Dryer Reminder

Michael Mohler

CST-451 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Revision: 1

Date: 11/21/21

**ABSTRACT**

The Dryer Reminder is a device that can pair with a phone application to tell the user when their dryer has stopped spinning. Notifications are so ingrained into people that they wish they had them for the appliances in their home. The way dryers notify people on their own isn’t enough for many. The noise alert from a dryer can be missed after the jingle has finished. The timers are not consistent either to just match it with a phone timer. A notification from a phone would easily grab any user’s attention. Stay at home spouses, the hard of hearing, and tech enthusiasts would be ready to grab this for the convenience of not missing when their dryer has finished.

It might seem hard to develop this for any kind of dryer, but all the device needs to do is check if it is shaking. Every dryer needs to move to dry the laundry out, which makes it vibrate. With research to find out what component to check for motion, it just needs some programmers for the device and phone app to get it off the ground. There should not be a need for a dedicated server, so the cost after it is finished would go to upkeep on the mobile application. The device could be sold by itself for much less than a new dryer that has the same feature installed. There are huge communities for some of the small computers they use for this. Focusing on that would allow for selling the application by itself too. There are a lot of opportunities with the Dryer Reminder project with just as many payoffs.

|  |
| --- |
| History and Signoff Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 11/21 | Michael Mohler | Initial draft for review/discussion |
|  |  |  |
|  |  |  |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

Yes  No

**TABLE OF CONTENTS**

Design Overview 4

Detailed High-Level Solution Design 5

Detailed Technical Design 6

Appendix A – Technical Issue and Risk Log 7

Appendix B – References 8

Appendix C – External Resources 9

Design Introduction

The Dryer Reminder is an embedded systems project where two devices will communicate with each other to help inform the user when their dryer has stopped spinning. Both devices will send data to each other using a RESTful service that will run on the device attached to the outside of the dryer. The mobile application used by the user will control the device on the dryer, using a REST client. These service and client will communicate over local Wi-Fi, which will allow a user to still use the device if their internet is down.

The device that will be attached to the dryer is a Raspberry Pi 3 Model B, with a Sense HAT attached to the top of it through the GPIO pins. The Sense HAT carries an accelerometer onboard which will be used by the Pi to calculate the range of each axis after about 15 seconds. The user will first have to calibrate the device by getting the range of the axes while the device is not moving. The ranges will then be sent to the phone and saved. When the user starts the dryer process it will receive the ranges from the phone and start checking the range every 15 seconds until it is less than or equal to the calibrated ranges, plus an offset. An API to tell the Raspberry Pi what to do will be a RESTful service running the Flask library and will be secured using Basic Http Authentication using a token.

An android application, developed specifically for a Samsung Galaxy S8+, will be used to communicate with the Pi on what processes to start and send the user a notification when the dryer has stopped. There will be multiple views to navigate the user to either start the device or go to a settings page to calibrate the pi, adjust the offset, or change how often their phone will remind them their laundry is done. The phone will use a REST client called OkHttp to get and send information to the Raspberry Pi. The application will send the Pi the variable used for the offset, get the ranges when the Pi calibrates its accelerometer, and send those saved ranges to check when the user wants to finally start their dryer. It will also need to know the token to comply with the Basic HTTP Authentication.

An API has been designed and put onto Swagger so that the developer will use it as a reference to develop the RESTful service. This will also allow for others to develop their own ways to communicate with the service on the Raspberry Pi.

There is a plan for a washer reminder to detect when the user’s washer has stopped, but it is out of scope. This will need to include changes to dryer service code on the Raspberry Pi, but otherwise will work the same from the android app until the method is called on the Raspberry Pi.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Deliverable Acceptance Log | | | | | | |
| ID | Deliverable Description | Comments | Evaluator (internal or external as applicable) | Status | Date of Decision |
| 1 | Sweagger documentation of the Raspberry Pi API using Flask |  | <https://app.swaggerhub.com/apis/Mmohler1/DryPi/1.0.0> | In Progress | 11/18/21 |

