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

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 have 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. Battery Life for the mobile application must be minimal as the application will require access within a Laundromat
2. There must be stable Network Connection in the Laundromat for users to connect to the mobile application
3. The single-board computer has a 1 to 1 relationship with each appliance. For every new appliance added, an additional Raspberry Pi is required. Therefore scalability must be taken into consideration.

4. Washing Machine Control Board connectivity are dependent on manufacturer and model. Finding a solution that is ideal for a large majority of washing machine controller will be a challenge.
5. Without "hacking" into a washing machine, the pulling of power data will need to be done using external components.
6. The application will be mobile and thus any intensive computing should happen on the server end of the architecture.
7. Limited accesses to AWS server under Free-Tier payment plan.
8. 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: 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 August

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

Phase 7 Delivery Date: A keypad with LCD display will be connected to the Raspberry Pi. The keypad will register an input and the raspberry pi will read and send it to the server for confirmation before powering on an appliance. This will be done by October.

Phase 8 Delivery Date: The raspberry pi and keypad w/ LCD will be enclosed in a waterproof compartment and attached to a portable washing machine by the end of November.

2. Specifications and Analysis

2.1 Proposed Design

Concept Diagram

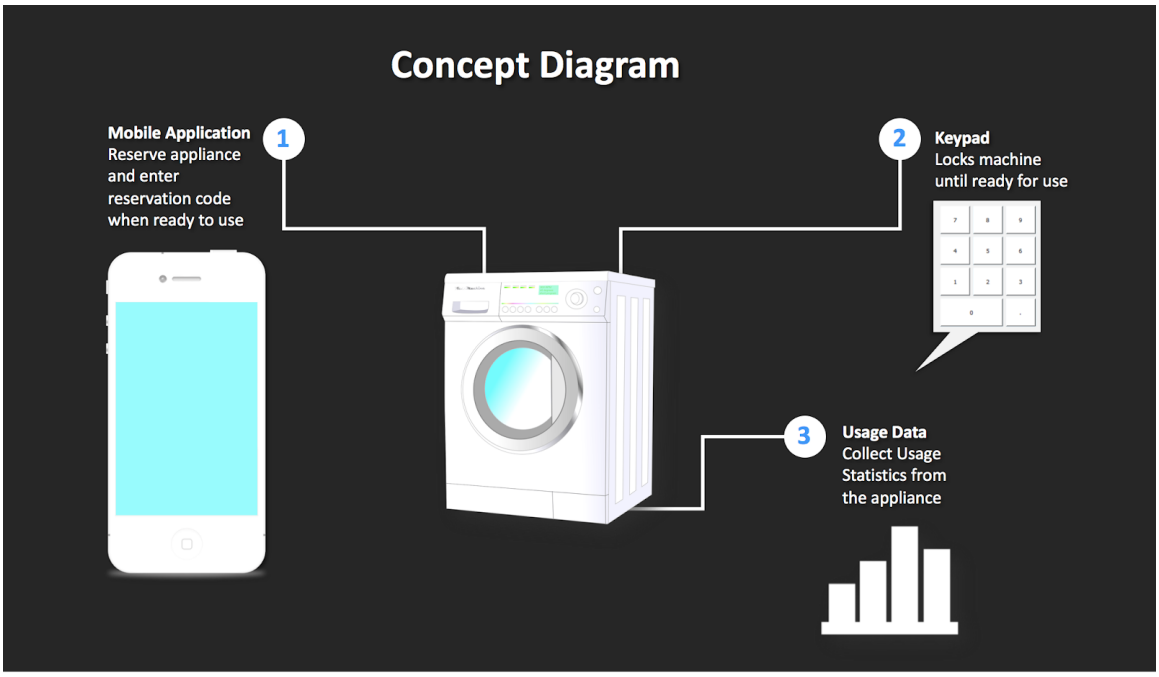


Figure 1: Concept Diagram

There were several possible ways of designing the solution to fulfill the requirements of this project, and the team had to figure out which one would be the most effective.

One design decision that was considered was whether the hardware controller should communicate directly with the database/back end or if it should use AWS IoT. It was decided to use AWS IoT because of the extendability it provides so that the database/back end can be moved or changed in the future without having to change the code for every microcontroller. It was also a suggestion from our client to use AWS IoT because they already had an AWS account, and it allowed for greater management for expanding to more than one physical location.

Another design decision that was considered was whether the solution should have a back end to centralize requests to the database or if every call should hit the database directly. It was decided that there should be a back end that will handle requests for data to allow the database location to change as needed due to updates and growth. This method also provided a method for expandability (using load balancers and more server instances) if necessary. It will also allow for analytic information to be calculated on the server instead of on mobile applications, which will reduce battery drain and provide a more centralized location for these analytical calculations.

