# Mobile Application for Alien RFID Reader (WKU Parking and Transportation Services)

Sprint 4
4-26-2024
Client: Kevin Werner


| **Name** | **Email Address** |
|---|---|
| Nicholas Johnson | nicholas.johnson769@topper.wku.edu |
| Cy Dixon | cy.dixon656@topper.wku.edu |
| Alex Godsey | alex.godsey750@topper.wku.edu |
| Michael Lynch | michael.lynch320@topper.wku.edu |


CS 496-001
Spring 2024
Project Technical Documentation

# Contents

# List of Figures

# 1 Introduction

## 1.1 Project Overview

This project is focused on creating a mobile application in tandem with RFID transmitters to streamline the parking and transportation department's car identification process. The Mobile Application for Alien RFID Reader project focused on enhancing parking management for WKU Parking and Transportation Services through RFID technology. The goal was to automate the verification of parking permissions within designated lots, aiming to streamline operations and improve the overall parking experience.

Central to the project is the implementation of an Alien F800 RFID reader with an Alien ALR-8698 RFID antenna, scanning RFID tags on parking permits. Complementing this hardware setup is a mobile application developed for Android devices, serving as the interface for parking management personnel. The app enables real-time scanning of parked vehicles' tags, checking if they are within their allocated parking area based on permissions stored in a database.

The system addressed several challenges currently faced in parking management, including time-consuming manual verification process and unauthorized parking. By automating these processes, the project aimed to streamline parking operations, speed up the process for checking if a car is parking in an illegal spot, and through this hopefully improve the overall parking experience for vehicle owners through.

The primary audience for this project includes parking management teams responsible for monitoring and enforcing parking regulations. The client themselves is the Senior Information Technology Consultant at the WKU Parking and Transportation Services, who has supplied us with a Alien ALR-8698 RFID Antenna for receiving the information from the tags, a Alien ALR F800 RFID Reader, and some sample RFID tags.

This RFID-based parking management system utilizes technology to offer a scalable, secure, and efficient solution for managing parking resources at WKU. The project followed a systematic approach, emphasizing clear documentation, client engagement, and the application of RFID technology to achieve its objectives. The development process is organized into distinct phases, aligning with a modified waterfall model to allow flexibility for adjustments and improvements as needed.

## 1.2 Project Scope

The scope of this project included several things, but the main development our team worked on consists of developing a mobile application to aid the WKU Parking and Transportation Services in parking pass scanning. This application was made by using Android Studio to develop an app that can be linked to an RFID reader to run functions to check to see if the parked car has proper permissions to be in the lot that it is currently located in. The specific functionality that was outlined that was deemed necessary by our client was the ability to track the location of the scanned vehicle based on its RFID tag, determine if the car is in the location that was just scanned, and gather the user ID from that tag.

The scope also consisted of utilizing the Alien F800 RFID scanner which was provided us by the WKU Parking and Transportation Services. This scanner and its associated technology allows us to scan parking passes of parked cars in WKU parking lots so the user ID associated with that parking pass can be obtained. We also created a mobile app that can be utilized alongside the Alien F800 RFID Reader to ensure cars are parked in correct locations based on the geolocation of the phone quickly after an RFID tag is read. So that the Alien F800 RFID Reader can read tags, we are also utilized an Alien ALR-8698 Antenna Unit to send and retrieve signals from the RFID tags in parked cars in WKU campus parking lots.

We also created documentation to show our overall progress and development to our client. This documentation also allowed our client and future teams to further develop, utilize, and/or troubleshoot our project. This documentation contains content revolving around the 4 main steps of the software development process, the Feasibility Study, Modeling and Design, Implementation, and Testing. In addition to documentation, we also created

presentations to present to our client about the progress we have made in each of the 4 main sprints throughout our project. These presentations closely followed the content of these technical and organizational documents.

Since the project consists of scanning RFID tags, we needed to obtain this hardware to allow us to develop our project and test it properly. Thankfully, WKU Parking and Transportation Services lent us a lot of equipment they had already obtained in the past to help us develop a product to assist them with parking-related enforcement inside of WKU parking lots.

Since the project revolves around the software development process, which is broken down into 4 main stages, our group had 4 main due dates linked with each stage in the software development process. These sprints are the Feasibility Study, Software Modeling and Design, Software Implementation, and Testing phases. During each phase of the software development process, we implemented several meetings a week that last a few hours each for every week we are working on the project. With these meetings and the work we are completed outside of meetings, we estimated that each group member will have worked between 10-20 hours every week on this project depending on the needs of that particular week. The below table shows the due dates for each of the 4 main software development stages.

| **Sprint #** | **Sprint # Due Date** |
|---|---|
| Sprint #1 | Due the week of January 28th - February 3rd |
| Sprint #2 | Due the week of February 25th - March 2nd |
| Sprint #3 | Due the week of March 24th - March 30th |
| Sprint #4 | Due the week of April 28th - May 4th |

Table 1: CS496 Senior Project Due Dates

## 1.3 Technical Requirements

### 1.3.1 Functional Requirements

| **Mandatory Functional Requirements** |
|---|
| 1. Alien F800 RFID Reader scans parking tags using the Alien ALR-8698 RFID Antenna. |
| 2. Phone is able to connect to Alien F800 RFID Reader through Network to allow communication across devices. |
| 3. Implementation of geolocation tracking for scanned tags in the mobile app. |
| 4. Phone can validate parking location accuracy post-tag scan. |
| 5. App is able to retrieve and display USER ID from scanned tags using the database. |
| 6. Data synchronization is ensured between the mobile app and RFID Reader. |
| **Extended Functional Requirements** |
| 1. The User should be able to navigate through the mobile application and move between different menus. |

The mandatory functional requirements listed are essential for the core functionality of the RFID-based parking management system. These requirements are further described below and are numbered in coordination with the table above.

**1. Alien F800 RFID Reader scans parking tags using the Alien ALR-8698 RFID Antenna.** The first functional requirement listed is probably the most important requirement of all of them. Ensuring that the Alien F800 RFID Reader and the Alien ALR-8698 RFID Antenna are properly set up and functioning properly is a must so that our team could gather correct and accurate readings of parked vehicles in WKU campus parking lots. If this step is not done properly, we could have potentially made mistakes inside our Android app development that could have harmed the final product. With proper testing in the provided gateway application and web application for the Alien devices, we avoided this error by changing settings, testing spare RFID tags, and by identifying the capabilities of the devices to ensure this step is not a future issue.

**2. Phone is able to connect to Alien F800 RFID Reader through Network to allow communication across devices.**

After correctly implementing the Alien hardware, we eventually needed to be able to link the Alien F800 RFID Reader to the mobile device that is utilizing the mobile application. This was essential to our project because we needed to obtain the information from the RFID tags, more specifically, the EPC of the RFID tag. With this EPC from the RFID tag, we then would be able to run data calls and determine if that user has the proper permissions with that parking pass to park in the parking lot that they are currently parked inside of. In the context of parking passes at WKU, we would have based these functions around the USER ID of the parking pass.

**3. Implementation of geolocation tracking for scanned tags in the mobile app.**
Another key functional requirement of the project was trying to obtain accurate geolocation. When implementing a parking pass solution for parking enforcement, we needed to be able to track the location of the vehicle if we have identified a car that has parked illegally in a particular WKU parking lot. To accomplish this, once our mobile application and Alien F800 RFID Reader has scanned nearby parked cars inside of parked locations, we then would track the geolocation of the nearby car based on the mobile device that is linked to the RFID Reader. We are aware that this will not be 100% accurate but would give a close range that can be used for further enforcement if necessary.

**4. Phone can validate parking location accuracy post-tag scan.**
Similar to functional requirement 3, functional requirement 4 deals with running logic to determine if the parked car is allowed to park in the lot based on the location of the scanned tag. Shortly after scanning the tag, our app must be able to determine if the car was located inside of the lot based on the tracked geolocation. If the car is illegally parked, our app will notify the user of illegally parked cars.

**5. App is able to retrieve and display USER ID from scanned tags using the database.**
Another key functional requirement of the mobile application and project is to be able to display scanned tags and display their USER ID. This is an integral part of this project because the end user utilizing the app must be able to the tags and their associated USER IDs that they have scanned inside of the app. If this did not function properly, then proper enforcement from the WKU Parking and Transportation Services would not be possible.

**1. The User should be able to navigate through the application without any bugs or issues.**
An additional functional requirement, while not strictly mandatory, involves the implementation of multiple menus, screens, and a user interface designed to enhance the end-user's efficiency in completing their tasks. This holds significance for us, as it enables users to seamlessly access recently processed data and execute various functions through a user-friendly app interface.

### 1.3.2 Non-Functional Requirements

| **Mandatory Non-Functional Requirements** |
|---|
| 1. App works on Android devices using Android OS 11.0 or later. |
| 2. App allows for multiple devices to work simultaneously. |
| 3. Secures scanned information using encryption to prevent data leaks. |
| 4. The phone needs 2-4 GB of RAM. |
| 5. The phone needs 26.9 MB of storage space. |
| **Extended Non-Functional Requirements** |
| 1. Code will be well documented so later adjustments can be made by other teams in the future. |
| 2. Mobile app is user-friendly and requires minimal training |

These nonfunctional functional requirements listed are about the mechanics of the project, and how it is designed to work in multiple circumstances. The extended non-functional requirements focus on the system's sustainability and user experience. These requirements are further described below and are numbered in coordination with the table above.

**1. App works on Android devices using Android OS 11.0 or later.**
The first requirement ensured that the application is accessible across a broad spectrum of Android devices,

increasing the system's usability and reach. This inclusivity is vital for a system intended for widespread use across various device generations, ensuring that the application remains accessible to all users regardless of their device's age. The decision to pick version 11.0 in particular is this was the version running on the test device provided to us by the client.

**2. App allows for multiple devices to work simultaneously.**
The second requirement is designed to support scalability and flexibility in operations. By allowing multiple devices to work simultaneously without interference or degradation in performance, the system could accommodate the needs of the parking management team. This is necessary during times when multiple staff members may need to verify parking permissions simultaneously across different parking lots.

**3. Secures scanned information using encryption to prevent data leaks.**
The third requirement addresses the critical aspect of data security. By encrypting scanned information, the system ensured the protection of sensitive user data against unauthorized access and potential data breaches.

**1. Code will be well documented so later adjustments can be made by other teams in the future.**
The first extended requirement emphasizes the importance of well-documented code, which facilitates future modifications, debugging, and enhancements by future development teams. This ensures that the system can change in response to differing needs or technology upgrades within the system without requiring a complete overhaul.

**2. Mobile app is user-friendly and requires minimal training**
The second extended requirement prioritized the end-user experience by emphasizing the need for a user-friendly interface and minimal training requirements. This approach reduced barriers to effective use of the system, ensuring that parking management personnel could efficiently utilize the application with minimal on-boarding time. This would allow for quick adoption of the technology developed in the project and effective operation of the parking management system.

## 1.4 Target Hardware Details

The following hardware details below list all of the hardware utilized throughout our project implementation. Some of the hardware consists of technology that was lent to us by the WKU Parking and Transportation Services. Some of this technology was provided by the group, like our personal computers to work on Android app development.

### 1.4.1 Alien F800 RFID Reader

Our client is allowed us to use an Alien F800 RFID Reader that can read and program an EPC RFID tag. After the Alien F800 reads a tag, the tags could then be viewed on a computer device through their gateway software or web application. This device can also act as a host system if you are using an ethernet cable for power over ethernet in the same network you are using your computer with. If you are using it as a host system, you can access the web interface on any device on that network if you have the IP address that the Alien F800 RFID Reader is using. This device also is connected to the Alien ALR-8698 Antenna Unit which is the next sub-section of this section.

### 1.4.2 Alien ALR-8698 Antenna Unit

The Alien ALR-8698 Antenna Unit directly links to the Alien F800 RFID Reader by an antenna cable that can be connected to one if its antenna ports on the Alien F800. This device is responsible for transmitting and receiving radio frequency between the RFID reader and the tags. It is an important component of the system that allows a user to get the RFID tag data to the reader. This particular antenna can emit radio waves from 865 - 928 MHz that can reflect off of RFID tags of parked cars inside WKU parking lots.


Figure 1: Image of Alien F800 RFID Reader

### 1.4.3 Android Device

Our client, Kevin Werner, gave us a SONIN XP8 mobile phone to develop an android app for. This phone is older and utilizes Android 11.0. Because this phone is an older device, we plan on developing an application that will run Android 11.0 and newer. We ended up not using this phone as much as one of our team members, Nicholas Johnson, has an android phone that we mainly used for actual physical testing not done through the Android Studio Emulator.

### 1.4.4 Computer for Android Studio Development

Since we developed a mobile application, we needed another device to create that app. To create the mobile application, our team is utilized Android Studio to complete the WKU Parking and Transportation application. To be able to effectively utilize the software, there were baseline system requirements that a developer should have so that when developing a mobile application, the system we used does not have any issues creating the application. Those system requirements are listed on their website and are listed in the table below.

### 1.4.5 Various Electronic Cables

The hardware that was lent to us by the WKU Parking and Transportation Services, like the Alien F800 RFID Reader and the Alien ALR-8698 Antenna Unit requires several different cables that were provided to us also by the WKU Parking and Transportation Services. We were given an antenna cable that is used to transfer radio


Figure 2: Image of Alien ALR-8698 Antenna Unit

signals from the Alien ALR-8698 Antenna Unit and the Alien F800 RFID Reader. We also needed to use an ethernet cable to link the Alien F800 RFID Reader directly to a computer or to an ethernet wall outlet for power over ethernet for communication with a computer device.

## 1.5 Software Product Development

The following is a long list of software tools that our team has utilized to help develop the product. This software includes tools to produce code, interact with the Alien RFID technology, and create documents for our clients.

### 1.5.1 Visual Studio Code

We utilized Visual Studio Code to write and edit the software and use the Live Share extension within the program to set it up so that we all had access and and the ability to edit the code simultaneously. This let everyone stay up-to-date on the code and no one works on the same thing.

### 1.5.2 Alien RFID Gateway

The Alien RFID Gateway Software could be accessed by either a demo software that can be downloaded from the company's website or the same information can be accessed by a local website that runs once the hardware is plugged in and running. The Gateway shows the information that is retrieved by the scanner and decoded by the


Figure 3: Image of Sonin XP8 Smartphone

reader. We wanted to use this software and plug all the information gathered into our mobile app where it can be sorted by whether it is a WKU Parking Tag or not and then sorted even further by referencing the GPS and Database.

### 1.5.3 Android Development Studio

We used Android Studio Development as our main IDE for the app development. The client, Kevin Werner, wanted a mobile app created that the WKU Parking Transportation Services can use while they patrol around the various parking lots on campus scanning license plates and RFID tags. The Parking and Transportation Services exclusively uses Android Phones for work. Android Studio is the perfect IDE to use when it comes to developing the front end of the mobile app as it was designed just for the purpose of creating mobile apps for Android Devices.

### 1.5.4 Alien SDK

A Java-based software development kit provided by Alien Technology. It's used to integrate the RFID reader's capabilities with our mobile application, which, combined with the antenna, allows for the scanning, processing, and management of RFID tag data in a customizable way that we can change depending on what works and what the client requests of us.

| System Requirement | Specification |
|---|---|
| Operating System | 64-bit Microsoft® Windows® 8/10/11 |
| CPU Architecture | x86_64; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor |
| RAM | 8 GB or more |
| Disk Space | 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) |
| Screen Resolution | 1280 x 800 minimum |

Table 2: System Requirements for Android Studio.

#### 1.5.5 Database SOAP Data Calls

Our project will require a connection to the database that the WKU Parking Transportation Services uses to hold all of the data about the cars currently registered such as what tag they have and their license plate. The app will need to cross-reference the RFID tags that scan as a WKU parking tag and reference the database and the GPS location of the phone to see if that tag is within the correct lot.

#### 1.5.6 GitHub

We are using GitHub as our version control and a way to easily share all of the programming work with one another. It allows us to easily test our new codes against what everyone else was working with to make sure there are no errors when combining the codes before being pushed into the main project. GitHub also allows for us to safely store our programs in a location that is widely accessible by our project team.

#### 1.5.7 Discord

We use Discord as our primary method of online communication. We have a server set up and different channels to keep it orderly. It allows us to keep track of useful links and to easily see when we are setting up in-person meetings and who is working on what while we are not together in person.

#### 1.5.8 Google Slides

We use Google Slides as the website to make our presentations. It allows us all to work on the slides together in real time and to keep up to date on all of what is currently in the presentation so no one ends up working on something that is already completed.

#### 1.5.9 Latex Documentation

Our team will also be utilizing Overleaf, an online tool that is a Latex editor. We chose Overleaf because it allows for easy online editing of a Latex document. It also allows for several users to modify a document at once. Overleaf allows for easy downloading of the compiled .tex file for quickly sending versions of the document. We were able to cut costs and not pay for OverLeaf to allow all of us to work simultaneously by using a shared email and password.

#### 1.5.10 ChatGPT and other Resources

This team is new when it comes to working with RFID Hardware and software that the client has provided us to work with. During the course of our work for the client, we will be working and learning simultaneously RFID Technology and interpreting the data through the software. ChatGPT is a great learning tool that will greatly assist in getting us in the right direction for the completion of our work. Other online resources such as StackOverflow and various other websites can help us with any errors we may come across or teach us better ways on how to progress. The client also provided us a textbook that was his that he used for his certification on working with RFID technology. This textbook starts from the very basics of RFID which will help each member of our group learn the technology we are working with.

# 2 Modeling and Design

## 2.1 System Boundaries

### 2.1.1 Physical

One major physical system boundary is the user must be using an Android phone that is running 11.0 or newer. This app is designed to work for the phones the WKU PTS employees are using while on the job. If a user attempts to use an Android phone that is running any earlier versions, then some of the app functionality may not work as intended. Utilizing a newer version of Android, like 11.0 Red Velvet Cake, will allow our client to utilize this software for a much longer lifespan than using older versions.

The Alien P.R.O.B.E app must also require the user to obtain location data from the app. Because of this requirement, the user must allow the application to access the GPS in the phone or else the application will not be able to determine where the scanned tags are. Without this, our calculation to determine if scanned RFID tags are in the correct lot will not be accurate or possible.

The Alien RFID technology equipment must also be set up properly and working. If problems arise from the Alien RFID Scanner and Reader caused by incorrect setup on the WKU PTS scan vans, various issues may cause the app to stop functioning properly or a complete halt in functionality. This inter-connectivity is due to the RFID reading capabilities being exclusive to the RFID reader which allows the RFID tag data to be displayed in our mobile application.

### 2.1.2 Logical

The logical system boundaries encompass the integration and functionality of hardware and software components to automate parking management. The core includes the mobile app interfacing with the Alien F800 RFID Reader via Bluetooth or Network, which then connects to the Alien ALR-8698 RFID Antenna for real-time scanning of RFID tags.

This process involves validating parking permissions against a database, utilizing geolocation to ensure vehicles are in their designated lots. The system is designed for scalability, allowing multiple devices to operate concurrently without performance degradation.

Development follows a modified waterfall approach, incorporating client feedback, with sprints documented for transparency and future enhancements. User interaction is streamlined through a user-friendly interface, requiring minimal training for parking management personnel. These boundaries define the operational framework, emphasizing efficiency, security, and user accessibility, in order to improve parking management for WKU PTS.

## 2.2 Wireframes and Storyboard

The storyboard wireframe shows the different views we will have in a mobile application and how to access each view. The first view in the frame is the home screen that has to buttons that lead to 2 different views. The Setup button takes the user to the view where they set up the connection to the Alien Reader and then once connection is established it will show a successful message to the user before taking them back to the home screen. Once the connection is established, the user can then click on the scan button that will take them to the next view that will show all the scanned tags in the vicinity. There are 3 more buttons on this view that the user can click do activate more functions. The start/stop button will allow the user to stop the Alien Reader from finding anymore tags or to start scanning for tags again. The save button will let the user save the scanned tags to a database. The clear button will clear the Reads textbox of all scanned tags. This button will come in handy when the user moves to a different lot and needs to start fresh with the tags in that lot.

