# IoT Remote Monitoring Mobile App for Commercial Appliances

DESIGN DOCUMENT

**Team Number**
17
**Client**
Greiner Jennings Holdings
**Adviser**
Goce Trajcevski
**Team Members/Roles**
*John Fleiner: Front End Lead*
*Ben Young: Hardware Lead*
*Thomas Stackhouse: Backend Lead*
*Hongyi Bian: Hardware Test*
*Yuanbo Zheng: Meeting Facilitator*
*Casey Gehling: Backend developer*
**Team Email**
sddec18-17@iastate.edu
**Team Website**
https://sddec18-17.sd.ece.iastate.edu

**Revised: Date/Version**
February 26th, Version 1

## Table of Contents

## List of figures/tables/symbols/definitions (This should be the similar to the project plan)

# 1 Introduction

## 1.1 Acknowledgement

Sddec18-17 would like to thank Taylor Greiner and Connor Jennings from Greiner Jennings Holdings, LLC for their contribution to the IoT Remote Monitoring Application for Commercial Appliances. Taylor Greiner and Connor Jennings submitted the proposal for the project and are providing our team with the necessary hardware and software components to complete our project. The items being provided include a washer control board, a dryer control board, a Raspberry Pi single-board computer, an Arduino Yun microcontroller, and an AWS IoT cloud service. Sddec18-17 would also like to thank the ECpE department at Iowa State University for funding our project.

## 1.2 Problem and Project Statement

According to the industry definition, a laundromat is a facility with washing machines and dryers available for public use. In fact, there are over 81,000 laundromats in the United States and the largest growing demand industry for on-site laundromats include apartments and dormitories. Despite the demand for on-site washing machine and dryer services, laundromats are often met with customer complaints. Customers tend to 'forget' that they are not the only ones doing laundry. Common complains often include 'having to wait for machines to become available' or lack thereof scheduling.

The purpose of our project is to find a method that mitigates scheduling conflicts between customers who want to access washing machines and dryers in a shared environment. To do so, our team will be utilizing the concept of IoT - Internet of Things. The internet of things consists of a network of physical hardware devices that can be controlled remotely.

Our proposed solution consists of two components: An IoT cloud service and a mobile application. An IoT cloud service will be used to register a set of washing machine and dryer control boards that can be controlled remotely. A multi-platform mobile application will be developed to connect to an IoT cloud service so that users may monitor, reserve, and control devices remotely. A reservation system on the mobile application will allow users to reserve a device for a set period of time. Once reserved, a time-stamped code will be generated for the user. during the reservation time, the reserved machine will be locked until the time-stamped code has been entered by the user, essentially gives users the opportunity to use a machine without it being taken. this will help prevent customers from traveling to a shared-appliance room only to find all of the machines in use.

## 1.3 Operational Environment

Since our proposed solution requires the use of several single-board computers, and microcontrollers, washer control boards, and dryer control boards, these hardware components may be subject to adverse operating conditions. Microcontrollers or single-board computers are also susceptible to overheating if overused or if located in a room with poor ventilation. It is expected that each washer and dryer will be frequently used, so we must account for standard wear-and-tear, damages, and out-of-service maintenance. Our microcontrollers and single board computers will be placed in an environment that is vulnerable to water damage. Neither microcontroller no single-board computer is water resistant, so caution must be taken when interfacing hardware components.

## 1.4 Intended Users and Uses

Greiner Jennings Holdings, LLC is dedicated to creating and delivering tech services for the industrial, electrical, and commercial space. they have collaborated with DPT Group and Critical Labs whom are dedicated to boosting productivity and control costs through synchronized communication, systems integration, and cloud computing. Therefore, the intended users of our mobile application include environmental and power systems manufacturers in the industrial, electrical, and commercial space.

The intended use cases of our mobile application can be divided into two separate sections: long-term and short-term. The short-term use case revolves around customers being able to remotely reserve a washing/drying machine using a third-party transaction platform. Each reservation generates a specialized time-dependent code. During the reservation period, the customer may enter the previously generated code to unlock and use the machine. Our client has mentioned that the long-term use case would include the expansion of IoT to other types of commercial appliances. However, we've been asked to direct our efforts towards the aforementioned short-term use case as integration of other types of commercial appliances should be trivial.

## 1.5 Assumptions and Limitations

**Assumptions**

1. The mobile application will only be presentable in English.
2. Each appliance will be differentiable from each other. We are being provided one washing machine component and one microcontroller to start. Thus, scalability should (hopefully) not be a problem.

**Limitations**

1. It is unknown how 'smart' the appliance component will be. The client has not purchased a component as of yet, but will do so with simplicity (for our sake) in mind. It is possible we will have do some configuration between the component and the microcontroller.
2. Clearly there is a deadline (December 2018) and thus a time constraint.
3. The application will be mobile and thus any intensive computing should happen on the server end of the architecture.
4. Hardware purchases may not exceed our $500 funded budget.

