Operation RM
Northern Arizona University
Flagstaff, AZ

William Rogers (Team Lead): wpr29@nau.edu
Isaac Faulkner: lwf2@nau.edu
Andrew Milizia: am5275@nau.edu
Nick Henderson: nsh67@nau.edu

Software Design Document

February 16th, 2024

Operation RM

Client:

General Dynamics Mission Systems

Mentor:

Tayyaba Shaheen

Team Members:

William Rogers

Isaac Faulkner

Andrew Milizia

Nicholas Henderson

Version:

1.2

Overview:

This document will provide a structured plan concerning the design and implementation of the software for this project. The development team will utilize this document in hopes to mitigate vulnerabilities within the software design and ease implementation efforts.

Table of Contents

1 Introduction 3
2 Implementation Overview 4
3 Architectural Overview 5
4 Module and Interface Descriptions 7
    4.1 Frontend Mobile Application 7
    4.2 Backend Java Application 10
    4.3 Backend C Controller Module 11
5 Implementation Plan 13
    5.1 Project Initiation 14
    5.2 Design and Development 14
    5.3 Feature Implementation 14
    5.4 Testing and Quality Assurance 14
    5.5 Final Adjustments 15
    5.6 Final Product 15
6 Conclusion 15

1 Introduction

In today's world, the need for secure, efficient, and tactical communication in the defense, public safety and intelligence communities can not be understated. General Dynamics Mission Systems has tasked Operation RM with developing a solution to solve the problem of inefficient communication with a radio modem on a mobile device.

General Dynamics Mission Systems (GDMS) is a defense contractor that specializes in the development of mission-critical products and systems. GDMS has developed many widely-used technologies and programs such as Rescue 21, which is a system that allows the Coast Guard to locate people and vessels in distress. Probably one of the most commonly known programs at GDMS is developing the next generation of GPS, or the global positioning system. Overall GDMS provides many critical products and systems to the defense, public safety, and intelligence communities.

GDMS has tasked Operation RM with developing an Android application that can provide two-way communication with a radio modem, in addition to being able to select a specific connection to use for transmitting and receiving. Developing a mobile application that is able to communicate with a radio modem efficiently is crucial for General Dynamics Mission Systems. Currently, communication with the radio modem is done through a web interface that is inefficient to access and easily add file attachments. With an Android app, the clients will be able to communicate in a more efficient and tactical manner. In addition, the advantages include not only the ability to send and receive files efficiently, but also the ability to easily send images in the field using the integrated Android camera application. GDMS has provided a radio simulator to simulate the behavior of the real software-defined radio, as well as a remote control interface (RCI) API that will be used to communicate with the radio simulator.

To meet the expectations of our client, it is imperative that certain requirements are met. These requirements provide Operation RM with a guide to what is needed for the project's solution.

- The application will utilize Java and C to enable communication between the application and RCI

- With the use of this application, emails will be able to sent and received through specified connections
- The application will be compatible with Android version 12,13, and 14
- Screen compatibility will be supported through the Android application
- Color blindness will be considered in the development of the UI
- For any loading process with a time over 3 seconds will display a progress bar

To implement this application, the software architecture was developed to ensure the software meets the requirements that GDMS has set forth. First, an overview of the systems architecture was created to begin to understand the connections and various components of the design. For each module or component of the architecture, a corresponding class diagram was developed to better understand the relationships between classes, as well as between modules.

2 Implementation Overview

This project primarily consists of creating a mobile Android application that will connect with the General Dynamics radio modem. This application will help make communicating with their modem easier on a mobile device, as the currently implemented interface is limited to a web application. The primary objective of the project is to get an email-like system on the mobile application that can send and receive files to and from the radio modem. One of the driving factors behind GDMS' desire for an Android application over the current web application is to have it integrate with other applications on the phone such as the camera. There will be many parts of the solution that should be considered when connecting the mobile application with the radio modem.

The first part of the solution is to create a mobile application that will look similar to the current web application in order to make it easier to train people who have already interacted with the web application. This Android application will serve as the primary user interface for communicating with the radio modem. The application will be similar to an email system where the user can compose, send and view emails sent and received to/from the radio modem. The user will be able to see the inbox and outbox of the radio modem as well as select a certain connection to use via presets. On the home screen there will be five different tabs to choose from: compose, inbox, outbox, sent and transfer history. There will also be settings, system health, and about tabs that will provide information and settings for the radio modem. In addition, there will be a status bar at the top of the application showing the current status of the connected radio modem. This status bar will show if the modem is currently connected and if it is actively receiving or sending. The Android application will be made on Android Studio using Java. Java is one of Android Studio's most compatible languages and can easily communicate with the given radio simulator from General Dynamics.

The second part of the solution is facilitating the communication between the Android application and the radio modem. To do this, the Android application will make use of the General Dynamics RCI API. This provides an interface with the radio modem and radio simulator that allows packets to be sent back and forth through a WiFi connection. One challenge during implementation is that the RCI API is written in C while the Android application is in Java. To facilitate this transition, the Android application will use the Java Native Interface (JNI). The backend Java will facilitate the conversion of the Java methods to the native C methods. In addition, the queue system will be implemented in the backend Java. To transition between Java and the RCI API, a controller written in C will be used. The controller will communicate with the radio modem through the RCI API to perform operations such as sending and receiving files.