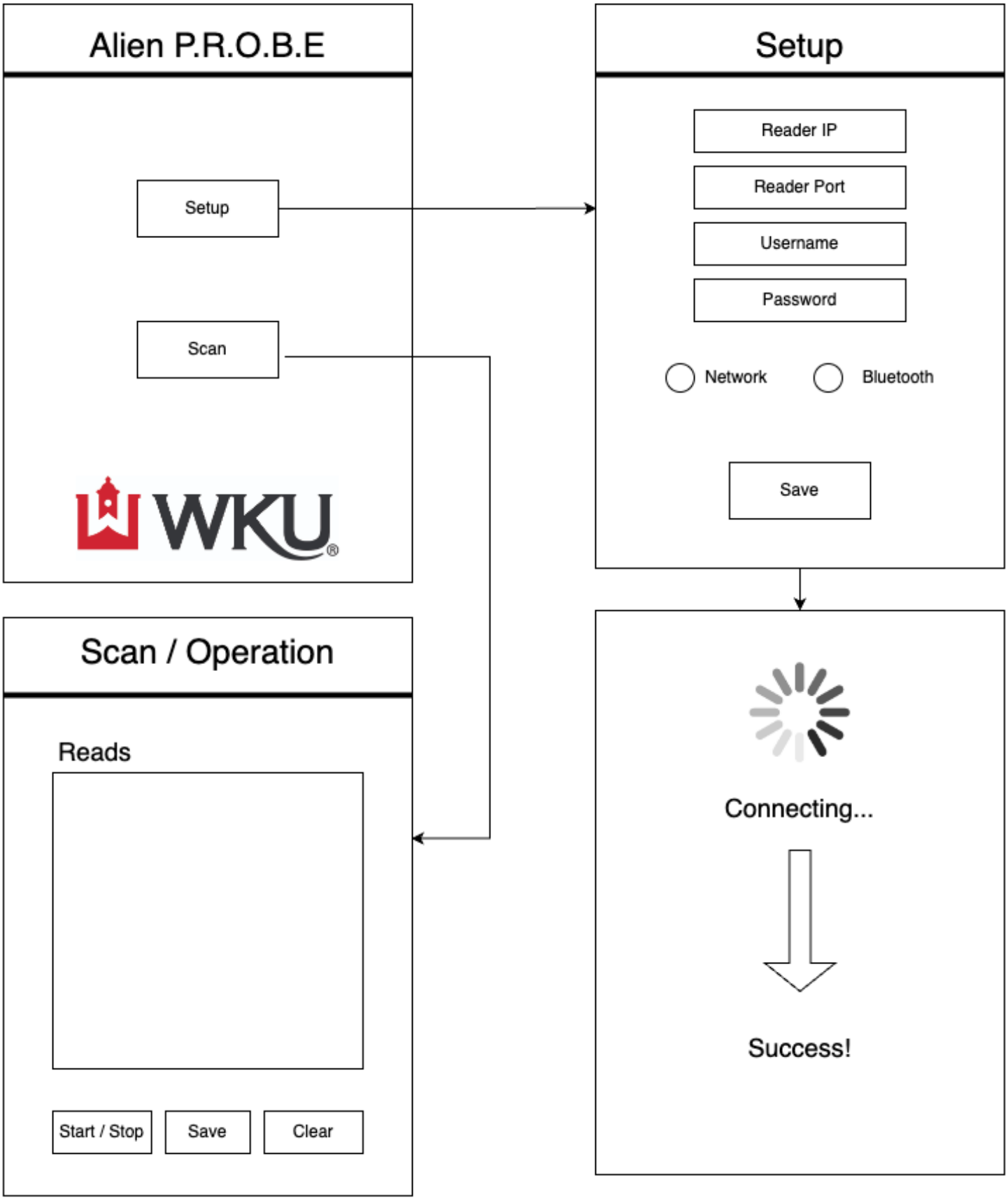


Figure 4: Storyboard for Alien P.R.O.B.E. app.

## 2.3 UML

### 2.3.1 Class Diagrams

Creational Design Pattern - Factory

The factory pattern is used to help show how object creation occurs. In our specific case, the factory pattern shows how RFID tag objects are created and the interaction of those objects with the client. In the diagram below, objects can be requested by the client. When an object is requested or asked for, the ScannerActivity.kt (Factory) will create a new RFID tag object. Once the RFID tag object is created, the client can use those objects when needed. The objects created will then be displayed in the Scanner Activity XML with its attributes. This approach will allow the end user with better options when it comes to accessing RFID objects for viewing. The ScannerActivity.kt code can be found in the appendix under C.5 ScannerActivity.kt.

Structural Design Pattern - Adapter

The adapter pattern is used to convert the raw data from the RFID reader into something more usable by the database. When the RFID Reader reads an RFID tag, the information from that tag is stored as raw data from the RFID Reader. This data is then sent to the Alien P.R.O.B.E. app through either a network connection or through Bluetooth. The Alien P.R.O.B.E. app then takes that raw data and edits it down to contain only the necessary information for the database, removing extraneous details that could slow down the process. This converted data, consisting of the RFID tag's userID and the location the tag was scanned at (discovered by the Android device's GPS functionality) is sent to the database through SOAP calls. The database gets the related information to the specified userID, and send back to the app whether or not their allowed parking matches with the sent location information, allowing the user to confirm if there is invalid parking or not.

Behavioral Design Pattern - Activity

This behavioral activity diagram illustrates the use of the app within the context of scanning, reading, and saving RFID tags. Pictured in the reference figure is the PTS (Parking and Transportation Services) employee as well as the Student/Faculty and their vehicle. The technology here is the Alien P.R.O.B.E. application and the database that will be used to cross-reference/store the tag lists and other meta data associated with the tags. Here we can see the 3 different activity that encapsulate all of these items. The Scan Tag will scan the tag from the Student or Faculty vehicle. This functionality is through the GetTagList function in the AlienScanner.java code which can be found in the Appendix under C.2 AlienScanner.java. That is then stored in the data base upon interaction with the save list activity. Then the Reader Setup, only accessible to the PTS employee, can be used to setup the IP, Port, Username and password to easily access the reader.

### 2.3.2 Use Case Diagrams

The Use Case diagram shows all of the available functions a WKU PTS employee should be able to use when they are using the Alien P.R.O.B.E. mobile app. They have the ability to set the connection to the Alien Reader, scan tags with the reader, view each tags extended information such as a rough GPS location, save what tags have been scanned into a database, and clear the table of scanned tags for when they move lots so the information is not cluttered in the list displaying all scanned tags.

### 2.3.3 Use Case Scenarios Developed from Use Case Diagrams (Primary, Secondary)

Use Case: Scan Tags

This covers the user instructing the app to scan for tags, which leads to the RFID Reader scanning for tags and sending the information to the app.

Use Case: View Tag Information.

This covers the user using the app to view the scanned tag information, which leads to the app displaying the information of the tags scanned so far.

Use Case: Set Up Connection.

This covers the user being in the app and without a connection method being set up.

Use Case: Stop Scanning.

This covers the app being in the scanning state and the user pressing the "Stop" button to stop scanning.

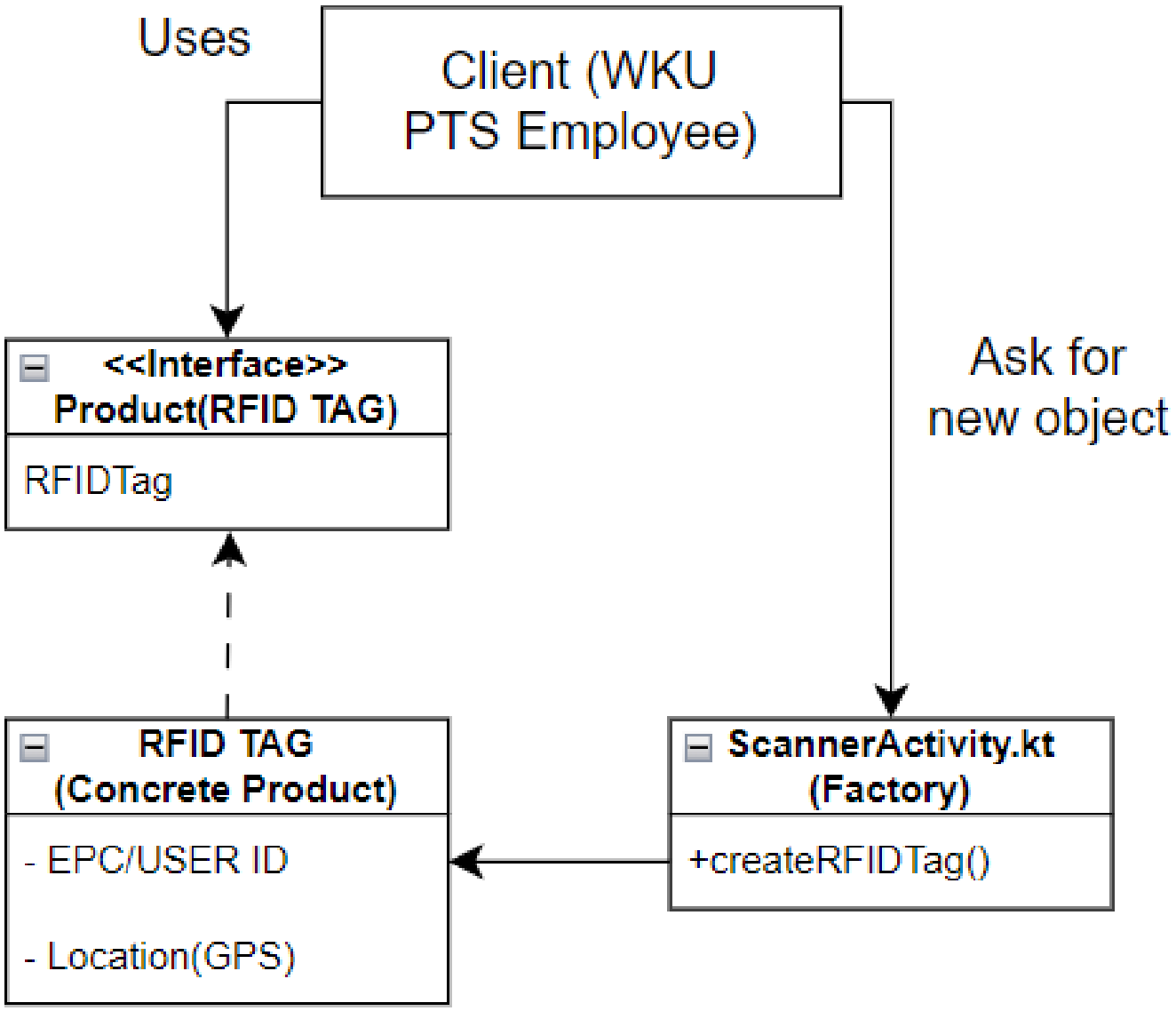


Figure 5: UML Class Diagram - Factory Adapter Diagram showing how RFID tag objects are created for the client.

| **Use Case: Scan Tags** |
|---|
| ID: UC01 |
| Actors: Employee |
| Preconditions: App is connected to RFID Reader |
| Primary scenario:<br>The "Scan Tags" button is pressed on the app, leading to the app communicating with the Alien RFID Reader and receiving the scanned tags. |
| The functionality for this is in the GetTagList function in the Appendix under C.2 AlienScanner.java. |

Use Case: Save Scanned Tags.
This covers pressing the "Save" button after the app has already received tag information.
Use Case: Clear Scanned Tags.
This covers when the app has scanned tags displayed, and the user pressed the "Clear Tags" button.

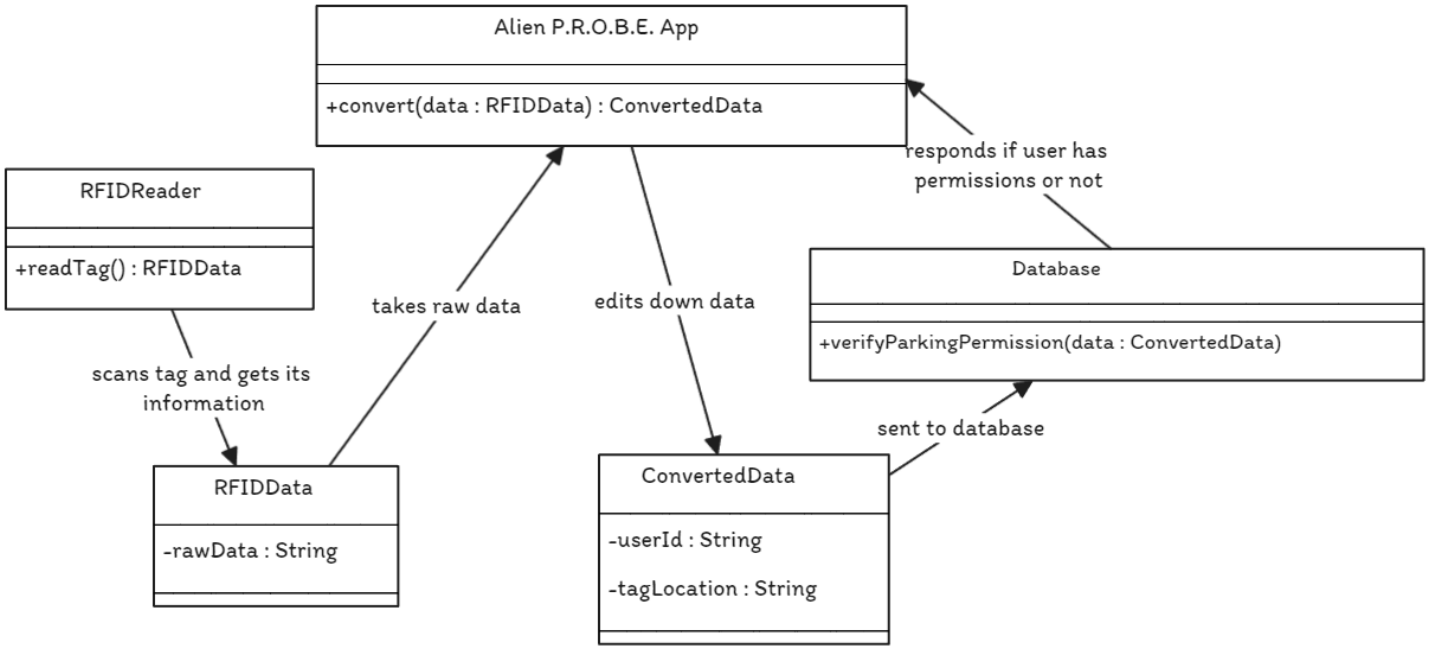


Figure 6: UML Class Diagram - Adapter Diagram of how the Alien P.R.O.B.E. app adapts the raw data for the database.

| **Use Case: View Tag Information** |
|---|
| ID: UC02 |
| Actors: Employee |
| Preconditions: App has received tag information from the RFID reader. |
| Primary scenario:<br>The "View Tags" button is pressed on the app, leading to the app accessing the stored tag information from the scans it has received and displaying their information. |
| The functionality for this is in the buttonClick val through the onCreate function in the Appendix under C.3 MainActivity.kt. |

| **Use Case: Set Up Connection** |
|---|
| ID: UC03 |
| Actors: Employee |
| Preconditions: App is open and either doesn't have a preset connection or the "Setup" button is pressed. |
| Primary scenario:<br>When the user presses the "Setup" button, they go to a screen that allows them to enter the Reader IP, Reader Port, Username, Password, and whether the connection is through Network or Bluetooth. |
| The functionality for this is in the settingsClick val through the onCreate function in the Appendix under C.3 MainActivity.kt. |

| **Use Case: Stop Scanning** |
|---|
| ID: UC04 |
| Actors: Employee |
| Preconditions: The app is scanning and the user presses the stop button. |
| Primary scenario:<br>When the user presses the "Stop" button while the app is currently scanning, the phone sends a signal to the reader to stop actively scanning for RFID tags. |
| The functionality for this is in the toggleOnOff val through the onCreate function in the Appendix under C.5 ScannerActivity.kt. |

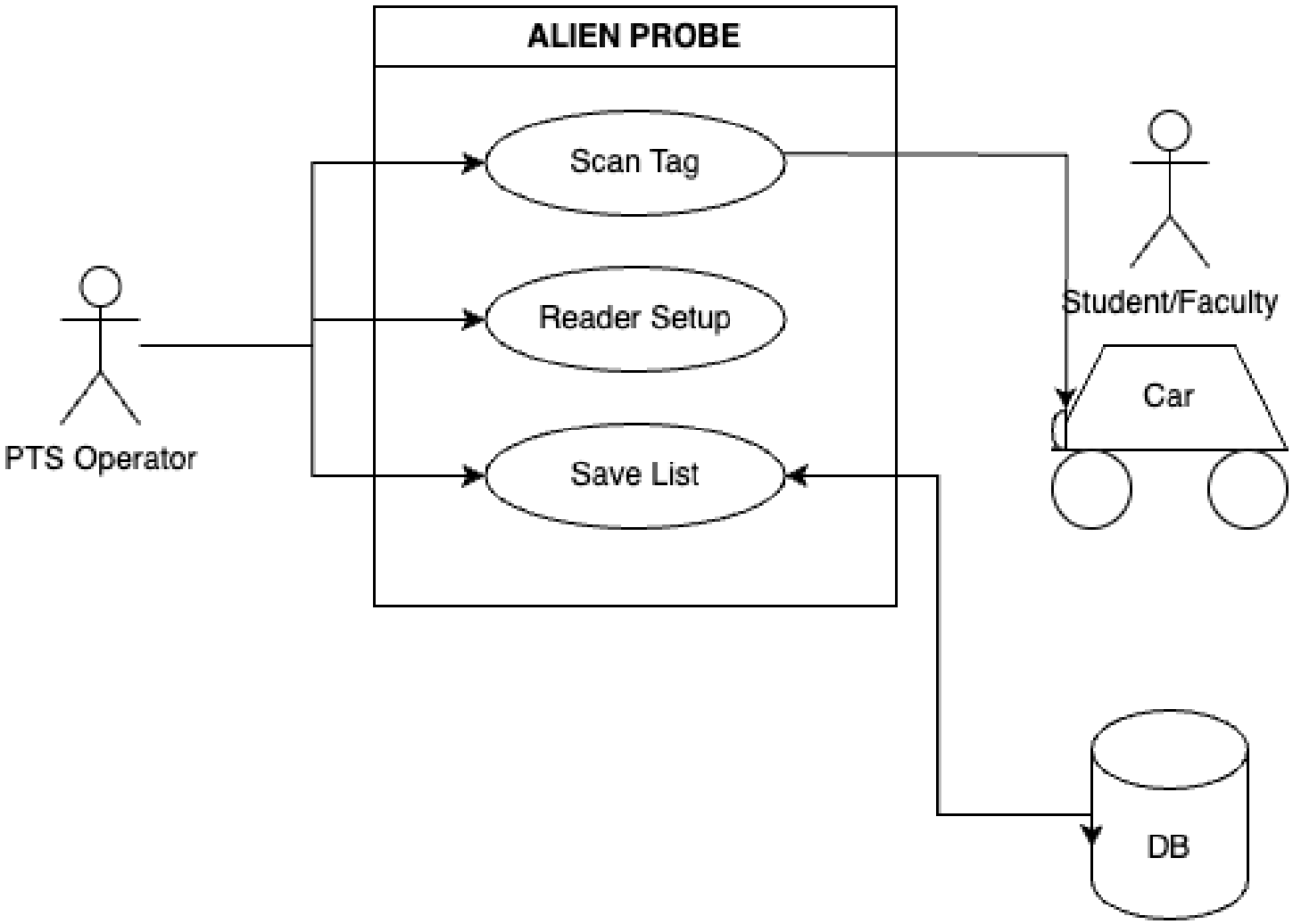


Figure 7: UML Class Diagram - Activity Diagram showing how the Alien P.R.O.B.E. takes in information and stores it.

