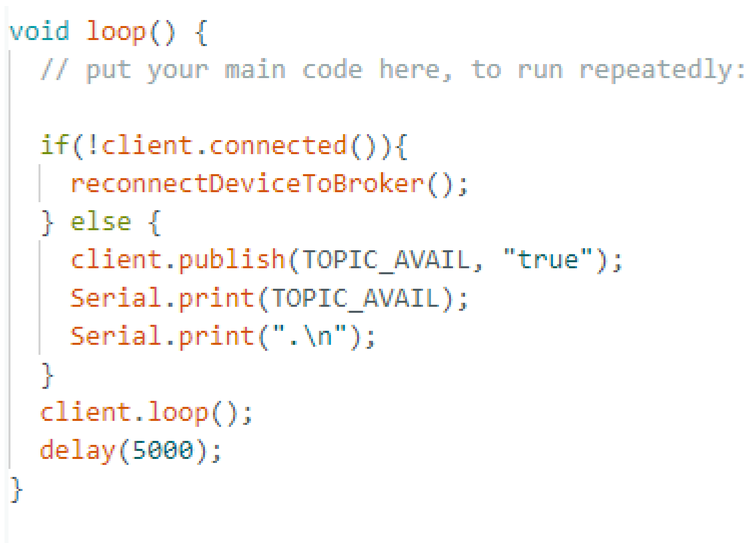


```
  // ---------------------------------------

  connectToWiFi();
  client.setClient(wclient);
  client.setServer(serverAddress, BROKER_PORT);
  client.connect(ID);
  client.subscribe(TOPIC);
  client.setCallback(callback);
}
```

Figure 3.19: ***Hardware Setup Function Code Snippet***

Figure 3.19 shows part of the code snippet for the setup function of the hardware device. The setup function is the function trigger once the device boots up. This part of the function first connects to the WIFI via the *connectToWifi()* function, then sets the client and connects to the MQTT server via the client.setClient and *client.setServer* functions. Once a connection has been made, it would register its client into the MQTT server with its ID and then subscribe to the TOPIC. Lastly, it sets the *setCallback* function, where it would boot whenever a message arrives at the subscribed TOPIC.

Hardware Loop Function

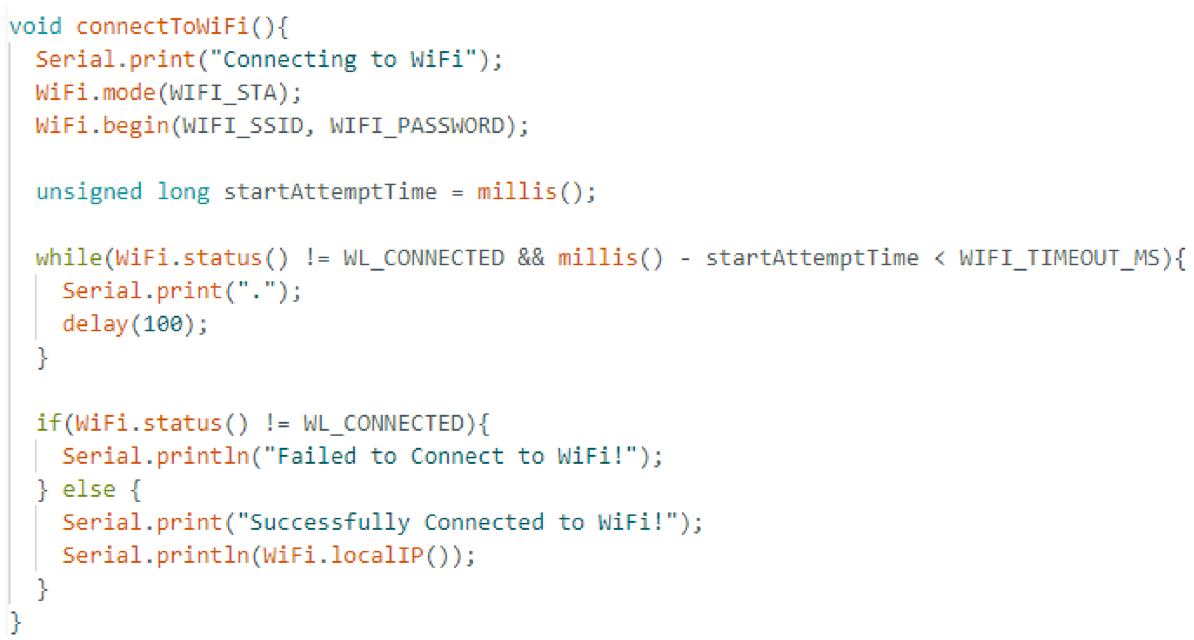


```
void loop() {
  // put your main code here, to run repeatedly:

  if(!client.connected()){
    reconnectDeviceToBroker();
  } else {
    client.publish(TOPIC_AVAIL, "true");
    Serial.print(TOPIC_AVAIL);
    Serial.print(".\n");
  }
  client.loop();
  delay(5000);
}
```

Figure 3.20: ***Hardware Loop Function Code Snippet***

Figure 3.20 shows the loop function of the device. The Loop function is a function that is triggered as long as the device is still connected to a power source. In this code snippet, the device would try to see if it is still connected to the MQTT server. The device would publish a "true" message to the TOPIC_AVAIL topic whenever the device is still connected or idle. This message is necessary for the server to check whether the device is still available. If not, it tries reconnecting to the server via the *reconnectDeviceToBroker* function. Since MQTT requires us to publish and subscribe in a loop manner constantly, it is necessary to place the *client.loop()* function to constantly check the connection.

Connection to WIFI

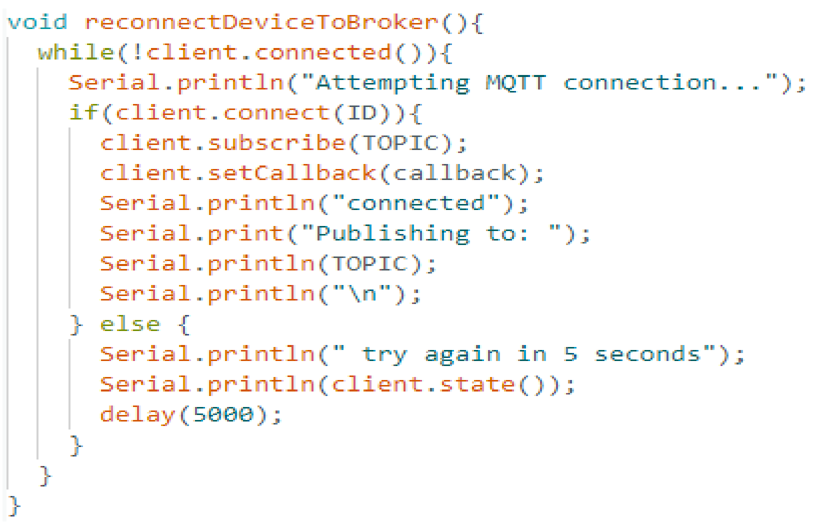


```
void connectToWiFi(){
  Serial.print("Connecting to WiFi");
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  unsigned long startAttemptTime = millis();

  while(WiFi.status() != WL_CONNECTED && millis() - startAttemptTime < WIFI_TIMEOUT_MS){
    Serial.print(".");
    delay(100);
  }

  if(WiFi.status() != WL_CONNECTED){
    Serial.println("Failed to Connect to WiFi!");
  } else {
    Serial.print("Successfully Connected to WiFi!");
    Serial.println(WiFi.localIP());
  }
}
```

Figure 3.21: ***Connection to WIFI Code Snippet***

Figure 3.21 shows the device's function to connect to the WIFI network. The function tries to connect to the WIFI network for the duration of the WIFI_TIMEOUT_MS listed in Figure 3.18 (20000ms = 20 seconds) and then prints out whether it has successfully connected.