3 Architectural Overview

This project has four main parts to it as seen in Figure 1. These parts are the front end mobile application, the backend Java layer, the backend C controller, and the intermediate layer between the backend Java and backend C. These are important parts of the architecture to consider in the project. It is crucial that the interaction between these components, as well as their individual structure are analyzed.

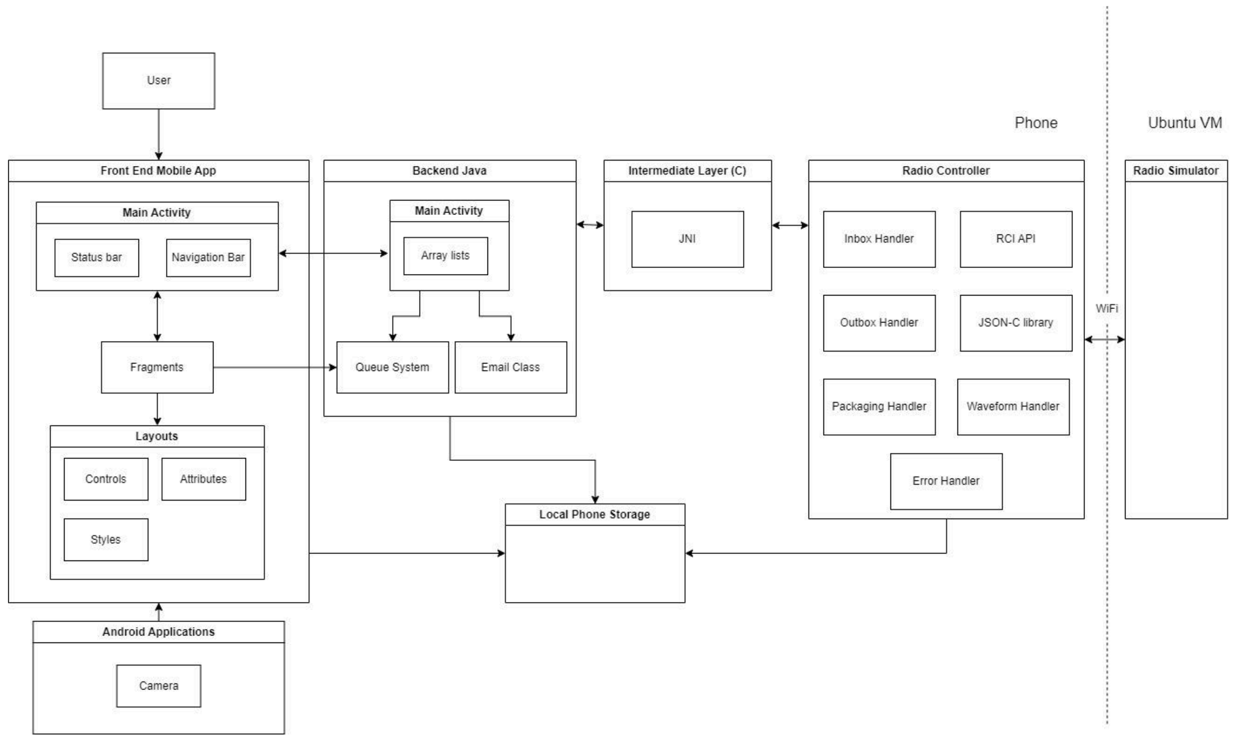

Figure 1 Systems Diagram

The front end mobile application is the primary user interface and executable of the project. It will control the mobile application and allow the user to access everything they need to communicate with the radio modem. Inside the mobile application most of everything will be controlled by fragments. These fragments will be used to facilitate the various functions of the radio simulator though a GUI. The main activity is the primary controller of the application that will be used the most. As part of the main activity, the status bar is used to check the status of the connected radio modem and see if it is connected, and whether it is actively sending or receiving files. In addition, the navigation bar will be used to navigate between the various tabs. In the layouts, there are controls, attributes and styles. The layout component consists of what the application looks like graphically. The front end mobile application serves as the primary facilitator for not only user interface, but communication with the radio modem entirely.

The backend Java will be used to store emails in a queue system, as well as interact with the backend C controller layer. This layer will facilitate the core functionality and logic of the application. This layer is composed of 3 parts: the main activity, queue system, and email class. The main activity is where most of the information and data is stored and used. The queue system will be used to store emails outbound for the radio modem. The queue system will follow First In First Out (FIFO) priority and will allow that queued email to be sent out in that order. Lastly, the email class will serve as an object to store the various components of each email. This class will store all necessary information for the emails to be displayed and sent properly. This layer will send and receive data from the back end C controller layer through the intermediate layer.

The intermediate layer is used between the back end Java and the back end C controller. This layer will connect any of the data that we gather from the radio modem on the C side and transfer them over to the Java side, and vice versa. In order to do this the Java Native Interface (JNI) will be used which is a Java to C interpreter. JNI will allow the mobile application to remain in Java and the RCI API to remain in C.