| **Use Case: Save Scanned Tags** |
|---|
| ID: UC05 |
| Actors: Employee |
| Preconditions: The app has already scanned some tags and the user pressed the "Save" button. |
| Primary scenario:<br>When the user presses the "Save" button, the app saves all the tag information scanned so far into the app's memory. |
| The functionality for this is in the saveClick val through the onCreate function in the Appendix under C.5 ScannerActivity.kt. |

| **Use Case: Clear Scanned Tags** |
|---|
| ID: UC06 |
| Actors: Employee |
| Preconditions: App has already scanned some tags and the user presses the "Clear Tags" button. |
| Primary scenario:<br>When the user presses the "Clear Tags" button, all the currently displayed tags that have been sent from the RFID Reader. These displayed tags are removed from the screen, clearing up screen space. |
| The functionality for this is in the clearClick val through the onCreate function in the Appendix under C.5 ScannerActivity.kt. |

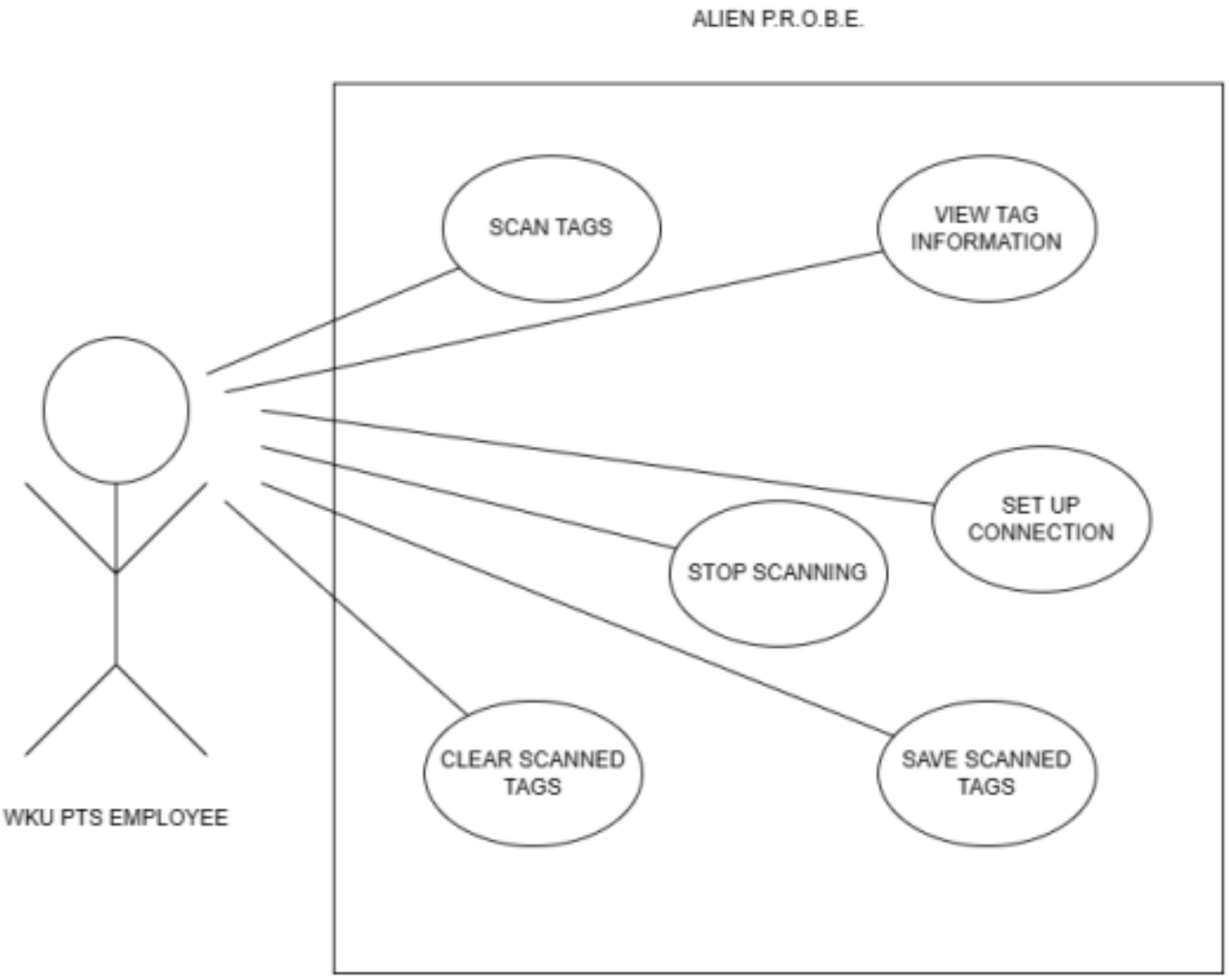


Figure 8: Use Case Diagram - Alien P.R.O.B.E. system.

### 2.3.4 Sequence Diagrams

The Sequence diagram shows the whole life cycle of the Alien P.R.O.B.E. mobile application. Once the Alien Reader is connected, steps 5-13 are repeatable to continue scanning more tags for as long as a WKU PTS employee may need to as they work going through the lots checking every car that has a scan-able parking tag in their window. The connection to the reader is set up in the openReader function, while the scanning for tags is within the GetTagList.java function, both of which are in the appendix under C.2 AlienScanner.java.

### 2.3.5 State Diagrams

The State Diagram of the Alien P.R.O.B.E. mobile app shows each state throughout its life-cycle from opening the mobile app and to when the user decides to stop using the application. Each state has connecting actions on what will get it to that state and what available actions they have to achieve a different state. This functionality is within the GetTagList function in the appendix under C.2 AlienScanner.java.

### 2.3.6 Component Diagrams

The component diagram breaks down a specific functionality of the software. In our component diagram, the functionality shown contains the steps it takes to scan and save tag data in our Alien P.R.O.B.E app. This specific

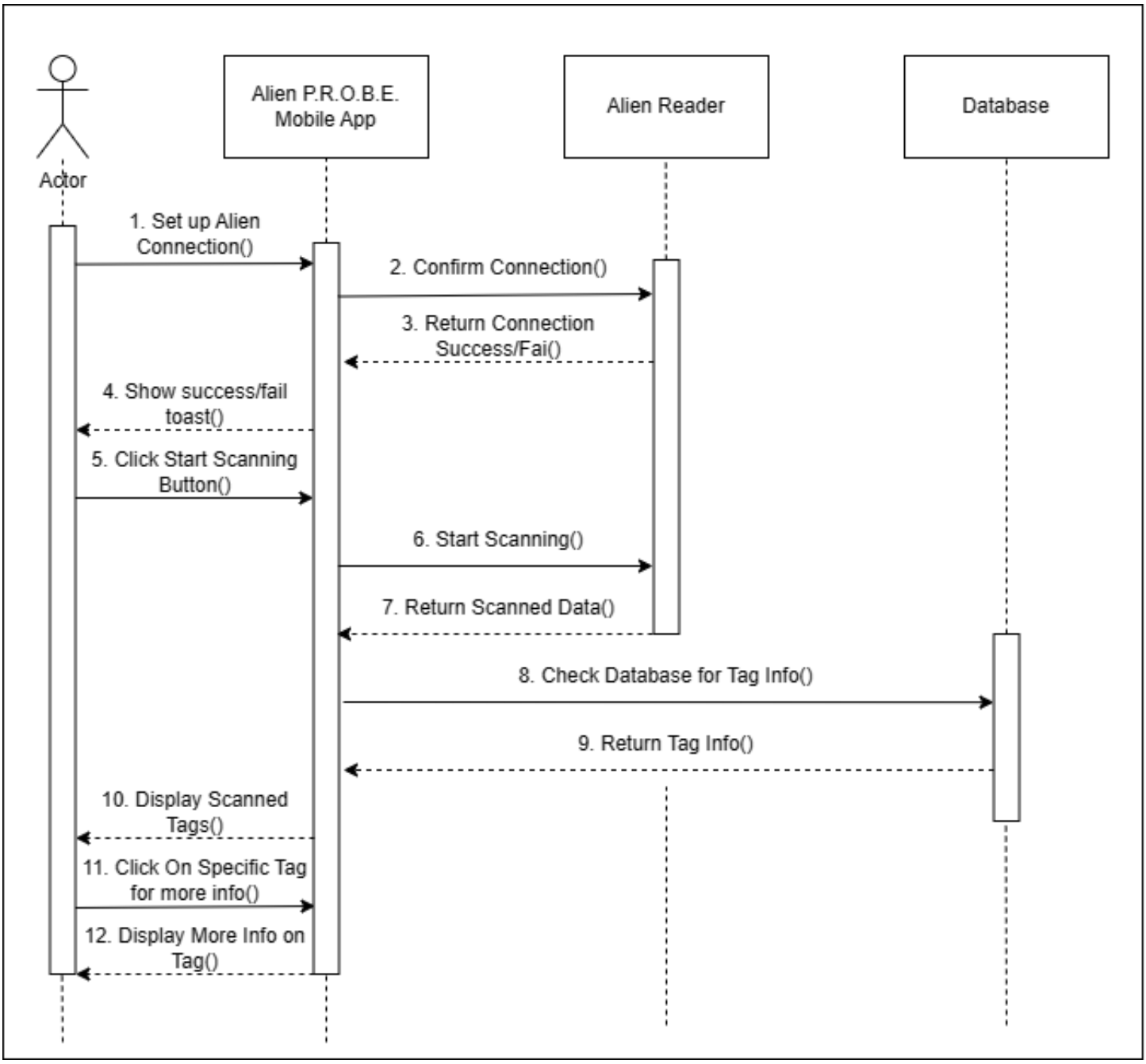


Figure 9: Sequence Diagram - Whole Life-cycle of Alien P.R.O.B.E.

component showcases the process of how scanning a tag functions. This is first done by determining if the Android Phone is connected to the RFID scanner. If a connection exists, the app checks to see if the isScan toggle button is on. If this toggle button is on, then the Alien P.R.O.B.E app requests the GetTagList() (found in appendix under C.2 AlienScanner.java) from the Alien RFID Scanner. This data is then stored within the database.

### 2.3.7 Deployment Diagrams

The deployment diagram shows how the technology is connected for our project, with the Alien RFID Reader being connected to RFID Antennas, as well as being connected to the network to link with the Android Device running the Alien P.R.O.B.E. app. The app can then make SOAP calls to the WKU Database to check whether the tags sent have permission to park using the geolocation of where they were scanned.

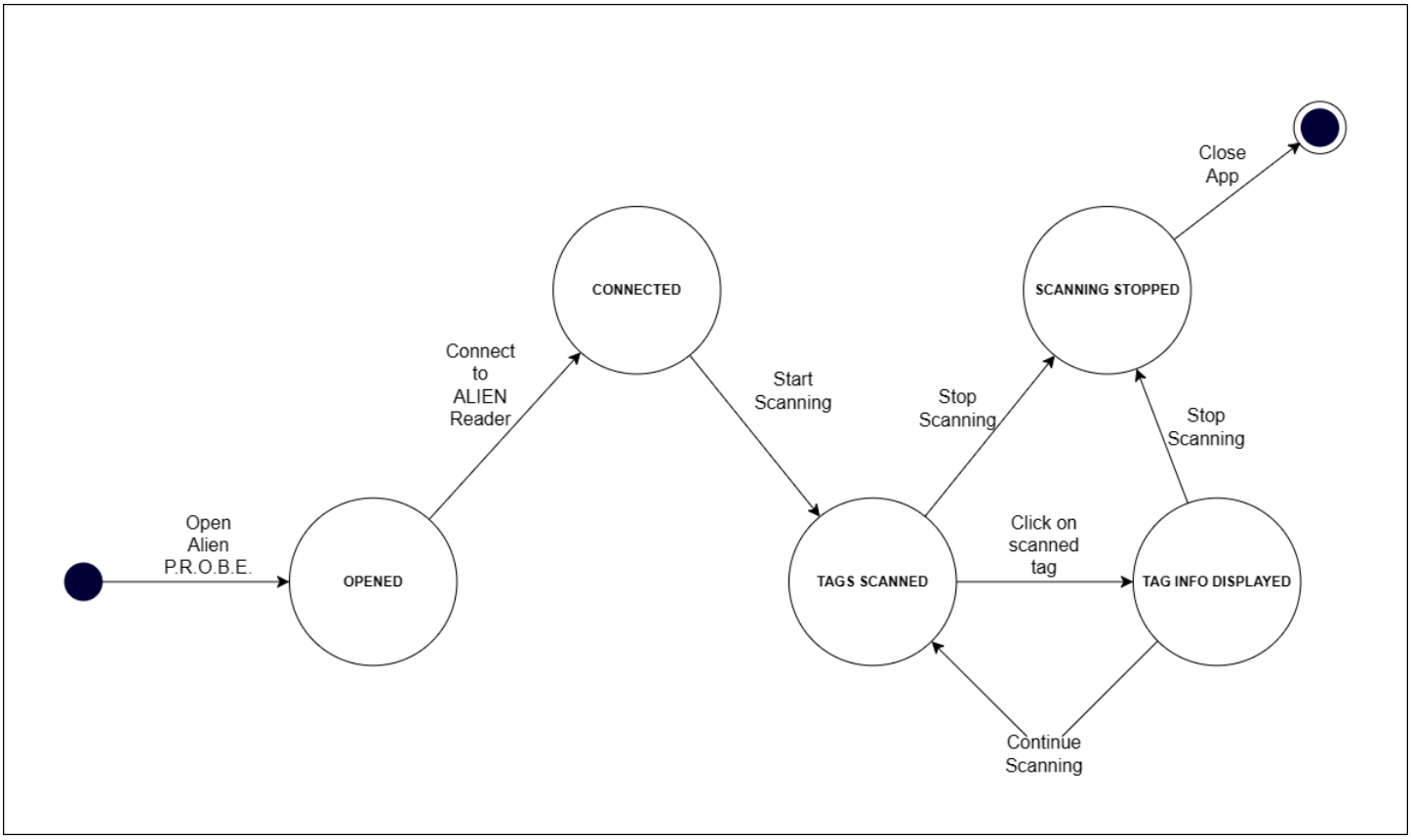


Figure 10: State Diagram - Alien P.R.O.B.E. app.

## 2.4 Version Control

The Alien P.R.O.B.E Android project team utilized GitHub to keep track of changes and have a safe place to store all versions of the app and other related files. Each member that was working on the project would work on the branch that had the most recent file change. If there was a major new implementation that was occurring, other than simple file changes like the addition of a new module like a new activity, we would usually branch off from the main and then merge the new branch with the main branch. If this new merge with the main branch was successful, we would continue on with the next implementation of the project. To utilize GitHub, we would use GitHub desktop to push, pull, commit, and merge changes to our GitHub repository. To keep this repository safe due to potential security issues, we also kept this repository private to reduce risks.

Table 3: Alien P.R.O.B.E Terms

| **Term** | **Definition** |
|---|---|
| Alien F800 Reader/Scanner | The reader is responsible for initiating and managing communication with RFID tags. It also consists of a radio frequency module, a microcontroller or processor, memory, and communication interfaces which allows us to connect to our mobile device. |
| Alien ALR-8698 Antenna | Emits and receives waves that allow us to detect RFID chips that are located inside of RFID tags placed inside of vehicles across Western Kentucky University. |
| MainActivity.kt | The kotlin activity file that is the main menu for the device. |
| AboutActivity.kt | The kotlin activity file that holds information about the creators and directions to use the app. |
| SettingsActivity.kt | The kotlin activity file that changes the preferences used to connect with the Alien F800 Scanner. |
| ScannerActivity.kt | The kotlin activity file that takes the preferences set in the SettingsActivity.kt and obtains a tag list from the Alien hardware. |
| AlienScanner.java | This is the class used to create an object that links with the Alien SDK to allow for connections with the reader. |
| RFIDTag.java | This is the class that is used for object creation when a new tag is discovered in the getTagList() function of the AlienScanner.java class. When a new RFIDTag is made, it gets put into a list to send back to the ScannerActivity to allow for displaying in the ScrollView. |
| .xml Layout Files | Each activity .kt file has an associated layout .xml file that is used for what the end user sees. This has buttons, views, and other tools for displaying things on a screen. |
| strings.xml | This file is used to hold strings that are used to display content in the .xml files that the end user can see. |
| getTagList() | This is the main function of the reader. If network mode is selected, this function will create a new thread because network functions cannot be made on the main(UI) thread of Android. After making a new thread, a socket connection between the Android device and the hardware will be made. The function will then open the reader and give the credentials for the scanner. Once validated, the command "t" will be sent to the reader and a string of tags will be sent back to the android device. RFIDTag.java objects are then created and added to a list that is returned back to the ScannerActivity.kt file. |

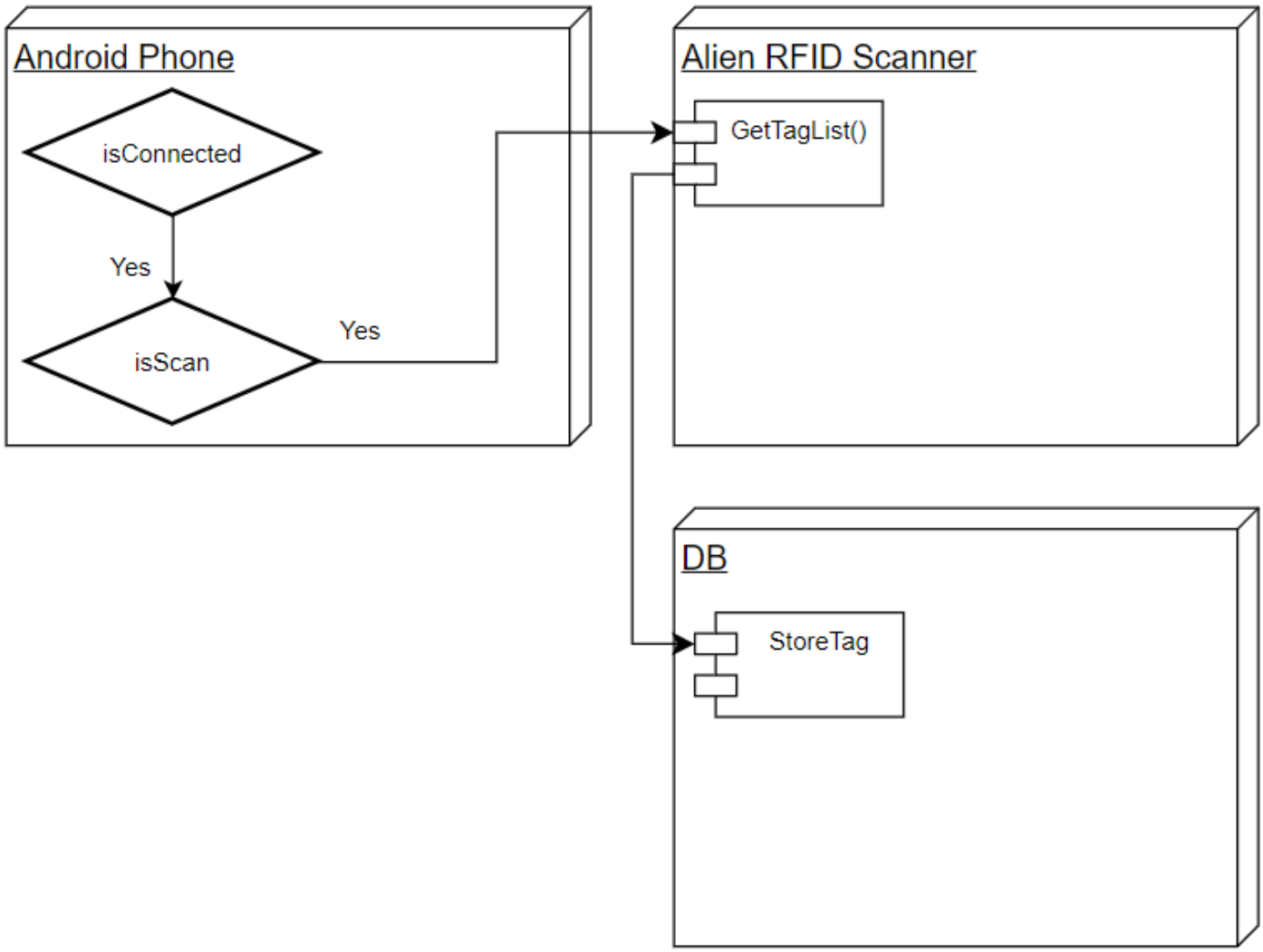


Figure 11: Component Diagram - Shows connectivity of scanning process in the Alien P.R.O.B.E. system.

