# IoT Remote Monitoring Mobile App for Commercial Appliances

PROJECT PLAN


**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 10th, Version 1

# Table of Contents

# List of Figures

N/A

# List of Tables



# List of Symbols

N/A

# 1 Introductory Material

## 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 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 Operating 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 Intended 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 Other 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:* Spring Boot server will be setup by end of January

*Phase 2 Delivery Date:* Spring Boot REST API will be created and database will be populated with data for users, locations, and reservations by end of February

*Phase 3 Delivery Date:* Server - Client (server to mobile) connectivity will be established by the end of March

*Phase 4 Delivery Date:* Server - AWS IoT (server to web service) connectivity will be established by the end of April

*Phase 5 Delivery Date:* AWS Iot - Server - Client connectivity will be established by the end of September

*Phase 6 Delivery Date:* Implementation of payment transaction platform will be established by the end of October

**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.

*Phase 1 Delivery Date:* A raspberry pi will be able to receive a command line signal to turn an LED light on/off by the end of February

*Phase 2 Delivery Date:* A raspberry pi connected to a lamp (mimic functionality of a washing machine) will be able to receive a command line signal to turn lamp on/off by Mid March

*Phase 3 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a lamp on/off by the end of March

*Phase 4 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a washing machine control board on/off by the end of March

*Phase 5 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a portable washing machine on/off by the end of September

*Phase 6 Delivery Date:* A raspberry pi will be able to receive portable washing machine usage data such as washing cycle by end of November

# 2 Proposed Approach and Statement of Work

## 2.2 Functional Requirements

- Locking Mechanism: The user must be capable of effectively locking the appliance via the mobile application.
- Payment Transactions: The mobile application must make use of a 3rd party payment system allowing users to lock and reserve an appliance.
- Data Tracking: The data from the appliances must be kept up-to-date and maintained in the cloud, allowing the user to browse available appliances on the mobile application.
- Appliance Control: The microcontroller must be capable of receiving requests from the mobile application and executing them appropriately by interacting with the 'smart' hardware component of the appliance.

## 2.3 Standards and Constraints Considerations

As a group we have decided on certain non technical requirements/constraints. Two of the constraints are creating a mobile application for android and iOS that does not use up a lot of battery or mobile data when the user is using the application. Another constraint that we have is a request from our client when it comes to the use of AWS. After a certain amount of accesses to the AWS server that start to charge you and so ou client requested that we keep the amount of times that we need to accesses the server to a minimum to keep the cost of the service down.

As for the standards, each team will use different coding standards depending on the language they will be using. The iOS team will be complying with the API Design Guidelines created by Apple Inc. The Android team will be complying with the ASOP Java Code Style for Contributors created by Google Inc. The hardware team will be complying with GNU Coding Standards. None of the standards that our three sub teams will be using are more of the industry standard and aren't considered unethical. Using these standards when coding for our project will allow our code to be clear and readable to other people in our group and people from outside the group.

## 2.4 Previous Work And Literature

The project that we are assigned is mostly related to the general idea of Internet of Things (IoT). Internet of Things (IoT) refers to the network connected objects, which in particular, the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors etc.

The term "The Internet of Things" was coined by Kevin Ashton from Auto-ID Labs at MIT in 1999. However, the idea of IoT was promoted decades before that. One of the very first examples of an Internet of Thing was developed in Carnegie Melon University: developers

made a Coca Cola machine which was connected to the inner network, people could connect to it through the network to check the status of drink products[1].

Since then, more and more things have been connected to the internet. As the underlying technologies gradually become more mature, the number of IoT devices & projects grows exponentially. According to a news post by IEEE SPECTRUM, there will be nearly 50 billion individual devices connected as IoT network in 2020[2].