Architecture Diagram

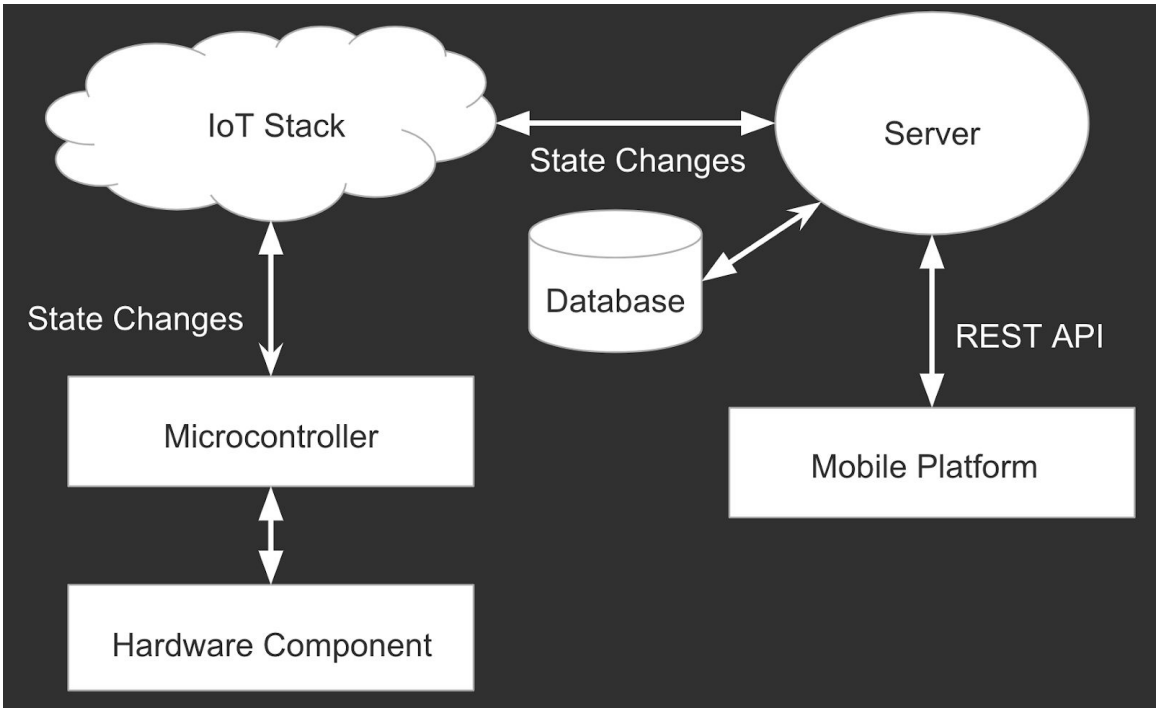


Figure 2: Architecture Diagram

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

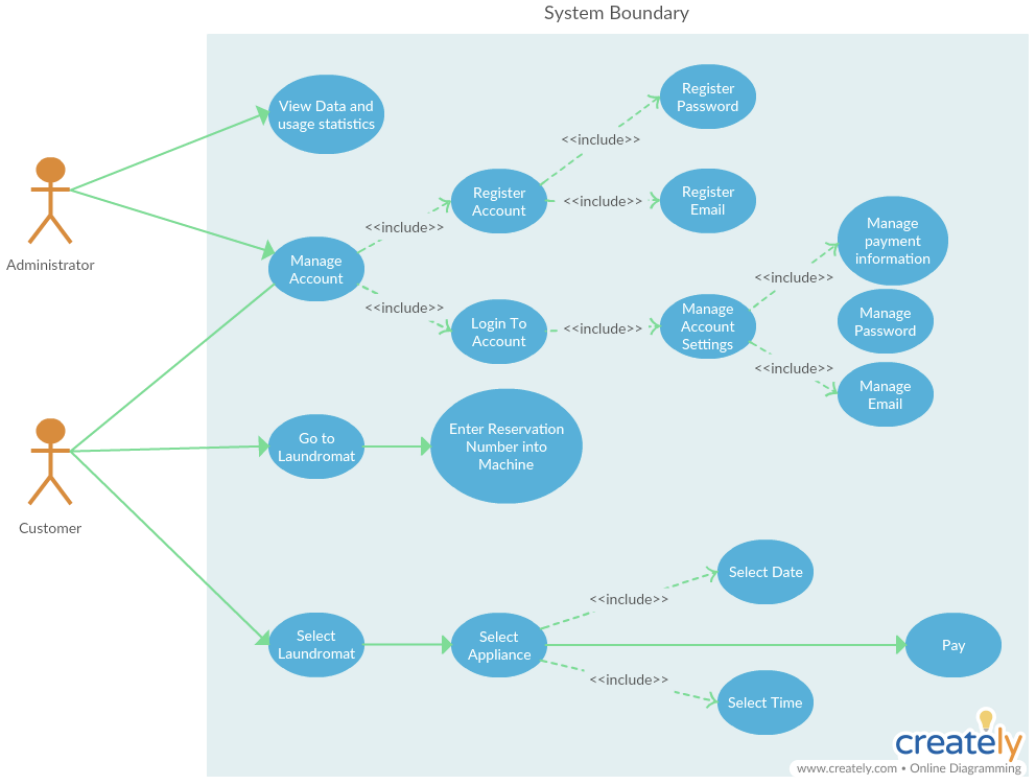


Figure 3: UML Use Case Diagram

A UML Use Case Diagram was designed by the mobile platforms team to identify the most crucial actions that a user may want to perform while using the mobile application. From a customer's perspective, it is required that our application allows them to handle account information, to view prior reservations while at a laundromat to enter reservation code, and to create new reservations. From a client or administrator's perspective, our application must allow for them to view data collected by our hardware team to analyze the performance and power usage of laundromat appliances.