## 1.6 Expected End Product and Deliverables

**Mobile Application**

It is expected that our team delivers a multi-platform mobile application supporting the Android and iOS operating systems. The mobile application will allow users to view an availability schedule for shared-room appliances across several locations. Users will have the ability to reserve an appliance at a specific time for a fee. During each reservation period, the user will be able to control the reserved appliance remotely from his or her mobile device.

*Phase 1 Delivery Date:* A prototype of the customer user interface presenting several different screen implementations for scheduling and reserving an appliance will be delivered during the last week of February.

*Phase 2 Delivery Date:* A prototype of the customer user interface with data populated from the external backend will be delivered during the last week of March. It can be expected that version 1 of the reservation system is completed.

*Phase 3 Delivery Date:* Functioning prototype of the customer version of the iOS and Android application will be delivered during the last week of April

*Phase 4 Delivery Date:* A prototype of the administrator user interface presenting different screen implementations for viewing usage, energy and pricing statistics, analytics and data for machines at each laundromat location will be delivered during the last week of September.

*Phase 5 Delivery Date:* A model for analyzing data will be delivered during the last week of October.

*Phase 6 Delivery Date:* A prototype of the administrator user interface with valid data analytics presented will be delivered during the last week of November.

*Phase 7 Delivery Date:* A Completed application for the administrator user interface with valid data analytics being viewable will be delivered during the last week of class in December

### Web Server

It is expected that our team provides a dedicated hosting server with a MySQL database. the web server will be responsible for facilitating communication between the washing machine control board and the mobile application. Requests made from the mobile application will sent to the dedicated web server, which will work with the Amazon IoT Web Service to control and provide feedback from the registered commercial appliances. The MySQL database will be used to store user profile information, user login information, and calendar scheduling data.

*Phase 1 Delivery Date:* MySQL database populated with calendar data up to one year in advance. A REST API will be created for the mobile applications to retrieve and submit calendar data. Both will be delivered during the last week of February.

*Phase 2 Delivery Date:* Device Simulation with AWS IoT and AWS Lambda will generate device sensors that report their state to the cloud by the second week of March.

*Phase 3 Delivery Date:* Server will retrieve and submit chance in device state to the AWS IoT by the last week of March.

### IoT-connected Hardware Controller

The microcontroller will be responsible for communicating and controlling the appliance controller (and therefore the appliance). It will provide a simple interface for the web server to ultimately control the appliance. It will also provide feedback to the server/IoT cloud so that the user stays up to date.

*Delivery Date:* End of the first semester (May 2018).

# 2. Specifications and Analysis

## 2.1 Proposed Design

*Include any/all possible methods of approach to solving the problem:*

*– Discuss what you have done so far – what have you tried/implemented/tested, etc?*

*– We want to know what you have done*

*• Approach methods should be inclusive of* ***functional and non-functional requirements*** *of the project, which can be repeated or just referred to in this section*

*If your project is relevant to any* ***standards*** *(e.g. IEEE standards, NIST standards) discuss the applicability of those standards here*

### 2.1.1 Back-End Server

The back end development team has been working on putting together a server that will serve REST requests from both the mobile applications and the IoT interface that will be connected to the hardware. The design we have been working on to serve these requests is a Spring Boot application managed by Gradle. There are three main layers to the back end, as well as connection to a database and Spring Security.

The first layer of the application is the Controller layer. This layer defines what REST requests the back end will be able to serve. It is divided up further into components, where each component serves a specific object that is mapped directly to its own table in the database. To interact with the Controller layer, requests need to be authenticated via Spring security. Without authentication, a request will not be served. This Controller layer also calls the next layer, the Service layer, to serve responses back to the client.

The Service layer is used to do most of the heavy lifting for processing requests. The Service layer is also divided into components, where each component serves a specific object corresponding to a database table. After a Controller calls the Service that corresponds to the request that needs processed, that Service will call the Repository layer (the third layer) to access the data stored in the database. Any further processing is then done in the Service layer, any edits are saved to the Repository layer, and the processed response is then passed back to the Controller layer.