Detailed High-Level Solution Design

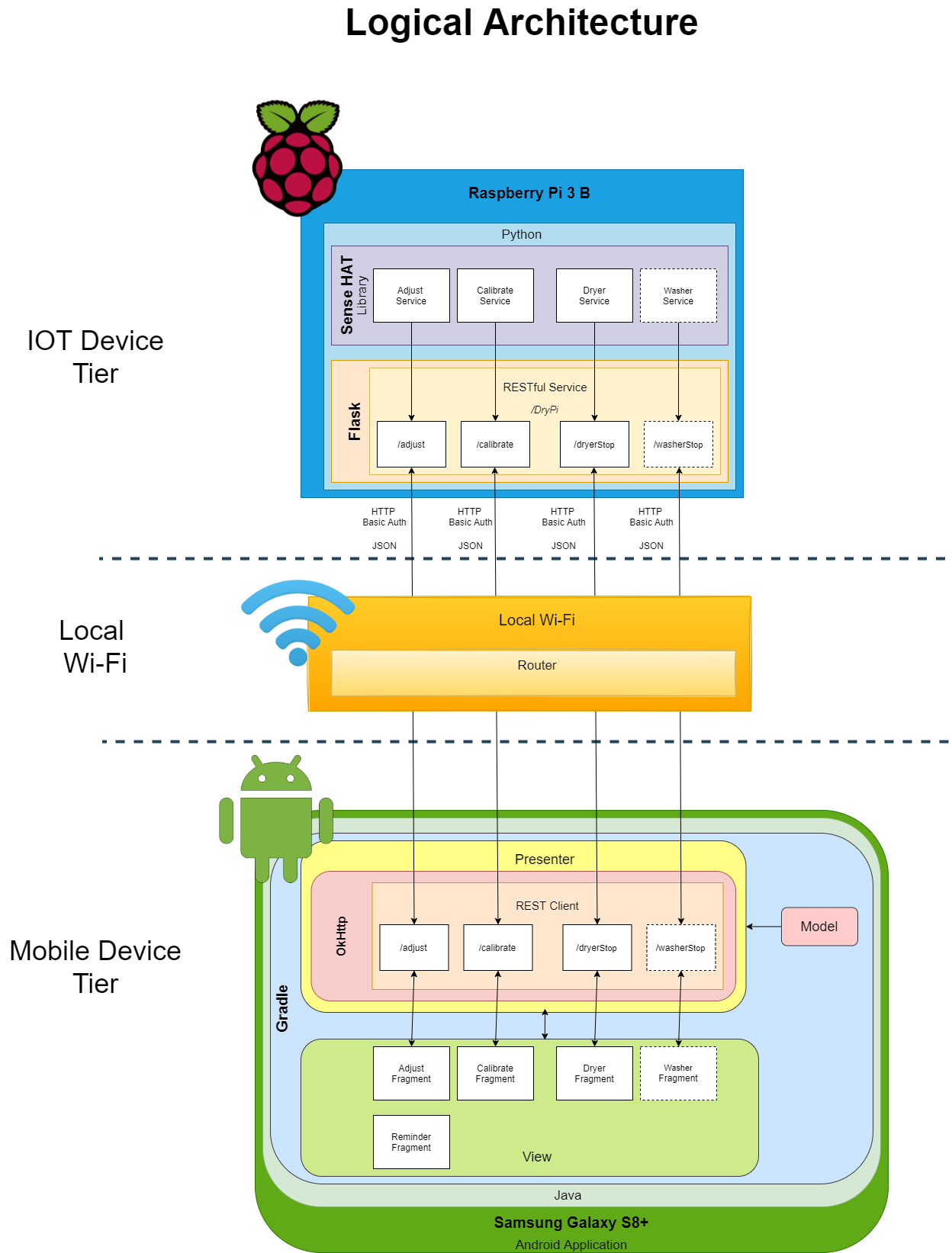
|  |  |  |
| --- | --- | --- |
| **Proof of Concepts** | | |
| **Description** | **Rationale** | **Results** |
| An accelerometer will show a notable difference in results in each axis when the dryer is on compared to when it is off. | When a dryer shakes it creates movement. An accelerometer could be used to read the axes of that movement and calculate the range of each axis. | After taking three minutes of data from an accelerometer (MPU6040), with the dryer on, then off, a noticeable difference was seen in the range of the axes. When comparing the axis there was as little as 1.5 times to as high as 10 times the difference in the range. |
| Using the Android device as a RESTful Service to make it easier to communicate between itself and the RESTful service on the Raspberry Pi. | The communication between both devices will be simpler. | The ability to turn an Android phone into a RESTful service is not supported by the software and as such no libraries are made for it. |
| A RESTful service will need to be used to communicate between the IOT device and the Mobile Phone. | Both devices will need to find a way to communicate with each other through an API for the Mobile phone | The RESTful service on a Raspberry Pi was used to send data to a Java Application that would have been used to tell the user the dryer stopped running. |
| The API on the IOT and Mobile device to communicate using Local Wi-Fi. | The API’s will need a way to communicate with each other and the local Wi-Fi from a router could be used to do this. | If the IP address on the Raspberry Pi is known, then they can communicate with each other over local Wi-Fi. An internet connection will not be needed, which can save in frustration. This will not work if they are not connected to the same network. |
| The API on the IOT and Mobile device to communicate using a service on the web | The API’s will need to find a way to communicate with each other and connecting to a web service can help. | The devices will not need to be configured to the Pi’s IP Address. This will not work if the internet is not working. More work will need be down to make a service that is deployed to a cloud to handle the communication. |
| Raspberry Pi 3 | The Raspberry Pi 3 is a cheap computer that has plenty of options to find or connect an accelerometer to. The 3rd model is cheap and has plenty of accessories being made for it. | The Raspberry Pi Models are suffering from distribution issues and the computer part shortages are catching up to it. It is still the best option for this project but may not be as cheap to find new computers. |
| MPU6050 | Relatively cheap accelerometer. | The team has no experience making a custom library for an embedded system. |
| Sense HAT | Free Libraries for the accelerometer that a part the device. Popular among those who like the Pi. Other features that could be taken advantage of in future software revisions. | The libraries being free works better for the small team on this project. It also reduces time for installation of the software for those who want to use their own Pi. |
| Sense HAT Library | A free library that will aid in the speed of development for the project. | Extremely easy to import into the project and read data from the accelerometer. The protype was refactored to work with it in only a couple of minutes. |
| Flask Library | The RESTful service is common for Raspberry Pi devices, which can read and send JSON data. | Was able to read and POST a string over local Wi-Fi to a Java Application. |
| Fast-Android-Networking | A very powerful REST Client for Android devices that can read and send JSON data. | The library is more powerful than project needs it to be. There is very little documentation on using it, compared to other REST Clients. |
| OkHttp | A simple REST Client to communicate with the Raspberry Pi’s RESTful service. | Very simple to use and implement. Works outside of android and there is a lot of documentation and examples. |