Proposed Screen Sketches:

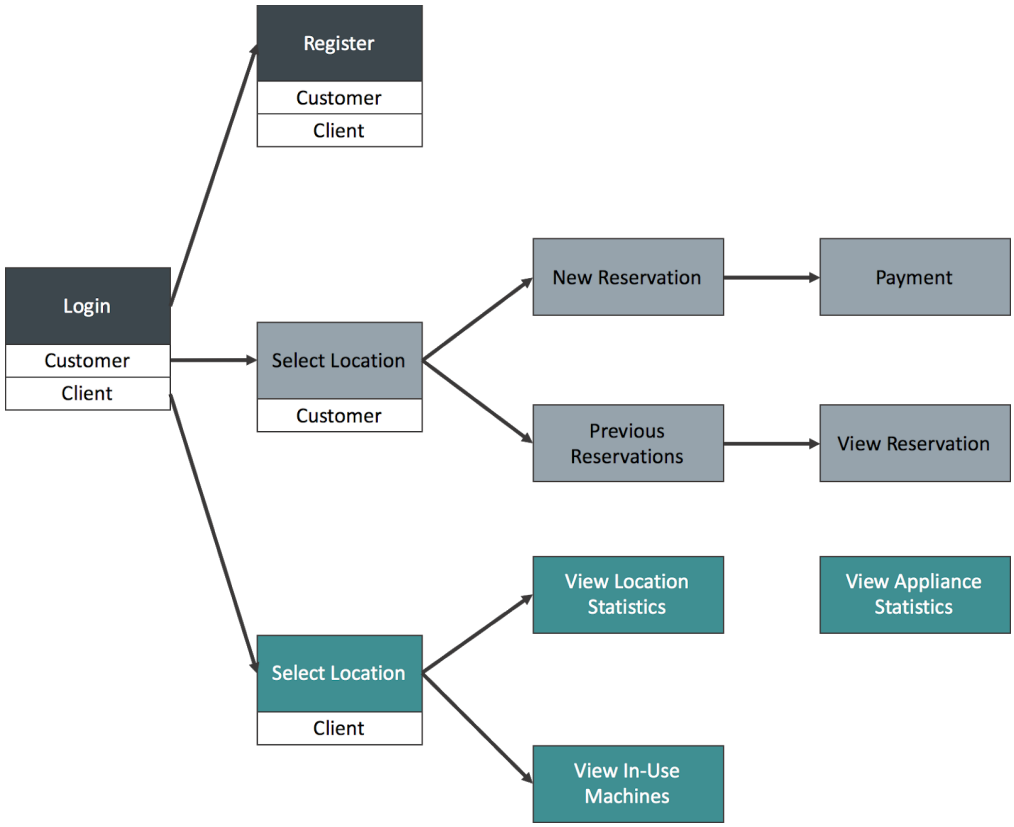


Figure 4: Screen Sketch Diagram

Our proposed screen sketch outlines the the various screens to be implemented by the mobile platforms team. Th screen sketch follows a similar model to *Uber*, the popular ride-sharing application. *Uber* introduced a dual UI application where a user may login as either a rider or a driver. A different UI will appear according to which user has logged into the application. If a customer signs-in to our mobile application, then they will be directed to a Google Maps screen where they may select a laundromat. Our maps screen will support a search bar that allows users to search for different laundromats. Additionally, they may enable user location to find laundromats within an x mile radius to them. Once a laundromat has been selected, they will be directed to a reservation form. The reservation form reserves a set of requested appliances on a specific day at a specific time. Once a reservation has been submitted, the user will receive a reservation code to be entered into the appliance at the time of use.

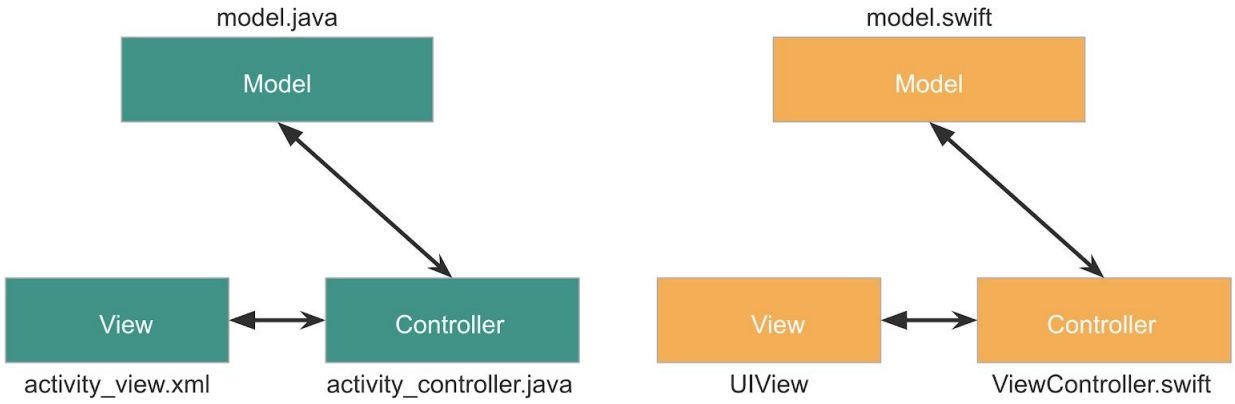


Figure 5: MVC Design Pattern

A detailed analysis regarding our choice of IDE and programming language has been described below in the following section. To develop both the Android and iOS application, we plan to use Android Studio as the IDE for Android and xcode as the IDE for iOS. The Android mobile application will be developed in the native language Java and the iOS mobile application will be developed in the native language Swift.

The application will be developed using the MVC Model View Controller Design Pattern. The image above depicts the similarities and differences between the implementation of MVC in android vs. iOS. In Android, a model.java file contains all API requests and database queries. The View is created in XML and associates id numbers with each tag. The id numbers can then be used by the controller, an activity class, to obtain input data or button actions. Likewise, a model.swift file contains all API requests and database queries. Rather than using an XML based storyboard, the iOS team will be implementing all views programmatically. The use of swift IBOutlets, or id-references, can then be used by the controller to receive input and button commands.