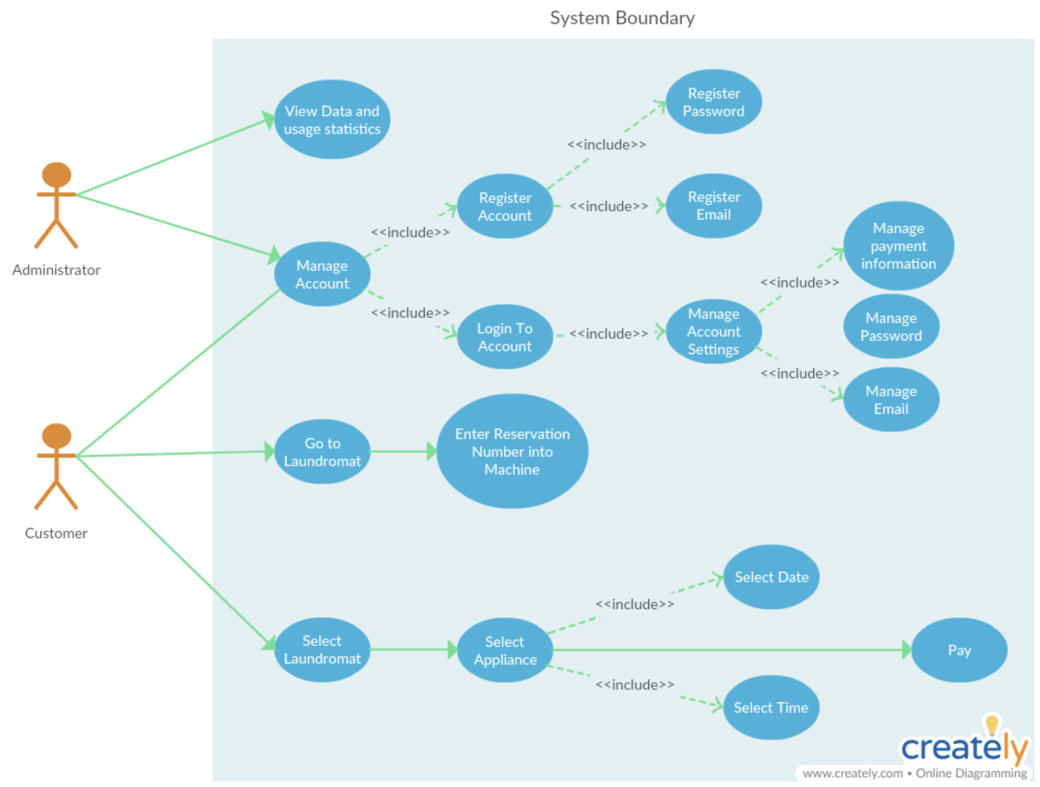
The Repository layer connects with the database and provides access to the data stored within. The Repository layer is also divided up into components, where each component serves a specific object by pulling it out of the corresponding table. The Repository layer in this application uses a JpaRepository that allows the direct manipulation of objects in the database via the Java object defined as an Entity that directly maps the object to the database table. Therefore, the manipulation of the database tables can be done directly through object manipulation in Java, and the results can be saved back into the database.

The Database is also a very important portion in the back end. There are actually three different databases, defined based on the stage of development the back end is currently in. For local development, the back end is using an H2 database embedded into the Spring Boot application itself. This database is automatically spun up according to the specifications of a schema, and is automatically populated by test data. However, this embedded database also ceases to exist after the back end powers down, and any changes to the data stored in the database are lost. Therefore, the embedded database is only viable for development and some initial testing with the mobile applications. The second database is a remote MariaDB database which was set up for this senior design project. The benefits of using this database are that data changes are persisted throughout multiple restarts of the back end, and it is remotely hosted which allows testing for the configuration of external sources before the third database is configured. This remote database will be used when the back end is set up on a remote server and will allow the mobile applications to hit it from actual phones. This will provide the basis for our User Acceptance Testing environment. The third database previously mentioned will be a production database, and will be like the remote database except for the fact that it will be hosted on Amazon Web Service and it will run the production data for the project once it is in the final stages of completion.

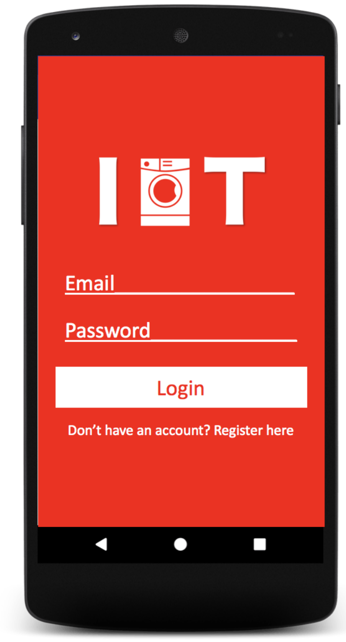
The final portion of the back end is Spring Security. Spring Security will allow authentication and authorization for our application, as well as preventing from many types of attacks such as clickjacking, cross site request forgery, session fixation, etc. It will be integrated with the Controller layer, but it will sit separately from the other layers and will be integrated with the application's custom user storage mechanism.