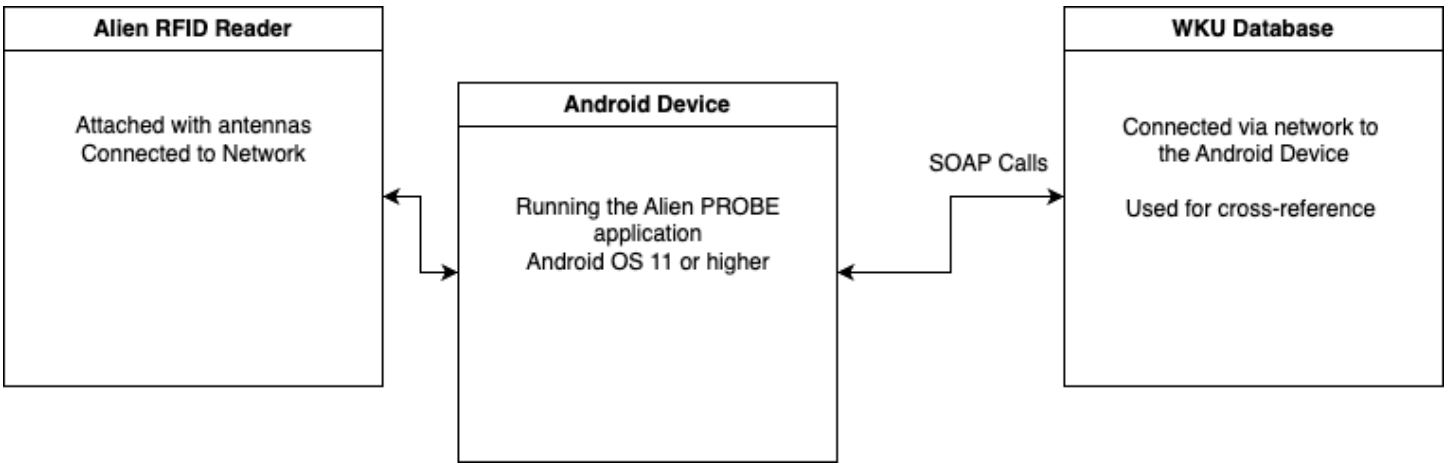


Figure 12: Deployment Diagram - Alien P.R.O.B.E. system.

## 2.5 Data Dictionary

## 2.6 Requirements Traceability Table

| Requirements | UC01 | UC02 | UC03 | UC04 | UC05 | UC06 |
|---|---|---|---|---|---|---|
| Alien F800 RFID Reader scans parking tags using the Alien ALR-8698 RFID Antenna. | X | | | | | |
| Phone is able to connect to Alien F800 RFID Reader through Bluetooth to allow communication across devices. | X | | X | | | |
| Implementation of geolocation tracking for scanned tags in the mobile app. | | | X | | | |
| Phone can validate parking location accuracy post-tag scan. | | X | | | | |
| App is able to retrieve and display USER ID from scanned tags using the database. | | X | | | X | |
| Data synchronization is ensured between the mobile app and RFID Reader. | X | | X | X | | |
| The User should be able to navigate through the mobile application and move between different menus. | X | X | X | | | |

## 2.7 User Experience

### 2.7.1 Alien P.R.O.B.E Flow Diagram

The Alien P.R.O.B.E. Flowchart shows the current functionality of what our app can achieve. It starts off with launching the app and immediately going into the settings and filling them out. If the settings are not filled out and you go to the scan screen, then the scanning function will not be able to find any tags. Filling out the correct settings is important to be able to connect to the reader and if a connection is not established its important to go back and check the settings to make sure they are correct. Once the settings are filled out correctly, you can then go to the scan menu and start scanning tags by clicking the save button. This sends a single scan command to the reader to find all the tags in the vincinity and save them as a list object in our app. Once the objects are saved, they are then displayed on the scan menu. Before new objects are saved, the app checks for repeated scanned tags in the list. If there are any repeats then it will not create a new object for that specific tag. Once the user is done scanning, they can clear the saved tags and exit the app.

### 2.7.2 Alien P.R.O.B.E Objectives

The objectives that we aim to achieve with this app is to help the WKU Parking Transportation Services find cars that are parked in the incorrect lots and parking structures. The reason for this is to hopefully reduce the amount of people parking wherever they want which will lead to more spaces being open for those who have paid to park in certain lots or structures. The WKU PTS employees already have some means to identifying cars that are parked incorrectly, this app is to act as another aid in identifying cars to help while they are driving through the lots to identify cars much quicker and show the PTS employee roughly where a car is in relation to their van so they can identify said car much quicker.

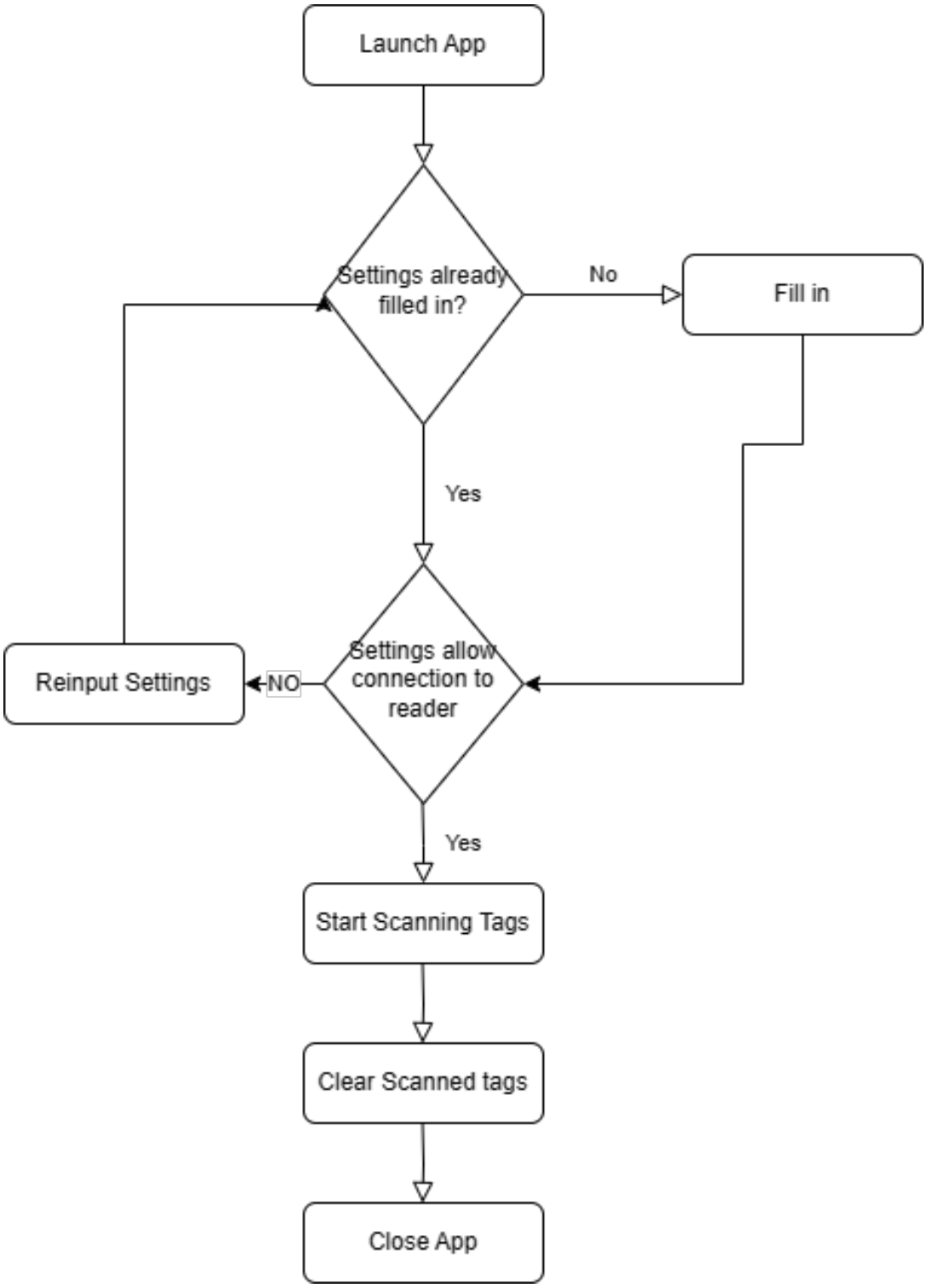


Figure 13: Alien P.R.O.B.E. Flowchart

### 2.7.3 Alien P.R.O.B.E Mechanics

The functionality of the app revolves around setting up a connection with the Alien F800 Reader and receiving the items scanned back to the app. The app connects to the reader through either a network connection or a blue tooth connection. The network connection is set up with sockets where the app sends the username and password to the reader first and once that is accepted, the app then tells the reader to scan all the tags in the vincinity. Once the tags are scanned, the reader then sends back all that information. The Bluetooth functionality has not been fully set up yet, but it will work roughly the same way but with "BluetoothSocket"s. Once the information is received by the app. The app then identifies each tag and checks if there is already an existing tag with the

same ID in the saved tag-list on the app. If there is not a pre-existing tag, it will then create a new object for that tag ID. If there is a tag that exists then no new object will be created. Once all new objects are created, it then is displayed on the scan screen of all the existing RFID tag objects that are saved in the tag-list. There is also a clear button that removes all scanned tags from the tag-list so when the user moves to a new lot, they can start fresh and not have the screen cluttered with previously scanned tags

#### 2.7.4 Alien P.R.O.B.E Menu Screens

In the current state of this app, there are 3 screens that the user sees and interacts with. The first screen the user will see is the menu screen. There are 3 buttons on this screen as well as the logo for the app. There is a small about section for the app that will pop up when clicked. If this is the first time a user is about to use the app, they should navigate to the settings screen first. This is where the user inputs all the information that is required to connect with the reader. The scan screen is where the user will spend most of there time while the app scans for RFID tags in the vincinity and displays them in a TextBox on this screen.

# 3 Non-Functional Product Details

## 3.1 Product Security

### 3.1.1 Approach to Security in all Process Steps

From the beginning our project had relatively little security risk involved, as with our goal of communicating the Electronic Product Code (EPC) and the current geolocation of the RFID tags with the WKU Parking and Transportation database was through SOAP calls. SOAP calls have built in security, so even though the EPC is not private information, we still have a layer of security in that sense. When receiving data from the database from the SOAP call return, it is also secure, but it only contains a yes/no boolean of whether that tag has permissions to park in that area. If the car is not allowed in that area, it will also provide the app with information to help the employee identify the car. All the information in this exchange is secure, but also not particularly needing of protection as it is unobtrusive. This level of security has been our objective the entire project, and hasn't particularly changed between sprints.

### 3.1.2 Security Threat Model

We currently only have a connection between our app and the RFID reader, where the app gets the tags read by the RFID reader and displays it on the app for the user to read. We reflect this approach in our diagram, with having the user, app, and reader as the only points, with a barrier between all. The app temporarily stores the tags as well, so there is an flow of data within the phone storage and app.

### 3.1.3 Security Levels

There is only one type of user, as there is no admin log-in function within the app, however, the user does have to input their username and password credentials, as well as the IP and port number to get the app to access the reader. This prevents unintended users from accessing and using the RFID Reader, while also being a part of the setup process for the RFID Reader. The app since it is not publicly available will only be use-able by the WKU Parking and Transportation Service employees on their work phones.

## 3.2 Product Performance

### 3.2.1 Product Performance Requirements

The Android version must be 11.0 or higher, this is the case because if we went back too far, the coding functions for Android development change, so we settled on 11.0 as it meets our clients needs and has all the functionality we need. The application itself is light on requirements, it requires 2-4 GB of RAM, this will keep the tags stored in the app for reference to the user, so they can find the cars parked in an invalid location by using the data stored within the app about the car. The phone the app is installed on also needs a minimum of 26.9 MB of storage space as that is the size of the app.

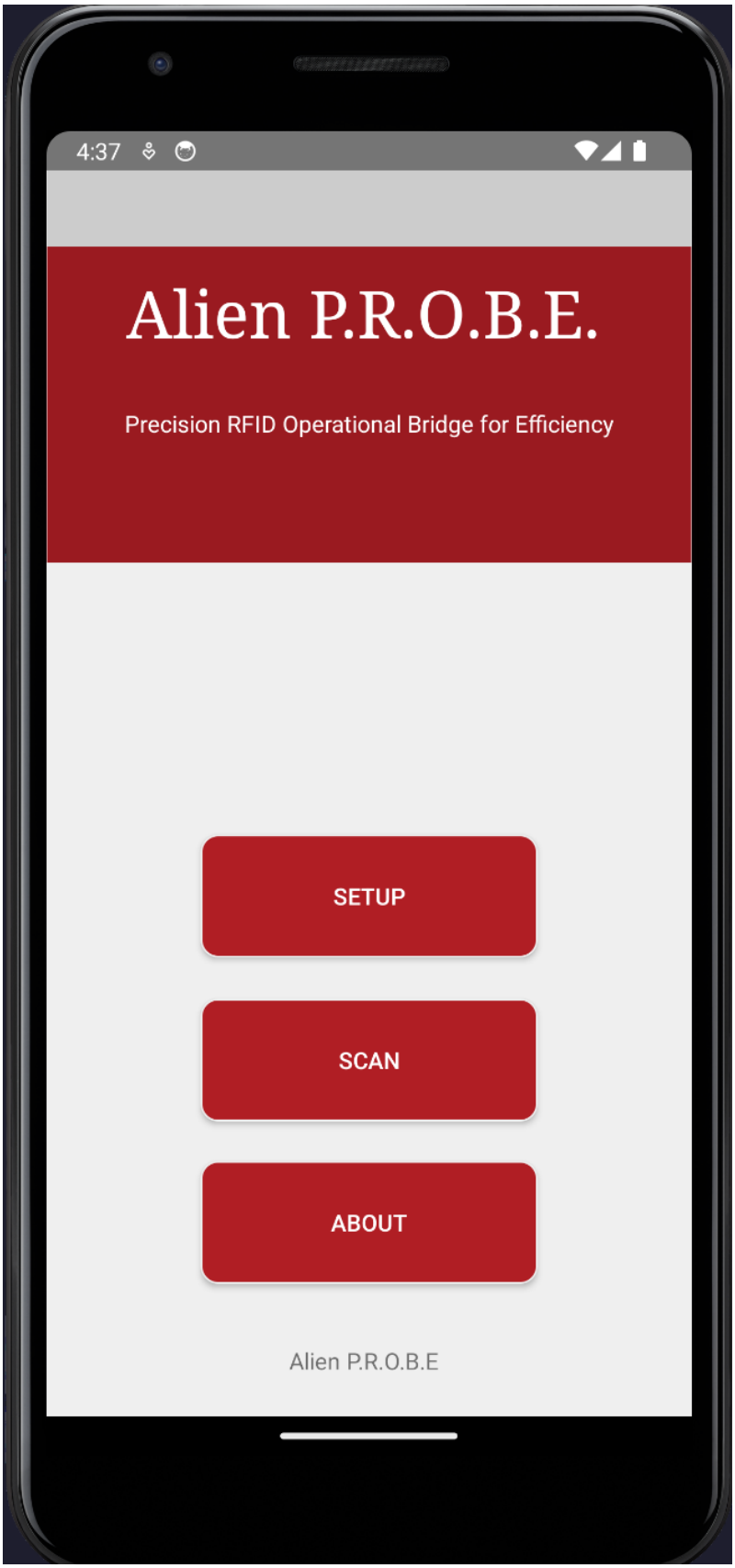


Figure 14: Alien P.R.O.B.E. Menu Screen

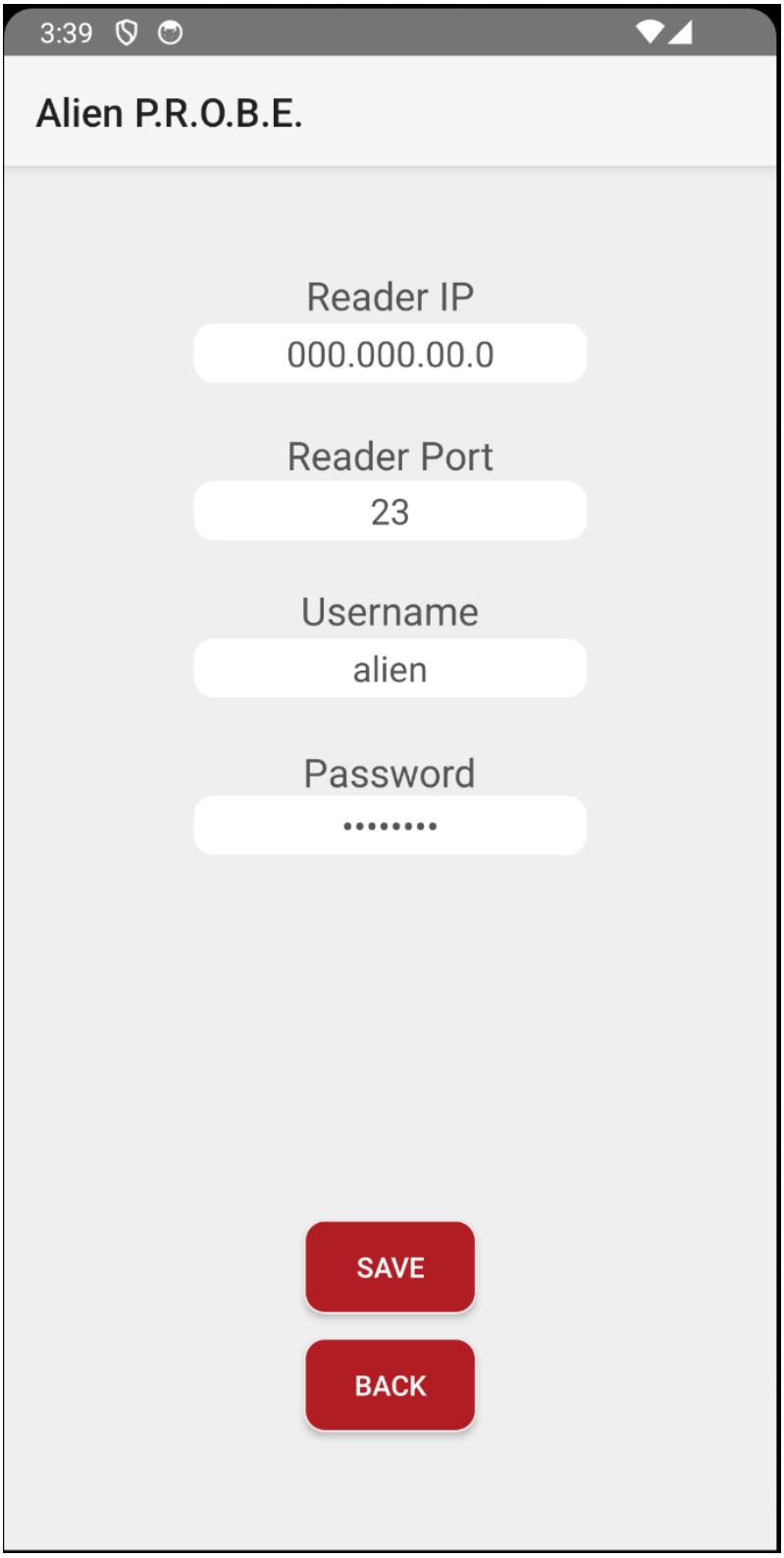


Figure 15: Alien P.R.O.B.E. Setting Screen

### 3.2.2 Measurable Performance Objectives

Since our project is small in size compared to other apps and does not have a large amount of intensive processing, our goal is to continue trying to keep our mobile application performance low by optimizing the app's functions. With this in mind, since the app will be continually running the getTagList() function, we need to do adequate testing of this feature to ensure that no crashing or other issues arise when scanning for parking tags


Figure 16: Alien P.R.O.B.E. Scan Screen

occurs.