2.1.3 Hardware

The hardware system design can be illustrated as below. The hardware system is consisting of four components as from left to right: User Interface, Control Unit, Power Source Control, and the Portable Washing Machine. In next a couple of paragraphs, we will introduce this hardware system in more details.

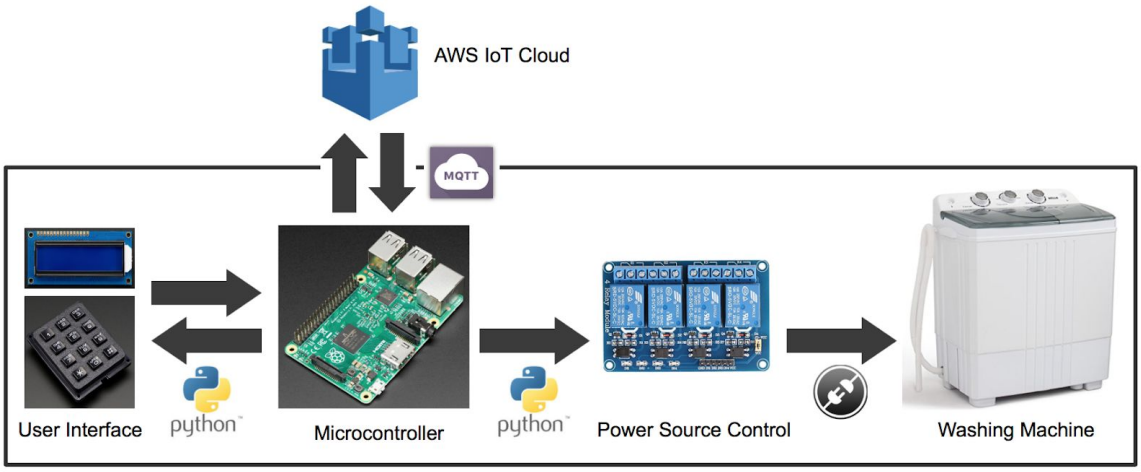


Figure 6: Component Diagram

Microcontroller

The first thing related to hardware components is comparing which microcontroller fits better to our project. As we done research on the topic of microcontrollers, we found three candidates as listed below along with spec details:

	A	B	C	D
1				
2				
3		Arduino Uno	Arduino Yun	Resberry Pi 3 Model B
4				
5	Processor	AVR ATmega328p	ATmega32u4 & Atheros AR9331	ARM1176JZF-S
6	Clock Speed	16MHz	16MHz & 400MHz	700MHz
7	RAM	2KB	64MB & 32KB	512MB
8	GPIO	20	20	8
9	Power	175mW	N/A (~300mW)	700mW
10	WiFi&Ethernet	shield	built-in	built-in
11	OS&language	none/C	Linino/C,python	Any Linux/any
12	Price	$23.64	$89.95	$36.9

We first eliminated the choice of Arduino Uno board due to the lack on computational powers to support MQTT protocol with Amazon Web Service. Also, Arduino Uno does not

come with WiFi module on board, so we will have to purchase a WiFi shield which leads to more complex work and expensive equipments.

For the rest two microcontrollers, we decided to go with Raspberry Pi 3 Model B for two major reasons:

1. Raspberry Pi comes with higher clock speed which means higher computational power it has. Although for the sake of our project, we do not need these extra clock speed. But later if this project goes into industry, a higher computational board would be used as a central hub deployed in each laundromat.
2. Since this project will eventually be given to our client and perhaps be used as part of business. Budget would be a big concern in this sense. As shown in above chart, the cost difference is nearly $50, which makes the Raspberry Pi a much better choice budget wise since both meet the least computational requirements for this project.

User Interface

The second portion comes with our hardware design is to combine the keypad/LCD screen into our system as the user interface. This user interface will allow users to enter their verification code once they arrive the laundromat. A picture of our keypad/LCD screen chois is as shown above. Both of the components will be connected to the Raspberry pi via jump wires. Since there are 40 GPIO pins on the board of Raspberry pi, it is more than enough for these two components (cost 23 pins in total) to work properly within our design. For the 16x2 LCD screen particularly, the 16 pins will be separated into 3 power source pins, 9 data bus pins, and 4 control bit pins. We will follow the diagram below.

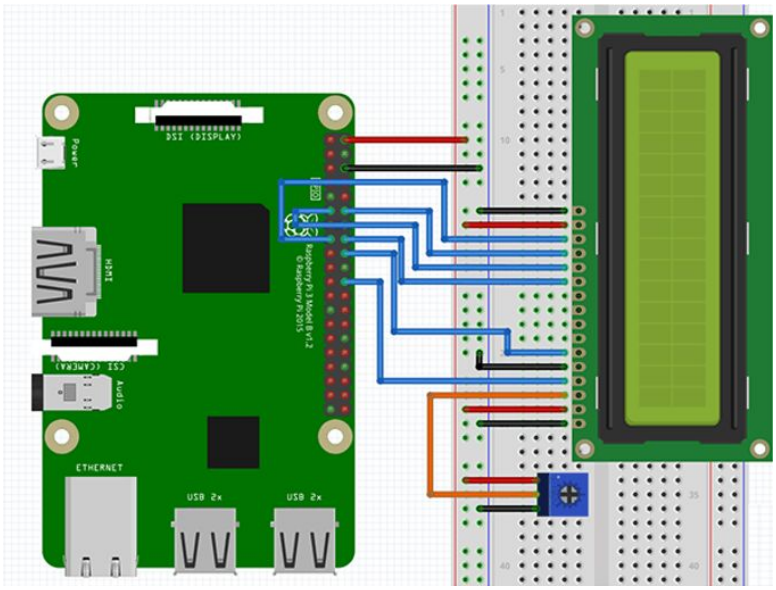


Figure 7: 16x2 LCD Design Diagram

Power Source Control Unit

The third portion of hardware design is to decide how exactly we will be control the wash machine. Since we have not gotten the actual wash machine, the strategy for now is to simply control the power of the wash machine. After research, we found that using relay in the circuit would be a decent choice to make this to work. There are 1/2/4/8 channel-relay modules available in the market (note: only considered 250V AC voltage / 10A current / 5V DC coil voltage to meet the load for our project), the costs are shown as below:

Channel Types	Price (according to Amazon.com)
	$5.80
	$6.79
	$7.86
	$8.98

The cost for either type of relay retains nearly the same. Thus our choice was not made based on the budget considerations. For our implementation, at least a 1-channel relay is needed. However, we choose to use the 4-channel relay due to two major reasons:

1. Safety reason. 1-channel relay module has potential safety issues since it always has one wire branch connected with 110V AC voltage in practice. This may cause electric shock if the circuit was not properly deployed. Therefore, we followed “at least 2-channels” rule to allow the high voltage detached from the circuit when not in use.
2. Based on the conclusion from reason #1, we decided to use a 4-channel relay module in our implementation. In this case, we would have two channels left besides two connected with power supply. The rest two channels could be used

later on to provide feedback information from the wash machine, so we set these two as “reserved channels” in our design.

As a result, we have designed a simple circuit shown as below:

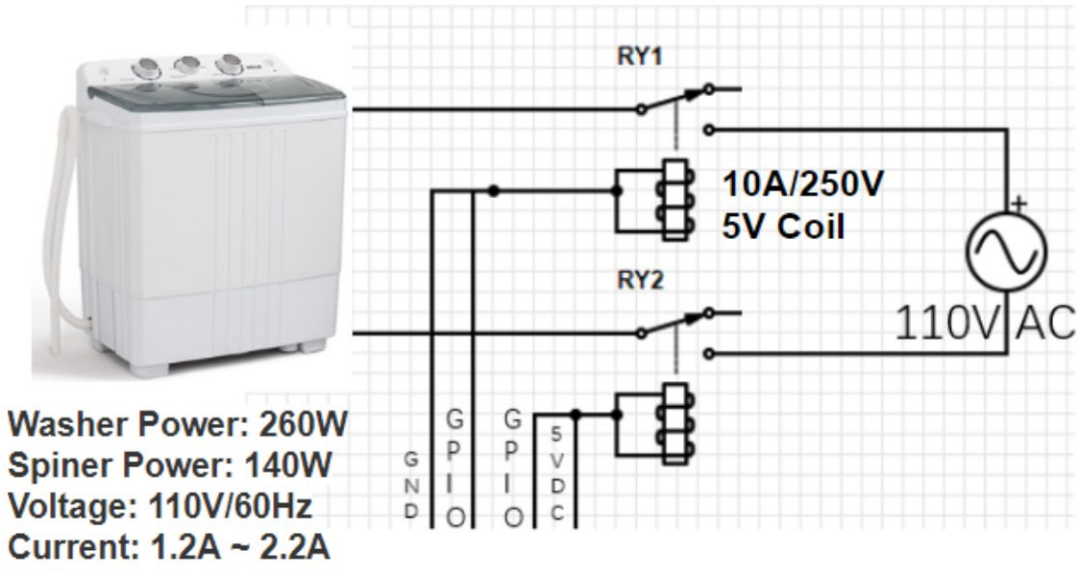


Figure 8: Circuit Diagram

Communication with AWS IoT

The communication between our local setup and AWS IoT is based on a state-of-art, light-weight networking messaging protocol called Message Queuing Telemetry Transport (MQTT). MQTT is becoming popular recently especially for IoT projects. Devices can “subscribe/public/listen” to the same “topic” to transmitting commands. This is what AWS use in their implementation and also meet our requirements perfectly.

Scripts

On Raspberry pi, we have two types of scripts running to make the hardware system to work. Two snips of our script are shown below this section.

1) Python: Python mainly in charge of the MQTT socket programming as well as the GPIO controls in our local setup. The library we used within Python scripts are GPIO and AWS_MQTT supporting libraries.
2) Bash: we use bash scripts to give the Raspberry pi abilities to run the Python scripts everytime it boots. Also we have the bash script to send us an e-mail with the IP address of the pi under current network environment. This way we could

easily log in to the pi and make changes remotely without worrying about figuring the IP address to use in SSH.

Python

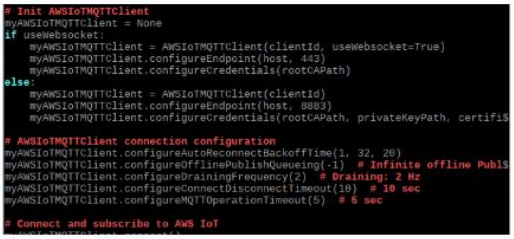


Figure 9: Python Script

Bash

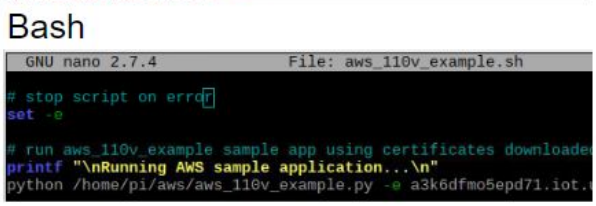


Figure 10: Bash Script

Implementation

After the design process, we have the following as our implementing process:

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 have established 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. Update: the current testing presents as controlling a 110v home lamp by sending commands from the AWS cloud.