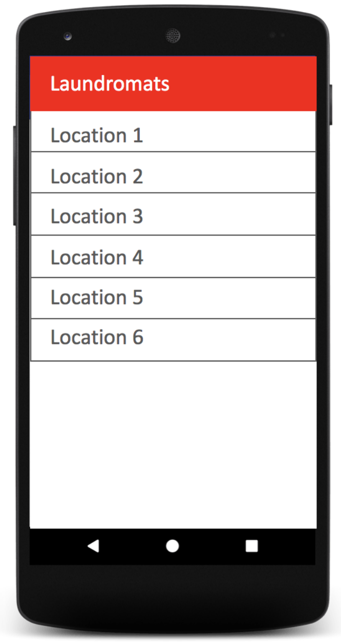
### 2.1.2 Front-End Mobile Application

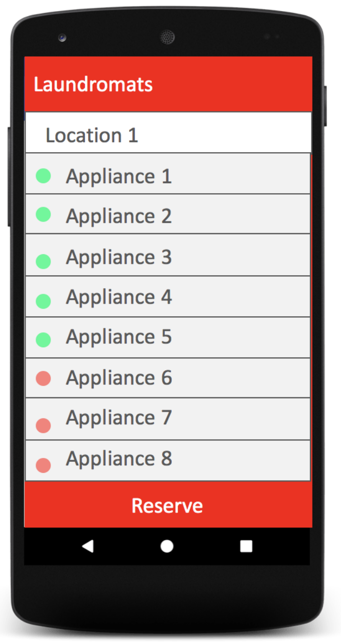
UML Use Case Diagram -

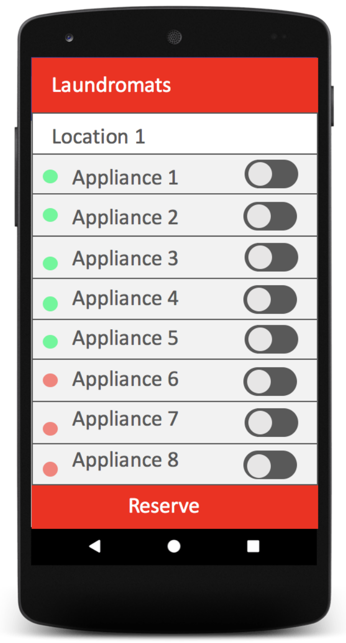


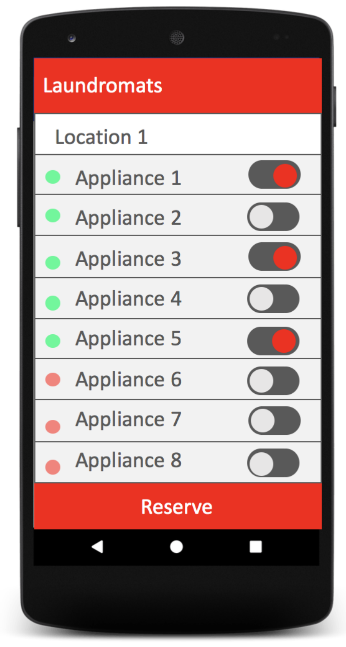
Proposed Screen Sketches:

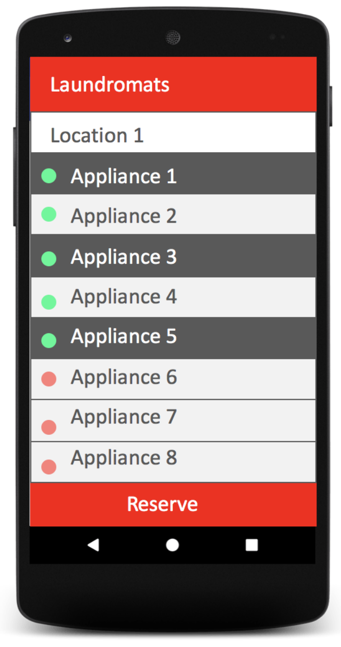


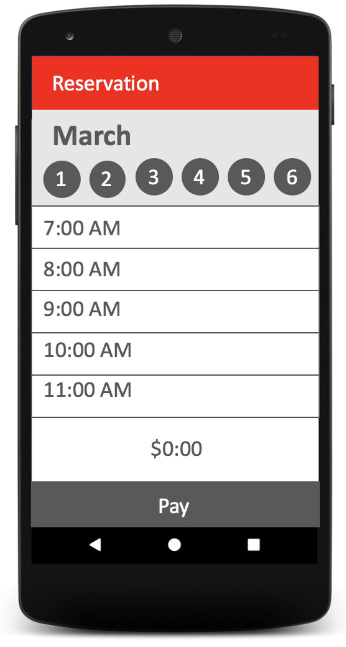


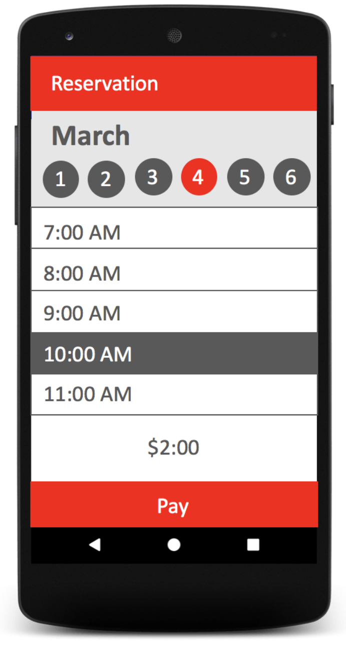


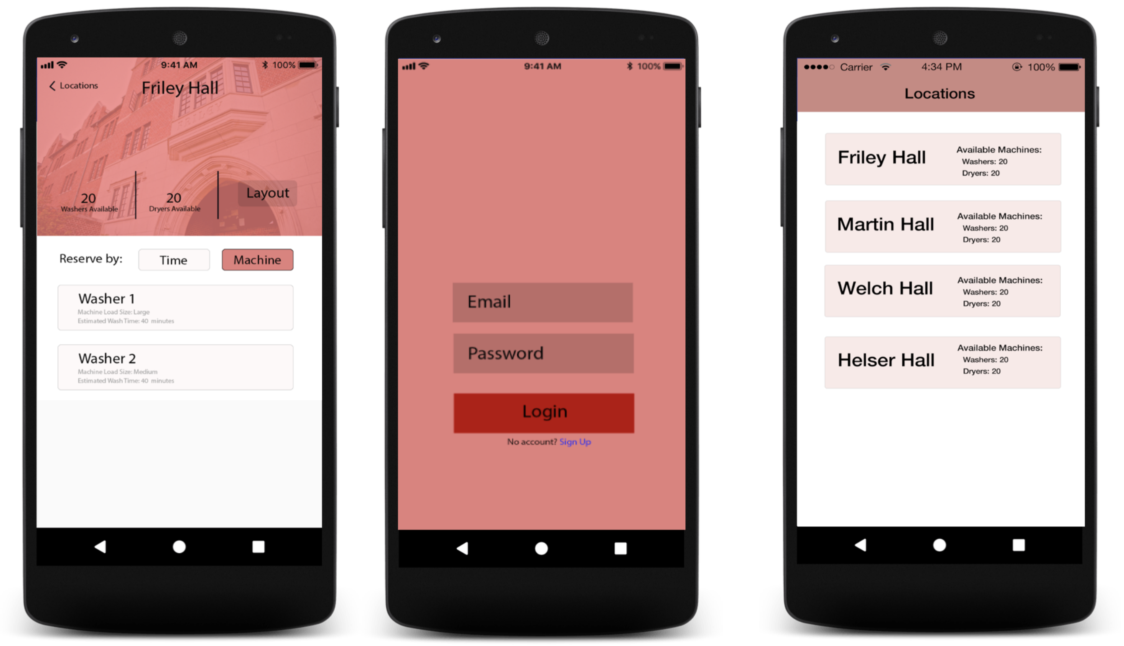












The mobile development team is now going through the process of deciding how we want our application to flow from screen to screen and how the user goes about creating a reservation at a laundromat. The first way that the team did it was to start the app on a login screen, have the user log in, pick a laundromat, pick a machine at that laundromat and reserved that machine. After thinking about this method for a few days the team believed that the amount of screens that the user has to go through was to high so we started rethinking our design. One of our non functional requirements was to make an application that flowed smoothly and allowed the user to accomplished what they wanted in a simple fashion.

For the first demo the mobile team will have with the client, the android and iOS applications won't be exactly the same because we want to give the client an idea what the route we want to take. Doing this will allow the clients to see more that is possible and give us feedback on what should be added or changed.

### 2.1.3 Hardware

The hardware team will first set up the raspberry Pi. Then the team will use the raspberry Pi to build a testbed based on the breadboard. After deployed the raspberry Pi which includes Linux Raspbian and Operating System essential setups along with network setups, we will use the Python to program the socket communication channel and embedded control which used for AWS IoT communication and GPIO circuit respectively.

For current stage testing, we are trying to set a communication channel between Raspberry Pi and AWS IoT cloud. On the other hand, the breadboard testbed along with a simple circuit consisted of LED and some resistors are used for GPIO testing.

The only standard which is relevant to IEEE standards in our Hardware component is IEEE 802.11ac (WiFi). The Raspberry Pi has a built-in WiFi model which we use to connect to the network through wireless channel.