Reconnect to Broker

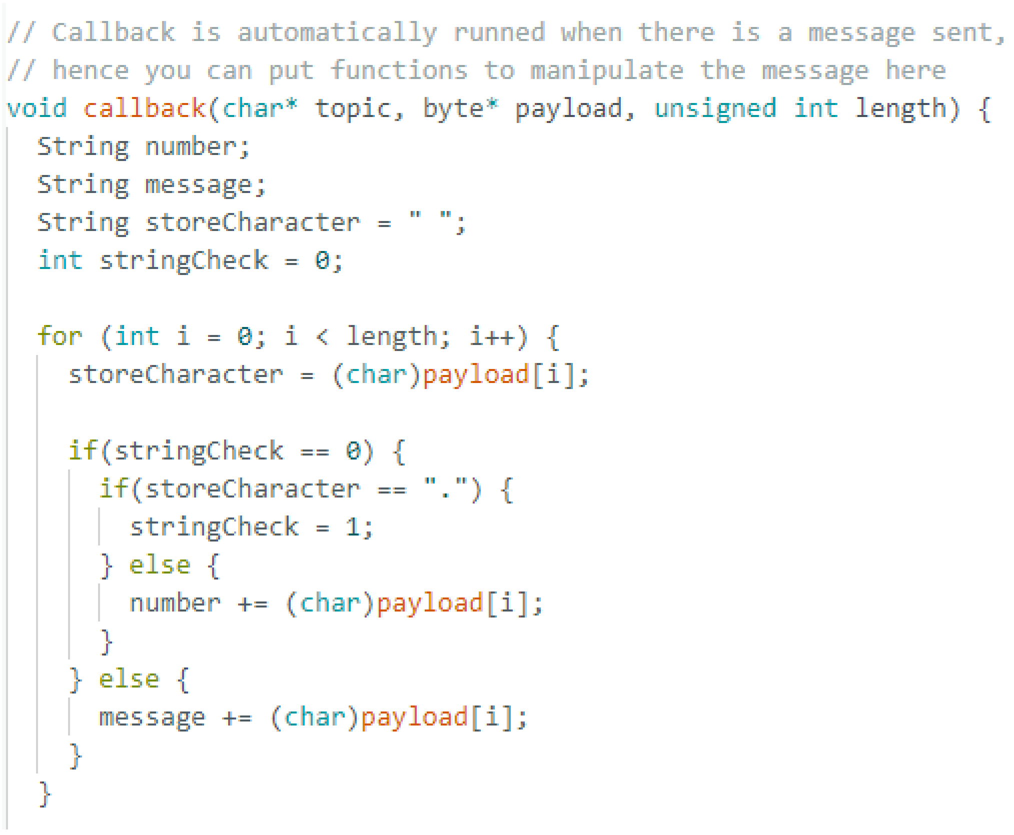


```
void reconnectDeviceToBroker(){
  while(!client.connected()){
    Serial.println("Attempting MQTT connection...");
    if(client.connect(ID)){
      client.subscribe(TOPIC);
      client.setCallback(callback);
      Serial.println("connected");
      Serial.print("Publishing to: ");
      Serial.println(TOPIC);
      Serial.println("\n");
    } else {
      Serial.println(" try again in 5 seconds");
      Serial.println(client.state());
      delay(5000);
    }
  }
}
```

Figure 3.22: ***Reconnect to Broker Code Snippet***

Figure 3.22 shows the reconnect function for the device. *client.connected()* is the function that returns a boolean that checks if the device is still connected to the MQTT network. If a connection did not exist, *client.connect(ID)* would try to reconnect to the MQTT server. Once connected, it would subscribe to the TOPIC and set the *setCallback* function again.

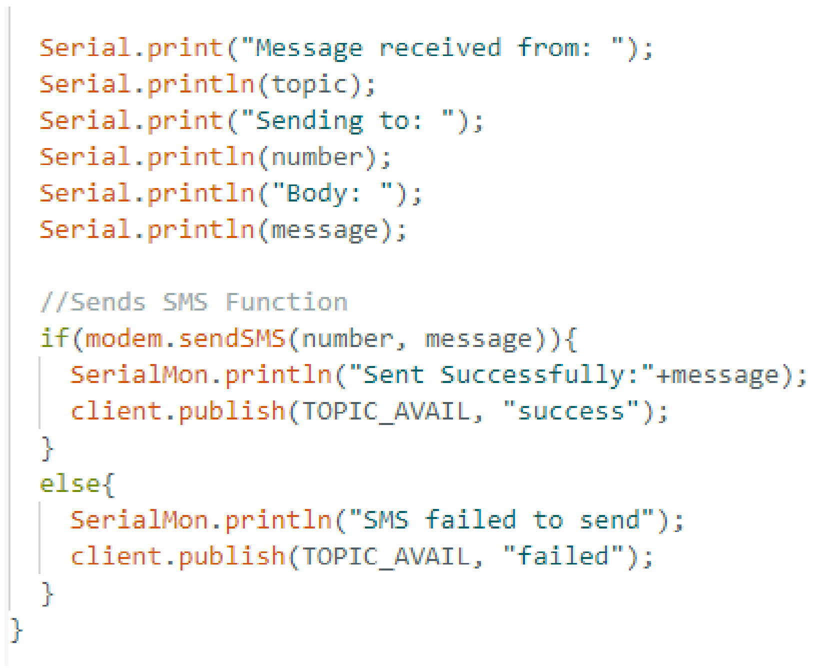
Callback Function



```
// Callback is automatically runned when there is a message sent,
// hence you can put functions to manipulate the message here
void callback(char* topic, byte* payload, unsigned int length) {
  String number;
  String message;
  String storeCharacter = " ";
  int stringCheck = 0;

  for (int i = 0; i < length; i++) {
    storeCharacter = (char)payload[i];

    if(stringCheck == 0) {
      if(storeCharacter == ".") {
        stringCheck = 1;
      } else {
        number += (char)payload[i];
      }
    } else {
      message += (char)payload[i];
    }
  }
```

Figure 3.23: ***Callback Function Code Snippet***

Figure 3.23 shows the code snippet for the callback function of the device. The callback function is the function that would be triggered once the device has received a message from the topic it was subscribing to, in this case, the TOPIC. Once a message has been received, it would separate the payload (which contains the mobile number and message content separated by ".") by iterating into every payload character and assigning them to the message and number variables.



```
  Serial.print("Message received from: ");
  Serial.println(topic);
  Serial.print("Sending to: ");
  Serial.println(number);
  Serial.println("Body: ");
  Serial.println(message);

  //Sends SMS Function
  if(modem.sendSMS(number, message)){
    SerialMon.println("Sent Successfully:"+message);
    client.publish(TOPIC_AVAIL, "success");
  }
  else{
    SerialMon.println("SMS failed to send");
    client.publish(TOPIC_AVAIL, "failed");
  }
}
```

Figure 3.24: ***Callback Function Code Snippet (contd.)***

Figure 3.24 is a continuation of the code snippet in Figure 3.22. Once the iteration for the payload has been completed, it will print out the necessary information into the terminal. Then, it triggers the *modem.sendSMS* function to send the message to the cellular tower. This function accepts a number and message and returns a boolean whether it is successful. Depending on the boolean value, if the message has been successful, the device would publish a "success" message into the TOPIC_AVAIL, which the server would receive. Otherwise, it would publish a "failed" message.

IV. System Sending Procedure

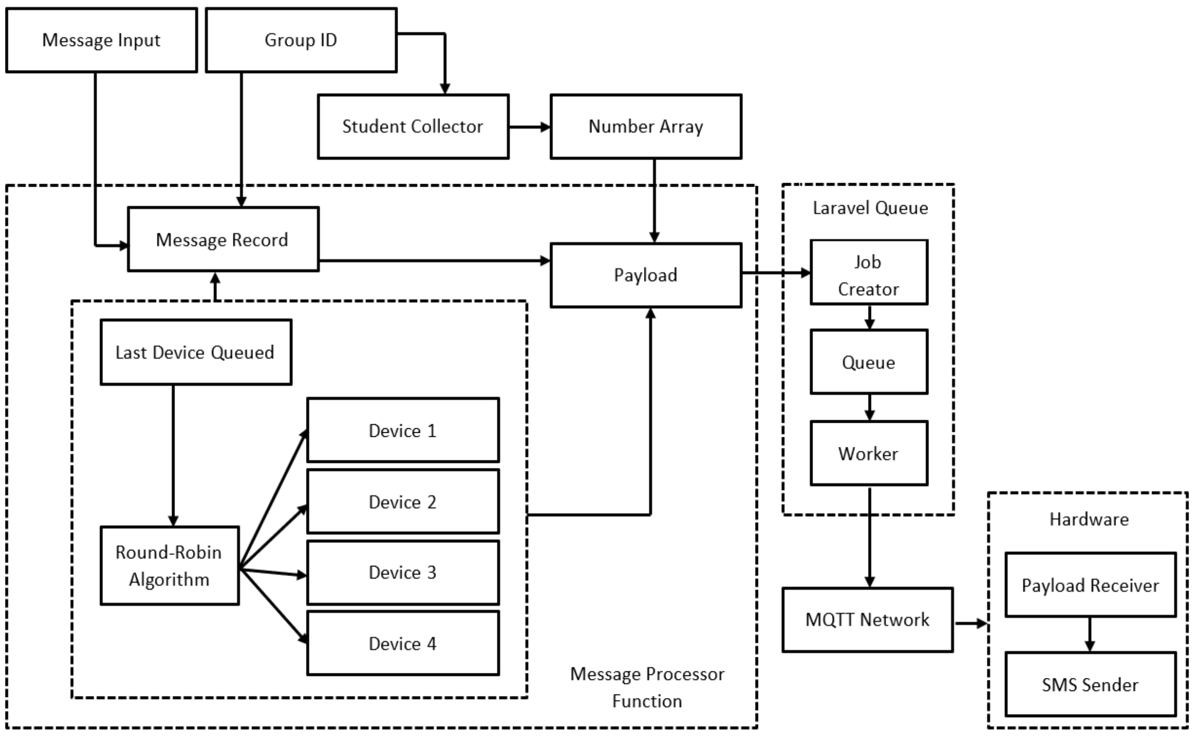


Figure 3.25: ***System Sending Procedure Diagram***

The System Sending Procedure is the procedure or process in which the system processes the message input and its recipient groups. The system must follow a specific set of instructions to achieve an efficient and effective way of delivering the message inputs to its intended recipients. It is also convenient for determining where an error might occur during the process. Figure 3.25 above shows the system overview of initiating the sending procedure.