In addition, 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

We will now discuss the specifics of our design, broken down into components.

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.

While Spring Boot was our eventual choice, we could have went with a couple of different alternatives to provide a REST server. Familiarity had a lot to do with our choice, as well as the large online community which surrounds the framework. However, Dropwizard has a similar online community, which is important to us as one of our original goals was to resolve issues internally as much as possible. While Spring Boot allows for thorough configuration by the developer, Dropwizard relies on its conventional technology, somewhat limiting configuration options. While Dropwizard limits your HTTP servlet container to Jetty, Spring Boot defaults to the embedded tomcat servlet but allows you to choose a different one, such as Jetty. Another feature which made Spring Boot attractive

was the built-in dependency injection. Spring Boot also seems more extendable and seems to integrate with other technologies better.

While Spring Boot's performance is arguably not up to par with more lightweight technology such as the Express framework (Node.js), it is generally more difficult to scale javascript to an enterprise application. We could have made a decision to move to a more lightweight technology for the sake of performance, but our clients have the vision of extending our application to an enterprise level and we believe java is the right technology to utilize.

2.2.2 Front-End Mobile Application

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 did not 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.

2.2.3 Hardware

So far we have set our Raspberry Pi up and built a breadboard testbed consists of LED and some resistors. The main reason we choose the Raspberry pi is that the clock speed of the Raspberry pi is faster than the Arduino Yun (The other choice) and we need a microcontroller with higher computational power. Also, the raspberry pi is cheaper so it will be a good choice for industrial business. The provided operating system of the

Raspberry pi is Raspbian, our hardware team then decided to use the Python and Bash script to implement our design which will allow the Raspberry Pi to communicate with AWS cloud.

We first find the GPIO ports on the Raspberry pi and then connect those with our breadboard. Then we implemented a Python program which will turn on/off the LED by sending PWM signal from the GPIO ports. Thus we had the fundamental deployments tested.

Next, we established the communication between Raspberry pi with AWS cloud. So we were able to control the LED remotely from AWS IoT test panel. In specific, the local Python script listened to the subscribed topic (Term in AWS IoT cloud, represents certain communication channel between AWS and device) for specific commands sent by AWS test panel. Once the expected commands received by local device, turns on/off the LED accordingly.

In the end, we added a 4-channel relay to our testing circuit to handle a 110V AC appliance. By combining this step with previous works, we were able to turn on/off a lamp remotely from AWS IoT test panel.

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:

1. Internet is not stable with our channel. The connecting and testing phase can only be proceed on the private network right now (It will not connect to a new WiFi easily).

2. In our project, it seems we cannot get the current status of our washing machine. For instance, we will not know if it was working since we cannot get its information anyway. The only thing we can control is the power of the washing machine.

2.2.4 Design Timeline

Attached below is a Gantt Chart that distinguishes between the design elements to be completed during Semester 1 and Semester 2 of Senior Design.

Mobile

The mobile team aims to develop a login screen, registration screen, google maps screen, reservation screen, and confirmation screen by the end of Senior Design 1. The payment transaction platform, user profile slide menu, reservation history, client login and registration, and usage statistic overview screens will be implemented during Senior Design 2.

Hardware

The hardware team plans to complete a hardware prototype where AWS IoT sends a Command to the raspberry pi. The raspberry pi reads the command and powers on/off a portable washing machine. It can be expected that the prototype is completed by the end of Senior Design 1.