## 2.2 Design Analysis

– *Discuss what you did so far*

– *Did it work? Why or why not?*

– *What are your observations, thoughts, and ideas to modify or continue?*

– *If you have key results they may be included here or in the separate "Results" section*

*-Highlight the **strengths, weakness**, and your observations made on the proposed solution.*

### 2.2.1 Back-End Server

So far, the back end development team has succeeded in creating a basic Spring Boot project that allows for basic object manipulation. It provides a way via REST calls to manipulate these objects, and it saves the manipulation in a local database. The remote database has just started being hooked up. The Spring Boot project layers have already been divided into components based on how our understanding of the project will be currently, and these components will grow over time as new requirements are added and as the project grows. The current approach is working very well, and it should be very simple to finish up enough of the project to have a working demo of the solution by the end of the semester.

There are still several things that still need to be set up however, such as Spring Security, different json object mappers that allow objects to be displayed in a better way, and functionality of what each component will be able to do needs to be further fleshed out. We also will need to figure out how to host the back end on Amazon Web Service, as well as creating a production database on AWS. Finally, we will need to create API documentation, probably through a service such as Swagger. However, the approach we have initially started on makes it easy to modify and extend the different segments of the back end, as well as work on multiple things at the same time without the work of two team members conflicting.

There are many strengths in our current design. One strength is that Spring Boot is an architecture that can take a pounding from users. By dividing up the application into separate parts, multiple users can access the application simultaneously without the fear of it crashing, and the user load can be spread out over all of the components more evenly than if they were combined. Another strength is the modularity of the design. Even though the application is divided into three main layers, since each layer is made up of different components, if we want to disable an API we just need to go to the controller layer and comment out the @Controller annotation. The modularity also makes extension easier, because there is a common strategy that all of the code follows, and each portion of the back end is made easy to find due to this breakup. Another strength is that the back end does all of the heavy lifting, and it is in one centralized location for extension into metric analytics that do not need to be running on the average user's phones. One final strength is that because the back end is just a server, it can run anywhere. We can have it set up on a server, in the cloud, or on our own computers for development and testing.

There are also a few weaknesses with the current design. Even though the back end is a centralized point for analytics, this also makes the back end a single possible point of failure. If it ever goes down, the entire solution goes down with it. Another weakness is that even though this overall solution was designed for industrial scalability, it may be over-engineered for the initial creation of the company that will be using this solution.

### 2.2.2 Front-End Mobile Application

So far the mobile development teams have implemented several different ideas on how we want to go about handling a user reservation and how the application should flow. Everything that we have worked on so far for the mobile development part is not set in stone, our main goal to to get something up to show the client and get feedback to what the client actually wants in terms of interface and application flow.

### 2.2.3 Hardware

So far we've set our Raspberry Pi up and built a breadboard testbed consists of LED and some resistors. Our hardware team also used to Python to program the communication channel which will communicate the channel with our Raspberry Pi and the AWS IoT. That works well since the AWS cloud was pretty reliable because it was provided by Amazon, and our WiFi can ensure the communication between our Raspberry Pi and AWS IoT cloud.

Through that part, we've observed that there are various operating system on the Raspberry Pi and the internet is not stable with our channel. So I think before our project, we need to make sure which operating system and language is the most convenient.

**Strengths: The breadboard testbed can be a testing machine for us to check if the LED was lighting up, then we will know whether the Raspberry Pi and the AWS IoT cloud was connected.**

**Weakness: Internet is not stable with our channel. The connecting and testing phase can only be proceed on the private network right now.**

# 3 Testing and Implementation

*Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or a software library*

*Although the tooling is usually significantly different, the testing process is typically quite similar regardless of CprE, EE, or SE themed project:*

> *1. Define the needed types of tests*
> *2. Define the individual items to be tested*
> *3. Define, design, and develop the actual test cases*
> *4. Determine the anticipated test results for each test case 5. Perform the actual tests*
> *6. Evaluate the actual test results*
> *7. Make the necessary changes to the product being tested 8. Perform any necessary retesting*
> *9. Document the entire testing process and its results*

*Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you've determined.*

## 3.1 Interface Specifications

*– Discuss any hardware/software interfacing that you are working on for testing your project*

### 3.1.1 Mobile Application/Backend Server

The backend team will provide an API for the mobile application development team, effectively allowing the frontend developers to access and manipulate the database, as well as the ability to send commands to the raspberry pi. While we haven't connected the server to Amazon Web Services yet, our interface will allow the mobile developers to reach the AWS technology. The backend team will be responsible for testing this provided interface.

### 3.1.2 Hardware

The interface we use for testing Raspberry Pi is to create a thing on the AWS IoT cloud, then through its shadow functionalities to hardcode the data and do the testing. On the other hand, we build a simple breadboard testbed for simple circuit testing. Since Raspberry Pi has GPIO ports and later we need to connect those to the wash machine, this testbed gives us abilities to simulate some of the input/outputs.