The procedure starts with receiving two inputs from the user, the Message Input, which is the content of the message to be sent, and the Group ID. This identifier would point to a group that contains the student recipients. Using the Group ID, a function would be used to retrieve and store an array of student information within the Student Collection. From the Student Collection, each student's mobile number is retrieved and converted into a format containing the country code of the Philippines (+63) and stored in an array iterated in the Message Processor Function.

The Message Processor Function is a function that prepares the creation of a Job, which is a record of instructions processed asynchronously by the Laravel Queue and payload, which would be sent to the hardware for processing. When the function receives the Message Input and Group ID, it creates a new instance of a Message model, containing the message and its recipient group and being recorded into the Message table within the database. Additionally, information about the target device, which is the hardware it is intended to be processed to, is generated and would also be included in the message record. This target device is determined via a round-robin algorithm using the last target device used in the message record within the database. Once the record is created, the payload will be generated by appending the current student number being iterated to the message input.

This payload is then passed to create a Job along with the record's target device and message ID. This Job is then dispatched into the Laravel Queue, which would be executed depending on the priority set during creation. Within the Laravel Queue, the Worker is responsible for initiating the instructions within the Job. This Worker operates so that higher-priority jobs would be executed before other lower-priority jobs. During execution, the Job connects with the hardware stipulated in the target device information over the MQTT network and then passes the payload toward the hardware. Once the sending is successful, it then makes use of the message ID passed to change the status of the message record within the database. Once the hardware receives the payload, it extracts the recipient's number and message from it and stores them in different variables. The hardware then executes an SMS send function to send the message to the recipient's number, completing the process of sending the SMS to its intended recipient.

Queue Process

The system's Queue Process uses the inherent Laravel Queue feature within every Laravel application. The Laravel Queue works by executing a series of instructions, called Jobs, in a manner that is asynchronous to other application processes. For example, a user wishing to send a series of emails may continue to do other processes within an application without the need to wait for all emails to be sent. This is possible because of Laravel Queue, where such methods are processed independently and can be executed on the backend of the application while the application executes other processes.

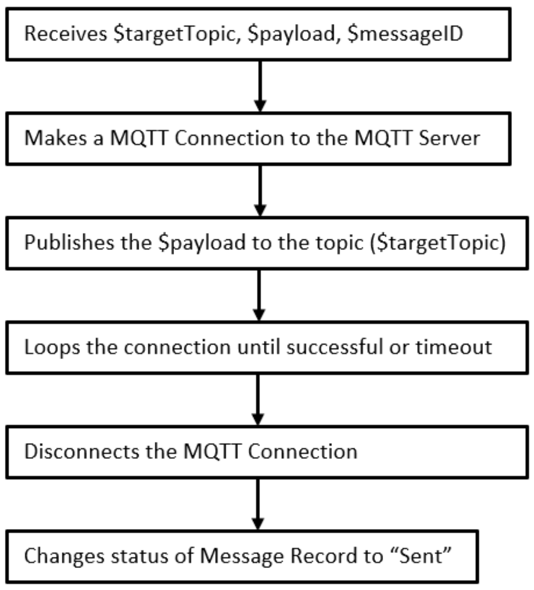

Receives $targetTopic, $payload, $messageID
Makes a MQTT Connection to the MQTT Server
Publishes the $payload to the topic ($targetTopic)
Loops the connection until successful or timeout
Disconnects the MQTT Connection
Changes status of Message Record to "Sent"

Figure 3.26: ***Process for Sending Payload to the Hardware Diagram***

The current Laravel application uses the queue to process the payload transfer from the application to the hardware. This transfer process is converted into a Job model, which is used every time the system wants to pass payload from the application to the hardware. Above is Figure 3.28 shows the Job's basic instruction process for sending payload to the hardware.

Whenever a job is created, it is stored in a specific table within the database called "jobs". This is where the queue references which Job to execute based on the priority value of that particular job data. Additionally, this is also where the queue of jobs is stored. The Worker runs each Job. The Worker is responsible for executing the jobs in the queue based on their priority value, settled upon activating the Worker in the console. If the Job fails to be executed after a certain number of attempts, it is redirected to the "failed_jobs" table within the database. This table contains the same attributes as the "jobs" table with only the addition of an error log that shows the content of the error when the Worker failed to complete the Job. Such entries in this table can be re-executed by activating another worker for failed jobs in the console.

Round-Robin Algorithm

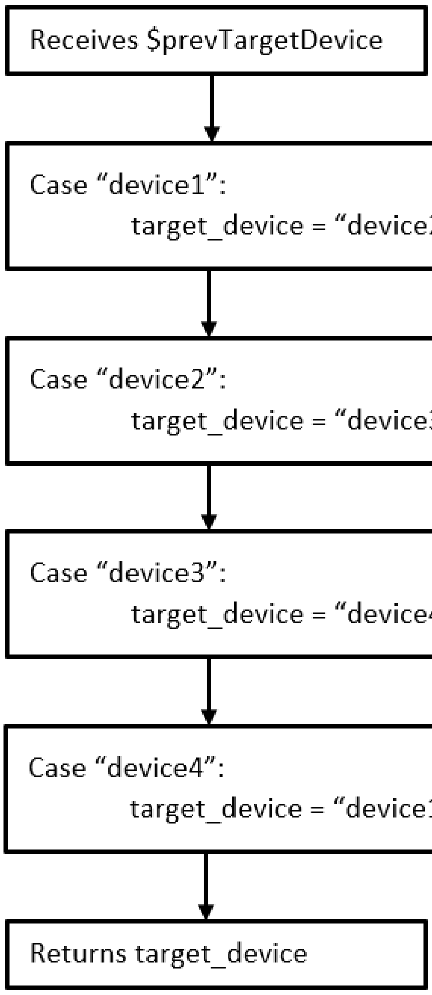

Receives $prevTargetDevice
Case "device1":
target_device = "device2"
Case "device2":
target_device = "device3"
Case "device3":
target_device = "device4"
Case "device4":
target_device = "device1"
Returns target_device

Figure 3.27: Basic Round-Robin Algorithm Diagram

The Round-Robin Algorithm used in the system is a simple algorithm where every message record inputted is paired to one device, and the next record is pointed to the next adjacent device, generally going back to the first after the last device. The algorithm works by retrieving the message record's last entry and extracting its target device's value. Once done, the new target device value would be decided based on the next device adjacent to the previous target device. For example, the previous value of the target device is "device1". Hence the value of the new target device would now be "device2". The current system only uses four devices. Thus the values of the target device can only be "device1", "device2", "device3", and "device4". The following figure above shows the diagram of how the Round-Robin Algorithm decides the value of the target device. You can refer to Figure 3.10.1 for a more detailed explanation of how the round-robin algorithm is handled when there are message failures or unavailable devices.

Sending Process

The Sending Process involves communicating between the server application and hardware via the MQTT network and sending SMS of the hardware to its intended recipients via the Cellular Network. From the server application, the Laravel Queue executes the Jobs in the queue where each Job would connect with the MQTT server and pass the payload to the hardware via publish method. When the hardware connecting to the MQTT server receives the payload, it executes a callback function where it processes the payload, then sends an SMS generated from the payload to the cellular network to its destination phone number.

MQTT

MQTT (MQ Telemetry Transport) is a lightweight, publish-subscribe model, machine-to-machine network protocol used for Message queue/Message queuing service. It is commonly used for connecting devices with limited network bandwidth and resource constraints. Hence, it is an OASIS standard messaging protocol for the Internet of Things (IoT). However, an MQTT Broker is needed for such situations to make communications between devices more convenient and successful. An MQTT Broker is a central software entity that liaises between clients sending messages and subscribers receiving those messages. This system uses the Mosquitto Broker as the broker for the MQTT Server.