|  |  |  |  |
| --- | --- | --- | --- |
| **List of Technologies and Tool Needed** | | | |
| **Technology or Tool** | **Type** | **Version or Model** | **Company or Owner** |
| Raspberry Pi | Computer | 3 B | Raspberry Pi Foundation |
| Raspberry Pi | Computer | 3 B+ | Raspberry Pi Foundation |
| Sense HAT | Board | Unspecified | Raspberry Pi Foundation |
| Thonny | IDE | 3.3.13 | Aivar Annamaa |
| Android Studio | IDE | 4.2.2 | Google, JetBrains |
| Galaxy | Smartphone | S8+ | Samsung |
| Flask | Library | 2.0.2 | Pallets |
| OkHttp | Library | 4.9.1 | Square |
| Python | Language | 3.7.9 | The Python Software Foundation |
| Java JDK | Language | 11 | Oracle |

**Logical Solution Design:**

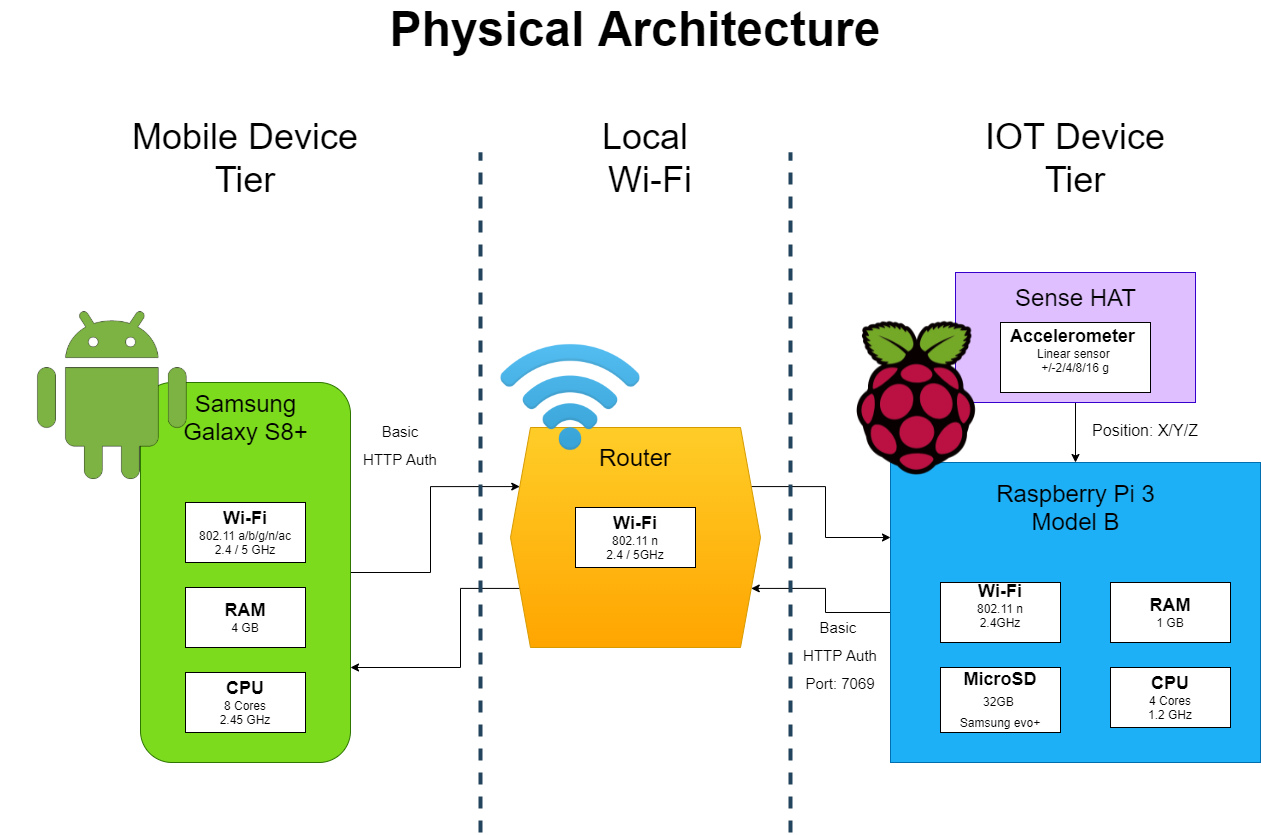
The logical architecture of the project. This gives an idea what languages, builds, and libraries each device will use. It also shows how the device will communicate with each other using a RESTful service and client. This will include the API’s that will be made for the project and what design principle the Android application will use.