Back to our project, the schema we have planned for our project is basically a terminal-cloud-terminal model of internet of things. Similar to the Coca Cola machine mentioned above, this is perhaps the most classic way to deploy an IoT project. Nonetheless, as the mobile application & cloud computing techniques become more mature and more individual developable, we slightly shifted the 'terminals' to be smart phones instead of large computing machines; and the 'cloud' to be third-party web services (such as AWS, Blue Mix etc.) instead of inner network connections. This way, we can build a full-lifecycle, spatial distributed IoT project which is similar to but not exactly like the 'ancient drink machines'.

### 2.5 Proposed Design

**Mobile Application**

To create a cross-platform mobile application, one of two approaches may be taken: Native approach, or Framework approach. The native approach refers to using the native or proper programming language for each platform. The current native programming language for iOS is Swift with the Objective-C being the native language prior to Apple's release of Swift. The everlasting native programming language for Android is Java, however recently Google announced that they were officially adding support for the kotlin programming language. Non-native programming languages for mobile development are often used as for cross-platform development in which only one set of source code needs to be maintained. Examples of cross-platform languages include Xamarin with Visual Studio. Our client had initially recommended the use of a single cross-platform language, however since our team has experience with Android - Java and iOS - swift, the native programming languages will be used.

**Web Server**

For the server that will support our project by doing many of the heavy calculations and connecting to a database for storage, there are many options available. Some options considered were Node, PHP, and Java Spring Boot. One person on the team had considerable experience with Java Spring Boot and all members of the team were relatively familiar with programming in Java, so it was decided that we would use Java Spring Boot. To satisfy the requirements of both connecting to the mobile application and the IoT, it was decided that the web server will be a REST API. The REST API will be able to be called by the mobile application to provide information about the states of commercial

applications connected to the solution, and will also be able to be hit by AWS IoT lambda expressions to update the state of connected commercial applications in the database.

The internal architecture of the web server will be divided into 'controller', 'service', and 'repo' segments that will do specific tasks. The 'controller' layer will define the API and accept requests, passing information onto the appropriate 'service' layer segment to fulfil the requests. The 'service' layer will do the bulk of the business logic, making calculations or calling appropriate 'repo' segments as needed. The 'service' layer will return results to the 'controller' layer. The 'repo' layer will allow querying from the database, and will return results to the 'service' layer. The 'repo' layer will be segmented up so that each object that will be stored in the database will have their own segment in the 'repo'. The objects that will be stored in the database and have their own 'repo' segment will be stored as 'entity's.

**IoT-connected Hardware Controller**

For the hardware portion of our project we want to use a microcontroller, whether that be a raspberry pi or an arduino yun, to pull the information that we want off the washing machine/dyer boards. When talking with our client we had the idea to try and get directly connected with the board but with further brainstorming we realized that connecting to the board might go beyond the scope of this project unless a USB connect was present. Taking this into consideration and suggestions from our advisor we learned that another senior design group has been able to obtain information from a washing machine using external sensors. Being able to connect directly to the washing board would be simpler but if it is not possible to get connect directly to the washer board we would have to use the alternative of external sensors. Once the microcontroller is connected to the commercial appliance, the state of that appliance will be sent via the AWS IoT and lambda function calls to the web server.

### 2.6 Technology Considerations and assessment

When we were considering what technologies we were going to use for the solution, we had to account for several strengths and weaknesses of our design, as well as make trade-offs based on what we found.

In our design, there are many apparent strengths. One is that we will be supporting both IOS and Android for our application. Another strength is that our design is scalable. New commercial appliances may be added added as desired to the AWS IoT. Lambda expressions from AWS IoT are spun up on the fly as needed. If the web server is overloaded, then another web server can be spun up to take some of the traffic by applying a load balancer in front of the server. The mobile component can be added over many phones, and will all connect to the central server network. Also, one final strength of the design is that the mobile component is battery-friendly. Since many of the larger calculations are made by the server, the phone's CPU is not used as much as it might be otherwise, saving battery and allowing the mobile component to be used on older or less powerful phones.