## 3.2 Hardware and software

- *Indicate any hardware and/or software used in the testing phase*
- *Provide brief, simple introductions for each to explain the usefulness of each*

### 3.2.1 Back-End Server

**JUnit**

Seemingly standard, we will use JUnit to write unit tests for our backend server. JUnit is a unit testing framework developed for Java and has been exposed to SE students extensively over the course of our higher education. JUnit allows you to instantiate and initialize defined objects, manipulate them by calling methods native to that object/class, and assert the results based on your understanding. While spring uses the notion of dependency injection, we can easily this modularity using functional tests in the form of JUnit.

**Mockito**

Alongside JUnit, we will also use Mockito to test our backend server. Mockito is also a testing framework for Java, but allows us to further test our beans and their runtime execution. While spring manages a lot of the dependencies within the project, mockito allows you to "mock" your java objects and exercise the dependencies that exist between your "dummy" objects.

### 3.2.2 Front-End Mobile Application

**Mockito**

To extend our testing capabilities, the Android team will integrate the Mockito test framework to test Android API calls to the backend in our local unit tests.

**Earlgrey**

EarlGrey is an iOS framework that is very similar to Espresso for Android, but offers support for network requests and API calls.

### 3.2.3 Hardware

**Breadboard Testbed**

As mentioned in the above sections, the hardware components need a circuit testbed for simple tests. We choose breadboard because this I what we used during previous EE labs, it is a simple, easy-to-use testing idea for our GPIO functioning tests.

**PyUnit**

PyUnit is a unit test framework frequently used in Python programming. For, our Raspberry Pi portion, socket programming along with embedded control functions are requiring Python involved. So we decided to use PyUnit as our Python testing framework.

## 3.3 Functional Testing

*Examples include unit, integration, system, acceptance testing*

### 3.3.1 Back-End Server

Testing the functionality of the backend spring boot server will be done in a couple of different perspectives. The first will be from the mobile application's standpoint; we will need to confirm the flow of data going in both directions is being stored or manipulated properly by running unit tests with predetermined outcomes. This will include retrieving information from the database as well as storing information generated by the client. Additionally, we will need to confirm the inner workings of the server using mockito and its mock objects.

The other standpoint from which the server will need to be tested is the interaction between the hardware/microcontroller and the server itself. Again, we will need to confirm the flow of data going in both directions, as the server will send status updates to the hardware, and the hardware will provide feedback to the server going through AWS. At this point in time we are unsure of how exactly testing works with AWS, but it will need addressed.

### 3.3.2 Front-End Mobile Application

**Android Studio, Espresso, and UI Automator**

Android studio allows for both JUnit tests and instrumented tests. JUnit tests run on the local JVM and instrumented tests run on a mobile device. We will integrate both Espresso and UI Automator test frameworks to perform user interactions during the instrumented tests.

**Xcode, UI Automation, and Earlgrey**

Xcode offers built-in support for both UI, Unit and performance testing.  Apple provides a javascript library called UI Automation that can be used to perform automated tests on both the iOS Simulator and external devices such as an iPad.

### 3.3.3 Hardware

The functional test for hardware components are relevantly trivial. Since the only functional requirements to this component is to lock/unlock the machine based on the reserved time, then acknowledge the cloud server with current status. We need to test the lock/unlock functionalities with hardcoded data information sent from AWS IoT cloud, adjust the hardware accordingly.

## 3.4 Non-Functional Testing

*Testing for performance, security, usability, compatibility*

### 3.4.1 Back-End Server

The most important nonfunctional requirement that we decided on prior to any implementation was performance. This is the reason we chose to implement a web server; by handling all of the processing on the server as opposed to the mobile application, we hope to improve perceived performance of the application as well as preserve battery life of the mobile phone. While we haven’t drilled down exact metrics yet, this is a major area of concern for our team. Compatibility will also need to be tested due to the fact that we are implementing mobile applications on both the ios and android platform; however, this should not be much of an issue for the server. Security is another concern; while spring boot performs 'basic authentication' on all HTTP endpoints, we will need to ensure the security of our backend server to protect the integrity of the entire project. This is the portion of the architecture which is most vulnerable to security threats, as we will be storing personal information and potentially handling monetary transactions. While we are using a third-party system to handle online transactions, this will need to be an area of focus as we progress on the project.

### 3.4.2 Front-End Mobile Application

**XCTest / XCUITest Performance Testing**

XCTest is an automation framework for testing iOS applications. XCTest framework tests are used for UI and performance. XCUITest is used for functional testing and workflow tests.

**Android Profiler**

The Android Profiler is a tool that measures an applications CPU, memory, and network activity.