<https://github.com/Mmohler1/DryerReminder/blob/main/Design/Logical.png>



**Physical Solution Design:**

The physical architecture of the project. This will show the hardware that is being used on the Raspberry Pi, Sense HAT, and Samsung Device. It will also show the router used to connect to both will need to be Dual Band.



Detailed Technical Design

**General Technical Approach:**

The Python application will be designed as if it is an Object-Oriented Programming language. Object models will be used for the axes. The services, API’s and models will also be in different files so the program is not in one massive python file that holds everything for the project.

The Android application will be designed with Object-Oriented Programing in mind and use the Model View Presenter (MVP) design principle. MVP is easier to implement in Android Studio then the Model View Controller (MVC) design principle that the developer has more experience with.

**Key Technical Design Decisions:**

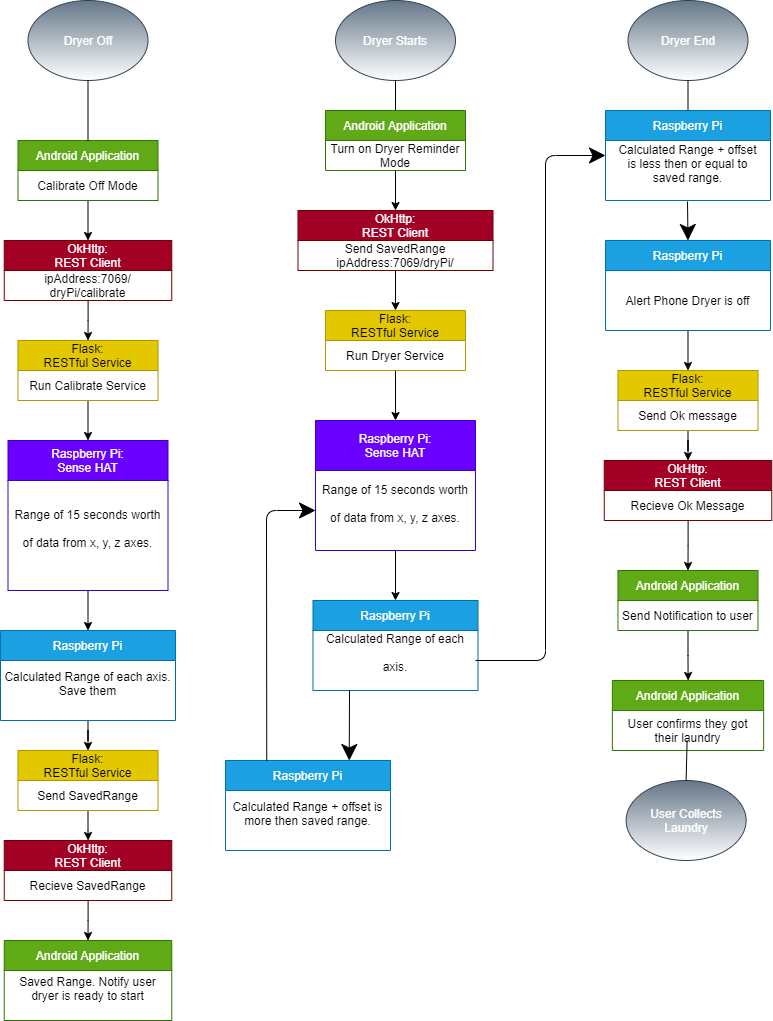
The hardware for the device that attaches to the dryer will be a Raspberry Pi 3 Model B and a Sense HAT, both made by the same company. Raspberry Pi’s are very popular small computers for users that like to implement projects themselves. Since the project will be designed in a way so that the user can make it themselves, using a popular IOT device is necessary for connecting to the internet. The Sense HAT is needed for the accelerometer on the device and since it easily attaches to the Raspberry Pi’s GPIO pins, this adds to the possibility that a user will try to implement this project. The Sense HAT is also powerful enough to determine the difference in small movement a dryer makes when it shakes, which is paramount for the entire project working.

The Raspberry Pi will use Python as it’s coding language since the developer has the most experience in it for Raspberry Pi development. The library used to control the sense HAT are also written in Python which will aid in the speed and development of the project. Flask will be used as the RESTful service on the Pi to communicate with itself and the android application. Flask was chosen for its ability to easily send JSON data and the documentation surrounding it. The library has been tested and proven to work with another RESTful service and client on Java applications running in Spring Boot and Android Studio.

The mobile application will specifically be made for the Samsung Galaxy S8+ and use Java JDK 11. The developer has more experience with Java over Kotlin, so he chose to use Java for this project. The RESTful client used will be OkHttp Networking for its simplicity and compatibility with Basic HTTP Authentication for android devices. The application will be developed in a way that it can work with other android devices, but the developer will specifically be making this for a Samsung Galaxy S8+. Compatibility on other android phones will vary because of this.