There are several weaknesses in our design as well, however. The first one is that the microcontroller that connects to the commercial appliances will need to be manually connected to each appliance, and it is possible that connecting microcontrollers to appliances may be 1 to 1, meaning each new appliance added would need its own microcontroller to connect to AWS IoT. Another weakness is that the dedicated database itself cannot be expanded. The database and its speed will be the choke point for the growth of our solution.

When deciding on our technologies, we had to consider several things. The first was whether we should use a relational or graph database. A relational database would allow fast querying and was more widely known by the team, whereas a graph database may allow for faster querying or less table joins. It was decided that we would use a relational database because more of the team knew SQL, SQL was supported across all relational databases (including the one supported by AWS), the querying we would be doing would most likely be dealing primarily with entire objects and not as much with the relationships between them, and AWS' graph database was queried from a language that none of the team members had experience with. Another thing that has to be considered is what microcontroller we should use. This is still being considered, as we will have access to several and testing has not started yet.

We felt like our solution was the most expandable, and offered the most benefits for our experience. Another possible solution would be to use a graph database instead of a relational database, however we have less experience with the AWS graph database than we do with relational databases. We could also have created our server in PHP or Node. However, with PHP the scope of available services can be limited. This may not have been an issue for our proof of concept, but when the project scales this could have become a pain. Also, for example, using collections in java is much easier than in PHP and could potentially increase our performance as we optimize queries on result sets returned by the database. Also, asynchronous processing could be necessary as our project scales up, and php struggles in this area compared to java. Node also would have been a decent choice; however, it falls behind spring boot in a couple of key areas, one being security. While spring configuration is typically more cumbersome than node, we had an experienced backend team lead to handle most of the configuration issues a newcomer might encounter.

It was also possible that we could forgo the server entirely and just have all of the information loaded straight from the database onto the application, however this would force more stringent CPU requirements on the phones themselves as well as use more of the phone's battery.

The mobile development team has looked through several options to develop the android and iOS applications. In the end we chose to make both applications using the native languages Swift for iOS and Java for Android. Before deciding on that we were going to use the swift and java to develop the applications we looked at frameworks such as Xamarin,

Cordova and Appcelerator that allow a developer to create an application that can be compiled to both the iOS and Android platforms.

Xamarin is a platform that allows a developer to use C# to create a single code base and compile for both iOS and Android platforms. According to the Xamarin website you as a developer are able to do anything with C# and the Xamarin framework that you can do with Swift and Java. The mobile development team decided to not go with Xamarin because we have not had any experience with C# and the Xamarin framework. We didn't want to take the time to learn a new framework and new language to be able to do what we want to given our prior knowledge with swift and java.

Next we looked at Cordova and Appcelerator which are frameworks that allows a developer to create a single code base using HTML, CSS and Javascript for Cordova and only Javascript for Appcelerator and compile it to to work as iOS and Android applications. Again same as when we looked at the Xamarin framework the mobile team is more versed in Swift and Java and have less experience with the two frameworks and the languages required.

In the end as a group we decided to create both the iOS and Android applications using the native languages Swift and Java. We chose this option because we both have experience with creating android and iOS application using the native languages. By choosing to develop the iOS and Android applications using the native languages we save ourselves some time learning technology to advance our project.

### 2.7 Safety Considerations

*Include any safety concerns you find applicable to you project.*

### 2.8 Task Approach

Our group of six people is being partitioned into three subgroups: frontend, backend, and hardware. Each subgroup consists of two people, with one individual assuming the lead role. Our project leads are the individuals with the most experience and ability in that particular role. During the initial development stage, our three subteams will work on their own for the most part. Each week we will meet to discuss the status of each group, as well as what needs to be done to stay on pace with the rest of the group. Our individual team leads will delegate tasks as they see fit. Each team has their own set of tasks, whether they are managed internally or on the gitlab issue board. The integration stage will be much more of a group effort; however, it is possible that new bugs arise during this stage and we may need to partition off for development rework.