As a publish-subscribe model, MQTT uses the Topics, Publish and Subscribe methods to facilitate transferring messages/data. Topics are information hubs where messages are transferred from one device to the other. In order for devices to receive any messages from the Topic, they need to Subscribe to a particular Topic. For example, Temperature LED devices that output temperature data need to subscribe to the "temperature_sensor" topic before receiving any messages/data from this Topic. On the other hand, devices that send messages/data need to Publish to a particular Topic the data they want to send to all subscribed devices to that topic.
MQTT also has Quality of Service (QoS) levels when publishing data on a Topic. QoS Level 0 is the default QoS which doesn't guarantee message delivery. QoS Level 1 guarantees message delivery but could get duplicates. Lastly, QoS Level 2 guarantees message delivery with no duplicates.

The Laravel Application in this system uses a package called "php-mqtt/laravel-client" to facilitate communication with the MQTT Broker. Among the functions in this package, it is possible to set the level of QoS when publishing data over the MQTT Network. These functions can be utilized in the Job model for sending the payload to the MQTT network.

Hardware

The hardware used for this system is an "ESP32 LILYGO TTGO T-Call V1.4". It is an ESP32 Microcontroller integrated with the SIM800L GSM Module that allows the microcontroller to send GPRS transmission, send and receive SMS and make and receive voice calls. In addition, it also has Bluetooth and Wi-Fi capabilities, making it possible to communicate with the device wirelessly. This hardware uses Arduino IDE to code the programs for the hardware and the "PubSubClient" library to enable the device to connect with the MQTT broker via Wi-Fi.

The responsibility of the hardware to the system is to send SMS messages to the Cellular Network via the SIM it is equipped with. Depending on the SIM, the number of messages sent by the hardware may be limited. Using Globe SIM would limit the device to 499 messages daily while using Smart SIM would enable the device to send 3,160+ messages daily. Hence, Smart SIM is used in the hardware to allow more messages to be sent per day.

Because the hardware only needs to wait for data to be received, the main role of the device in the MQTT Network is to serve as a subscriber for a particular topic. In this system, the researchers make use of four devices, and each device would have its Topic, namely, "device1", "device2", "device3", and "device4". Each device would be subscribed to these topics waiting for any published data within their respective Topics before sending the SMS. The process of sending the SMS message to its intended recipients is composed of waiting for the payload from the server and processing the payload as an SMS to be sent to the cellular network. The device constantly connects to the MQTT Network through the MQTT Subscribe function, waiting for any published payload. Once a payload is received, a callback function, a function that is triggered when there is published data, is initiated to process the payload and then send an SMS message to its recipient via the cellular network.

MQTT Subscribe Function

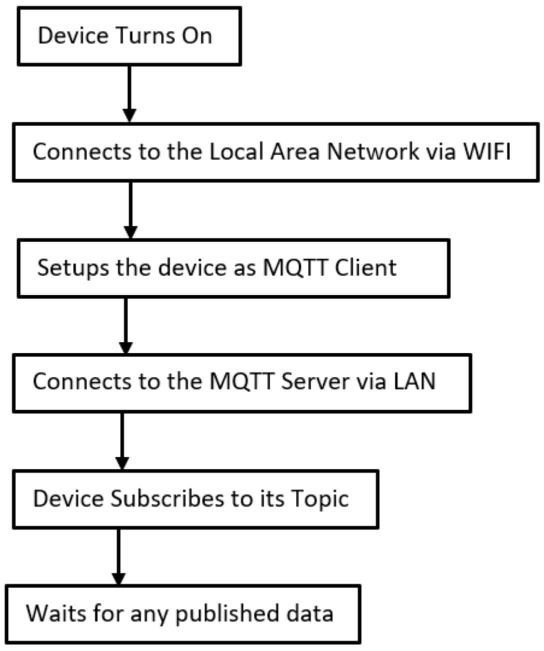

Device Turns On
Connects to the Local Area Network via WIFI
Setups the device as MQTT Client
Connects to the MQTT Server via LAN
Device Subscribes to its Topic
Waits for any published data

Figure 3.28: MQTT Subscribe Function Diagram

Figure 3.28 shows the process diagram of how the device connects to the MQTT Network, subscribes to a topic, and then waits for published data. When the device turns on, it waits 10 seconds to boot up the SIM800L module and allow it to connect to the cellular network. A blinking red light in the device indicates a successful connection every three seconds. After that, it starts to connect with a Local Area Network via Wi-Fi. Once it connects to the LAN, the device will set up its client information and try to connect with the MQTT Server via the LAN. If it fails to connect to the MQTT Server, it will re-attempt to connect every 5 seconds until it reconnects. Once connected to the MQTT Server, it would subscribe to its Topic and wait for any published data under that topic. If it received published data under its Topic, it would initiate the callback function for processing the payload and sending an SMS to the cellular network.

Send SMS Function

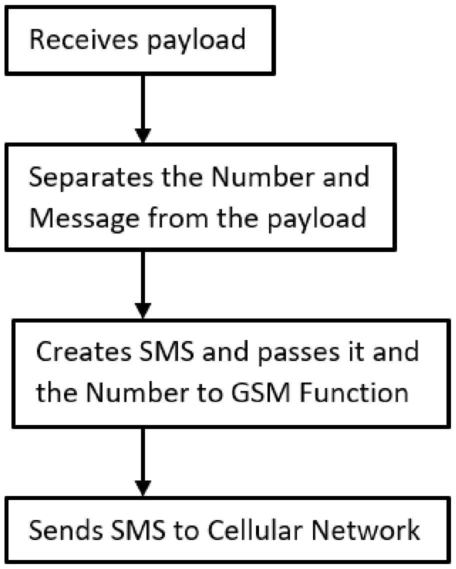

Receives payload
Separates the Number and Message from the payload
Creates SMS and passes it and the Number to GSM Function
Sends SMS to Cellular Network

Figure 3.29: Send SMS Function Diagram

Figure 3.29 shows the callback function's process diagram for processing the payload and sending the SMS to the cellular network. Once the device receives a payload from the topic, it separates any characters before the first period (.) and stores them in a variable called "number." In contrast, characters after the first period would be stored in a variable called "message ."It is important to note that during the payload creation, the recipient's number was appended at the front of the message string and was separated with a period. Once the payload has been processed, the number and message are passed onto a GSM function, which takes in an SMS and the recipient's number. The GSM function then connects to the SIM800L and sends an SMS to the recipient's number via the cellular network. Hence, completing the procedure of processing the payload and sending the SMS to its intended recipient's number via the cellular network.

Testing Process

This section covers the functional and performance testing results done on the system during development. Each table represents a test case performed on a major use case defined in the system, along with its description, platform used (web or mobile), expected and actual results, and test data. These tests ensure that the system works as expected and its requirements are met.

I. Functional Suitability

Test Case 001: Set Current Year and Term

Use Case	Set Current Year and Term
Platform	Web
Description	Enter valid year and term
Test Data	Selects from dropdown: 2nd Semester Moves year roulette to: 2022-2023
Expected Result	System sets the current term and year as inputted.
Actual Result	System sets the current term and year as inputted.
Status	Pass

Table 3.1: Set Current Year and Term Test Case

The following table shows the test case of the *Set Current Year and Term* use case with proper inputs. With the user selecting the inputs for the term and year, changes to the system would be applied.

Test Case 002: Import University Data

Use Case	Import University Data
Platform	Web
Description	Upload valid student details
Test Data	StudNumber: 2022010001 Name: Cuico, Kenneth
Expected Result	- System displays a notice of the successful operation - Student added to the master list
Actual Result	- System displays a notice of the successful operation - Student added to the master list
Status	Pass

Table 3.2.1: Import University Data Test Case 1

The following table shows the *Import University Data* use case test case with the proper CSV format and headers (StudNumber and Name). With the user providing the correct entries for each field, the appropriate changes would be reflected in the database with its corresponding success notification.

Test Case 003: Import University Data

Use Case	Import University Data
Platform	Web
Description	Upload invalid student details
Test Data	StudNumber: 20220100 1

	First Name: Kenneth Last Name: Cuico
Expected Result	System displays a notice indicating that there is an error in the format
Actual Result	System displays a notice indicating that there is an error in the format
Status	Pass

Table 3.2.2: Import University Data Test Case 2

The following table shows the test case of the *Import University Data* use case with the incorrect CSV format and header names. With the user providing the incorrect entries for each field, the system would produce an error with its corresponding error message.