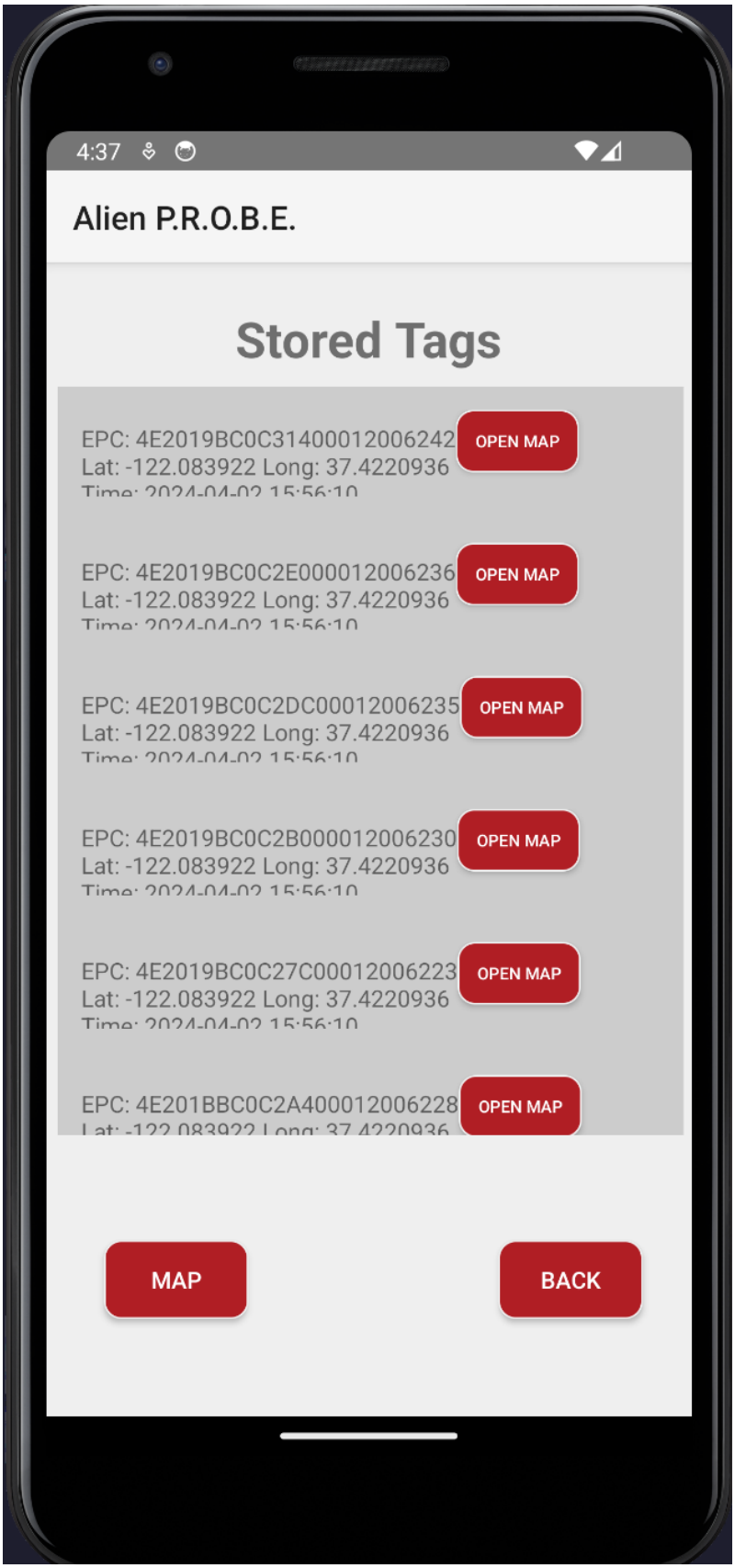


Figure 17: Alien P.R.O.B.E. View Tags Screen

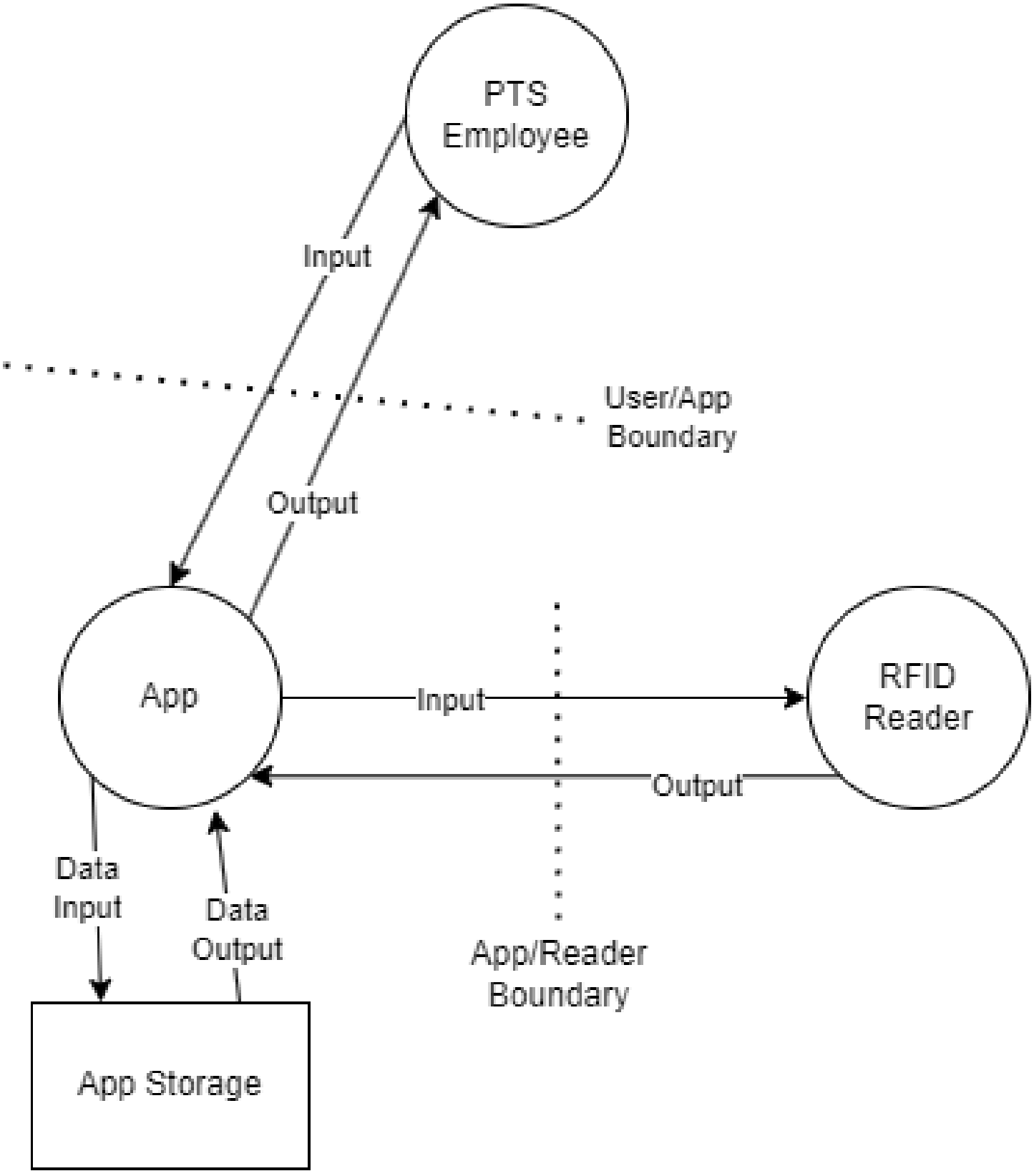


Figure 18: Information passes between user, app and reader.

### 3.2.3 Application Workload

To gather accurate time data to calculate the percentage of time used in each section, we added a timer to each activity to determine how much time was spent on each screen/activity. We then averaged this and got the percentage data below averaged to the nearest whole percent.

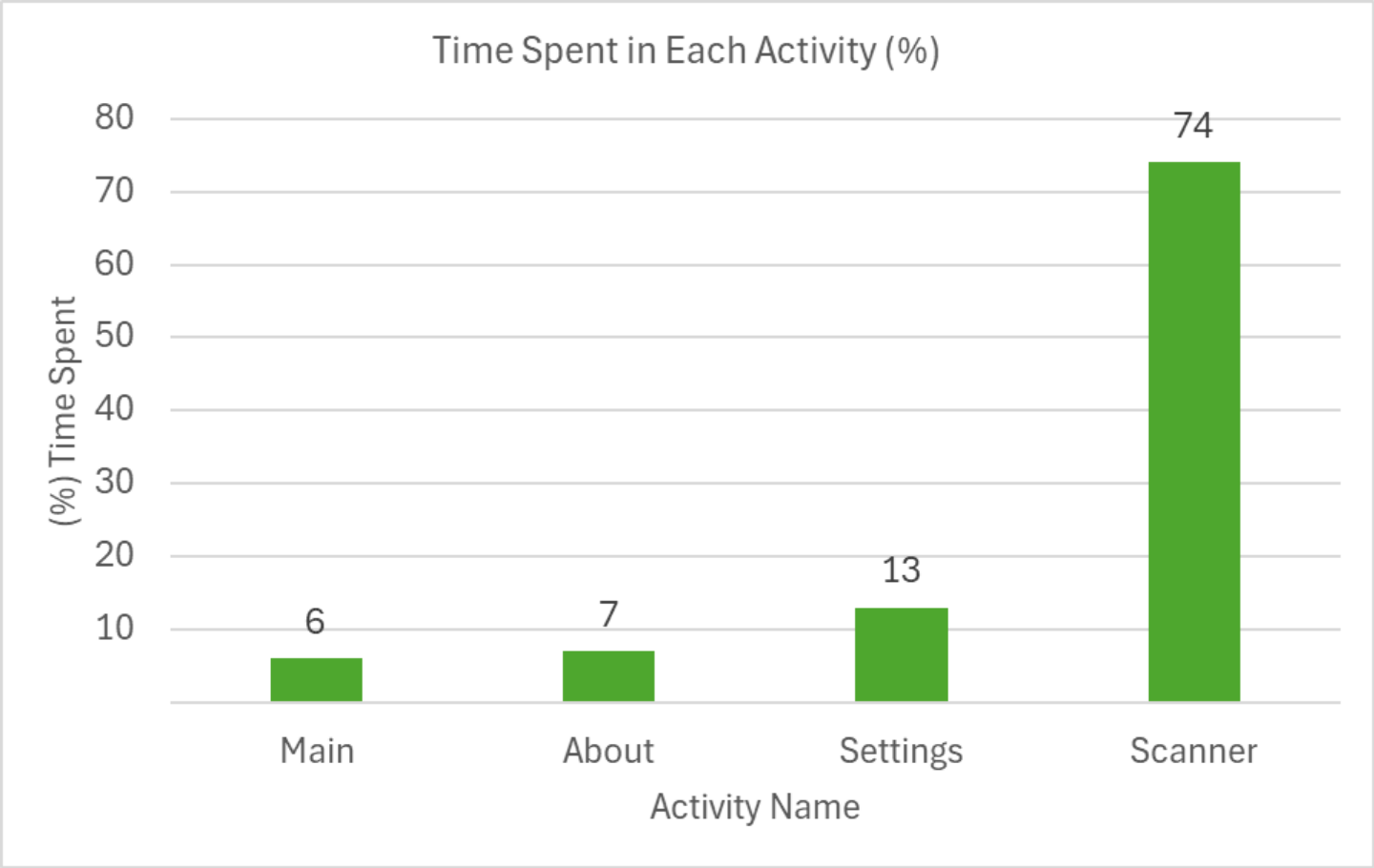


Figure 19: Application Workload across Alien P.R.O.B.E Activities

#### 3.2.4 Performance Tests

For gathering performance metrics, we used delta times for multiple functions and tested them 5 times each, averaging them out between the five trials to get an idea of how long each function will generally take. This allows us to see what functions may need to be optimized and which functions could be potential bottlenecks to watch out for. In our tests getTagList() takes the most time by a fair margin, as it is the most intensive function in our code. clearTags() took the least time as it has fewer complications. Both functions averaged out below 150ms, so the effect of these wait times should be negligible in practice.

#### 3.2.5 Hardware and Software Bottlenecks

The main, and largest function in the app, GetTagList(), which adds tags from the RFID reader to the list in the app, while checking for duplicates, takes O($n^2$) in Big O notation based on Round Trip Time for the call. As this is the biggest complexity of the functions in the app, it would be the most likely software bottleneck. While in practice we have had no issues as despite its O($n^2$) complexity it runs near instantly, on slower phones it would likely be the cause for a software bottleneck. When testing the hardware through the built-in Android Studio Profiler, we found that doing the various functions on the mobile app on average used 10-22%The memory usage ranged from 80-100MB, increasing as the user continued to use the app due to the increasing tag elements being stored.

## 4 Software Testing

### 4.1 Software Testing Plan Template

**Test Plan Identifier:**
**Introduction:**

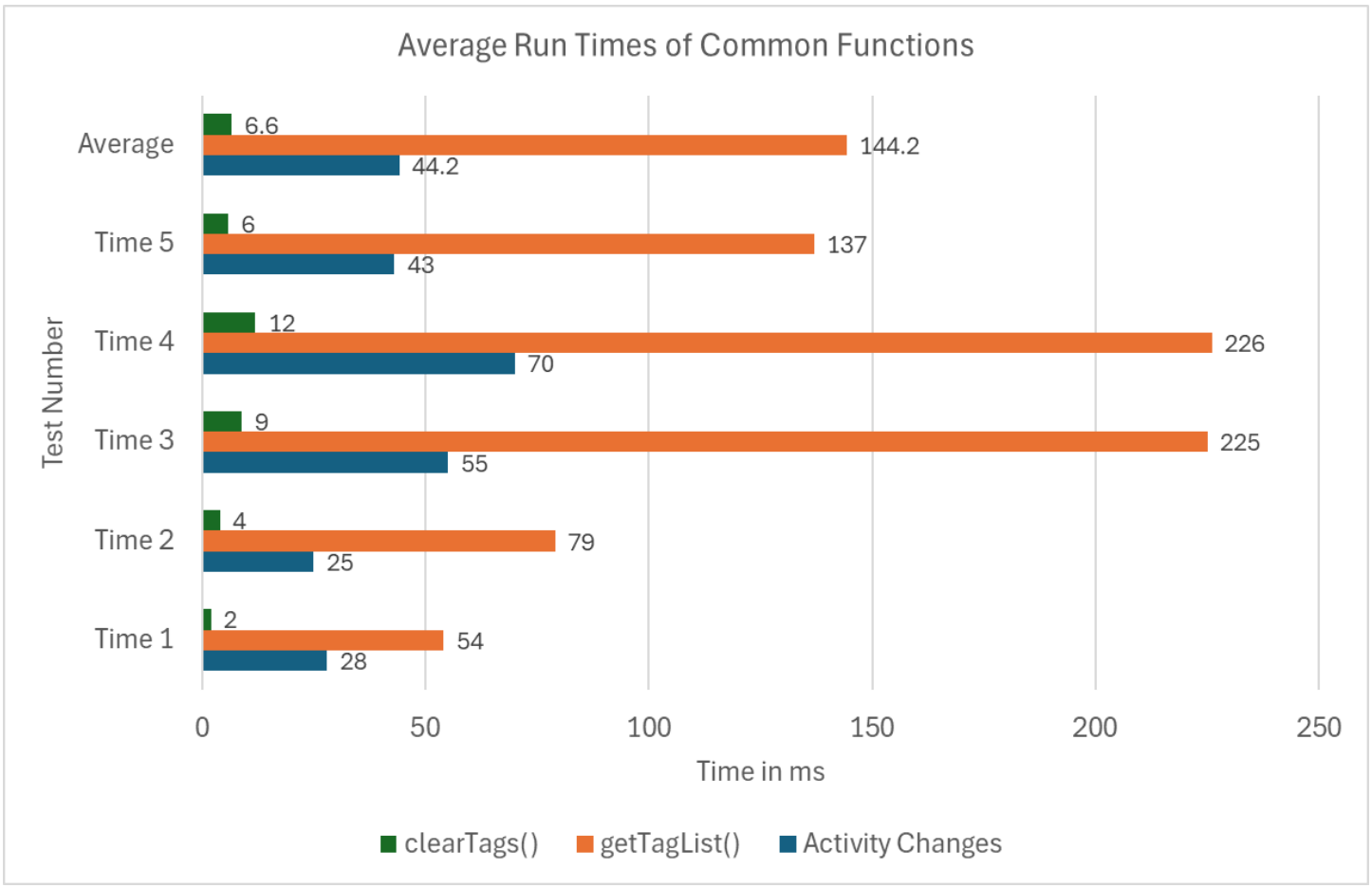


Figure 20: Alien P.R.O.B.E Performance Tests

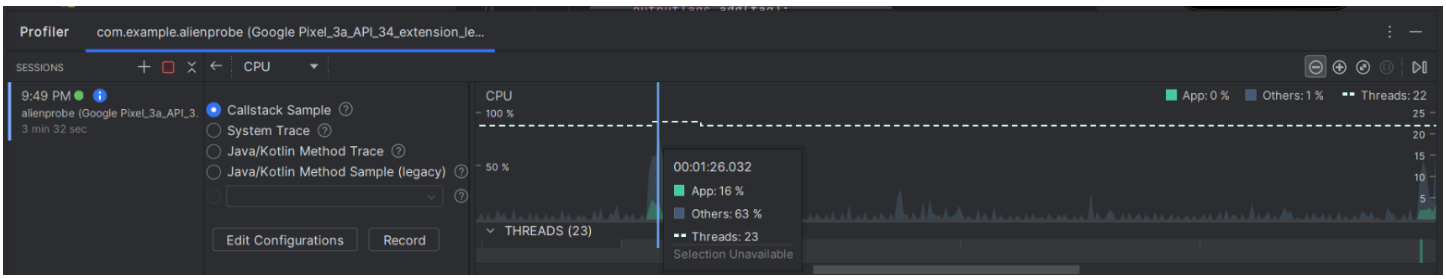


Figure 21: Alien P.R.O.B.E CPU Bottleneck Testing

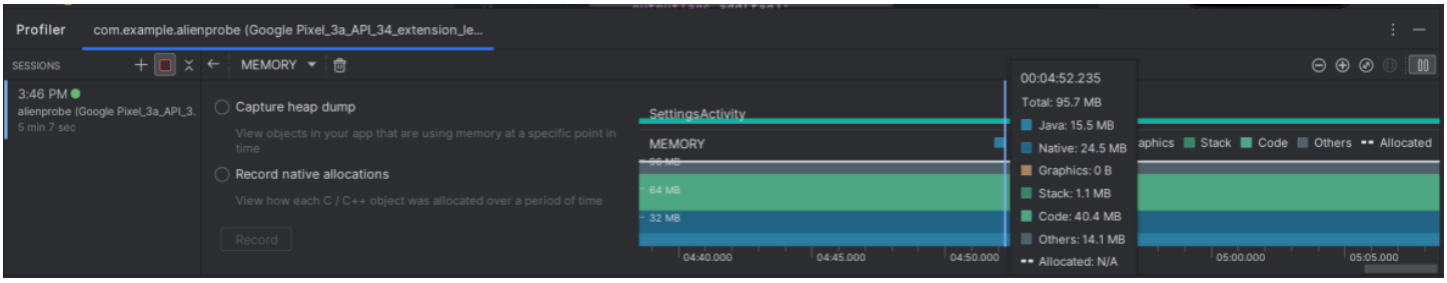


Figure 22: Alien P.R.O.B.E Memory Bottleneck Testing

**Test item:**
**Features to test/not to test:**
**Approach:**
**Test deliverables:**

**Item pass/fail criteria:**
**Environmental needs:**
**Responsibilities:**
**Staffing and training needs:**
**Schedule:**
**Risks and Mitigation:**
**Approvals:**

## 4.2 Unit Testing

**Unit Tests List**

| Unit Test | Goal |
|---|---|
| AlienScanner.Java | Tests to determine accuracy of obtaining data from Alien F800. |
| SetupActivity.kt | Tests to determine if settings are properly stored for future setup. |
| Overall UI Functionality | Tests to determine if UI is functional. |

Table 4: Unit Test Results

**Test Plan Identifier:** PROBE-UNIT01

**Introduction:** PROBE-UNIT01's objective was to test all portions of the network connection between the reader and another device using an internet connection to ensure a good connection quality and minimal issues when obtaining EPC from RFID tags.