### 2.9 Possible Risks And Risk Management

1. **Time Restrictions**: As with any project tied to a deadline, there is a potential for the team to fall behind schedule and ultimately not meet the level of expectation placed upon us by the client. We also all have other responsibilities, and it is important that the team stays cohesive during seemingly downtime throughout

the semester. We plan on mitigating this risk by meeting on a weekly basis as well as keeping our scrum board up to date in gitlab.

2. **Hardware Compatibility**: As the client's' choice of hardware component for the appliance is still up in the air, we face numerous risks associated with our hardware. While it is possible the component is capable of plugging into our microcontroller via USB, we need to be aware this is not guaranteed. If we have to find another solution, we run the risk of falling behind at a critical point in the semester as it could be a couple of weeks before we are notified of the selection.
3. **AWS Restrictions**: Due to the fact that our project is going to be cloud based, we have to try and limit the amount of traffic we are running across the cloud for our clients sake. While he promotes the use of Amazon Web Services, he acknowledged that he would like us to try and minimize the amount of financial resources that need to be allocated for Amazon.

### 2.10 Project Proposed Milestones and Evaluation Criteria

**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:* Spring Boot server will be setup by end of January

*Phase 2 Delivery Date:* Spring Boot REST API will be created and database will be populated with data for users, locations, and reservations by end of February

*Phase 3 Delivery Date:* Server - Client (server to mobile) connectivity will be established by the end of March

*Phase 4 Delivery Date:* Server - AWS IoT (server to web service) connectivity will be established by the end of April

*Phase 5 Delivery Date:* AWS Iot - Server - Client connectivity will be established by the end of September

*Phase 6 Delivery Date:* Implementation of payment transaction platform will be established by the end of October

**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.

*Phase 1 Delivery Date:* A raspberry pi will be able to receive a command line signal to turn an LED light on/off by the end of February

*Phase 2 Delivery Date:* A raspberry pi connected to a lamp (mimic functionality of a washing machine) will be able to receive a command line signal to turn lamp on/off by Mid March

*Phase 3 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a lamp on/off by the end of March

*Phase 4 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a washing machine control board on/off by the end of March

*Phase 5 Delivery Date:* A raspberry pi will be able to receive a signal from AWS IoT to turn a portable washing machine on/off by the end of September

*Phase 6 Delivery Date:* A raspberry pi will be able to receive portable washing machine usage data such as washing cycle by end of November

### 2.11 Project Tracking Procedures

We are going to track our progress using gitlab's built-in issue board. Here we can create, assign, and track the status of tasks as development continues throughout the semester. The creation and assignment of tasks will typically happen at one of our weekly meetings as we adhere to the scrum philosophy. Using this technology, we can see what everyone is working on during the week as well as request either help or a code review prior to merging new software into the master branch.

While gitlab will host most of the development-specific tasks, we have a timeline laid out detailing functional requirement deadlines. In addition to using gitlab, we will track progress at a higher level by evaluating which tasks align with a certain deadline. Using this information, we can ensure the status of our current project.

### 2.12 Expected Results and Validation

**Mobile Application**

The Android and iOS mobile application are responsible for hosting the calendar-based reservation system to checkout external appliances. The application must feature a login system that stores user reservation data and secures payment transaction information. If a user would like to reserve an appliance for a given period of time, the application must allow for them to select a day and time frame, generate a code in response to the reservation time and machine #, and send reservation lock data to the IoT stack. During a

reservation time, the application must send signal data to the IoT to update usage status and record usage analytics.

To validate that our iOS mobile application performs as expected, the iOS team will be writing a series of iOS Unit Tests following apple guidelines at https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/04-writing_tests.html. Xcode, the IDE for iOS offers a test navigator to test several components of an app including: application model, asynchronous methods, interactions with libraries, interactions with system objects, UI, and performance.