Test Case 004: Import University Data

Use Case	Import University Data
Platform	Web
Description	Upload missing student details
Test Data	StudNumber: 2022010001 Name:
Expected Result	System displays a notice indicating that there is an error in the format
Actual Result	System displays a notice indicating that there is an error in the format
Status	Pass

Table 3.2.3: Import University Data Test Case 3

The following table shows the test case of the *Import University Data* use case with some CSV headers left undefined. With the user providing no entries for some fields, the system would produce an error with its corresponding error message.

Test Case 005: View Hardware Status

Use Case	View Hardware Status
Platform	Web
Description	View hardware status together with the message it is currently sending
Test Data	Click “View Device Status”
Expected Result	System displays a modal containing all the device status and the message it is currently sending.
Actual Result	System displays a modal containing all the device status and the message it is currently sending.
Status	Pass

Table 3.3: View Hardware Status Test Case

The following table shows the *View Hardware Status* use case test case with the user clicking the "View Device Status" on the Alert Logs screen. With the user's actions, it displays the details of the device status. The modal displays the following: Device, Availability, and Status.

Test Case 006: Manage Users

Use Case	Manage Users
Platform	Web
Description	Accept registration request
Test Data	Registrant User Id: 306 Name: Gene Abello College Id: 100 (SCS) Role Id: 201 (faculty)
Expected Result	System grants Gene Abello the privileges of a faculty
Actual Result	System grants Gene Abello the privileges of a faculty
Status	Pass

Table 3.4.1: Manage Users Test Case 1

The following table shows the test case of the *Manage Users* use case with the user clicking the “Approve” button in the registrations screen. With the user’s action, the user “Gene Abello” would now have access to the system using the credentials of user_id 306.

Test Case 007: Manage Users

Use Case	Manage Users
Platform	Web
Description	Deny registration request
Test Data	Registrant User Id: 307

	Name: Al Galinea College Id: 100 (SCS) Role Id: 201 (faculty)
Expected Result	System denies the registering user, and removes their credentials and data from the database
Actual Result	System denies the registering user, and removes their credentials and data from the database
Status	Pass

Table 3.4.2: Manage Users Test Case 2

The following table shows the test case of the *Manage Users* use case with the user clicking the “Deny Entry” button in the registrations screen. With the user’s action, the row entry having the user_id 307 would be declined. The row would then be deleted from the *users* table.

Test Case 008: Manage Groups (Create)

Use Case	Manage Groups
Description	Enter valid group name and description
Test Data	Group Name: 12032 Group Description: Mobile Development 3
Expected Result	System displays the group with 12032 as its name
Actual Result	System displays the group with 12032 as its name
Status	Pass

Table 3.5.1: Manage Groups Test Case 1

The following table shows the test case of the *Manage Groups* use case. The corresponding group would be created with the user supplying the correct details into the respective fields.

Test Case 009: Manage Groups (Create)

Use Case	Manage Groups
Description	Enter missing group name and/or filled description
Test Data	Group Name: Group Description: Mobile Development 3 Group Name: Group Description:
Expected Result	System focuses on group name field with pop-up error saying 'Please fill out this field'
Actual Result	System focuses on group name field with pop-up error saying 'Please fill out this field'
Status	Pass

Table 3.5.2: Manage Groups (Create) Test Case 2

The following table shows the test case of the *Manage Groups* use case. With the user not supplying the details into the fields, an error will occur, highlighting the missing field and an error message.

Test Case 010: Add Students

Use Case	Add Students
Platform	Web
Description	Add individual students to their course or group
Test Data	Group id: 512 Group name: Web Dev Selects: 1. Kenneth Cuico 2. Joshua Libradilla 3. Shane Kian Salsbury
Expected Result	System adds Cuico, Libradilla and Salsbury to the Web Dev group
Actual Result	System adds Cuico, Libradilla and Salsbury to the Web Dev group
Status	Pass

Table 3.6: Add Students Test Case

The following table shows the test case of the *Add Students* use case. With the user's action of selecting the checkboxes beside the students in the Web Application and the user's action of tapping the tiles of the students in the Mobile Application, it would select the respective students. After pressing the "Add to Member List" button, the selected students would then be included in the group.

Test Case 011: Send Alert Messages (Send to Specific Group)

Use Case	Send Alert Messages
Description	Send a SMS Message to students in his/her assigned courses and assign specific recipients
Test Data	Recipient: Group Id: 500 Group Name: 11021 Group Description: Net 2
Expected Result	Notification of SMS message sent status
Actual Result	Notification of SMS message sent status
Status	Pass

Table 3.7.1: Send Alert Messages Test Case 1

The following table shows the test case of the *Send Alert Messages* use case. In the Web Application, the user selects a specifically created group in the "Manage Groups" view and presses the envelope icon. In the Mobile Application, The user presses the "Send a Group Text" card on the Home Screen, then selects a group from the recipients and the subgroup.

The user then inputs the message in the text field, and the system accordingly processes the message to create a row in the *message_queue* table.

Test Case 012: Send Alert Messages (Send to Individual/Custom Group)

Use Case	Send Alert Messages

Description	Send a SMS Message to students in his/her assigned courses and assign specific recipients
Test Data	Recipients: Group Id: 520 (pseudo) Group Name: Libradilla, Cuico, Salsbury Student Ids: 2022010001 2022010002 2022010005
Expected Result	Notification of SMS message sent status
Actual Result	- SMS queued in database - Notification of SMS message sent status
Status	Pass

Table 3.7.2: Send Alert Messages Test Case 2

The following table shows the test case of the *Send Alert Messages* use case. In the Web Application, the user selects unique students from the "Send Indv. Message" card in the Dashboard and taps the envelope icon. In the Mobile Application, the user presses the "Send a Text" card on the Home Screen and selects unique students from the recipients.

The user then inputs the message in the text field, and the system accordingly processes the message to create a row in the *message_queue* table.

II. Performance Testing

Cellular Network Provider Constraints

The development process starts with identifying the different cellular network constraints when sending SMS messages over their network, specifically the number of SMS messages each cellular network sim can send before being restricted. Once those constraints are identified, they will be the basis for determining the System Sending Procedure. Hence, the researchers must explore and identify the constraints of the available cellular networks in the area. For this study, the researchers have identified two major cellular networks (Smart and Globe) that can be utilized for our system. It is important to note that the hardware used for this study, the SIM800L, only works in areas where a 2G network is available. Only Smart and Globe cellular towers allow 2G network communications among the cellular network providers. As a new cellular network provider, DITO does not have a cellular tower that allows 2G network communication. Lastly, the cellular network SIMs utilized in this study would use Unlimited-Text to All Network Promos to minimize expenses when sending SMS over the cellular network.

1.0 Globe

Globe Telecom, Inc., commonly known as Globe, is one of the major telecommunication companies that offers its services in the Philippines. With almost 88 million subscribers, Globe operates the largest mobile network in the Philippines and offers its customers commercial wireless services through its 2G, 3G, 3.5G

HSPA+, 4G LTE, and LTE-A networks, with 5G. Hence, Globe was chosen as one of the cellular network providers in this study.

1.1 Fair Use Policy

Based on its website's Fair Use Policy for Unlimited Promos, Unlimited Promos in Globe Cellular Network does not offer limitless chances for its subscribers to send SMS messages to all networks. Globe asserts its rights to terminate any exploitation of the promos with a 30-day deactivation of the Sim. Despite mentioning a threshold on usage, it needed to be clarified with the exact number of SMS messages per day or the process of how their system would determine spam messages. Hence, further tests would be required to determine the threshold for the Globe Cellular Network.

1.2 Test Case for Globe Unli-Text Promo

This test determines the actual threshold of Globe Cellular Network in sending SMS messages. In the following test, four different test cases were done. Each test would test the cellular network to determine how many similar messages would be sent consecutively and the total amount of messages sent on that day. It is important to note that the promo used for this test is Unli-Text P20 of the Globe Promos, which provides unlimited text to all networks, 100 MB of mobile data, and unlimited calls to all networks for one day.

Max Number of Messages Sent	10	50	100	Unlimited
Number of Messages Sent	10	50	100	325
Misc. Messages	14			

Total Messages	499

Table 3.8: ***Test Case for Globe Unli-Text***

Table 3.8 shows the test result done with the Globe Cellular Network Sim. Based on the result, the Globe could consecutively send consecutive messages by 10, 50, and 100. In the last test case done, which would run an infinite loop of sending messages, Globe could only send 365 messages. Including those miscellaneous messages sent before the test, there were a total of 499 messages sent on that day before the Globe Sim was restricted from sending any more messages. It is important to note that even though the SIM was restricted to sending messages, it could still make calls and use mobile data. Hence, this test did not wholly deactivate the SIM but only restricted its ability to send messages, which implies that the threshold of Globe SIMs for sending SMS messages is 499 messages per day.