Lastly there is the backend C controller layer. In this layer, the application will connect to and communicate with the radio modem. This is where the RCI API will be implemented into the application. The RCI API will then be used to communicate with the radio modem and send and receive data. There are also handlers for the different application parts such as the inbox, outbox, and connection presets. These are used to handle the respective parts of the application. Any error that comes from the radio modem in connecting, sending, or receiving will be dealt with in this area and forwarded to the appropriate Java handler.

4 Module and Interface Descriptions

4.1 Frontend Mobile Application

The frontend mobile application module serves as the primary point of communication between the user and the radio modem. All user interface descriptions and code are included in this module. This module connects with the backend Java module in order to execute tasks requested by the user and to receive messages from the radio modem. Figure 2 shows the overall architecture of the frontend mobile application module.

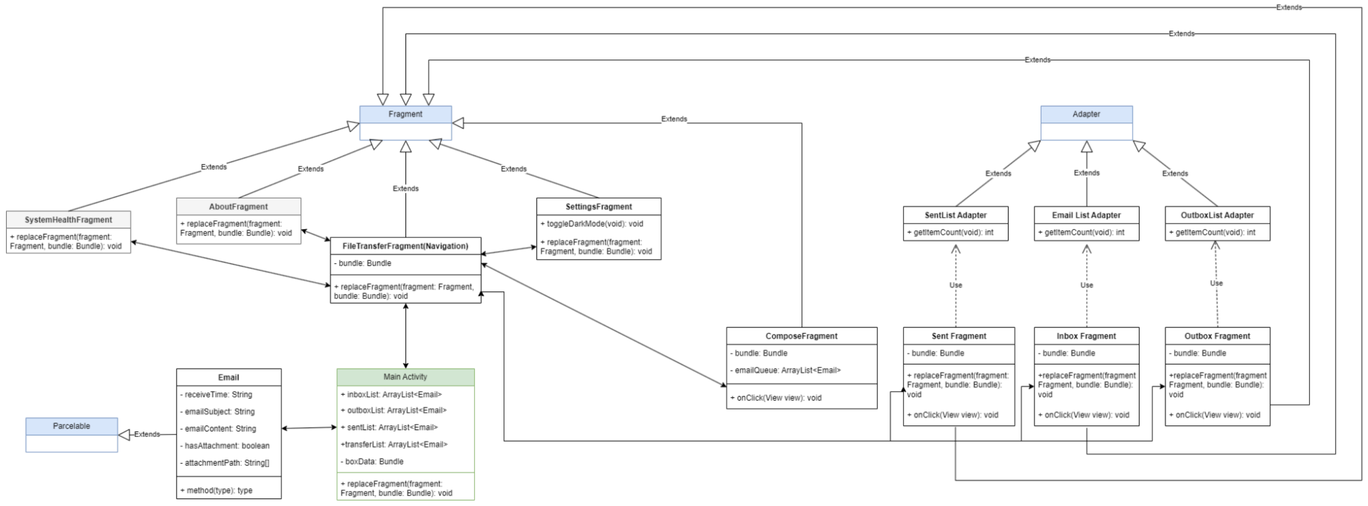
Fragment
Extends
SystemHealthFragment
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
AboutFragment
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
FileTransferFragment(Navigation)
- bundle: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
SettingsFragment
+ toggleDarkMode(void): void
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
Email
- receiveTime: String
- emailSubject: String
- emailContent: String
- hasAttachment: boolean
- attachmentPath: String[]
+ method(type): type
Parcelable
Main Activity
+ inboxList: ArrayList<Email>
+ outboxList: ArrayList<Email>
+ sentList: ArrayList<Email>
+transferList: ArrayList<Email>
- boxData: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
Adapter
SentList Adapter
+ getItemCount(void): int
Email List Adapter
+ getItemCount(void): int
OutboxList Adapter
+ getItemCount(void): int
Use
ComposeFragment
- bundle: Bundle
- emailQueue: ArrayList<Email>
+ onClick(View view): void
Sent Fragment
- bundle: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
+ onClick(View view): void
Inbox Fragment
- bundle: Bundle
+replaceFragment(fragment Fragment, bundle: Bundle): void
+ onClick(View view): void
Outbox Fragment
- bundle: Bundle
+replaceFragment(fragment Fragment, bundle: Bundle): void
+ onClick(View view): void

Figure 2 Frontend Class Diagram

There are two primary components within this module, the main activity and general fragments component, and the adapter component.

First, the main activity and general fragments component consists of the Main Activity class which handles the main front-end code. In addition each fragment of the Android application will be linked and used by this class and act as screens that will appear on top of the main activity. This is done to keep the program's email data contained in a centralized area where the fragments can access it by reference. Figure 3 displays the structure of this component.