To validate that our Android mobile application performs as expected, the Android team will be writing a series of Android Unit tests following training from https://developer.android.com/training/testing/unit-testing/index.html.

**Web Server**

The job of the web server is to connect the commercial appliances in AWS IoT to our database and make that data available to both of our mobile applications. The main purpose of the web server this first semester is to store information about the machine such as if it is on and if a user has reserved a certain washer/dryer at a particular location for a specific time. To validate that our database works we will show that with the mobile applications we will be able to reserve a machine by changing a field in the data and showing that we can turn the machine off and on from the mobile applications.

**IoT-connected Hardware Controller**

The desired outcome for the hardware portion of the project would to be able to do the following things. One, to be able to turn the washing machine and dryer on. Two, to be able to pull various information from the washer/dryer boards such as power consumption for analysis. To show that this functionality work by turning the washer/drying and displaying the information pulled from the washer/dryer board on the mobile application.

## 2.13 Test Plan

| Functional Requirements | Test Plan |
|---|---|

| | |
|---|---|
| Reservation Lock: mobile application must generate a number to be entered into the appliance | This could be separated into two test scenarios:<br><br>1. For the mobile app side: make a fake signal from the cloud, try to generate the number on the mobile app. (Eventually can send multiple signals pretending different users, test this on multiple smartphones, also across platform between IOS and Android)<br><br>2. For the microcontroller side: hardcode a number from the cloud, see if this number is able to unlock the machine. Also, test different numbers in terms of reservation time, then test if those numbers could unlock the machine respectively. |
| Reservation fees - payment transactions | This could be tested by creating a test account, process the reservation and finish the payment to see if the balance changed on the test account. In addition to this, test if the reservation can be successfully marked on the cloud after payment. |
| Hardware appliance use information must be tracked | This can be tested in three phases:<br><br>1. Manually run the wash machine, test if the microcontroller can track the information such as running time.<br><br>2. After phase 1, add another layer onto it. Manually run the wash machine, to see if the information that been tracked by the microcontroller can be send to the cloud server and database.<br><br>3. Combine two phases above or create a set of fake data on the cloud. Then try to update these information on the mobile terminals. Test if these information can be presented on the mobile apps. |
| Scheduling and real-time data are synchronized | These will be synchronized at the database level. This may be tested by opening two database transactions at the same time and |

| | |
|---|---|
| | making sure that only one of them is able to reserve a commercial appliance at a time. |
| User profiles | This can be tested by creating a test user account. Editing different user info to see if the cloud side can track the updates. |
| Server maps to microcontroller given a request from the mobile platform | Similar to the reservation lock test cases. Create test data sets from mobile application to the cloud server, then create test data sets from server to request the microcontroller. |

Table 1: Test Plan

# 3 Project Timeline, Estimated Resources, and Challenges

## 3.1 Project Timeline

Due to the nature of Senior Design, our project will follow a revised version of the waterfall model. The waterfall model is a linear sequential design approach for the development of hardware and software. Throughout the next two semesters, our timeline will be divided into five phases, with several phases overlapping.

Phase 1: Requirements

The first phase of our timeline consists of project planning. Items include capturing software and hardware requirements, identifying project scope and end goals, and building team expectations and roles.

Phase 2: Design

The second phase of our timeline focuses on design thinking to create sketches, diagrams, and simplistic prototypes that resemble features to be implemented in the end product. Activities such as design thinking and design documentation occur during the second phase.

Phase 3: Development

The third phase of our timeline is the development stage. Unlike standard waterfall methods, our development stage will be a bit revised since there are deliverables that must be presented during the last week of Senior Design 1 and Senior Design 2. Therefore, we will have two development phases, one per semester. During semester 1, development will focus on prototyping the core functionality of the mobile application, interfacing the hardware components, and forming a connection between the hardware and software.

During semester 2, development will focus on incorporating data analytics into our mobile application.