2.0 Smart

Smart Communications Inc., commonly known as Smart, is one of the subsidiary digital services of PLDT Inc, a telecommunications company based in the Philippines. Together with Globe Inc., both Smart and Globe are the two major cellular network companies that comprise more than 80% of the cellular network subscribers in the Philippines. Like Globe, Smart also offers its subscribers commercial wireless services through its 2G, 3G, 3.5G HSPA+, 4G LTE, and LTE-A networks, with 5G. Hence, for this study, Smart is one of the cellular network providers.

2.1 Fair Use Policy

Similar to Globe Fair Use Policy, Smart Fair Use Policy for Unlimited Promos also restrict subscribers with some of its services, such as the number of messages sent daily. It is stated in their policy that only a maximum of 1,200 texts can be sent and a maximum of 180 minutes per call transaction per day. Unlike the vague Globe Policy, Smart Policy clearly stated their threshold for sending messages per day. However, it did not state what messages were considered spam messages. To determine the authenticity of this policy, tests were conducted to validate and determine the threshold for the Smart Cellular Network.

2.2 Actual Test Case for Smart Unli-Text Promo

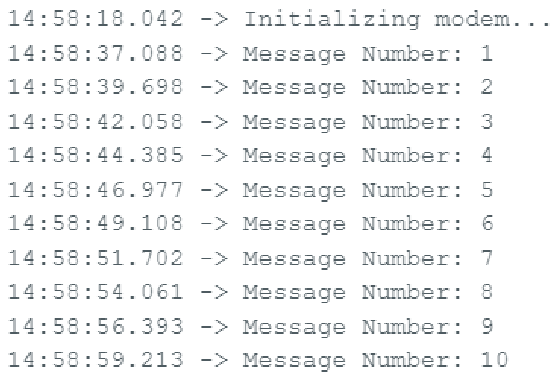
Similar to the test done with Globe Cellular Network, this test is done to determine the actual threshold of Smart Cellular Network in sending SMS messages. In the following test, four different test cases were done. Each test case would test the cellular network, how many similar messages would be sent consecutively, and the total amount of messages sent on that day. It is important to note that the promo used for this test is Unli-Text P30 of the Smart Promos, which provides only unlimited text to all networks for one day.

Max Number of Messages Sent	10	50	100	Unlimited
Number of Messages Sent	10	50	100	3,000+
Misc. Messages	0			
Total Messages	3,160+			

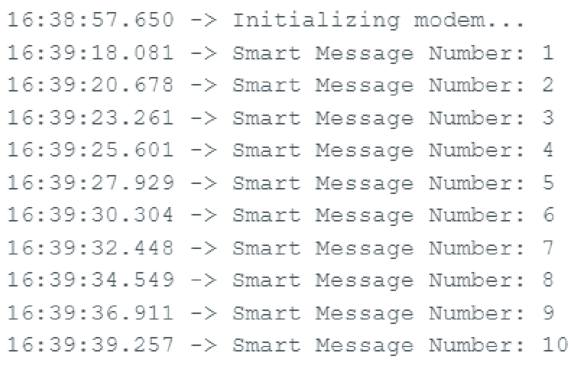
Table 3.9: Actual Test Case for Smart Unli-Text

Table 3.9 shows the result of the Smart Cellular Network Sim test. Based on the result, the Smart SIM could consecutively send consecutive messages by 10, 50, and 100. Surprisingly, the Smart SIM could send 3,000+ messages consecutively, six times greater than the threshold for Globe. Including all test cases, the Smart SIM could send 3,160+ messages in one day. This is considerably more than the stated maximum threshold of 1,200 texts per day in their Fair Use Policy. It is also important to note that the test was stopped at the mark of 3000 since it would be unnecessary to continue further.

Hardware Message Sending Time Interval



```
14:58:18.042 -> Initializing modem...
14:58:37.088 -> Message Number: 1
14:58:39.698 -> Message Number: 2
14:58:42.058 -> Message Number: 3
14:58:44.385 -> Message Number: 4
14:58:46.977 -> Message Number: 5
14:58:49.108 -> Message Number: 6
14:58:51.702 -> Message Number: 7
14:58:54.061 -> Message Number: 8
14:58:56.393 -> Message Number: 9
14:58:59.213 -> Message Number: 10
```



```
16:38:57.650 -> Initializing modem...
16:39:18.081 -> Smart Message Number: 1
16:39:20.678 -> Smart Message Number: 2
16:39:23.261 -> Smart Message Number: 3
16:39:25.601 -> Smart Message Number: 4
16:39:27.929 -> Smart Message Number: 5
16:39:30.304 -> Smart Message Number: 6
16:39:32.448 -> Smart Message Number: 7
16:39:34.549 -> Smart Message Number: 8
16:39:36.911 -> Smart Message Number: 9
16:39:39.257 -> Smart Message Number: 10
```

Figure 3.30.1 and Figure 3.30.2: Hardware Message Sending Timestamp for Globe and Smart

The above figures show the timestamp of the device sending consecutive messages in each cellular network. The result shows an average time of 2-3 seconds between each message sent by the hardware to the cellular tower, regardless of what cellular network the hardware uses. Assuming that this is the average time and a good signal reception is present, a single hardware may be able to send a maximum of 20-30 messages per minute.

Actual Message Sending Test Simulation 1

Number of Devices	Cellular Network	No. of Messages Sent	No. of Messages Received	Time Elapsed
1 Device	Smart	100	100	4 mins 32 sec (272 seconds)
2 Devices	Smart	100	100	2 mins 42 sec (162 seconds)
3 Devices	Globe	100	98(1)	1 min 35 sec (95 seconds)
4 Devices	Smart	100	99 (1)	1 min 4 seconds (64 seconds)

Figure 3.31: Test Simulation 1 (Indoors Near the Window)

The table above shows the result of a test conducted to determine the time elapsed and the number of messages received when more devices are added to the system. For this test, we had a fixed number of messages sent per number of devices integrated into the system, 100 messages. Results show that as you increase the number of devices integrated into the system, the time elapsed to receive the messages also reduces. However, one result shows an outlier to the whole. When integrating the third device with a Globe cellular network, the total messages received were not equal to those sent by the hardware.

Further investigations show that out of 100 messages sent, only 98 were received, making two messages lost. Later, one of the two missing messages was received after 5 hours. The same situation also happened when four devices were integrated into the system. The researchers believe that the problem may have occurred due to cellular network problems since it started to appear when the third device, with a Globe SIM card, was added to the system. The researchers also believe that the location contributes to the lost message, as Globe may not have the best signal reception in the area.

Actual Message Sending Test Simulation 2

Number of Devices	**Cellular Network**	**No. of Messages Sent**	**No. of Messages Received**	**Time Elapsed**
1 Device	Smart	100	100	5 mins 2 sec (302 seconds)
2 Devices	Smart	100	100	3 mins 35 sec (215 seconds)
3 Devices	Globe	100	88(4)	2 min 47sec (167 seconds)
4 Devices	Smart	100	89 (2)	1 min 33 seconds (93 seconds)

Figure 3.32: Test Simulation 2 (Indoors/No windows)

The table above shows you the result of test simulation 2, which was conducted in a location where signal reception is low. Test simulation 2 is a test conducted to reinforce test simulation 1 by changing the location of the hardware to a location where signal reception may not be good, i.e., indoors where it is not adjacent to an outer wall. The researchers believe that location may also be one of the factors. Hence, they conducted a simulation with unfavorable signal reception to determine its effect. The result shows an increase of 10% in time elapsed compared to the results taken from test simulation 1. This implies that the hardware had taken more time to send the message when placed in a location with low signal reception. Results also show that more messages were also lost when the sending process was conducted.

Further investigation shows that although Smart sim card devices were affected by the low signal reception, it was still able to send all messages despite a more extended time duration to send all messages. However, Globe sim card devices might be heavily affected by the low signal reception, justified by the result that 22 messages were lost out of 100, and 4 were only received after 3-5 hours. Hence, the researchers concluded that signal reception is a significant and vital aspect to consider when using the system.

Importing University Data

The following shows the testing done on the bulk imports of university data through a CSV file. The group, student, and student load entities have been identified as those that may cause performance issues, as these files usually contain many rows to be imported. Hence, these are also the entities used during testing.

Two types of tests were performed with varying amounts of data, one importing all of the group, student, and student load files in bulk, and another importing the three separately and according to the university's colleges. Three tests are performed for each category, recording the number of entries, the time the upload started, the time the upload ended, the elapsed time, and the average duration of the three tests.