**Test item:** The SUT for this unit test had to deal with the AlienScanner.java class file. This class file contained classes from the Alien F800 Java SDK, more specifically, the com.alien.enterpriseRFID.reader.AlienClass1Reader class.

**Features to test/not to test:** The features that PROBE-UNIT01's tested were to obtain a clean taglist from the Alien F800 reader. This was of very high priority, because we needed this data to later on be able to pull data from a database that pertains to that specific tag item.

**Approach:** Our approach to this test was to first test this snippet of code using a temporary GUI for initial testing. Once we ensured this code was valid, we moved on to integrating it into our Alien P.R.O.B.E mobile application. Once this integration was complete, we tested this inside of our app.

**Test deliverables:** The approach for both of these unit test cases were to first implement the code in both systems, the GUI then the mobile app. Once doing this we first ensured we could get output from the reader and display it inside of the log. We then moved onto displaying this information inside of our Android mobile application.

**Item pass/fail criteria:** To pass this test, we needed to be able to see a list of EPC code that the Alien F800 obtained and sent over a network through our AlienScanner.java code. The fail criteria for this would be if there would be no output or lack of a connection with the device.

**Environmental needs:** To test our code, we first accomplished this over an IDE with a basic implementation with a Java file while using the Alien F800 SDK to be able to obtain data with the device. After obtaining the correct data in the log, we moved on to implementing this into the Alien P.R.O.B.E mobile application.

**Responsibilities:** The team worked together to write the code and ensured it worked properly with the device.

**Staffing and training needs:** To accomplish this testing, we needed to be familiar with Java and the Alien F800 device. Linking the SDK with the Java code was also essential to learn before testing could be completed.

**Schedule:** Unit Tests were performed throughout the third week of Sprint 4 to ensure we have the information and everything is working as intended.

**Risks and Mitigation:** There were minimal risks in testing this particular code due to our team closely following the provided coding manual and SDK.

**Approvals:** This unit testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/19/24.

**Test Plan Identifier:** PROBE-UNIT02

**Introduction:** PROBE-UNIT02's objective was to test for the SetupActivity.kt page to ensure proper settings are stored when testing the device. This was important so that we could properly connect with the right internet connection, such as the correct IP and port that is associated with the Alien F800 device.

**Test item:** The software under test for this test was the Alien P.R.O.B.E mobile application with a focus on the SettingsActivity.kt file and setup.xml file used for the end user display. This file stores preferences inside of the share preferences of the device.

**Features to test/not to test:** To test this device, we needed to focus on focus on being able to store data to the shared preferences using the SettingsActivity.kt file. To accomplish this, we also needed an intuitive UI that correctly allowed the user to store data through the setting.xml page.

**Approach:** To test PROBE-UNIT02, we first had to understand how shared preferences work inside of Android Studio. We then had to integrate the proper UI with out setting.xml and correct code implementation inside of the SettingsActivity.kt file. After completing this, we moved onto testing inside of Android Studio to ensure our implementation was correct.

**Test deliverables:** The approach for this was to complete the UI and the SettingsActivity.kt first and then move toward testing the application through the Android Studio emulator and on the team's available Android Devices.

**Item pass/fail criteria:** To pass this test, the settings page would need to have 4 fields that save the credentials and internet connection details to connect with the device. After this, the save button would need to save those fields to the preferences. If this was a fail, the preferences would not be saved properly.

**Environmental needs:** To test properly, our team utilized the emulator inside of Android Studio and an android device to further test this.

**Responsibilities:** The team worked together to write the code and ensured it worked properly with the device.

**Staffing and training needs:** The team needed to properly understand how activities and preferences work inside of the Android environment. We also needed to be able to utilize fields and buttons to properly save fields inside of the shared preferences within the app.

**Schedule:** Unit Tests were performed throughout the third week of Sprint 4 to ensure we have the information and everything is working as intended.

**Risks and Mitigation:** Since we were first testing our code within the emulator, there were little to no risks in testing our code. After testing within the emulator, we used our android devices to further test to see if the preferences were saved properly.

**Approvals:** This unit testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/19/24.

**Test Plan Identifier:** PROBE-UNIT03

**Introduction:** This unit test covers the user experience of the Alien P.R.O.B.E. Android application. The user experience is a key functionality of the app and it needs to work smoothly and efficiently for the app to provide a good experience as a whole. We will delve deeper into the testing of the below.

**Test item:** The UI consist of xml files associated with the corresponding activities. These need to be legible and well put together as to allow for future developers to change and modify if not add content. The intent that is passed between these different activities also has to provide the correct context between activities to allow the UI to be formed and inflated properly.

**Features to test/not to test:** Testing the xml within the unit test consisted of continually going back and forth the different activities with changing values to insure that the interface could handle anything we threw at it. Further testing includes the intent and context passed between activities to ensure no data leak or missing data is detected, as well as determining what data to place where on the screen when a new activity is loaded.

**Approach:** We will implement methods to text this. Android Studio provides testing tools to automate different tests. Here we will have it test the capabilities of the app by changeling content and context to ensure the UI is well functioning.

**Test deliverables:** The deliverables will be in the form of a table. This table will contain a pass or fail criteria based on the given test created. For example there will be rows of the table containing the test such as 50 contextual switches and or changes to the input content. This will be tested as pass and fail to insure there is no faltering in the UI.

**Item pass/fail criteria:** Each unit test must be able to receive both valid and invalid inputs, and properly handle both to receive the expected outputs.

**Environmental needs:** The testing is setup in Android Studio with access to local databases and a connection to our Alien RFID Reader. We also setup a constant non changing xml files for a consistent test.

**Responsibilities:** The team all worked to implement the xml and the test will be conducted together to thoroughly determine the best use case for our user experience.

**Staffing and training needs:** No staffing and training needs are required.

**Schedule:** Unit Tests were performed throughout the third week of Sprint 4 to ensure we have the information and everything is working as intended.

**Risks and Mitigation:** There was no particular risks involved given the testing, as we had the original code in case the writing code for testing introduced issues, and the only requirement was the time required to write and execute the tests.

**Approvals:** This unit testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/19/24.

### 4.2.1 Source Code Coverage Tests

The source code coverage tests involve calculating cyclomatic complexity, which helps in understanding the complexity of the program's decision structure and the number of linearly independent paths through the program. For the unit AlienScanner.java, which interacts directly with the Alien F800 RFID Reader, the cyclomatic complexity represents the complexity in handling RFID data retrieval, including error handling and response to different RFID tag states. The SetupActivity.kt is focused on user settings and preferences storage, with complexity arising from the configurations and user inputs that can be stored. For the overall UI functionality, the complexity involves the interaction between different UI components and the transitions between activities.

### 4.2.2 Unit Tests and Results

**Unit Tests and Results:**

| **Method** | **Pass** | **Fail** |
|---|---|---|
| AlienScanner.Java | X | |
| SetupActivity | X | |
| Overall UI Functionality | X | |

Table 5: Unit Test Results

## 4.3 Integration Testing

**Test Plan Identifier:** PROBE-INT01

**Introduction:** This test checks the interaction between the software and the external factors in the project with a focus on all 3 of our unit tests in the previous section(PROBE-UNIT01, PROBE-UNIT02, & PROBE-UNIT03).

**Test item:** The software under test was the UI of Alien P.R.O.B.E along with the ScannerAcitivity.kt and AlienScanner.java files. We wanted this test to test all UI functionality to ensure that gathering tags based on the SetupActivity.kt and AlienScanner.java was working properly.

**Features to test/not to test:** The features to test were the above-listed files from the unit tests to ensure obtaining tags based on user preferences was correct all while having proper UI that functions well.

**Approach:** To test this, we created activities and XML files during the project cycle of the project. Once we needed to test UI functions, we first did it inside of the Android studio emulator and then moved onto testing on available mobile devices. To test the 2 activities within the app, we tested that settings were properly saved to shared preferences and the app was able to properly obtain a tag list from the reader based on those shared preferences.

**Test deliverables:** To test this we tested through the Android Studio emulator and with our available devices and provided a table below to show a pass or fail from our testing.

**Item pass/fail criteria:** The pass criteria was if the UI functioned well within our device all while being able to properly store shared preferences and obtain a tag list with a fully functional UI. If the UI was slow, buggy, or made the app crash, then it would be considered a failure. If the tag list or shared preferences were not set properly then this would also be considered a fail.

**Environmental needs:** We set up the RFID Reader to constantly scan and send tags, this way when we connect the Reader to the app we see if the tags are correctly showing up. We then utilized Android Studio and our Android mobile devices to properly test all of these features.

**Responsibilities:** Nicholas Johnson set up the RFID Reader and Antenna in his room to constantly scan for tags while on the WKU network, and the team developed the internal database and connection method ahead of time, with no new system being needed for testing purposes, just stress-testing the existing infrastructure.

**Staffing and training needs:** These connections are a core part of the app, and as such knowing how to use the regular app will allow for users to perform these tests, although a further understanding of RFID technology may be needed to troubleshoot if problems arise during testing.

**Schedule:** As the Integration tests are core to the app, we have been running the connection to the RFID Reader since we implemented the technology in Sprint 2. We did perform more rigorous testing during Sprint 4 to be sure there were no problems with the preferences, obtaining the taglist, and overall UI.

**Risks and Mitigation:** This test posed little risks to us as with the integration there are little risks present. The GitHub repository also contains all of our previous pushes and code. This results in a very easy mitigation if a risk or problem were to arise. We simply can go back to the git page and pull a previous version of our code that works, or simply reference it to find a solution to our problem.

**Approvals:** This integration testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/19/24.

#### 4.3.1 Integration Tests and Results

| Test Case | Pass | Fail |
|---|---|---|
| PROBE-UNIT01 & PROBE-UNIT02 | X | |
| PROBE-UNIT01 & PROBE-UNIT03 | X | |
| PROBE-UNIT02 & PROBE-UNIT03 | X | |

Table 6: Integration Testing Results

### 4.4 System Testing

**Test Plan Identifier:** PROBE-SYS01

**Introduction:** The system testing was performed on the completed software of the app to find any potential issues or bugs. This include going through all of the functions of the app and making certain that all the aspects of the app interact and working fluently together.

**Test item:** We tested the entire Alien P.R.O.B.E software system including all of the functions and connections it has. This includes the scanning the database and the UI functions.

**Features to test/not to test:** We tested the functionality of the entire experience of the app by going running through how a user would interact with the app. Going through each page of the app and using all of the available functions to find any issues. There were no particular exclusions as the purpose was to test all functions in a session of the app. We thoroughly tested edge cases as well.

**Approach:** The app was tested on both the Android Studio emulator and on the physical Android device we had. The testing would open the app, go through the About page, the setup page, and the scanning page, making sure all aspects of each page were working as intended as well as the transitions between these states were working.

**Test deliverables:** The deliverable for the system tests were a table in which there each tested criteria either gets a check mark that it passed or failed.

**Item pass/fail criteria:** The criteria changes between the different items, with the main checks being that the buttons that change scenes worked quickly and correctly, that the scanning was correct in it's reads and was

receiving the scans quickly, and checking that the setting up of the network connection to the RFID Reader did not result in any errors.

**Environmental needs:** As this was a comprehensive testing phase, we used the standard methods of running the app with no special requirements necessary. The RFID Reader was continually scanning tags to ensure we could properly test the communication between the app and the reader, and the app in the emulator and on the actual Android were on the WKU network to test connectivity to the database.

**Responsibilities:** The roles were split between the entire team as creating the app was a multi-person process, in testing every member tested the app, with Nicholas Johnson having access to the Android phone with the app installed to be able to perform a more realistic test case.

**Staffing and training needs:** As the test is testing a standard usage of the app as long as the user knows how the app works and how to run it there should be no additional training required.

**Schedule:** We performed tests covering the entirety of the app multiple times as we developed the project in all Sprints, with the most intensive and complete tests being done throughout Sprint 4, this was because not only was this the sprint we focused on testing, but also the sprint where the Alien P.R.O.B.E. app had the most features implemented that we could test.

**Risks and Mitigation:** No risks were involved given that it was a testing of the main features of the app, with the only investment being the time used to perform these tests.

**Approvals:** This system testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/20/24.

### 4.4.1 System Tests and Results

| Test Case | Pass | Fail |
|---|---|---|
| App opens successfully | X | |
| About Page opens | X | |
| Setup Page opens | X | |
| Setup Page saves info | X | |
| Scan Page opens | X | |
| Scan Page receives scans | X | |
| Tag List opens | X | |
| Tag List displays tag info | X | |
| Tag List can be cleared | X | |

Table 7: System Testing Results

## 4.5 Acceptance Testing

**Test Plan Identifier:** PROBE-ACC01

**Introduction:** Our acceptance testing was to ensure that the Alien P.R.O.B.E. project met all the laid out requirements properly, as well as being easy to use and understand for our target user base.

**Test item:** The tests involved the entirety of the app, and cross-referencing the experience with both the list of requirements we created as well as the expectations required for a new user and how reasonable the app is to navigate in this sense.

**Features to test/not to test:** We tested the app for the laid-out software requirements first, and then tested from a user acceptance perspective, specifically on the ease-of-use of the app. This phase of testing didn't involve testing the back-end of the software that didn't relate to the requirements of end user experience.

**Approach:** The testing for the requirements went through all functions of the app and checking whether or not within those functions all the requirements that we set out to accomplish for our client were met. For the user experience, involved going through each of the app's scenes, through the perspective of someone with a minimal understanding on how the app functions and how to properly use it. This involved close examination of the About page's tutorial, as well as how the UI is laid out.

**Test deliverables:** The deliverable for this tests was a table containing all of our laid out requirements and whether or not they were met within the app.

**Item pass/fail criteria:** The requirements section of testing had a simple pass/fail criteria as it is a binary answer whether or not the app in it's current form met each one of the requirements. By going down the list of requirements one-by-one and checking it's equivalence in the app we are able to either pass or fail each requirement. For the user experience, it is more open-ended, with it relying on our perspective of how understandable and usable our app is. If there was a part of the app that felt unexplained or obtuse then that would fail that portion of the testing.

**Environmental needs:** The testing required minimum resources, we only needed to be able to run the app. Given we wanted multiple perspectives to ensure the app was immediately understandable we did all work on the testing which did use some time, but that was the only cost of the testing.

**Responsibilities:** All members contributed to testing the app, both on their own through the Android Studio emulator and when we met together we were able to all test the user experience on the actual Android as an app.

**Staffing and training needs:** As the testing was performed to make the app easily usable by someone with little knowledge of the app, minimal training is required to do this portion. For testing whether the app meets all the requirements, the tester would need to have the list of requirements as well as have an understanding of what all the requirements mean, which may require some degree of knowledge of databases and RFID technology.

**Schedule:** While we did continuously consider how the app met the requirements that were laid out throughout the project, we did intensify these checks during Sprint 4 in these tests to ensure nothing was overlooked. As for the user experience, it was not as much a focus earlier on in the project as we were more focused on getting the base app working. In Sprint 4 however we were able to test with this user-base in mind.

**Risks and Mitigation:** The only risks involved in this testing phase was that we would need to make changes to the app either in order to meet a unmet requirement or in order to make the app more understandable to users, which does require a time commitment that we were prepared for as our mitigation as the concern of this time loss.

**Approvals:** This acceptance testing plan was approved by Nicholas Johnson, Cy Dixon, Alex Godsey, and Michael Lynch on 4/19/24.

### 4.5.1 Acceptance Tests and Results

| Requirement | Pass | Fail |
|---|---|---|
| Alien F800 RFID Reader scans parking tags using the Alien ALR-8698 RFID Antenna | X | |
| Phone is able to connect to Alien F800 RFID Reader through Bluetooth | | X |
| Implementation of geolocation tracking for scanned tags in the mobile app | X | |
| App is able to retrieve and display USER ID from scanned tags using the database | X | |
| Data synchronization is ensured between the mobile app and RFID Reader | X | |
| The User can move through the mobile app and between different menus | X | |
| App works on Android devices using Android OS 11.0 or later | X | |
| App allows for multiple devices to work simultaneously | X | |
| Secures scanned information using encryption to prevent data leaks | X | |
| Code will be well documented | X | |
| Mobile app is user-friendly and requires minimal training | X | |

Table 8: Acceptance Testing Results

In the end the Bluetooth functionality didn't work out as it was not a native option for the Alien Reader as we first thought.

# 5 Conclusion

In conclusion, there were many purposes of the technical document. One main purpose of writing it was that it keeps track of our goals for the project and the goals we were trying to reach. Having it ready allowed us as a team to have another document that tracks our progress. When we had meetings to discuss and write the technical document, we also had to discuss the future of the project. Writing this document allowed us to get more practice

for writing technical documents before we graduate and move on to professional practice. This semester we had to remember everything we learned from our previous classes and then apply it in this class with a completely new project.

There were some shortcomings during our work on our project. In the end we could not get Bluetooth functionality working. This was due to the Alien Reader not having native Bluetooth functionality. This was an oversight for us and unfortunately we were unable to add the functionality given our time scope. We recommend any future work on this aspect of the project be put into the effort of adding a Bluetooth adapter device and creating an interface API for the reader through that adapter.

The future work for our project rests entirely on how the WKU Parking Transportation Services decides to use our software. They can decide to use bits and pieces of it for other projects they work on. They can also decide to continue on our work and continue developing the app and adding more functions to it. We left our code very organized with many comments explaining what everything does so if WKU Parking and Transportation Services decides to continue the work, they can pick up where we left off much easier.