Phase 4: Testing

The fourth phase of our timeline is the verification and validation stage. During the testing stage, Unit tests will be written for both the iOS and Android mobile platforms. Similarly, hardware components will be tested following the methods outlined in the validation and verification section of the project plan.

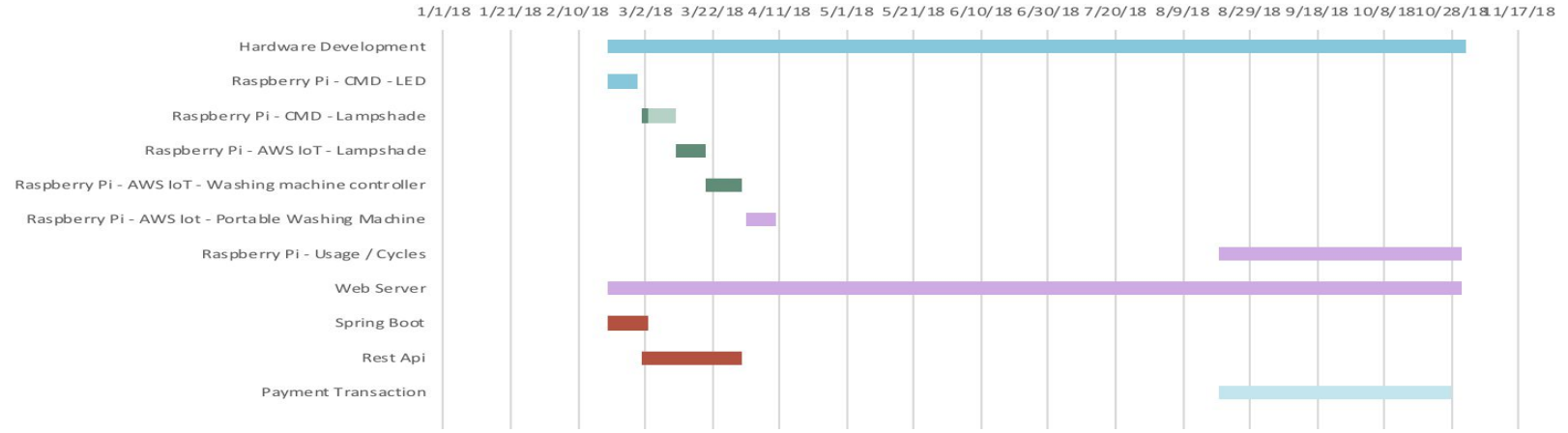
Phase 5: Deployment

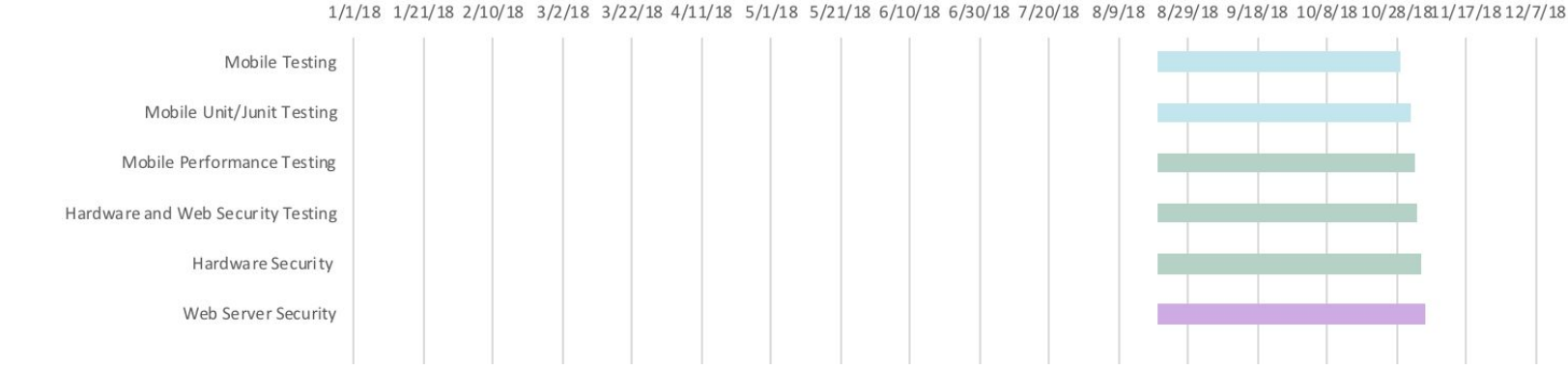
The fifth phase of our timeline is the deployment stage. The deployment stage is where our milestone deliverables will be demonstrated as part of a panel presentation, client presentation, final IRO presentation, and final client presentation.