The tests were performed on a desktop computer running the web application, MySQL workbench, and server. The computer runs on 16 gigabytes of memory with an i5 Generation 4 processor (4 cores).

Furthermore, the MaatWebsite/Excel package is utilized to support importing CSV files in Laravel and have them converted into database entries. Batch inserts and chunked reading from the package are also applied to limit the number of queries and minimize memory usage, as the uploaded files are relatively large. Both the batch and chunk sizes are set to 1000 rows, which means that for a given CSV file, a maximum of 1000 rows are processed in each run.

1.0 Importing Student Load in Bulk

ID	OfferID	Year	Term
4068	7047	2021	2nd Sem.

Table 3.10: Student Load CSV Format

The *Student Load* CSV file has 54,411 rows and contains the ID (student ID), the Offer ID of the subject the student is enrolled in, and the Year and Term.

Test Cases	Upload Start Time	Upload End Time	Time Elapsed
001	2023/05/04 1:58 PM	2023/05/04 2:03 PM	5.22mins
002	2023/05/04 2:10 PM	2023/05/04 2:15 PM	4.90mins
003	2023/05/04 2:20 PM	2023/05/04 2:25 PM	4.93mins
Average Upload Time: (5.22 + 4.90 + 4.93) / 3 = 5.01mins			

Table 3.11: Importing Student Load in Bulk Test

All 54,411 rows were inserted into the database for the three tests performed, with an average upload time of 5.01 minutes. The entries were added to the system with a mixture of the MaatWebsite/Excel package code and the Laravel Eloquent ORM filtering methods to query and sync the subject and student data to the *Group_Student* pivot table.

The initial code involved a for loop to iterate each row and query the corresponding OfferID in the database before attaching it with the Student ID to the pivot table. And although it worked in smaller CSV files, it was ultimately deemed unusable for larger files. We eventually received a Laravel-issued memory leak error due to the number of rows and queries performed.

2.0 Importing Students in Bulk

StudNumber	First Name	Last Name	Course	Level	Year	Term	Phone
4270	Bradley	Allison	BSIT	4	2021	2nd Sem.	

Table 3.12: Students CSV Format

The *Student* CSV file has 7,483 rows and contains student information such as their StudNumber, First and Last Name, Course, Level, Year and Term, and Phone Number.

Test Cases	Upload Start Time	Upload End Time	Time Elapsed
001	2023/05/04 2:32 PM	2023/05/04 2:33 PM	0.78mins
002	2023/05/04 2:37 PM	2023/05/04 2:37 PM	0.78mins
003	2023/05/04 2:42 PM	2023/05/04 2:43 PM	0.77mins
Average Upload Time: (0.78 + 0.78 + 0.77) / 3 = 0.78mins			

Table 3.13: Importing Students in Bulk Test

All 7,483 rows were inserted into the database for the three tests performed, with an average upload time of 0.78 minutes or 47 seconds.

3.0 Importing Subjects in Bulk

OfferID	Name	Desc.	Day	Time	Year	Term	Teacher
11007	Prac	Practicum	MTWThF	08:00 am to 05:00 pm	2021	2nd Sem.	21

Table 3.14: Subject CSV Format

The *Subject* CSV file has 1,939 rows and contains information regarding subject offerings such as the Offer ID, Name, Description, Day, Time, Year, Term, and the Teacher handling the subject.

Test Cases	Upload Start Time	Upload End Time	Time Elapsed
001	2023/05/04 3:00 PM	2023/05/04 3:01 PM	0.18mins
002	2023/05/04 3:04 PM	2023/05/04 3:05 PM	0.12mins
003	2023/05/04 4:18 PM	2023/05/04 4:18 PM	0.20mins
Average Upload Time: (0.18 + 0.12 + 0.20) / 3 = 0.17mins			

Table 3.15: Importing Students in Bulk Test

All 1,939 rows were inserted into the database for the three tests performed, with an average upload time of 0.17 minutes or 10 seconds.

4.0 Importing Student per College

For this particular test, the entries in the *Student* CSV file are grouped by the university's six colleges and into six separate CSV files. Starting with the School of Allied Medical Sciences (SAMS) with 385 rows, the School of Arts and Sciences (SAS) with 1,352 rows, the School of Business and Management (SBM) with 4,036 rows, the School of Computer Studies (SCS) with 459 rows, the School of Education (SED) with 348 rows, and the School of Engineering (SOE) with 903 rows, totaling up to the 7,483 rows of the CSV file. It also maintains the same format and columns as the original file.

Test Cases	**Upload Start Time**	**Upload End Time**	**Time Elapsed**
001	2023/05/04 4:46 PM	2023/05/04 4:47 PM	1.23mins
002	2023/05/04 5:07 PM	2023/05/04 5:09 PM	1.13mins
003	2023/05/04 5:20 PM	2023/05/04 5:21 PM	1.17mins
Average Upload Time: (1.23 + 1.13 + 1.17) / 3 = 1.17mins			

Table 3.16: Importing Student per College

Each CSV file is uploaded in alphabetical order. Starting with SAMS (385), SAS (1,352), SBM (4,036), SCS (459), SED (348), and SOE (903). And for the three tests performed, the six CSV files were completed with an average upload time of 1.17 minutes or 1 minute and 10 seconds. This already includes the wait time and the processing of the next file to be uploaded.

5.0 Importing Student Load per College

Similar to the previous test, the entries in the *Student Load* CSV file are divided into six separate files representing the university's different colleges. SAMS has 2,386 rows, SAS has 8,605 rows, SBM has 27,976 rows, SCS has 4,391 rows, SED has 3,293 rows, and SOE has 7,760 rows, totaling up to 54,111 rows of the Student Load file. It also maintains the same format and columns as the original file.

Test Cases	**Upload Start Time**	**Upload End Time**	**Time Elapsed**
001	2023/05/04 5:22 PM	2023/05/04 5:27 PM	4.97mins
002	2023/05/04 5:32 PM	2023/05/04 5:36 PM	4.03mins
003	2023/05/04 5:41 PM	2023/05/04 5:45 PM	3.77mins
Average Upload Time: (4.97 + 4.03 + 3.77) / 3 = 4.25mins			

Table 3.17: Importing Student Load per College

Each CSV file is uploaded in alphabetical order. Starting with SAMS (2,386), SAS (8,605), SBM (27,976), SCS (4,391), SED (3,293), and SOE (7,760). And for the three tests performed, the six CSV files were completed with an average upload time of 4.25 minutes or 4 minutes and 25 seconds. This already includes the wait time and the processing of the next file to be uploaded.

Test Summary

Various tests were conducted to determine the performance tests of the system. The tests were conducted to evaluate the system's Data Management, Cellular Network utilized, and Sending Message process. Results show various factors the researchers identified as a point of consideration when using the system.

In the test made for Cellular Networks, the researchers found the threshold for two cellular networks, Globe and Smart. In the results for Globe SIM, the researchers found that the Globe Cellular Network only allows its Unli-Text Promo subscribers 499 messages per day, which can be consecutively by 10, 50, and 100 messages. As for the other cellular network, the researchers discovered that the Smart Cellular Network allows its Unli-Text Promo subscribers 3,160+ messages per day. This is twice the amount of maximum text per day that they stipulated in their Fair Use Policy. With four devices running on the Smart SIM, the system may be able to send nearly 10,000 messages per day, far greater than using Globe Sims. Hence, it is more efficient to utilize Smart Sims over Globe Sims.

In the following test simulation conducted, the researchers were able to identify that the more devices added to the system, the lesser time is required to successfully send all messages sent by the system. However, the researchers also discovered an important factor to consider when utilizing the system. Test simulation 1 shows that the signal reception of the hardware's location may be the reason why some messages were lost during the sending process. This is further proved by the results of Test Simulation 2, where the location was changed to a low-signal reception area. Hence, the researchers concluded that location is a major factor to be considered when utilizing the system. The researchers also noted that this might be mitigated by using a more powerful antenna for the hardware instead of the generic antenna used by the tested hardware.

For bulk imports, some slight differences in upload time can be noticed when the CSV files are uploaded per college. As a reference, bulk imports with the Student Load file had an average upload time of 5.01 minutes, while importing per college showed an average upload time of 4.25 minutes, a 0.76 minute or 46-second difference between the two processes. Conversely, bulk imports with the Student file had an average upload time of 0.78 minutes, while importing per college had a slower average upload time of 1.17 minutes, a 0.39 minute or 23-second difference. The MaatWebsite/Excel batch inserts and chunked reading features also helped immensely by mitigating the memory leak and query issues posed by the CSV file sizes.