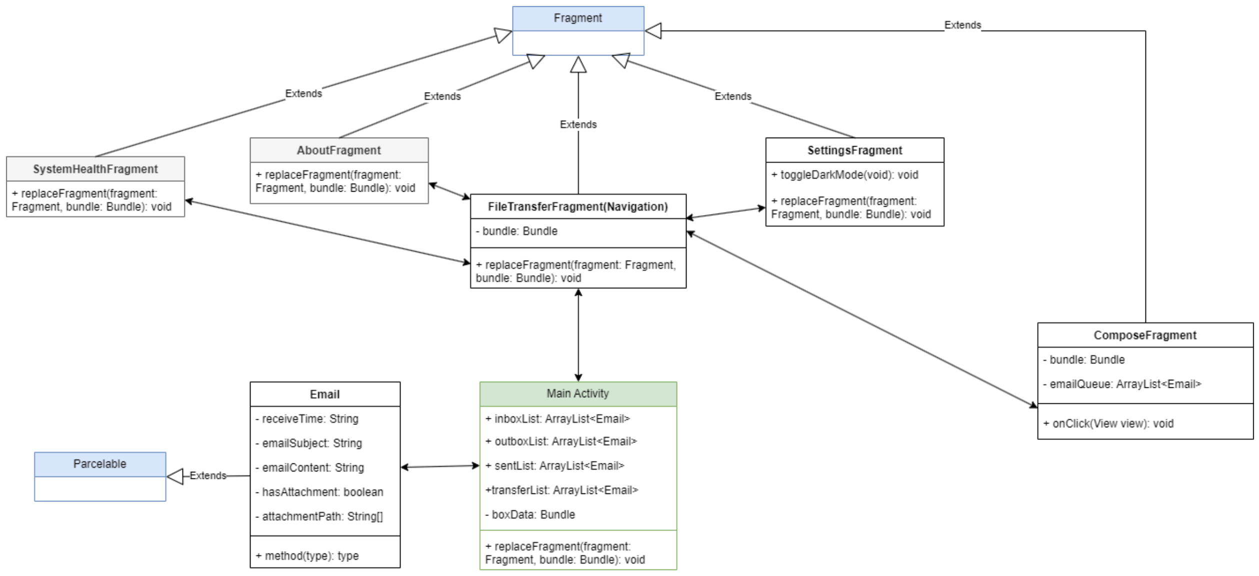
Fragment
Extends
SystemHealthFragment
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
AboutFragment
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
FileTransferFragment(Navigation)
- bundle: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
SettingsFragment
+ toggleDarkMode(void): void
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
ComposeFragment
- bundle: Bundle
- emailQueue: ArrayList<Email>
+ onClick(View view): void
Parcelable
Email
- receiveTime: String
- emailSubject: String
- emailContent: String
- hasAttachment: boolean
- attachmentPath: String[]
+ method(type): type
Main Activity
+ inboxList: ArrayList<Email>
+ outboxList: ArrayList<Email>
+ sentList: ArrayList<Email>
+transferList: ArrayList<Email>
- boxData: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void

Figure 3 Main Activity and General Fragments Class Diagram

The public interface of this component consists of the frontend, UI code that will serve as an interface for the radio modem on the mobile device. This component redirects the user input into the backend Java module.

The main activity serves as the initial main class of the application. This will use the Email class to access email objects. In addition, it will use the file transfer fragment to navigate to the various file transfer and setting pages. The system health and about fragments can be navigated to which are added as a possible future implementation as a stretch goal. The settings fragment will be used to toggle the dark mode. The main activity will serve as a bridge between the frontend and Java backend while also containing some of its own frontend functionality such as the navigation between the previously mentioned fragments. In addition, the compose, inbox, outbox, and sent fragments can be navigated to which is covered below.

The second components of the system are the fragments. These are the screens that will sit on top of the main activity and will be used to conduct the operations needed to use all the functionality regarding emails. These fragments are:

- Compose fragment: This is where the user can create and email including the subject, body, and attachments. The user can also specify if they want to automatically send the email or send it to the outbox.
- Outbox fragment: This is where the created emails will be stored that are awaiting transmission. Here the user can manage which emails will be sent out or remove an email from being sent. This fragment will also include emails that have been automatically sent and their current status of transmission.
- Inbox fragment: This is where the received emails will be accessed and opened. Managing emails in the inbox can also be conducted in order to remove emails that are no longer needed and to refresh the inbox similar to commonly used email systems.
- Sent fragment: This is where the emails that have been sent can be viewed and managed. The status of an email's transmission will be included to indicate if it was successfully transferred or if there was an error during the transmission process.

Further detail about the outbox, inbox, and sent fragments can be found below and in Figure 4 which covers how emails will be displayed on these screens through the use of adapters.