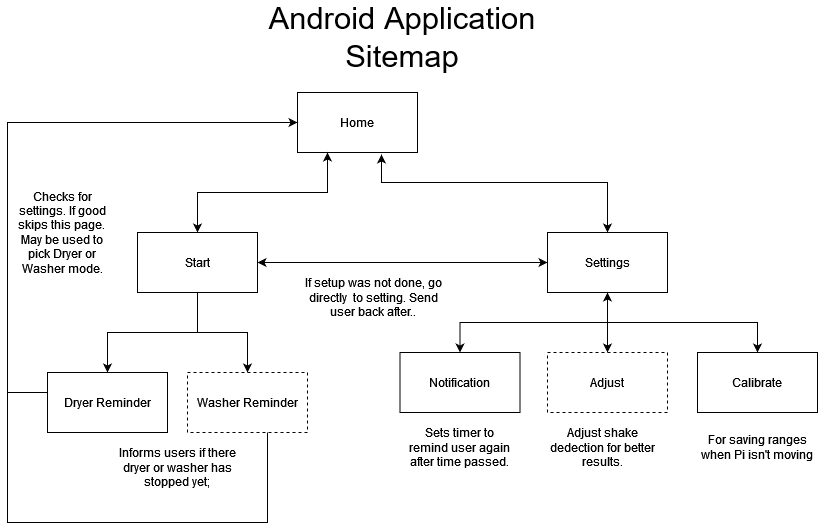
**Flow Charts/Process Flows:**

This flow chart shows the basic flow of the project that the user will have to do at the bare minimum to get their device to work. The part on the left is how the device will calibrate itself when it is off. The right part is the process of checking if the dryer has stopped. This flowchart does not go over every possible outcome, just the most likely.



**Sitemap Diagram:**

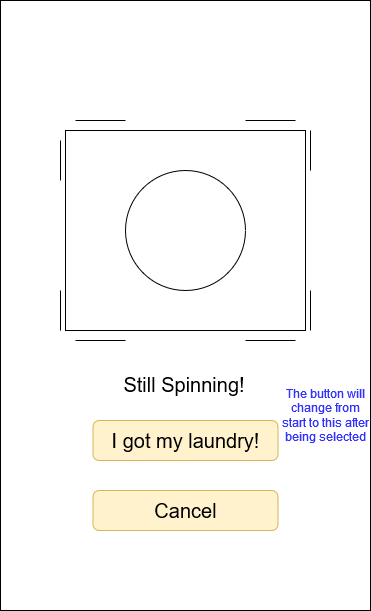
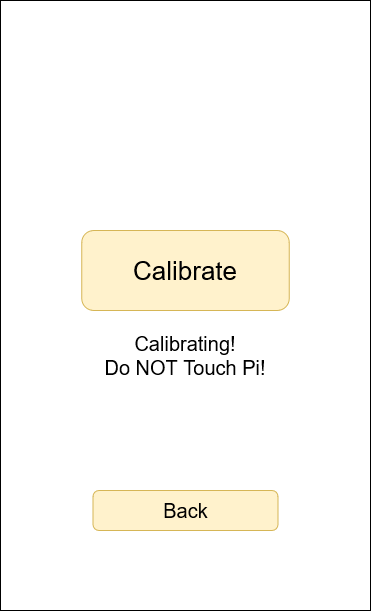
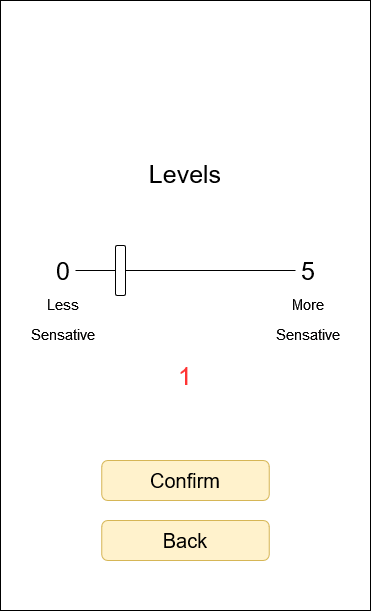
The sitemap gives an idea of how the user will navigate the android application.