Whether or not the CSV files will be grouped during the upload will be left to the admin's discretion. However, to achieve the most optimal performance, the researchers recommend limiting the number of rows of the CSV file to only below the 10,000 mark. As larger files ultimately take longer to upload.

CHAPTER IV

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

Summary of Findings

The researchers have implemented tests on the system's working parts, namely, Web Application, Mobile Application, and Hardware.

In the Web Application part, all modules have been tested and checked to ensure working performance in production. Registering, Logging in, Imports, Group Management, Message Sending, Routes, and UI/UX have been thoroughly simulated and used to confirm working order. The APIs have also been tested using a mix of Postman and actual usage in the Mobile Application.

All modules have been tested and checked in the Mobile Application part to ensure working performance. Logging in, Routes, UI/UX, Group Management, Message Sending, and History Logs have been thoroughly simulated and used to confirm working order. Network connectivity considerations have also been tested to ensure the software does not crash nor behave inconsistently when disconnected and later connected to the network.

On the Hardware side, the findings extracted from the SMS limit for each cellular network were acted upon. Testing on both SIM packs has revealed that the

SMART SIM pack is more reliable in mass messaging and was therefore used in the hardware. The Round-Robin algorithm was also thoroughly tested in simulations, as can be seen in Figure 3.33 and Figure 3.34.

During the debugging process, the researchers were able to find that there was a problem with the location they conducted the test simulation. The test simulation was conducted on the second floor of a building inside an indoor room far from the outer wall. The researchers concluded that the Globe SIM device could not send the message fully, and somehow the message was lost between the device and the cellular tower. In the second test simulation, the researchers changed the location to a room with windows with high signal reception for the SIM cards. The results show that out of the six messages that were sent, all six were able to reach their expected recipients. It is also important to note that in both tests, the maximum number of characters allowed (around 230 characters) was used as the content of the message.

Conclusion

In this study, the researchers were able to create the AdelanteSMRS System with both a web application and a mobile application. They also tested the data transmission and processing between the web, mobile, and hardware. Two test simulations were done in the University of San Jose - Recoletos with the system. The results showed a successful SMS sending from the mobile and web application to the recipient's smartphone. In these test simulations, the researchers were able to find that there was a problem with signal reception with the hardware placed indoors. Nevertheless, when the signal reception was addressed, it could still send the message to its intended recipients.

Hence, the study has fulfilled its objective of developing a working mobile and web-based campus short message relay system using MQTT Protocol with the integrated SIM800L and ESP32 devices. By offering a wide area of availability, a sizable number of reachable recipients, high open rates that recipients can view in seconds, a straightforward communication method for most people, and a complementary method to other communication channels, the AdelanteSMRS can improve the institution's efficient information dissemination. With the recent tests done, the researchers believe that this system would apply not only to educational institutions but to any institution or organization with a demand for effective information dissemination. Although less convenient and advanced than existing applications and systems that utilize the internet, the AdelanteSMRS can provide a simple, quick, wide-range method for information dissemination as an alternative for these.

Recommendations

Many recommendations can be made with the study and the current AdelanteSMRS. For the study, no survey or test was done on users outside the research group. These surveys or inputs from test users would give more diverse and substantial suggestions that could improve the overall design and processes of the AdelanteSMRS. Hence, the study recommends that further test methodology be implemented to get more diverse and substantial outputs from actual system users.

The researchers also believe that new frameworks and packages could be used to improve the versatility and usage of the AdelanteSMRS. For example, the current Laravel Framework is not convenient for asynchronous processes. Hence, the researchers worked around this by utilizing Laravel Queues as a means for background processing of creating and sending SMS messages to the hardware.

A scheduled-send system would be a great addition to this project as it would allow users to send messages whenever they want. The application's load-balancing system could be improved through data analytics. A round-robin algorithm is a simple yet effective load-balancing algorithm, but through data processing of the application run time, there could be a more suitable one.

Improvements in the system should also be considered for production and deployment. During the course of development, the web APIs were locally hosted for testing purposes. The APIs published in the web application would be better suited to be placed in a dedicated server in production as this could lead to an efficient and robust system. Additional SIM800L and ESP32 devices should also be provided as the workload for an entire department using the system 24/7 might be too much for only four instruments. This would especially be true if moved to the university scale.

Lastly, the researchers recommend finding better modules for the system's hardware. The two test simulations proved a problem with signal reception within indoor rooms. Hence, a better antenna module is recommended for the SIM800L component of the system, as it only uses a generic flat-type antenna, which is not a suitable module for indoor reception.

REFERENCES

Caluza, L. J. B. (2016). Effects of Global Warming of ICT Products in the Philippines. *Journal of Communication and Computer*, *13*, 181-184. https://www.davidpublisher.com/Public/uploads/Contribute/57a1b57f8d932.pdf

Cope, B., & Kalantzis, M. (2016, June). Big Data Comes to School: Implications for Learning, Assessment, and Research. *AERA Open*, *2*(2), 1-19. 10.1177/2332858416641907

Hayati, A., Alireza, J., & Amir, M. (2013). Using Short Message Service (SMS) to teach English idioms to EFL students. *British journal of educational technology*, *44.1*(2013), 66-81.

Huang, J. (2021). Information Dissemination Control Algorithm of Ecological Changes in the New Media Communication Environment. *Mobile Information Systems*, *2021*(6274856), 1-10. https://doi.org/10.1155/2021/6274856

Joseph, E. C. (2019, December). Development of an IoT-based Students' Attendance Monitoring System. *International Journal of Engineering Research & Technology*, *8*(12), 653-658. https://www.ijert.org/research/development-of-an-iot-based-students-attendance-monitoring-system-IJERTV8IS120312.pdf

Kadirire, J. (2005). The short message service (SMS) for schools/conferences. *Recent Research Developments in Learning Technologies*, (2005), 997-981.

Kedare, T., Swami, V., Deshmukh, A. H., Dumbre, V., & Shaikh, A. (2021, January). ATTENDANCE MONITORING SYSTEM USING GSM. *International Research Journal of Engineering and Technology*, *8*(1), 1840-1847. https://www.irjet.net/archives/V8/i1/IRJET-V8I1306.pdf

Lumauag, R. G. (2016, November). SENT SMS : School Event Notification Through SMS. *Asia Pacific Journal of Multidisciplinary Research*, *4*(4), 61-68.

Mojares, P. V., Litan, G. A. T., & Mojares, J. G. (2013, Summer). INOTIFIED: AN SMS AND RFID-BASED NOTIFICATION SYSTEM OF LIPA CITY COLLEGES, LIPA CITY, BATANGAS, PHILIPPINES. *Journal of Applied Global Research*, *6*(18), 36-47. https://studylib.net/doc/8791495/inotified--an-sms-and-rfid-based-notification-system-of-l

Olaleye, O., Olaniyan, A., Eboda, O., & Awolere, A. (2013). SMS-Based Event Notification System. *Journal of Information Engineering and Applications*, *3*(10). https://d1wqtxts1xzle7.cloudfront.net/73814337/7637-10053-1-PB-libre.pdf?1635529199=&response-content-disposition=inline%3B+filename%3DSMS_Based_Event_Notification_System.pdf&Expires=1681328145&Signature=EYeq3ZbckuwEuRFsyfQjBZAaub2F8OTwnt9K0rebA7I-1CqT9C1S

Republic Act No. 10844. (2016, May 23). Official Gazette. Retrieved April 12, 2023, from https://www.officialgazette.gov.ph/2016/05/23/republic-act-no-10844/

Roberts, T., & Hernandez, K. (2019, February 8). Digital Access is not Binary: The 5'A's of Technology Access in the Philippines. *Wiley*. https://doi.org/10.1002/isd2.12084

Sadiku, P. O., Ogundokun, R. O., Ogundokun, O. E., & Adebayo, A. A. (2021, February). Interactive website on information dissemination. *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, *19*(1), 115-123. 10.12928/TELKOMNIKA.v19i1.15048

Satria, D., Zulfan, Munawir, & Mulyati, D. (2018, October). FINAL PROJECT CONSULTATION INFORMATION SYSTEM INTEGRATED NOTIFICATION SYSTEM BASED ON SMS GATEWAY. *Cyberspace: Jurnal Pendidikan Teknologi Informasi*, *2*(2), 135-140. https://jurnal.ar-raniry.ac.id/index.php/cyberspace/article/view/4002

Xu, L. D., He, W., & Li, S. (2014, January 16). IEEE Transactions on Industrial Informatics. *Internet of Things in Industries: A Survey*, *10*(4), 2233 - 2243. 10.1109/TII.2014.2300753

CURRICULUM VITAE