**Dumpsys**

Dumpsys is an Android tool that runs on android devices. The tool measures the performance of the system by dumping status information about the application.

**Frame Stats**

The frame stats command prints frame rate statistics about the running application.

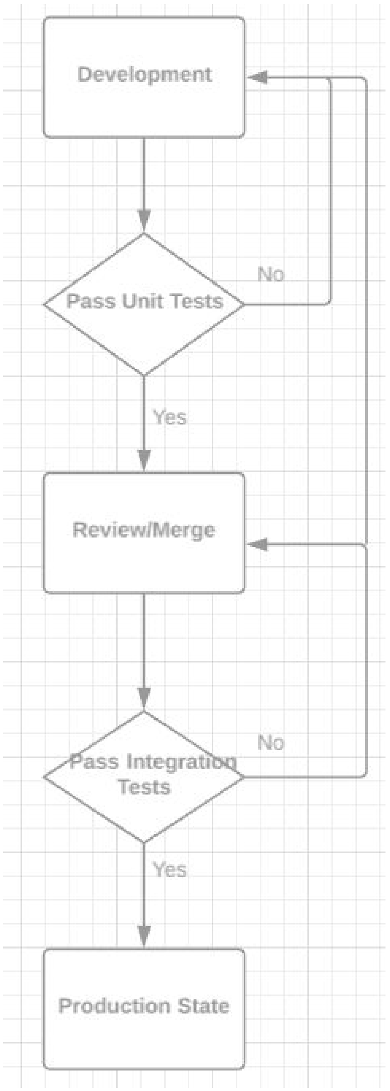
### 3.4.3 Hardware

The non-functional test for hardware components will be focused on: how to set up the wireless environment so that the raspberry pi has a clean, stable, safe network to connect with AWS IoT cloud. Since the data we will be transmitting is relevantly small (line of commands instead of video streams), plus AWS IoT provide a secure certificated communication channel. Meeting non-functional requirements from hardware perspective is not complicated.

## 3.5 Process

– *Explain how each method indicated in Section 2 was tested*

– *Flow diagram of the process if applicable (should be for most projects)*



Following development on a new feature, our teams will run local tests on their code to ensure not only that the new feature didn't introduce any new bugs, but that the new code didn't break existing functionality. When the developer passes the defined tests, the updated code will be pushed to a remote branch on gitlab. That leads us to our next testing stage.

The next stage involves optional code review and merging of the remote branch with the master branch. The code review is listed as optional depending on the nature or complexity of the new feature or code being merged. If it is a trivial update, there is no need for a code review. The merge process will typically include resolving merge conflicts to get the updates into the master branch.

After merging the code into the main branch, some (as of now) undefined integration tests will need to be ran. It is possible that a portion of the integration testing will be done manually by exploring the application searching for bugs. If the integration tests fail, clearly the developer will need to revert changes and potentially resume development.

If the updates pass integration testing, the code will be deemed to be in a 'production' state.

## 3.6 Results

– *List and explain any and all results obtained so far during the testing phase*

- – *Include failures and successes*
- – *Explain what you learned and how you are planning to change it as you progress with your project*
- – *If you are including figures, please include captions and cite it in the text*

  • *This part will likely need to be refined in your 492 semester where the majority of the implementation and testing work will take place*

-***Modeling and Simulation****: This could be logic analyzation, waveform outputs, block testing. 3D model renders, modeling graphs.*

*-List the* ***implementation Issues and Challenges****.*

# 4 Closing Material

## 4.1 Conclusion

*Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.*

The problem with shared-appliance rooms has been around as long as public laundromats and college dormitories; due to no time restrictions after entering the allotted fee, appliances often sit occupied but out of commission as users either forget about their belongings or neglect to collect them immediately. Our solution aims to enforce priorities via a scheduling application. By reserving a machine for a specified amount of time, a user will essentially broadcast to all other interested parties the status of its targeted machine. This will prevent the frustrating scenario of a user wishing to use an appliance only to find it is currently unavailable.

Our approach consists of a mobile application connected to a server which retrieves data from the cloud and presents it to the user in the form of a calendar. The mobile application will send requests to the server (for example, when reserving an appliance) which will communicate with the Amazon Web Services and translate the request to the SDDEC18-17 19 hardware controller. In the microcontroller will reside software capable of locking or monitoring the status of the targeted appliance

## 4.2 References

*This will likely be different than in project plan, since these will be technical references versus related work / market survey references. Do professional citation style(ex. IEEE).*

## 4.3 Appendices

*Any additional information that would be helpful to the evaluation of your design document.*

*If you have any large graphs, tables, or similar that does not directly pertain to the problem but helps support it, include that here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc. PCB testing issues etc. Software bugs etc.*