# 6 Documentation Testing

## 6.1 Consistency Checklist for Documentation

| Section | Consistent | Inconsistent |
|---|---|---|
| 1.1 Project Overview | X | |
| 1.2 Project Scope | X | |
| 1.3.1 Functional Requirements | X | |
| 1.3.2 Non-Functional Requirements | X | |
| 1.4 Target Hardware Details | X | |
| 1.5 Software Product Development | X | |
| 2.1.1 Physical System Boundaries | X | |
| 2.1.2 Logical System Boundaries | X | |
| 2.3.1 Updated UML Class Diagrams (3 OO Design Patterns w/documentation) | X | |
| 2.3.2 Updated UML Use Case Diagrams (w/documentation) | X | |
| 2.3.3 Updated UML Use Case Scenarios (w/documentation) | X | |
| 2.3.4 Updated UML Sequence Diagrams (w/documentation) | X | |
| 2.3.5 Updated UML State Diagrams (w/documentation) | X | |
| 2.3.6 Updated UML Component Diagram (w/documentation) | X | |
| 2.3.7 Updated UML Deployment Diagram (w/documentation) | X | |
| 2.4 Usage of Micro or Macro Software Version Control | X | |
| 2.5 Data Dictionary | X | |
| 2.6 User Experience/Game Design | X | |
| 3.1 Product Security | X | |
| 3.2 Product Performance | X | |
| 4 Software Testing | X | |
| 4.1 Completed Software Testing Checklist/Testing Plan(s) | X | |
| 4.2 Defined and executed Unit Tests and Results (With Testing Plan) | X | |
| 4.2 Code coverage tests using Flow Graphs to generate test cases | X | |
| 4.3 Defined and executed Integration Tests and Results (With Testing Plan) | X | |
| 4.4 Defined and Executed System Tests and Results(With Testing Plan) | X | |
| 4.5 Defined and Executed Validation, Verification, and Acceptance Tests and Results | X | |
| 5 Concluding Remarks/Future work | X | |
| 6 Documentation Testing | X | |
| 6.1 Consistency checklist for all documentation sections | X | |
| 6.2 Checklist that all required sections are complete | X | |
| 7 Appendix (source code with comments, project build instructions) | X | |
| 7.1 Software Product Build Instructions | X | |
| 7.2 Software Product User Guide | X | |
| 7.3 Source Code with comments | X | |

Table 9: Project Section Consistency Checklist

### 6.2 Completion Checklist for Documentation

| Section | Complete | Incomplete |
|---|---|---|
| 1.1 Project Overview | X | |
| 1.2 Project Scope | X | |
| 1.3.1 Functional Requirements | X | |
| 1.3.2 Non-Functional Requirements | X | |
| 1.4 Target Hardware Details | X | |
| 1.5 Software Product Development | X | |
| 2.1.1 Physical System Boundaries | X | |
| 2.1.2 Logical System Boundaries | X | |
| 2.3.1 Updated UML Class Diagrams (3 OO Design Patterns w/documentation) | X | |
| 2.3.2 Updated UML Use Case Diagrams (w/documentation) | X | |
| 2.3.3 Updated UML Use Case Scenarios (w/documentation) | X | |
| 2.3.4 Updated UML Sequence Diagrams (w/documentation) | X | |
| 2.3.5 Updated UML State Diagrams (w/documentation) | X | |
| 2.3.6 Updated UML Component Diagram (w/documentation) | X | |
| 2.3.7 Updated UML Deployment Diagram (w/documentation) | X | |
| 2.4 Usage of Micro or Macro Software Version Control | X | |
| 2.5 Data Dictionary | X | |
| 2.6 User Experience/Game Design | X | |
| 3.1 Product Security | X | |
| 3.2 Product Performance | X | |
| 4 Software Testing | X | |
| 4.1 Completed Software Testing Checklist/Testing Plan(s) | X | |
| 4.2 Defined and executed Unit Tests and Results (With Testing Plan) | X | |
| 4.2 Code coverage tests using Flow Graphs to generate test cases | X | |
| 4.3 Defined and executed Integration Tests and Results (With Testing Plan) | X | |
| 4.4 Defined and Executed System Tests and Results(With Testing Plan) | X | |
| 4.5 Defined and Executed Validation, Verification, and Acceptance Tests and Results | X | |
| 5 Concluding Remarks/Future work | X | |
| 6 Documentation Testing | X | |
| 6.1 Consistency checklist for all documentation sections | X | |
| 6.2 Checklist that all required sections are complete | X | |
| 7 Appendix (source code with comments, project build instructions) | X | |
| 7.1 Software Product Build Instructions | X | |
| 7.2 Software Product User Guide | X | |
| 7.3 Source Code with comments | X | |

Table 10: Project Section Completion Checklist

# 7 Appendix

### 7.1 Software Product Build Instructions

With the project file, download the Alien P.R.O.B.E application. After the file is downloaded, open up the project in Android Studio. From there, any person continuing the development now has access to the project.

### 7.2 Software Product User Guide

The user, who would be a WKU Parking Transportation Service employee, would first launch the app. They can either go to the about view to learn more about how the app function or they can go straight to settings and

set up the app for use. In the settings view, they would write the IP, port number, username, and password for the Alien Reader which they will have access to from working for the PTS. After that they can navigate back to the home screen and move onto the scanning view. On the scanning view they can manually scan once or set the toggle to "on" to have it continuously scan while they drive around one of the lots. The employee can then go into the database and view the information of the scanned tags and where the tags first got scanned while driving around. If there are any tags that scanned that had errors that made it into the database then the user can click on that tag and delete it from the database. If there are too many tags in the main scan screen, the user can clear them off and have a fresh screen to work with.

## 7.3 Source Code with Comments

### 7.3.1 C.1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.NEARBY_WIFI_DEVICES" /> <!--
       Permissions for Bluetooth -->
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />

    <!-- Declare Bluetooth feature -->
    <uses-feature
        android:name="android.hardware.bluetooth"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/play_store_512"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/play_store_512"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.DayNight"
        tools:targetApi="30">
        <activity
            android:name=".presentation.ViewTagsActivity"
            android:exported="false" />
        <activity
            android:name=".presentation.AboutActivity"
            android:exported="false" />

        <profileable android:shell="true" />

        <activity
            android:name=".presentation.ScannerActivity"
            android:exported="true" />
        <activity
            android:name=".presentation.SetupActivity"
```

```
            android:exported="true" />
        <activity
            android:name=".presentation.MainActivity"
            android:exported="true"
            android:theme="@style/Theme.AppCompat.DayNight">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="preloaded_fonts"
            android:resource="@array/preloaded_fonts" />
    </application>

</manifest>
```

### 7.3.2 C.2 ApiService.java

```
package com.example.alienprobe.api;

import com.example.alienprobe.java.Vehicle;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface ApiService {
    //Get data from DB for vehicle permit information - Vehicle info & permitID
    @GET("Lookup/GetVehicleListByPermit")
    Call<List<Vehicle>> getVehicleListByPermit(@Query("PermitNumber") int permitId);
}
```

### 7.3.3 C.3 fetcherVehicles.kt

```
package com.example.alienprobe.api

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import com.example.alienprobe.java.Vehicle
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

fun fetchVehicles(permitId: Int?): LiveData<List<Vehicle>> {
    val vehiclesLiveData = MutableLiveData<List<Vehicle>>()
    if (permitId != null) {
        //obtain permit info from RetrofitClient API call
        RetrofitClient.getApiService().getVehicleListByPermit(permitId).enqueue(object
            :
            Callback<List<Vehicle>> {
            override fun onResponse(call: Call<List<Vehicle>>, response: Response<List
                <Vehicle>>) {
```

```
                if (response.isSuccessful) {
                    response.body()?.let { vehicles ->
                        Log.d("Vehicle fetched: ", vehicles[0].make + " " + vehicles
                            [0].model + " " + vehicles[0].plate);
                        // Update LiveData with the fetched vehicles
                        vehiclesLiveData.value = vehicles
                    }
                } else {
                    Log.d("Error", "Failed to fetch vehicles: ${response.errorBody()?.
                        string()}")
                    vehiclesLiveData.value = emptyList()
                }
            }

            override fun onFailure(call: Call<List<Vehicle>>, t: Throwable) {
                Log.d("Error", "Error fetching vehicles: ${t.message}")
                vehiclesLiveData.value = emptyList()
            }
        })
    }
    return vehiclesLiveData
}
```

### 7.3.4 C.4 RetrofitClient.java

```
package com.example.alienprobe.api;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitClient {
    private static final String BASE_URL = "(Change this to the URL API you are using)
        ";

    private static final Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

    public static ApiService getApiService() {
        return retrofit.create(ApiService.class);
    }
}
```

### 7.3.5 C.5 DataBaseHelper.java

```
package com.example.alienprobe.database;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteConstraintException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.Cursor;

import java.util.ArrayList;
import java.util.List;
```

```
12
13  import androidx.annotation.Nullable;
14
15  public class DataBaseHelper extends SQLiteOpenHelper {
16      public static final String RFIDTAG_TABLE = "RFIDTAG_TABLE";
17
18      //DB Fields
19      public static final String COLUMN_ID = "ID";
20      public static final String COLUMN_EPC_STRING = "EPC_STRING";
21      public static final String COLUMN_LAT_DOUBLE = "LATITUDE";
22      public static final String COLUMN_LONG_DOUBLE = "LONGITUDE";
23      public static final String COLUMN_TIME = "TIME";
24      public static final String COLUMN_VEHICLE = "VEHICLE";
25      public DataBaseHelper(@Nullable Context context) {
26          super(context, "RF" + COLUMN_ID + "Tag.db", null, 1);
27      }
28
29      //Create SQLite DB for holding RFIDTag info data
30      @Override
31      public void onCreate(SQLiteDatabase db) {
32          String createTableStatement = "CREATE TABLE " + RFIDTAG_TABLE + " (" +
33                  COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
34                  COLUMN_EPC_STRING + " TEXT NOT NULL UNIQUE, " +
35                  COLUMN_LONG_DOUBLE + " DOUBLE NOT NULL, " +
36                  COLUMN_LAT_DOUBLE + " DOUBLE NOT NULL, " +
37                  COLUMN_TIME + " TEXT NOT NULL, " +
38                  COLUMN_VEHICLE + " TEXT NOT NULL)";
39          db.execSQL(createTableStatement);
40      }
41
42      //updates DB
43      @Override
44      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}
45
46      //Add TagModel to DB (This is info from the ScannerActivity.kt file)
47      public boolean addOne(TagModel tag) {
48          SQLiteDatabase db = this.getWritableDatabase();
49          ContentValues cv = new ContentValues();
50
51          cv.put(COLUMN_EPC_STRING, tag.getEPC());
52          cv.put(COLUMN_LAT_DOUBLE, tag.getLongitude());
53          cv.put(COLUMN_LONG_DOUBLE, tag.getLatitude());
54          cv.put(COLUMN_TIME, tag.getTime());
55          cv.put(COLUMN_VEHICLE, tag.getVehicle().toString());
56
57          try {
58              // insertOrThrow() will throw SQLiteConstraintException if a UNIQUE
                    constraint is violated
59              db.insertOrThrow(RFIDTAG_TABLE, null, cv);
60              return true;
61          } catch (SQLiteConstraintException e) {
62              // You could log or handle the specific SQLiteConstraintException here if
                    needed
63              return false;
64          } finally {
65              db.close();
66          }
67      }
68
69      //Tag delete occurs inside of TagAdapter view
```

```
70      public boolean deleteTag(String tagId) {
71          SQLiteDatabase db = this.getWritableDatabase();
72          return db.delete(RFIDTAG_TABLE, "id = ?", new String[]{tagId}) > 0;
73      }
74
75      //Used inside of ViewTagsActivity.kt
76      public List<TagModel> getAllTags() {
77          List<TagModel> returnList = new ArrayList<>();
78
79          // Get data from the database
80          String queryString = "SELECT * FROM " + RFIDTAG_TABLE;
81          SQLiteDatabase db = this.getReadableDatabase();
82          Cursor cursor = db.rawQuery(queryString, null);
83
84          int i = 0;
85          if (cursor.moveToFirst()) {
86              do {
87                  int tagID = cursor.getInt(0);
88                  String epcString = cursor.getString(1);
89                  double longitude = cursor.getDouble(2);
90                  double latitude = cursor.getDouble(3);
91                  String time = cursor.getString(4);
92                  String vehicle = cursor.getString(5);
93
94                  TagModel tmpTag = new TagModel(tagID, epcString, longitude, latitude,
                        time, vehicle);
95                  System.out.println("New tag create with id: " + tmpTag.getId() + " epc
                        : " + tmpTag.getEPC());
96
97                  returnList.add(i,tmpTag);
98                  System.out.println(returnList);
99                  i++;
100             } while (cursor.moveToNext());
101         }
102
103         cursor.close();
104         db.close();
105         System.out.println(returnList);
106         return returnList;
107     }
108 }
```

### 7.3.6 C.6 TagModel.java

```
1   //Tag DB Model
2   package com.example.alienprobe.database;
3
4   import androidx.annotation.NonNull;
5   import com.example.alienprobe.*;
6
7   public class TagModel {
8       private int id;
9       private String epc;
10      private String time;
11      private final double longitude;
12      private final double latitude;
13      private final String vehicle;
14
```

```
    public TagModel(int id, String epc, double longitude, double latitude, String time
        , String vehicle) {
        this.id = id;
        this.epc = epc;
        this.longitude = longitude;
        this.latitude = latitude;
        this.time = time;
        this.vehicle = vehicle;
    }
    @NonNull
    @Override
    public String toString() {
        return "tagModel{" +
                "id=" + id +
                ", epc='" + epc + '\'' +
                ", longitude=" + longitude +
                ", latitude=" + latitude +
                ", time=" + time +
                ", vehicle=" + vehicle +
                '}';
    }
    public int getId() {
        return this.id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public void setTime(String time) { this.time = time; }
    public String getTime() {return this.time; }
    public String getEPC() {
        return this.epc;
    }
    public void setEpc(String epc) {
        this.epc = epc;
    }
    public double getLatitude() { return this.latitude; }
    public double getLongitude() { return this.longitude; }
    public String getVehicle() { return this.vehicle; }
}
```

### 7.3.7 C.7 AlienScanner.java

```
package com.example.alienprobe.java;

import android.content.Context;
import android.content.SharedPreferences;

import com.alien.enterpriseRFID.reader.AlienClass1Reader;

import java.net.Socket;
import java.util.*;
import java.util.stream.Collectors;

//PASS CONTEXT WHEN CREATING THIS BY USING 'this'
public class AlienScanner {
    public static String readerIP;
    public static Integer readerPort;
    public static String readerUserName;
    public static String readerPassword;
```

```
18      public static AlienClass1Reader reader = new AlienClass1Reader();
19
20      public AlienScanner(Context context) {
21          loadPreferences(context);
22      }
23      // openReader and closeReader are custom for testing
24      public void openReader(){
25          try {
26              reader.setConnection(readerIP, readerPort);
27              reader.setUsername(readerUserName);
28              reader.setPassword(readerPassword);
29              reader.open();
30              System.out.println("Connection established with RFID reader.");
31          } catch (Exception e) {
32              System.out.println("error");
33          }
34      }
35      public void closeReader(){
36          reader.close();
37          System.out.println("Connection Closed.");
38      }
39      public List<RFIDTag> GetTagList() {
40          List<RFIDTag> outputTags = new ArrayList<>();
41          new Thread(() -> {
42              try {
43                  Socket socket = new Socket(readerIP, readerPort);
44                  reader.setUsername(readerUserName);
45                  reader.setPassword(readerPassword);
46
47                  if (reader.isValidateOpen()) {
48                      reader.open();
49                      System.out.println("connection opened");
50
51                      Thread.sleep(100);
52
53                      String commandOutput = reader.doReaderCommand("t");
54
55                      System.out.println(commandOutput);
56                      List<String> outputLines = Arrays.stream(commandOutput.split("\\r
                            ?\\n"))
57                              .collect(Collectors.toList());
58
59                      for (String line : outputLines) {
60                          // Assuming each line represents an RFID tag
61                          RFIDTag tag = new RFIDTag(line);
62
63                          outputTags.add(tag);
64                      }
65                      reader.close();
66                      System.out.println("connection closed");
67                      socket.close();
68                  }
69
70              } catch (Exception e) {
71                  System.out.println(e);
72                  e.printStackTrace();
73              }
74          }).start();
75          return outputTags;
76      }
```

```
    private void loadPreferences(Context context) {
        SharedPreferences sharedPreferences = context.getSharedPreferences("
            AppPreferences", Context.MODE_PRIVATE);

        readerIP = sharedPreferences.getString("IP", "DefaultIP");
        readerPort = sharedPreferences.getInt("Port", 23); // Assuming default port 23
        readerUserName = sharedPreferences.getString("Username", "DefaultUsername");
        readerPassword = sharedPreferences.getString("Password", "DefaultPassword");

        reader.setConnection(readerIP, readerPort);
        reader.setUsername(readerUserName);
        reader.setPassword(readerPassword);
    }
}
```

### 7.3.8 C.8 RFIDTag.java

```
// Tag Model for ScrollView
// This has the vehicle attribute but it is not used.
// The plan was to have a JSON call to the DB and then store the data in a Vehicle
    object attached
// to an RFID tag that would be added to the local DB

package com.example.alienprobe.java;

import androidx.annotation.NonNull;

public class RFIDTag {
    private String epc;
    private Vehicle vehicle;
    public RFIDTag(String epc) {
        this.epc = epc;
    }
    @NonNull
    @Override
    public String toString() {
        return "RFIDTag{" +
                "EPC = " + this.epc;
    }
    public void setEpc(String epc) {
        this.epc = epc;
    }
    public String getEPC() {
        return this.epc;
    }
    public Vehicle getVehicle(){ return this.vehicle; }

}
```

### 7.3.9 C.9 Vehicle.java

```
package com.example.alienprobe.java;

import androidx.annotation.NonNull;

import com.google.gson.annotations.SerializedName;

```

```
public class Vehicle {
    @SerializedName("Id")
    private long id; // Using long because the Id looks quite large

    @SerializedName("Plate")
    private String plate;

    @SerializedName("Make")
    private String make;

    @SerializedName("Model")
    private String model;

    @SerializedName("Color")
    private String color;

    @NonNull
    @Override
    public String toString() {
        return "Vehicle{" +
                "id=" + id +
                ", plate='" + plate + '\'' +
                ", make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", color='" + color + '\'' +
                '}';
    }

    // Constructor
    public Vehicle(long id, String plate, String make, String model, String color) {
        this.id = id;
        this.plate = plate;
        this.make = make;
        this.model = model;
        this.color = color;
    }

    // Getters and Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getPlate() {
        return plate;
    }

    public void setPlate(String plate) {
        this.plate = plate;
    }

    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
```

```
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

### 7.3.10 C.10 AboutActivity.kt

```
package com.example.alienprobe.presentation

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.LiveData
import androidx.lifecycle.Observer
import com.example.alienprobe.api.ApiService
import com.example.alienprobe.R
import com.example.alienprobe.java.Vehicle
import com.example.alienprobe.api.fetchVehicles

class AboutActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.about)

        setupListeners()
    }
    private fun grabVehicle() {
        val id = 101234
        val vehicles: LiveData<List<Vehicle>> = fetchVehicles(id)
        vehicles.observe(this, Observer { vehicles ->
            // This block will be called every time the 'vehicles' LiveData changes.
            vehicles?.forEach { vehicle ->
                Log.d("Vehicle", "Make: ${vehicle.make}, Model: ${vehicle.model},
                    Plate: ${vehicle.plate}")
            }
        })
    }
    private fun setupListeners() {
        val backButton = findViewById<Button>(R.id.back_button)
        backButton.setOnClickListener {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
        }
```

```
    }
}
```