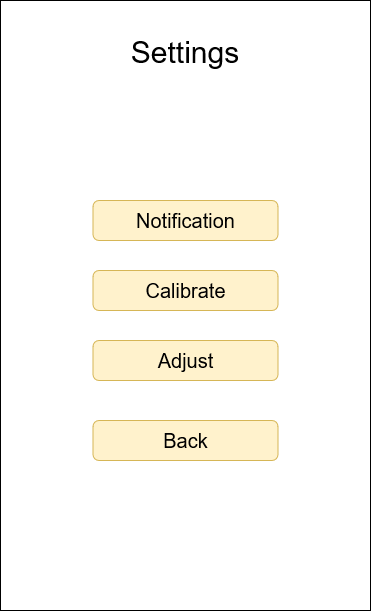
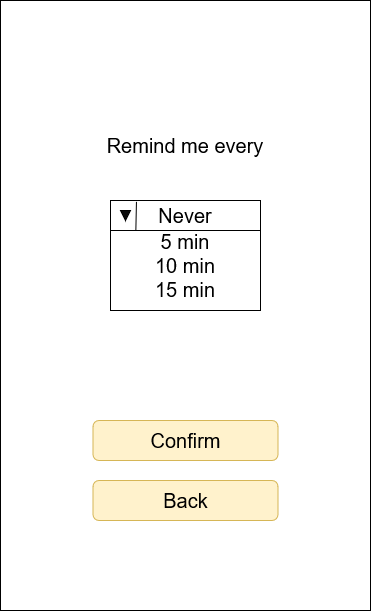
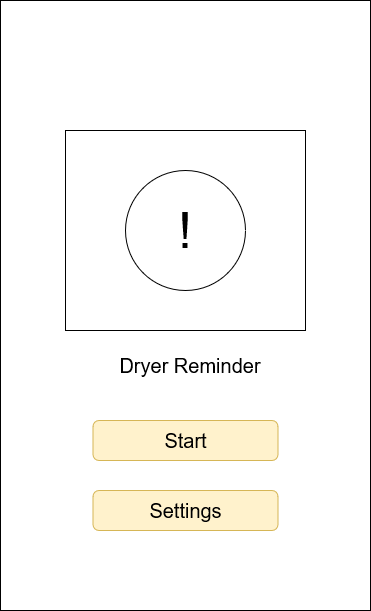
**User Interface Diagrams:**

These are wireframes of what every page should look like in the Android application.

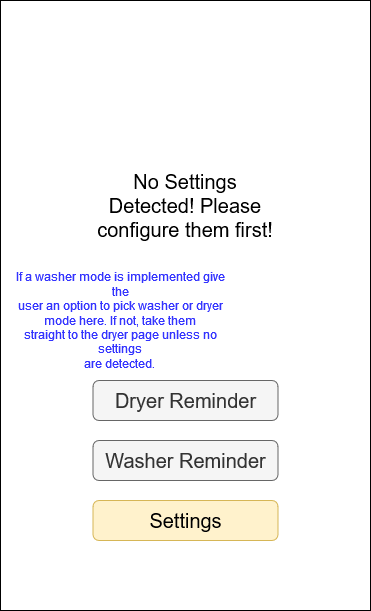
Adjust Calibrate Dryer/Washer Reminder



Home - Dryer Reminder Notify Settings



Start

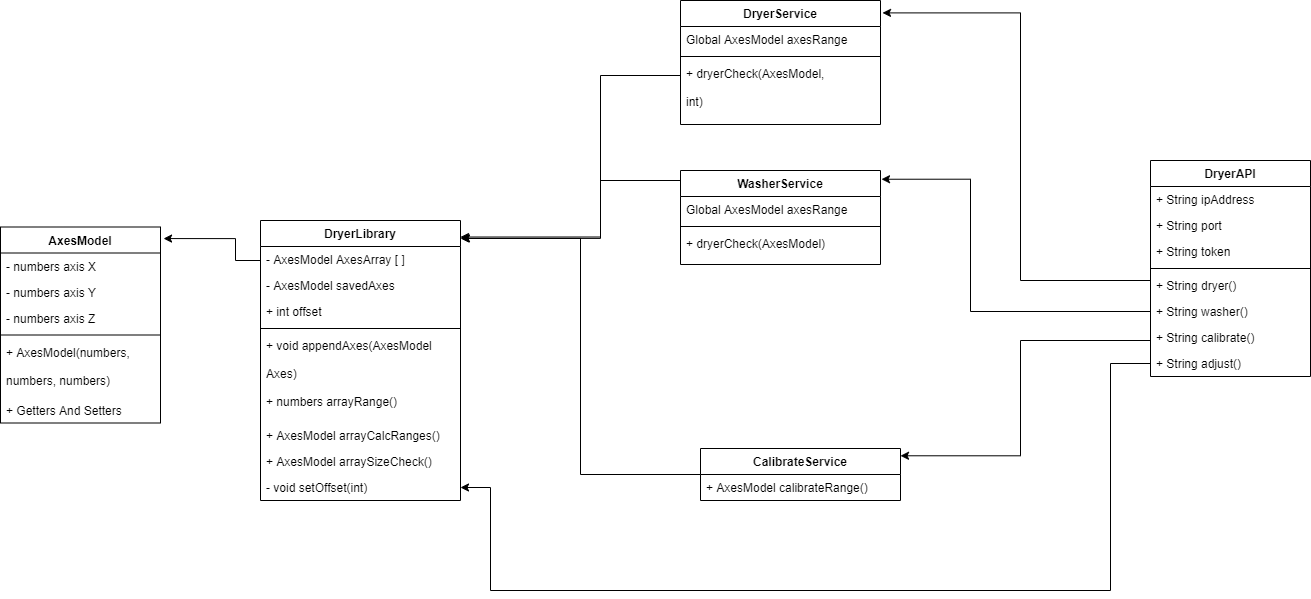


**UML Diagrams:**

Python Application UML Class Diagram

Since the Python application will be designed as if it is an Object-Oriented programming language, the UML has been structured to match that design as closely as possible.