The gantt chart provides a more comprehensive timeline and will be updated per change in expectations.

**[Gantt Chart on Next Page]**


1/1/18
1/21/18
2/10/18
3/2/18
3/22/18
4/11/18
5/1/18
5/21/18
6/10/18
6/30/18
7/20/18
8/9/18
8/29/18
9/18/18
10/8/18
10/28/18
11/17/18
Project Planning and Requirements
Identify Team Expectations and Roles
Identify High-Level Overview of Architecture
Identify Functional and Non-Functional Reqs
Research And Analysis: Front-end, backend, hardware
Identify Client Expectations and Project Scope
Project Plan v1
Design
Design Thinking Pre Workshop
Design Architecture Diagram
User Interface Sketches
Design Thinking Post Workshop
Development
Mobile Application Development - User
Google Maps Screen
Reservation Screen
Payment Screen
Mobile - Database Connectivity
Mobile Application Development - Client
Data and Analytics Screens


1/1/18
1/21/18
2/10/18
3/2/18
3/22/18
4/11/18
5/1/18
5/21/18
6/10/18
6/30/18
7/20/18
8/9/18
8/29/18
9/18/18
10/8/18
10/28/18
11/17/18
Hardware Development
Raspberry Pi - CMD - LED
Raspberry Pi - CMD - Lampshade
Raspberry Pi - AWS IoT - Lampshade
Raspberry Pi - AWS IoT - Washing machine controller
Raspberry Pi - AWS Iot - Portable Washing Machine
Raspberry Pi - Usage / Cycles
Web Server
Spring Boot
Rest Api
Payment Transaction


1/1/18
1/21/18
2/10/18
3/2/18
3/22/18
4/11/18
5/1/18
5/21/18
6/10/18
6/30/18
7/20/18
8/9/18
8/29/18
9/18/18
10/8/18
10/28/18
11/17/18
12/7/18
Mobile Testing
Mobile Unit/Junit Testing
Mobile Performance Testing
Hardware and Web Security Testing
Hardware Security
Web Server Security

### 3.2 Feasibility Assessment

**Project Description**:

The goal of our project is to use the IoT consists of a network of physical hardware to help consumers use the washing machine and dryer remotely. By using the mobile applications(Android and IOS), the consumers can reserve the washing time and enter the generated code to turn the machine on. Whole operations can be accomplished remotely so the consumer won't waste time travelling shared laundry.

**Possible Challenges and Solutions:**

**1. Redundant Operation:** The challenge is that we will add the operation on the mobile application to ensure the consumers can turn on the machines remotely. But this step seems redundant since even if the consumers reserved the washing machine, they still need to bring the clothes there. If we add this operation, the project will be extremely complicated. We need to connect the board from the wash machine to our application and show the information. This will be a hard EE work.

The solution will be a little tricky. We can buy an electric locker with specific functions which is able to connect our mobile apps. Adding the locker on the lid will be easy to lock the washing machine. When we are going to use the washing machine, we can use the mobile to unlock that locker and the lid can be open.

2. **Rebuild the Laundry:** This challenge we are facing is that many laundry are not able to connect to wifi since the old buildings will not provide the internet in the laundry. Also, rebuilding the laundry and adding internet connection will be a big cost.