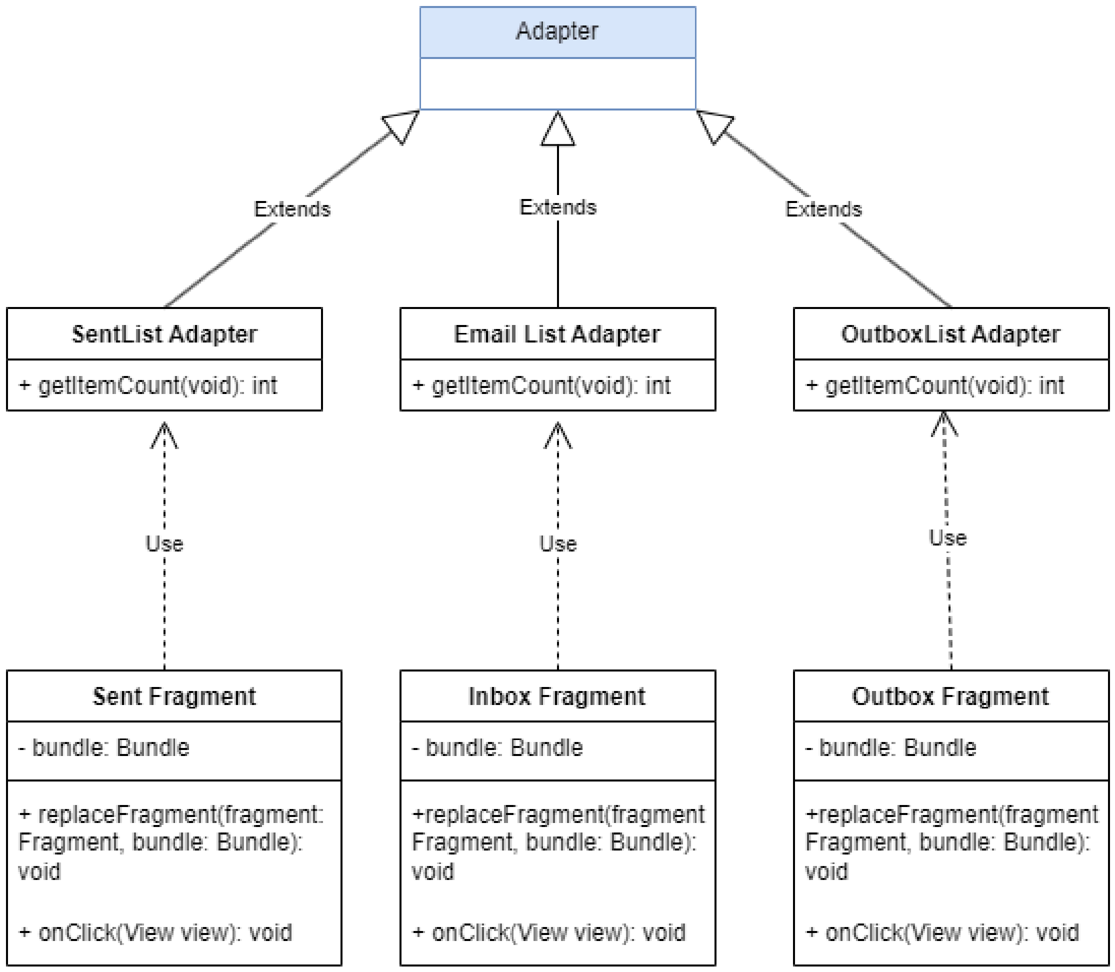
Adapter
Extends
Extends
Extends
SentList Adapter
+ getItemCount(void): int
Email List Adapter
+ getItemCount(void): int
OutboxList Adapter
+ getItemCount(void): int
Use
Use
Use
Sent Fragment
- bundle: Bundle
+ replaceFragment(fragment: Fragment, bundle: Bundle): void
+ onClick(View view): void
Inbox Fragment
- bundle: Bundle
+replaceFragment(fragment Fragment, bundle: Bundle): void
+ onClick(View view): void
Outbox Fragment
- bundle: Bundle
+replaceFragment(fragment Fragment, bundle: Bundle): void
+ onClick(View view): void

Figure 4 Adapter Class Diagram

The last components of the frontend are the adapters for the fragment classes which can be seen in Figure 4 above. The adapters have to implement Android's Adapter interface so it can effectively use the implemented adapters. These adapters are used to tell Android how to populate the list boxes on the fragments based on the data contained within the Email class. The list boxes are what will contain the individual email entries. To do this, adapter classes are needed for each fragment with a list box since each fragment displays different data. The adapters will use an Extensible Markup Language (XML) document to format each individual entry in terms of appearance.

Then, the adapter class will populate the entry's fields and allow Android to display it. All of the adapters will also include a check box which is used for email management. An example of this is deleting emails from the system, but other operations will take advantage of this as well. To do this, the adapter will keep track of the list of which checkboxes are checked and give it back to the fragment where the email can be deleted.

4.2 Backend Java Application

The backend Java application module serves as the primary logic layer for handling the various functionalities in the user interface. This module will utilize the JNI library to talk to the C code present in the backend C controller module. This module will be placed as an intermediate layer between the frontend mobile module and the backend C controller module. Figure 5 shows the structure of this module.