### 7.3.11 C.11 MainActivity.kt

```
package com.example.alienprobe.presentation

import android.Manifest
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.content.pm.PackageManager
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import com.example.alienprobe.R

class MainActivity : ComponentActivity() {
    private lateinit var permissionLauncher: ActivityResultLauncher<Array<String>>
    private lateinit var sharedPreferences: SharedPreferences
    companion object {
        const val PERMISSIONS_GRANTED_KEY = "permissions_granted"
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        obtainPrefs()
        initializePermissions()

        setupListeners()
    }
    private fun obtainPrefs() {
        sharedPreferences = getSharedPreferences("MyAppPreferences", Context.
            MODE_PRIVATE)
    }
    private fun setupListeners() {
        val scannerStartButton = findViewById<Button>(R.id.btnViewScanner)
        scannerStartButton.setOnClickListener {
            startActivity(Intent(this, ScannerActivity::class.java))
        }
        val settingsStartButton = findViewById<Button>(R.id.btnViewSetup)
        settingsStartButton.setOnClickListener {
            startActivity(Intent(this, SetupActivity::class.java))
        }
        val aboutStartButton = findViewById<Button>(R.id.aboutButton)
        aboutStartButton.setOnClickListener {
            startActivity(Intent(this, AboutActivity::class.java))
        }
    }
    private fun initializePermissions() {
        permissionLauncher = registerForActivityResult(
            ActivityResultContracts.RequestMultiplePermissions()
        ) { permissions ->
            if (permissions.values.all { it }) {
```

```
                Toast.makeText(this, "Permissions are granted.", Toast.LENGTH_SHORT).
                    show()
                sharedPreferences.edit().putBoolean(PERMISSIONS_GRANTED_KEY, true).
                    apply()
            } else {
                Toast.makeText(this, "Some permissions are not granted.", Toast.
                    LENGTH_SHORT).show()
                sharedPreferences.edit().putBoolean(PERMISSIONS_GRANTED_KEY, false).
                    apply()
            }
        }
        val requiredPermissions = mutableListOf<String>()
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.
            ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
            requiredPermissions.add(Manifest.permission.ACCESS_FINE_LOCATION)
        }
        if (requiredPermissions.isNotEmpty()) {
            permissionLauncher.launch(requiredPermissions.toTypedArray())
        }
    }
}
```

### 7.3.12 C.12 ScannerActivity.kt

```
package com.example.alienprobe.presentation

import android.Manifest
import android.content.Intent
import android.content.pm.PackageManager
import android.database.sqlite.SQLiteConstraintException
import android.icu.text.SimpleDateFormat
import android.icu.util.Calendar
import android.location.Location
import android.media.MediaPlayer
import android.net.Uri
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.provider.Settings
import android.util.Log
import android.widget.Button
import android.widget.LinearLayout
import android.widget.TextView
import android.widget.Toast
import android.widget.ToggleButton
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.appcompat.app.AlertDialog
import androidx.lifecycle.LiveData
import androidx.lifecycle.Observer
import com.example.alienprobe.java.AlienScanner
import com.example.alienprobe.database.DataBaseHelper
import com.example.alienprobe.R
import com.example.alienprobe.java.RFIDTag
import com.example.alienprobe.database.TagModel
import com.example.alienprobe.java.Vehicle
import com.example.alienprobe.api.fetchVehicles
import com.google.android.gms.location.FusedLocationProviderClient
```

```
import com.google.android.gms.location.LocationServices
import java.util.Locale

var tagList: MutableList<RFIDTag> = mutableListOf()

class ScannerActivity : AppCompatActivity() {
    companion object { private const val LOCATION_PERMISSION_REQUEST_CODE = 1 }

    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private var lastLocation: Location? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.scanner)

        fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)

        checkAndRequestLocationPermissions()

        setupUI()
    }
    private fun setupUI() {
        getLastLocation()
        //make sure to initialize this here or else scannerActivity will crash
        val reader = AlienScanner(this)
        val linearLayout = findViewById<LinearLayout>(R.id.linearLayout)
        //back button
        val buttonClick = findViewById<Button>(R.id.btnViewScanToMain)
        buttonClick.setOnClickListener {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
        }
        //view tags button
        val viewTags = findViewById<Button>(R.id.viewTagsButton)
        viewTags.setOnClickListener {
            val intent = Intent(this, ViewTagsActivity::class.java)
            startActivity(intent)
        }
        //clear button
        val clearClick = findViewById<Button>(R.id.btnScannerClear)
        clearClick.setOnClickListener {
            tagList.clear()
            linearLayout.removeAllViews()
        }

        val toggleOnOff = findViewById<ToggleButton>(R.id.toggleScanner)
        toggleOnOff.setOnCheckedChangeListener { _, isChecked ->
            if (isChecked) {
                // Start background thread when toggle is ON
                Thread {
                    while (toggleOnOff.isChecked) {
                        getLastLocation()
                        playSound()
                        val tempTagList: MutableList<RFIDTag> = reader.GetTagList()
                        // Use a handler to perform UI operations on the main thread
                        Thread.sleep(1250)
                        Handler(Looper.getMainLooper()).post {
                            checkForDuplicateTags(linearLayout, tempTagList)
                            addTagsToView(linearLayout)
                        }
```

```
                    }
                }.start()
            } else {
                // Handle what to do when toggle is OFF if needed
            }
        }
        // getList button
        val getList = findViewById<Button>(R.id.getTagListButton)
        getList.setOnClickListener {
            getLastLocation()
            playSound()
            val tempTagList: MutableList<RFIDTag> = reader.GetTagList()
            Thread.sleep(500)
            checkForDuplicateTags(linearLayout,tempTagList)
            addTagsToView(linearLayout)
        }
    }
    private fun addTagToDB(tag: RFIDTag) {
        // thread this to improve performance
        val currentTime = getCurrentTime()
        val dataBaseHelper: DataBaseHelper =
            DataBaseHelper(this)

        /* Uncomment this when needed to use api service for tag info
        val epc = tag.epc.takeLast(5).toIntOrNull()
        val vehicles: LiveData<List<Vehicle>> = fetchVehicles(epc)
        //
        var firstVehicle: Vehicle? = null
        //
        vehicles.observe(this, Observer { vehicles ->
            if (!vehicles.isNullOrEmpty()) {
                firstVehicle = vehicles.first()
            }
        })
        */

        //temp car name
        var firstVehicle = "Lightning McQueen"

        try {
            val long: Double = lastLocation!!.longitude
            val lat: Double = lastLocation!!.latitude
            val time: String = currentTime
            val tagModel: TagModel =
                TagModel(
                    -1,
                    "${tag.getEPC()}",
                    long,
                    lat,
                    time,
                    firstVehicle.toString()
                )
            val success = dataBaseHelper.addOne(tagModel)
            if (success) {
                Log.d("Insertion", "new tag: ${tag.getEPC()} added.")
            }
        } catch (e: SQLiteConstraintException) {
            // Handle the duplicate entry case, maybe log it or inform the user
            Log.d("Insertion", "Duplicate EPC: ${tag.getEPC()} not added.")
        } catch (e: Exception) {
```

```
156             Log.d("Insertion", "ERROR: ${tag.getEPC()} not added.")
157         }
158     }
159     private fun addTagsToView(linearLayout: LinearLayout) {
160         if (tagList.isNotEmpty()) {
161             for (tag in tagList) {
162 
163                 //Add Tag Data to Scroll View
164                 val textView = TextView(this).apply {
165                     text = "EPC: ${tag.getEPC()}"
166                 }
167                 linearLayout.addView(textView)
168                 addTagToDB(tag)
169             }
170         } else {
171             val textView = TextView(this).apply {
172                 text = "No tags found."
173             }
174             linearLayout.addView(textView)
175         }
176     }
177     private fun checkForDuplicateTags(linearLayout: LinearLayout, tempTagList:
        MutableList<RFIDTag>) {
178         for (tempTag in tempTagList) {
179             if (!tagList.any { it.epc == tempTag.epc }) {
180                 tagList.add(tempTag)
181             }
182         }
183         linearLayout.removeAllViews()
184     }
185     private fun playSound() {
186         val mediaPlayer = MediaPlayer.create(this, R.raw.alien_blaster)
187         mediaPlayer.start()
188         mediaPlayer.setOnCompletionListener {
189             it.release()
190         }
191     }
192     private fun getCurrentTime(): String {
193         val currentTime = Calendar.getInstance()
194         val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())
195         return dateFormat.format(currentTime.time)
196     }
197     private fun getLastLocation() {
198         if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
            ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
199             // Permission check failed. Exit the method.
200             return
201         }
202         fusedLocationClient.lastLocation.addOnSuccessListener { location: Location? ->
203             // Got last known location. In some rare situations, this can be null.
204             if (location != null) {
205                 lastLocation = location // Update the lastLocation variable with the
                    new location
206                 // Optionally, use the location data immediately for some task
207                 // For example, updating the UI or logging
208                 val latitude = location.latitude
209                 val longitude = location.longitude
210                 Log.d("LocationUpdate", "New location received: Lat $latitude, Lon
                    $longitude")
211             } else {
```

```
212                 // Handle the case where location is null
213                 Log.d("LocationUpdate", "No location received")
214             }
215         }
216     }
217     private fun showLocationToast() {
218         val locationMessage = if (lastLocation != null) {
219             "Latitude: ${lastLocation!!.latitude}, Longitude: ${lastLocation!!.
                longitude}"
220         } else {
221             "Location not available"
222         }
223 
224         Toast.makeText(this, locationMessage, Toast.LENGTH_LONG).show()
225     }
226     /// PERMISSION FUNCTIONS ///
227     private fun checkAndRequestLocationPermissions() {
228         if (ContextCompat.checkSelfPermission(this, Manifest.permission.
                ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED ||
229             ContextCompat.checkSelfPermission(this, Manifest.permission.
                ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
230 
231             ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.
                ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION),
                LOCATION_PERMISSION_REQUEST_CODE)
232         }
233     }
234     override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out
            String>, grantResults: IntArray) {
235         super.onRequestPermissionsResult(requestCode, permissions, grantResults)
236         when (requestCode) {
237             LOCATION_PERMISSION_REQUEST_CODE -> {
238                 if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
239                     // Permission was granted. Continue with location-related
                        functionality
240                 } else {
241                     // Permission was denied. Provide an explanation to the user and
                        guide them to enable it through settings
242                     if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                        Manifest.permission.ACCESS_FINE_LOCATION)) {
243                         showPermissionDeniedExplanation()
244                     } else {
245                         // User also checked "Don't ask again". Guide them to app
                            settings.
246                         guideUserToAppSettings()
247                     }
248                 }
249             }
250         }
251     }
252     private fun showPermissionDeniedExplanation() {
253         AlertDialog.Builder(this)
254             .setMessage("This app requires location permissions to scan and associate
                tags with their locations. Please allow location access.")
255             .setPositiveButton("OK") { _, _ ->
256                 checkAndRequestLocationPermissions()
257             }
258             .setNegativeButton("Cancel", null)
259             .create()
```

```
            .show()
    }
    private fun guideUserToAppSettings() {
        Toast.makeText(this, "Please enable location permissions in app settings",
            Toast.LENGTH_LONG).show()
        val intent = Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS, Uri.parse("
            package:$packageName"))
        startActivity(intent)
    }

}
```

### 7.3.13 C.13 SetupActivity.kt

```
package com.example.alienprobe.presentation

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.view.Gravity
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationServices
import com.example.alienprobe.*
import com.example.alienprobe.databinding.SetupBinding

class SetupActivity : AppCompatActivity() {

    private lateinit var binding: SetupBinding
    private lateinit var fusedLocationClient: FusedLocationProviderClient

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = SetupBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupListeners()

        loadPreferences()

        fusedLocationClient = getLocationServices()

    }

    private fun getLocationServices(): FusedLocationProviderClient {
        return LocationServices.getFusedLocationProviderClient(this)
    }

    private fun setupListeners() {

        binding.backBtn.setOnClickListener {
            val intent = Intent(this@SetupActivity, MainActivity::class.java)
            startActivity(intent)
        }

        //NEED TO CHANGE THIS TO ONLY SAVE THE MODIFIED FIELDS//
```

```
        binding.saveButton.setOnClickListener {
            savePreferences(
                binding.readerUsernameInput.text.toString(),
                binding.readerPasswordInput.text.toString(),
                binding.readerIPInput.text.toString(),
                binding.readerPortInput.text.toString().toIntOrNull() ?: 0,
            )

            val toast = Toast.makeText(applicationContext, "Preferences Saved", Toast.
               LENGTH_LONG)
            toast.setGravity(Gravity.TOP or Gravity.CENTER_HORIZONTAL, 0, 0)
            toast.show()
            loadPreferences()
        }
    }

    private fun savePreferences(username: String, password: String, ip: String, port:
       Int) {
        val sharedPreferences = getSharedPreferences("AppPreferences", Context.
           MODE_PRIVATE)
        val editor = sharedPreferences.edit()
        editor.putString("Username",username)
        editor.putString("Password",password)
        editor.putString("IP",ip)
        editor.putInt("Port",port)
        editor.apply()
    }
    private fun loadPreferences() {
        val sharedPreferences = getSharedPreferences("AppPreferences", Context.
           MODE_PRIVATE)
        val savedUsername = sharedPreferences.getString("Username", "alien")
        val savedPassword = sharedPreferences.getString("Password", "password")
        val savedIP = sharedPreferences.getString("IP", "161.6.219.3")
        val savedPort = sharedPreferences.getInt("Port", 23)

        val readerIPInput = findViewById<TextView>(R.id.readerIPInput)
        readerIPInput.hint = savedIP
        val readerPortInput = findViewById<TextView>(R.id.readerPortInput)
        readerPortInput.hint = savedPort.toString()
        val readerUsernameInput = findViewById<TextView>(R.id.readerUsernameInput)
        readerUsernameInput.hint = savedUsername
        val readerPasswordInput = findViewById<TextView>(R.id.readerPasswordInput)
        readerPasswordInput.hint = savedPassword
    }
}
```

### 7.3.14 C.14 TagsAdapter.ky

```
package com.example.alienprobe.presentation;

import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.net.Uri;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
```

```
12 import android.view.ViewGroup;
13 import android.view.Window;
14 import android.widget.Button;
15 import android.widget.TextView;
16 import android.widget.Toast;
17
18 import androidx.annotation.NonNull;
19 import androidx.cardview.widget.CardView;
20 import androidx.recyclerview.widget.RecyclerView;
21
22 import com.example.alienprobe.R;
23 import com.example.alienprobe.database.DataBaseHelper;
24 import com.example.alienprobe.database.TagModel;
25
26 import java.util.List;
27 import java.util.Objects;
28
29 public class TagsAdapter extends RecyclerView.Adapter<TagsAdapter.ViewHolder> {
30     private final List<TagModel> tagsList;
31     private final LayoutInflater inflater;
32     private final Context context; // Added to use for launching an Intent
33     private final DataBaseHelper dbHelper;
34
35     public TagsAdapter(Context context, List<TagModel> tagsList, DataBaseHelper
          dbHelper) {
36         this.context = context;
37         this.inflater = LayoutInflater.from(context);
38         this.tagsList = tagsList;
39         this.dbHelper = dbHelper;
40     }
41
42     @NonNull
43     @Override
44     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
45         View view = inflater.inflate(R.layout.tag_item, parent, false);
46         return new ViewHolder(view);
47     }
48
49     @Override
50     public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
51         TagModel tag = tagsList.get(position);
52         String text = "EPC: " + tag.getEPC();
53
54         holder.epcTextView.setText(text);
55         holder.tagContainer.setOnClickListener(v -> showDialog(tag));
56         // Set the click listener for the mapButton instead of epcTextView
57         holder.mapButton.setOnClickListener(new View.OnClickListener() {
58             @Override
59             public void onClick(View v) {
60                 // Open Google Maps with the specified coordinates
61                 Uri gmmIntentUri = Uri.parse("geo:" + tag.getLatitude() + "," + tag.
                       getLongitude() + "?z=10");
62                 Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
63                 mapIntent.setPackage("com.google.android.apps.maps");
64                 if (mapIntent.resolveActivity(context.getPackageManager()) != null) {
65                     context.startActivity(mapIntent);
66                 } else {
67                     // If Google Maps is not installed, open the location in a web
                           browser
```

```
                    Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("
                        https://maps.google.com/?q=" + tag.getLongitude() + "," + tag.
                        getLatitude()));
                    context.startActivity(browserIntent);
                }
            }
        });
    }

    private void showDialog(TagModel tag) {

        final Dialog dialog = new Dialog(inflater.getContext());
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
        dialog.setContentView(R.layout.bottom_sheet_layout);


        Button delete = dialog.findViewById(R.id.deleteButton);

        TextView epc = dialog.findViewById(R.id.epcView);

        String textToSet = "EPC: " + tag.getEPC() + "\nLat: " + tag.getLatitude() + "
            Long: " + tag.getLongitude() + "\nTime: " + tag.getTime();
        epc.setText(textToSet);

        delete.setOnClickListener(v -> {
            // Delete the brew from the database
            if (dbHelper.deleteTag(String.valueOf(tag.getId()))) {
                // Remove brew from the list and notify adapter
                int position = tagsList.indexOf(tag);
                tagsList.remove(position);
                notifyItemRemoved(position);
                Toast.makeText(inflater.getContext(), "Tag deleted successfully",
                    Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(inflater.getContext(), "Failed to delete tag", Toast.
                    LENGTH_SHORT).show();
            }
            dialog.dismiss();
        });
        dialog.show();
        Objects.requireNonNull(dialog.getWindow()).setLayout(ViewGroup.LayoutParams.
            MATCH_PARENT,ViewGroup.LayoutParams.WRAP_CONTENT);
        dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT))
            ;
        dialog.getWindow().setGravity(Gravity.CENTER);
    }
    @Override
    public int getItemCount() {
        return tagsList.size();
    }
    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView epcTextView;
        Button mapButton;
        CardView tagContainer;
        ViewHolder(View itemView) {
            super(itemView);
            epcTextView = itemView.findViewById(R.id.tagView);
            mapButton = itemView.findViewById(R.id.mapButton);
            tagContainer = itemView.findViewById(R.id.tagContainer);
        }
```

```
    }
}
```

### 7.3.15 C.15 ViewTagsActivity.kt

```
package com.example.alienprobe.presentation

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.alienprobe.database.DataBaseHelper
import com.example.alienprobe.R

class ViewTagsActivity : AppCompatActivity() {

    private lateinit var tagsRecyclerView: RecyclerView
    private lateinit var adapter: TagsAdapter
    private lateinit var dataBaseHelper: DataBaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.view_tags)

        setupRecycler()

        setDataBaseHelper()

        setupAdapter()

        setupListeners()
    }
    private fun setupRecycler() {
        tagsRecyclerView = findViewById(R.id.tagsRecyclerView)
        tagsRecyclerView.layoutManager = LinearLayoutManager(this)

    }
    private fun setDataBaseHelper() {
        dataBaseHelper = DataBaseHelper(this)
    }
    private fun setupAdapter() {
        val allTags = dataBaseHelper.allTags
        adapter = TagsAdapter(
            this,
            allTags,
            dataBaseHelper
        )
        tagsRecyclerView.adapter = adapter
    }
    private fun setupListeners() {
        val backButton = findViewById<Button>(R.id.backButtonTagView)
        backButton.setOnClickListener {
            val intent = Intent(this, ScannerActivity::class.java)
            startActivity(intent)
        }
    }
```

```
    fun openGoogleMaps(latitude: Double, longitude: Double) {
        // Create a Uri from an intent string. Use the result to create an Intent.
        val gmmIntentUri = Uri.parse("geo:$latitude,$longitude?q=$latitude,$longitude")

        // Create an Intent from gmmIntentUri. Set the action to ACTION_VIEW
        val mapIntent = Intent(Intent.ACTION_VIEW, gmmIntentUri)

        // Make the Intent explicit by setting the Google Maps package
        mapIntent.setPackage("com.google.android.apps.maps")

        // Attempt to start an activity that can handle the Intent
        if (mapIntent.resolveActivity(packageManager) != null) {
            startActivity(mapIntent)
        } else {
            // If Google Maps is not installed, open the location in a web browser
            val browserIntent = Intent(Intent.ACTION_VIEW, Uri.parse("https://maps.google.com/?q=$latitude,$longitude"))
            startActivity(browserIntent)
        }
    }
}
```