The solution is that what we are working is just a project. We can work with the laundry limit to have internet connection and some 5V socket so we can connect the microcontroller to the washing machine through the socket.

**Evaluation Criteria:**

**1.Practical**

The project should not be too complicated since a it just take months so a complicated project will make our work inefficient.

**2.Cost-Efficient**

The cost of our project should not be big. And the project should be realistic, we should not change the construction of the laundry. Our main goal is to improve the situation rather than change it.

**Outcome:**

After we figure out how to solve the potential challenges, I think our project turns to be cost-effective, practical and feasible.

### 3.3 Personnel Effort Requirements

| Task | Team | Effort Level | Explanation |
|---|---|---|---|
| Initialize Spring Boot Server | Backend | Medium | While there are a number of configuration issues to hash out, our backend lead has industry experience with the framework. |
| Set up remote database | Backend | Low | Simply request a database and build out the necessary tables and relationships. |
| Configure gradle profiles | Backend | High | Our backend lead customized three gradle profiles: local, remote and prod. These basically correspond to which datasource is being utilized. |
| Outline REST API | Backend | Low | This mainly consists of identifying use cases and providing API endpoints via controllers. |
| Implement API | Backend | Medium | While some of the queries were straightforward and required minimal effort thanks to spring boot, a few implementations took an extended period of time to return the correct results. |
| Connect AWS to Raspberry pi | Hardware | High | Connecting the pi to AWS cloud proved more difficult than originally thought, with a lot of configuration involved. |
| Extract data from the washing machine component | Hardware | High | Due to the lack of intelligence in our washing machine component, reading data from the machine is going to be |

| | | | |
|---|---|---|---|
| | | | difficult. We may need to measure current flowing to/from the machine and make assumptions based on that. |
| Develop UI | Frontend | High | Our mobile solution is stemming both iOS and android platforms. We have had numerous redesigns based on feedback and collaboration between the two mobile developers. |
| Connect to server | Frontend | Low | While this shouldn't take too much time to figure out, this is a critical point in the project. |
| Implement functionality | Frontend | High | Due to redesigns and slightly altered use cases, the mobile team has spent a lot of time implementing the necessary functionality, such as utilizing google maps to search for nearby laundromats. |

### 3.4 Other Resource Requirements

Our team requested and received a dedicated remote database server to use for remote testing. We will need to host our backend server eventually as well. However, for the time being, we are running our server on localhost.

### 3.5 Financial Requirements

| **Microcontrollers** | |
|---|---|
| Arduino Yun Microcontroller | $89.95 |
| Raspberry Pi 3 Motherboard (alternate option for Arduino Yun) | $36.37 |
| **Components might associate with above** | |
| Power supply or battery | $10 - $20 |

| | |
|---|---|
| Basic electronic kit (breadboard, resistors, wires etc.) | ~$15 |
| **Others** | |
| Washer embedded board | unknown |
| AWS account | unknown |
| **Total:** | **$161.32 (for the moment)** |

Table 2: Finnancial Requirements

# 4 Closure Materials

## 4.1 Conclusion

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 hardware controller. In the microcontroller will reside software capable of locking or monitoring the status of the targeted appliance.

## 4.2 References


1. "A Brief History of the Internet of Things." DATAVERSITY, 6 Aug. 2016, www.dataversity.net/brief-history-internet-things/.

2. 18 Aug 2016 | 13:00 GMT. "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated." IEEE Spectrum: Technology, Engineering, and Science News, 18 Aug. 2016, spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

## 4.3 Appendices

*If you have any large graphs, tables, or similar that does not directly pertain to the problem but helps support it, include that here. You may also include your Gantt chart over here.*

*– Any additional information that would be helpful to the evaluation of the project plan or should be a part of the project record shall be included in the form of appendices*

*– Examples of project documentation that might be included are property plat layouts or microprocessor specification sheets germane to the proposed project.*