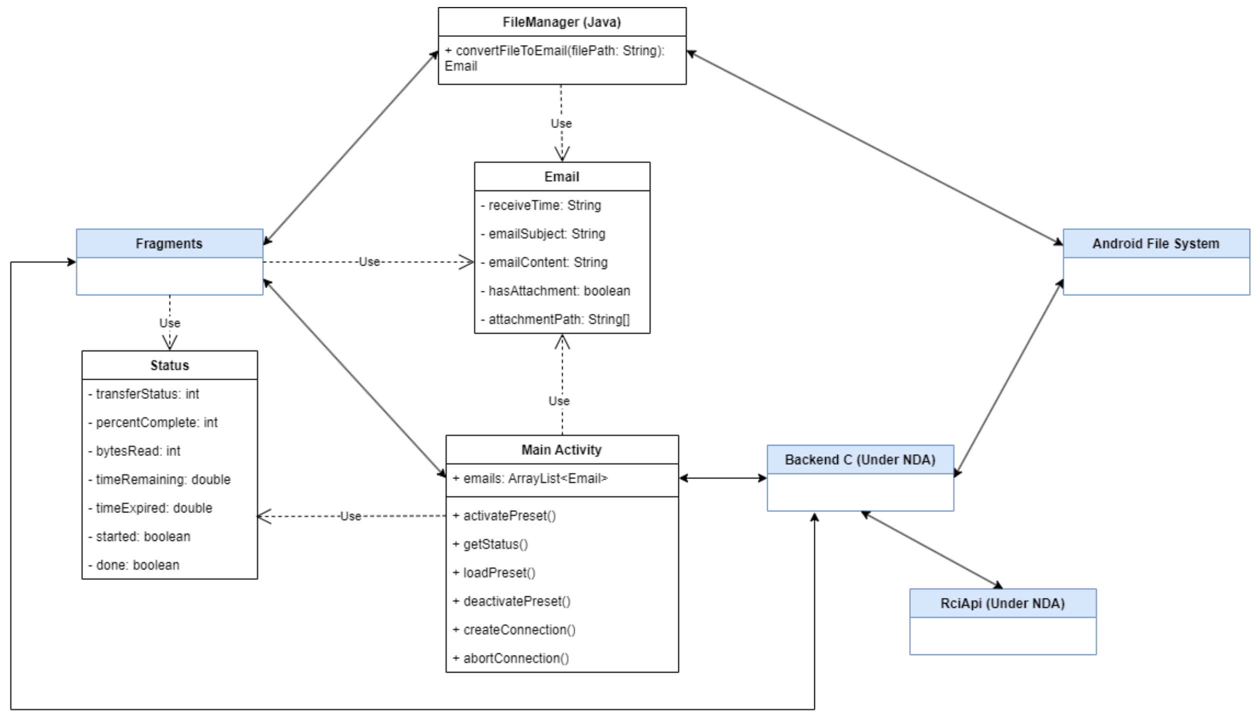

Figure 5 Backend Java Class Diagram

The primary public interface of this module is the various methods associated with calling the backend C controller module methods. These are native methods that use the JNI library to take place.

The main activity will use the Email class to create and access email objects. In addition the class will use Status objects that contain the status of the file transfer operation. The file manager class will be used to convert files to emails using the android file system.

4.3 Backend C Controller Module

The backend C controller module acts as the interface between the backend Java application and the radio simulator or radio modem. This module contains the custom handlers for the various functionality of the project. In addition, the given RCI API and requisite libraries will be included to interface with the radio.

Due to an active Non-Disclosure Agreement, Figure 6 has been simplified to a black-box diagram.