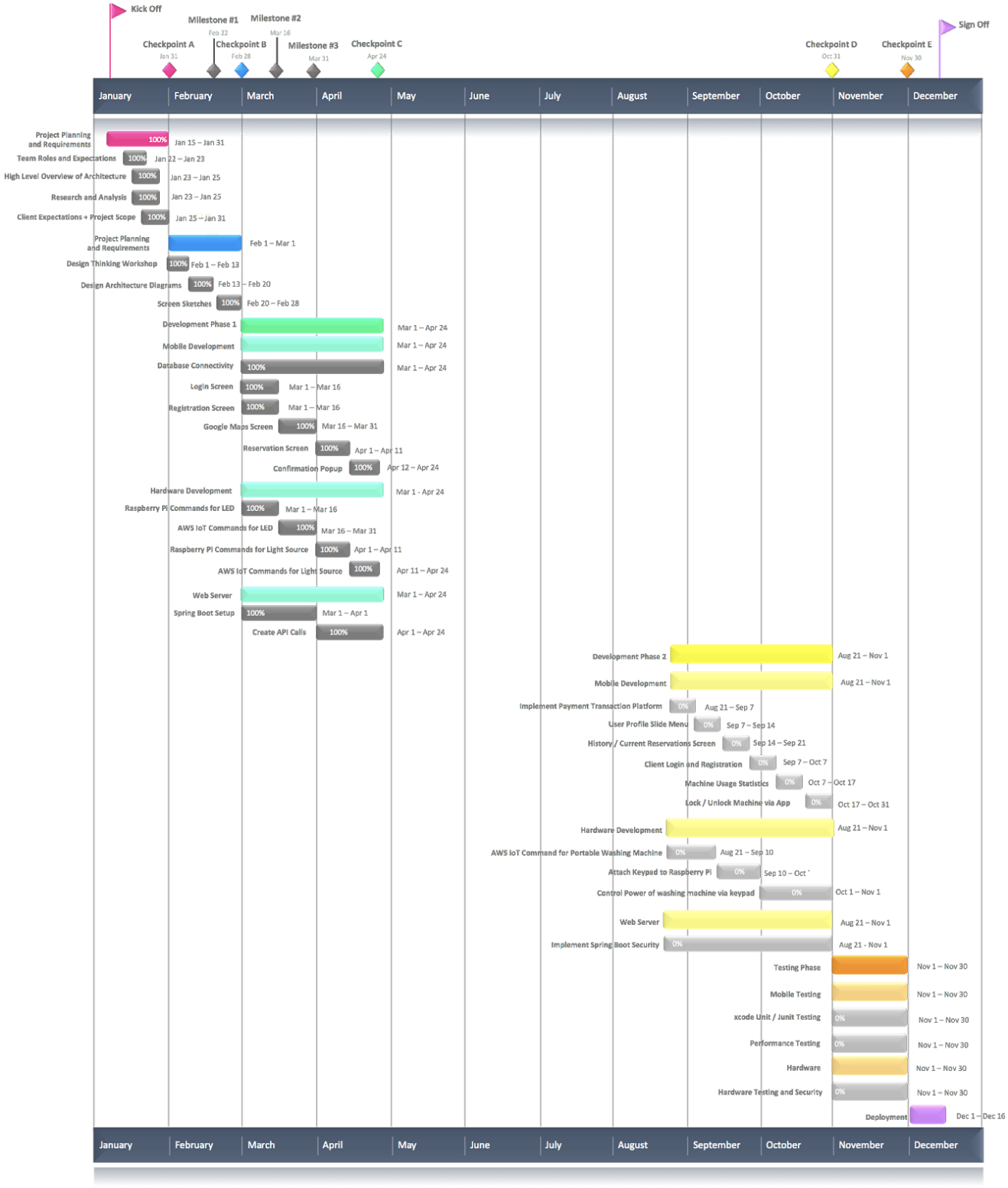
The keypad and LCD implementation of our project will be developed during Senior Design 2.

Backend

The backend team seeks to complete Spring Boot Setup and to write all API calls required by the mobile team. It can be expected that our Spring Boot solution is implemented by the end of Senior Design 1.

Testing and Security

Testing and Security as described below will be implemented during the second semester of Senior Design.


Kick Off
Milestone #1
Feb 22
Milestone #2
Mar 16
Checkpoint A
Jan 31
Checkpoint B
Feb 28
Milestone #3
Mar 31
Checkpoint C
Apr 24
Checkpoint D
Oct 31
Checkpoint E
Nov 30
Sign Off
January
February
March
April
May
June
July
August
September
October
November
December
Project Planning and Requirements 100% Jan 15 – Jan 31
Team Roles and Expectations 100% Jan 22 – Jan 23
High Level Overview of Architecture 100% Jan 23 – Jan 25
Research and Analysis 100% Jan 23 – Jan 25
Client Expectations + Project Scope 100% Jan 25 – Jan 31
Project Planning and Requirements Feb 1 – Mar 1
Design Thinking Workshop 100% Feb 1 – Feb 13
Design Architecture Diagrams 100% Feb 13 – Feb 20
Screen Sketches 100% Feb 20 – Feb 28
Development Phase 1 Mar 1 – Apr 24
Mobile Development Mar 1 – Apr 24
Database Connectivity 100% Mar 1 – Apr 24
Login Screen 100% Mar 1 – Mar 16
Registration Screen 100% Mar 1 – Mar 16
Google Maps Screen 100% Mar 16 – Mar 31
Reservation Screen 100% Apr 1 – Apr 11
Confirmation Popup 100% Apr 12 – Apr 24
Hardware Development Mar 1 - Apr 24
Raspberry Pi Commands for LED 100% Mar 1 – Mar 16
AWS IoT Commands for LED 100% Mar 16 – Mar 31
Raspberry Pi Commands for Light Source 100% Apr 1 – Apr 11
AWS IoT Commands for Light Source 100% Apr 11 – Apr 24
Web Server Mar 1 – Apr 24
Spring Boot Setup 100% Mar 1 – Apr 1
Create API Calls 100% Apr 1 – Apr 24
Development Phase 2 Aug 21 – Nov 1
Mobile Development Aug 21 – Nov 1
Implement Payment Transaction Platform 0% Aug 21 – Sep 7
User Profile Slide Menu 0% Sep 7 – Sep 14
History / Current Reservations Screen 0% Sep 14 – Sep 21
Client Login and Registration 0% Sep 7 – Oct 7
Machine Usage Statistics 0% Oct 7 – Oct 17
Lock / Unlock Machine via App 0% Oct 17 – Oct 31
Hardware Development Aug 21 – Nov 1
AWS IoT Command for Portable Washing Machine 0% Aug 21 – Sep 10
Attach Keypad to Raspberry Pi 0% Sep 10 – Oct '
Control Power of washing machine via keypad 0% Oct 1 – Nov 1
Web Server Aug 21 – Nov 1
Implement Spring Boot Security 0% Aug 21 - Nov 1
Testing Phase Nov 1 – Nov 30
Mobile Testing Nov 1 – Nov 30
xcode Unit / Junit Testing 0% Nov 1 – Nov 30
Performance Testing 0% Nov 1 – Nov 30
Hardware Nov 1 – Nov 30
Hardware Testing and Security 0% Nov 1 – Nov 30
Deployment Dec 1 – Dec 16

3 Testing and Implementation

3.1 Interface Specifications

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 have not 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

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

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.