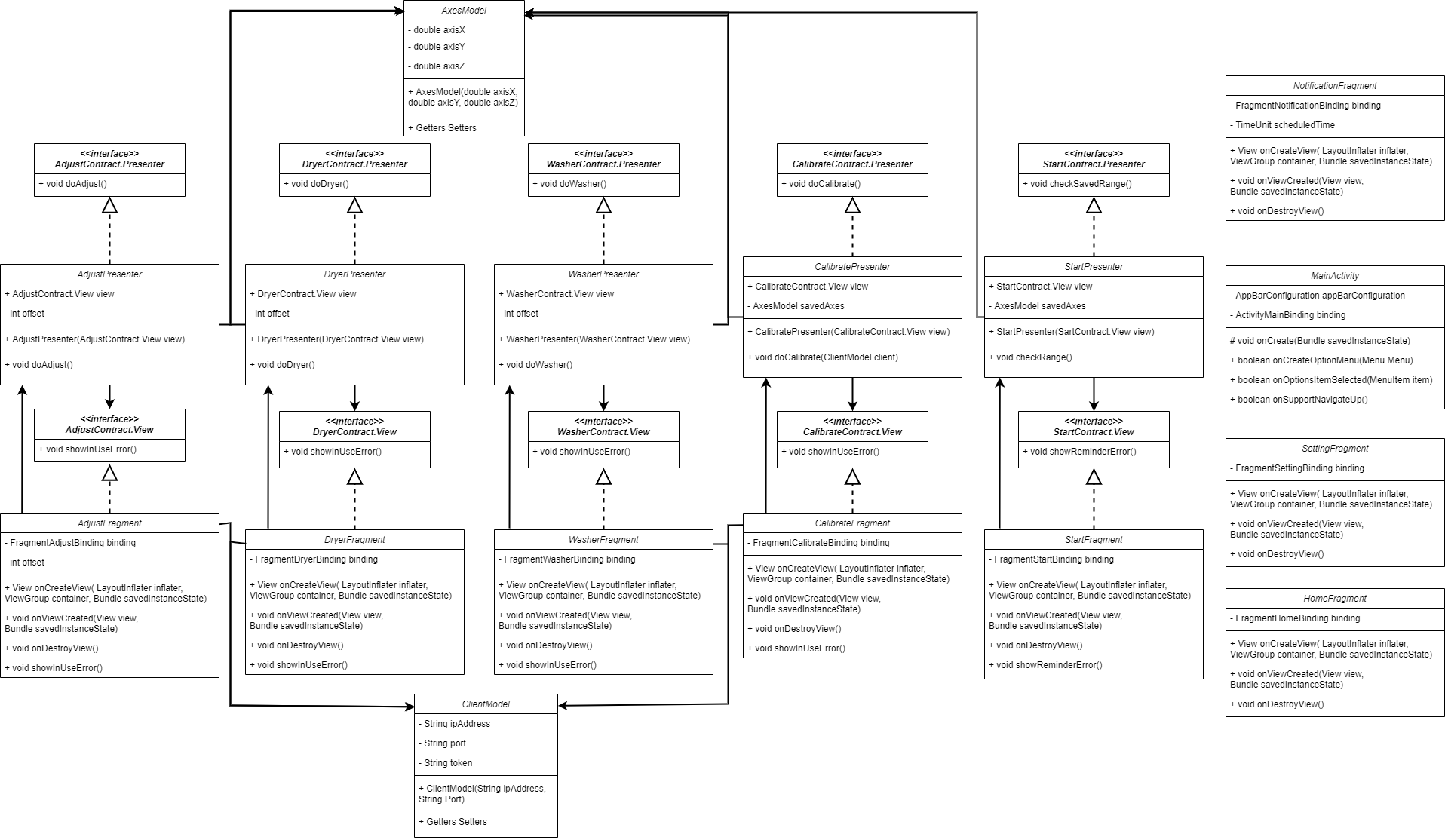
<https://github.com/Mmohler1/DryerReminder/blob/main/Design/PythonUML.drawio.png>