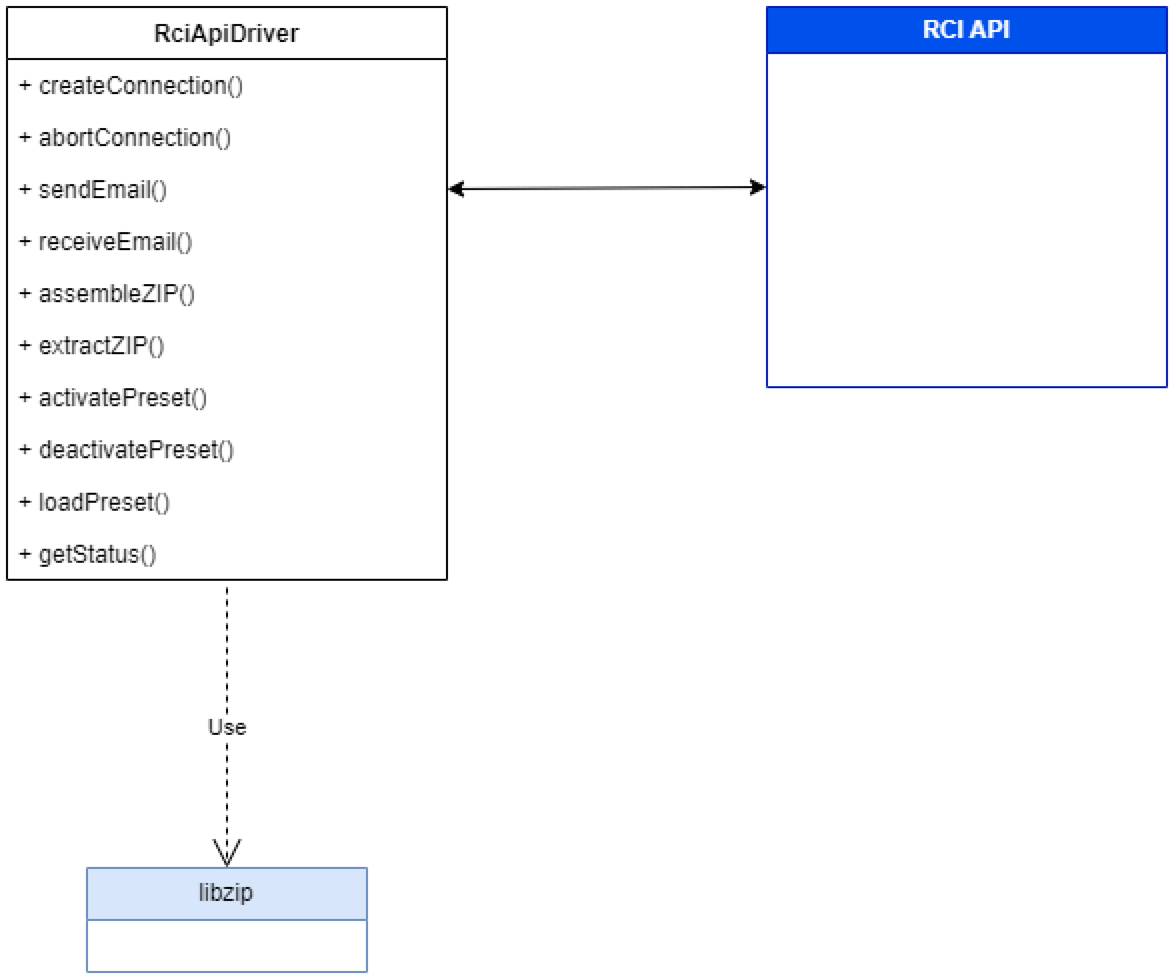

Figure 6 Backend C Controller Class Diagram

The primary public interface of this component is the methods that talk to the RCI API which then talks to the radio simulator or radio modem. This component will handle radio status and preset activation, which will be specified in the frontend layer.

The RCI API driver class will be used to call the various functions in the RCI API, and will be used to establish the socket connection with the radio modem. The createConnection function will establish the socket connection with the simulator. The abortConnection function will abort the socket connection with the simulator. The sendEmail function will send the email and attachments to the simulator. The receiveEmail function will receive emails and attachments. The assembleZIP function will zip all email with its attachments. The extractZIP function will unzip the email with its attachments. The activatePreset function will activate a desired preset on the simulator. The deactivatePreset function will deactivate the current preset on the simulator. The loadPreset function will load a desired preset on the simulator. The getStatus function will get the current status of the radio simulator, such as whether it is currently transferring files.

5 Implementation Plan

The implementation plan was created to ensure that the team stays on track due to the many tasks that are imperative to the success of this project. Implementation is split up into 6 different development phases. Each phase has a specific goal that the team plans to accomplish that will support the team's road to successful implementation. Be aware that this chart is flexible and changing based on possible successes and failures the team will encounter. Figure 7 displays the current Gantt chart for this plan.