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.

3.3.3 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

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 have not 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.

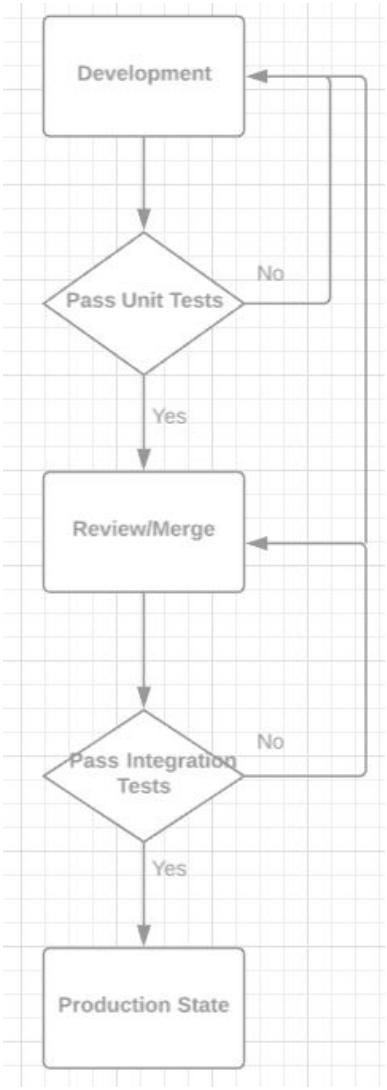
3.4.2 Front-End Mobile Application

To conduct usability testing, our mobile application team will be releasing a usability survey during the second half of Senior Design. The usability survey will be distributed to both our clients, advisor, and a selection of on-campus students. The usability survey will be used to identify UI flaws in regards to flow, step-by-step processes, color, look, and feel.

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



Starting in the second semester, our teams will run local tests on their code to ensure not only that their did not introduce any new bugs, but that the new code did not 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 (eventual) 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

Our team has faced several different technical challenges throughout the semester.

Mobile

xcode is a powerful IDE for developing iOS applications. However, xcode tends to have version control issues when using Gitlab. This is because all UI elements are usually done in a special iOS file called a storyboard. The storyboard is an xml file that hides the xml from the developer by allowing for a drag and drop design approach. When multiple developers overwrite the storyboard file, gitlab has trouble merging the file and thus causes file corruption. To mitigate this issue, the iOS team removed the storyboard file and programmatically built the user interface.

A second issue faced by the mobile team involved the creation of the actual UI. Both the Android team and iOS team developed several different UI prototypes throughout the semester, but a final decision wasn't made until March when we were able to receive feedback from our clients. The delay in finalizing a UI design created setbacks with our plans for backend integration.

Hardware

The hardware team ran into several issues throughout the semester regarding the purchase of components. Our team developed a prototype that controls the power to a lightsource instead of a washing machine as our team didn't receive a portable washing machine until April 23, 2018.

another issue faced by the hardware team is determining a method for collecting the power draw of a washing machine. At the request of our clients, it would be ideal for us to find a solution that does not involve attaching a sensor device to a washing machine. However, "hacking" into a washing machine violates IEEE standards and isn't a valid final solution.

During the second semester of Senior Design, it can be expected that the hardware team faces challenges with interfacing a keypad and LCD to the raspberry pi. Our team does not have an EE so we'd prefer to stay away from soldering wires.

Backend

The backend team has faced difficulty extended the Spring Boot Framework so that it can be hosted externally. Currently, Spring Boot is localized per developer which makes long-term testing between both mobile applications and the backend difficult.

4 Closing Material

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 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

Mackrory, Mike. “Building Java Microservices with the DropWizard Framework.” *Sumo Logic*, 8 May 2017, www.sumologic.com/blog/devops/building-java-microservices-with-the-drop wizard-framework/.

Shelat, Mihir. “Node.js for Enterprise Applications! Are You Kidding? - DZone Web Dev.” *Dzone.com*, 2 May 2016, dzone.com/articles/nodejs-for-enterprise-applications-are-you-kidding.

“Tomcat vs Jetty - Two Great Servlet Containers. Which One to Choose?” *DailyRazor.com*, 20 Dec. 2017, www.dailyrazor.com/blog/tomcat-vs-jetty/.

4.3 Appendices

Single-Board Computer - Raspberry Pi Model 3 Design Documents

Schematics
https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf

Broadcom Processor used in Raspberry Pi 3
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md

Raspberry Pi Boot Modes
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/README.md

Raspberry Pi Mechanical Drawings
https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/rpi_MECH_3b_1p2.pdf

Raspberry Pi Power Supply
https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md

Raspberry Pi USB
https://www.raspberrypi.org/documentation/hardware/raspberrypi/usb/README.md

Raspberry Pi GPIO
https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md

Raspberry Pi SPI
https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md

Raspberry Pi DPI (Parallel/RGB Display)
https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/README.md

Peripheral Addresses
https://www.raspberrypi.org/documentation/hardware/raspberrypi/peripheral_addresses.md

Standard Conformity Documentation
https://www.raspberrypi.org/documentation/hardware/raspberrypi/conformity.md

Revision Codes Reference
https://www.raspberrypi.org/documentation/hardware/raspberrypi/revision-codes/README.md

AWS IoT Documentation

Thing Shadow REST API

https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

API Reference
https://docs.aws.amazon.com/iot/latest/apireference/iot-api.pdf

AWS IoT Command References
https://docs.aws.amazon.com/cli/latest/reference/iot/index.html
https://docs.aws.amazon.com/cli/latest/reference/iot-data/index.html