Android Application UML Class Diagram

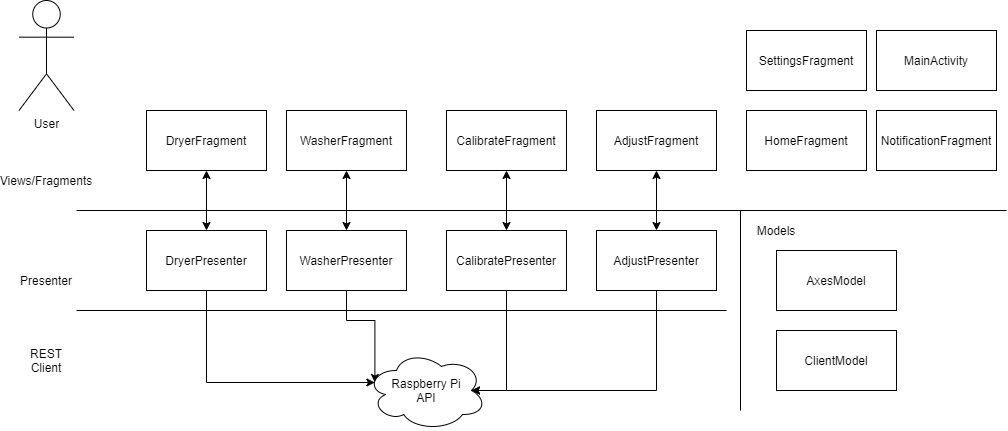
Since this is the first time the developer has made an android application and used the MVP principle, there are some issues with the diagram. Methods are used within the onViewCreated methods to form the actions for each button and each of those methods are called onClick that use a listener to decide which button is pressed. This made it difficult to showcase on the UML and there was not much information on how to present it. Since the wireframes are details enough to show what each button does, the listeners were left out of the UML. The contacts interfaces hold the interface for their respective view and presenter but use a period to separate each one. To better showcase the flow, they were separated on the diagram.

<https://github.com/Mmohler1/DryerReminder/blob/main/Design/AndroidUML.drawio.png>



Android Application UML Sequence Diagram

An overview of what the user interacts with, how the fragments connect to the presenters, and where the app connects to the API that communicates with the Python application on the Raspberry Pi.



**Security Design:**

Security is an important part of any application as it can hold valuable information on a user or their device to get their information. While this project does not hold information on its users, it does have an API that, if the IP address and port is known, just about anybody can try to connect to the DryPi service on the Raspberry Pi. To help with this, all information will be sent over a POST on the RESTful service to keep the information secure on the device. The reason for this is that information sent by a GET is transmitted over plaintext, making the information sent vulnerable to being exposed.

The most important way this device will be secured is with the help of Basic HTTP Authentication. By setting up a token on the Raspberry Pi, that only the Android application should have, a malicious user would need to know that token to be able to communicate with the Raspberry Pi. The Raspberry Pi’s RESTful service, Flask, has a library for setting up Basic HTTP Authentication with usernames, passwords, and tokens. The REST client OkHttp, on the Android application, can connect to an API that requires this authentication, if it is given the token first.

These two methods will be very helpful in keeping the RESTful service safe and hopefully allowing users to operate the product without fear that either of their devices will be at risk.

**Operational Support Design:**

Python Logging

The Python application will use the built-in logging library to properly log the start and end of a service and API. This includes the data, sent, received, or modified. It will also log any errors or exceptions in the application that were caught, to help debug problems that were, or were assumed, to be caught. This level of logging should give an idea of what the Pi was doing at any point when a user is looking at the log file. Since the Pi will log the data being received form the accelerometer too, if a log file reads a larger then average change in the axes, then one could assume the device fell or was moved by another party.

The Python logger will be configured to save all levels of logging to a text file called PiLogger. This includes the level debug, which is, by default, not saved by the logger. The logs will show the level, time, and message of the log. The time is important to showcase since the device will work in unison with the Android application and if an error is showcased on both devices, it is important to pair up their interaction.

Android Logging

Android Studio already uses a logger called Logcat. Additional logs will be added to logcat for the Android application at the start and end of a fragment and presenter. This will include the data, sent, received, or modified. The application will also log any errors or exceptions that are caught by the application. The format of the logs will display the time, level, and message from the log. Time is important to show on the log if the device crashes the software on both the Android application and the Raspberry Pi. This will help decipher where the error may have occurred first. The device will also save the logs to a text file on the phone, in the case it crashes when it is used outside of it being connected to Android Studio.

**Other Documentation:**

No other documentation was created for the design of the project up to this point.

Appendix A – Technical Issue and Risk Log

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Issues and Risk Log | | | | | | | |
| **Issue or Risk** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date Resolved** |
| 1 | Accelerometer Library might not be Open Source | Work will need to be delegated to find or develop for the accelerometer. | One will have to either find an Open-Source Option or Develop our own for | Michael | High | 9/16/2021 | 09/29/2021 |
| 2 | Time Management with other projects | Reduces time to study, practice, and research solutions for the challenges. | Delegating time using proper time management software. | Michael | Low | 9/20/21 |  |
| 3 | Trouble Making Android Application | The entire app portion of the project must be changed. | Study up on android app development | Michael | High | 9/20/21 |  |
| 4 | Restful API between Application and Pi | Notifications may need to be delivered by another method. | Work with the Pi’s Bluetooth instead. | Michael | Medium | 9/20/21 | 10/18/21 |
| 5 | The Pi or accelerometer breaking | Slowing down the project development, especially in testing phases. | Have multiple Pi’s and accelerometers when the final versions are decided. | Michael | Low | 9/20/21 |  |

Appendix B – References

Appendix C – External Resources

|  |  |
| --- | --- |
| **GIT URL:** | [*https://github.com/Mmohler1/DryerReminder*](https://github.com/Mmohler1/DryerReminder) |
| **SWAGGER URL:** | [*https://app.swaggerhub.com/apis/Mmohler1/DryPi/1.0.0*](https://app.swaggerhub.com/apis/Mmohler1/DryPi/1.0.0) |