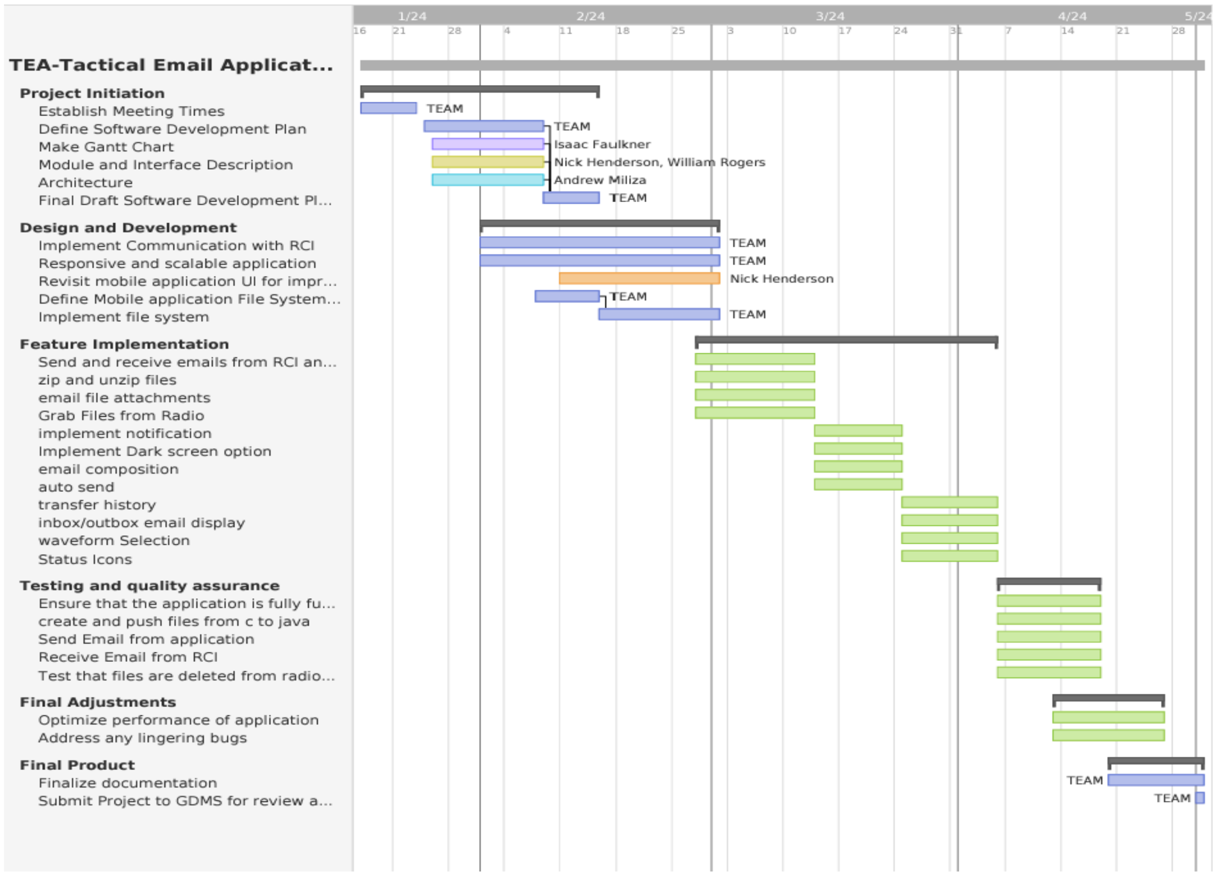

Figure 7 Gantt Chart

5.1 Project Initiation

In the implementation phase of project initiation the team will set up meeting times to ensure efficient communication throughout the entirety of the project. Furthermore, this software design document will be constructed and utilized to give the team a structure to follow and deadlines to meet. This document will be used as a map to ensure that the project is on task and to give each member of the team their portion to complete to meet the end goal.

5.2 Design and Development

The Design and Development phase has the goal of setting up the structure of the mobile application so that it may be efficiently utilized later in the project when features are added. The goal during this phase is to create a stable connection between the RCI to the mobile application and vice versa. Furthermore, the team will design and implement a file system so that later in the project when handling emails and attachments we have a foundation of how these items will be stored and accessed.

5.3 Feature Implementation

Next is the feature implementation phase. This phase has the largest amount of time sectioned out due to the amount of tasks that need to be accomplished. There are a total of 12 imperative features that need to be completed by the group. These tasks have not yet been assigned but will be as we discover the niches of the group and can assign tasks based on efficiency. Each group member will be responsible for implementing 3 features during this phase.

5.4 Testing and Quality Assurance

In the Testing and Quality Assurance phase the group will test the mobile application to ensure that the code the group has brought together in the feature implementation are working seamlessly. The group will also ensure that the application meets the standards of GDMS allowing time for the group to make any minor tweaks or additions to improve quality.

5.5 Final Adjustments

During the final adjustments implementation phase the team will take time to reassure that any bugs found during the testing stage have been resolved. As well as, take measures to optimize the performance of the mobile application. This may consist of multiple aspects that the team are uncertain of at this time in development.

5.6 Final Product

This last stage of development will consist of preparing the documentation for the email application. Allowing the team to ensure that the product can be replicated on by our clients. Lastly, the project will be submitted to General Dynamics for revision and approval.

6 Conclusion

The development of the mobile Android application for the client, General Dynamics Mission Systems, presents a convenient solution to enhance communication efficiency within GDMS respective fields. Through recognizing the limitations of the current web interface, this project aims to surpass those limitations by providing a user-friendly, intuitive Android application.

The significance of this project is found in its ability to enhance efficient and secure communication. Furthermore, the ability to replace the current web interface enhances the potential by providing a more convenient interface. With this solution the team hopes to enable users to send, receive messages and attachments efficiently.

The architectural overview displays a well structured system with four main components: the mobile application's front end, the backend Java side, the intermediate layer, and the backend C side. This modular design will allow the team to implement software that will support efficient communication and a seamless connection. The roles of these components are defined in the module and interface description. The mobile application's frontend will act as the primary point of interaction for the user, while the backend Java handles the logic and interfaces with the radio. The intermediate layer, utilizing Java Native Interface, will facilitate communication between the Java and C backends. The C backend controller utilizes the RCI API to interact with the radio modem.

The project will progress through various implementation phases, from initiation to final adjustments. These phases will ensure the team maintains a systematic approach to development of software and features. The Gantt chart illustrates the timeline for these phases.

The development of this document will give us a structured path to follow through many tasks as the team develops the Android application for GDMS that will address communication inefficiency. This will be conducted through careful planning, modular architecture, and a systematic implementation plan. Operation RM aims through this document to deliver a reliable, efficient, user-friendly solution that surpasses the expectations of General Dynamics Mission Systems. The team hopes that this solution will bring an efficient streamlined form of communication.