IoT Remote Reservation Mobile Application for Washing Machines and Dryers

Final Report

Team Number
17

Client
Greiner Jennings Holdings
Advisor
Goce Trajcevski

Team Members and Roles
John Fleiner: Android Mobile Developer
Ben Young: iOS Mobile Developer
Thomas Stackhouse: Backend Developer
Case Gehling: Backend Developer
Hongyi Bian: Hardware Engineer
Yuanbo Zhen: Hardware Engineer

Team Email:
sdded18-17@iastate.edu
Team Website:
https://sddec18-17.sd.ece.iastate.edu

Table of Contents

I. List of Figures

II. List of Tables

1. Introduction

1.1 Acknowledgement

Clients:

Our team 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 have provided our team with the necessary hardware and software components to complete our project. The items they have provided include a washing machine, a Raspberry Pi single-board computer, and an AWS IoT cloud service subscription.

Advisor:

Our team would also like to thank Goce Trajcevski for his continued support and guidance.

1.2 Problem Statement

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.

Purpose

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.

Solution Approach

Our proposed solution consists of three main components: AWS IoT cloud service, a mobile application, and an LCD Keypad locking system. The AWS IoT cloud service will be used to register a set of washing machine and dryers. A cross-platform mobile application will be developed to connect to an IoT cloud service and allow users to submit laundromat reservations. The reservation system on the mobile application will allow users to pay to reserve a washing machine or dryer for a set period of time. Once a reservation has been submitted and a payment transaction has been confirmed, a unique reservation code will be generated for the user. During the reservation period, the reserved machine will be locked and unable to be turned on until the unique reservation code has been entered into a keypad by the user. 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 Intended Uses

Industrial Users

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.

Commercial Users

It is also expected that laundromat customers of our clients shall use the mobile application to reserve washing machines and dryers.

Uses (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.

Uses (Long-Term)

Our clients plan to expand 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 can be done once a proof-of-concept has been developed.

1.5 Assumptions and Limitations

Assumptions

- A website reservation system shall not be included in the final prototype as the requested solution is aimed for mobile devices
- A laundromat facility must profit from a reservation, regardless if the user is absent during a reservation period
- The requested solution is expected to worth with a single washing machine appliance

Limitations

- Battery Life for the mobile application must be minimal as the application will require access within a Laundromat
- There must be stable Network Connection in the Laundromat for users to connect to the mobile application
- 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.
- 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.
- Without “hacking” into a washing machine, the pulling of power data will need to be done using external components.
- The application will be mobile and thus any intensive computing should happen on the server end of the architecture.
- Limited accesses to AWS server under Free-Tier payment plan.
- Hardware purchases may not exceed our $500 funded budget.

1.6 Expected End-Product and Deliverables

Mobile Application

Our mobile development team is expected to deliver a cross-platform mobile application that supports both the Android and iOS operating systems. The mobile application will include a secure login-system, a secure payment transaction system, and a reservation system with the ability to view receipts of their present and previous reservations.

End-Product 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.

Hardware

The hardware that will be supplied to our clients include a microcontroller with an LCD display and a numeric keypad attached to a portable lightweight washing machine. The microcontroller will be responsible for communicating with AWS IoT web service and for controlling the washing machine appliance.

Deliverables

- iOS Mobile Application
- Android Mobile Application
- Spring Boot Web Service
- Amazon Web Services Internet of Things w/ registered washing machine
- Portable Washing Machine equipped with a raspberry pi, LCD display and a keypad

1.7 Development Process

For the entirety of the project, our group followed Agile methodologies in the development process. This revolved around incremental development with continuous refactoring at the request of the three subteams, the advisor, and most importantly, the clients. We essentially participated in two week sprints, however; some sprints were a week or three weeks long, depending on circumstances with client availability as well as the schedule of each team member. Typically we held bi weekly team meetings to review progress from the previous sprint as well as plan for the upcoming spring based on feedback from bi weekly check ins with the client and the advisor. Our client meetings revolved around status updates and often included demos of the mobile application to receive feedback.

While we set out to follow test driven development methods, admittedly, we struggled to follow this process during the first semester. We achieved most of our development on the mobile and back-end server portions during the first semester and were rapidly changing/adding code to both components. While it would have helped both our development efforts in the first semester as well as our testing in the second semester to follow TDD concepts with more detail, the reality is we didn’t always achieve this.

However, during the second semester, we followed TDD concepts rather thoroughly. Because we had a solid code base in place, we were able to focus on testing early in the semester and thus could continuously test our incremental development during each sprint.

2. Requirements Specification

2.1 Context Diagram

A context diagram is a requirements model that that helps to identify the responsibilities of the project's work and the responsibilities of adjacent systems. Responsibilities are characterized as flows of data between systems to complete the work of reserving a washing machine.

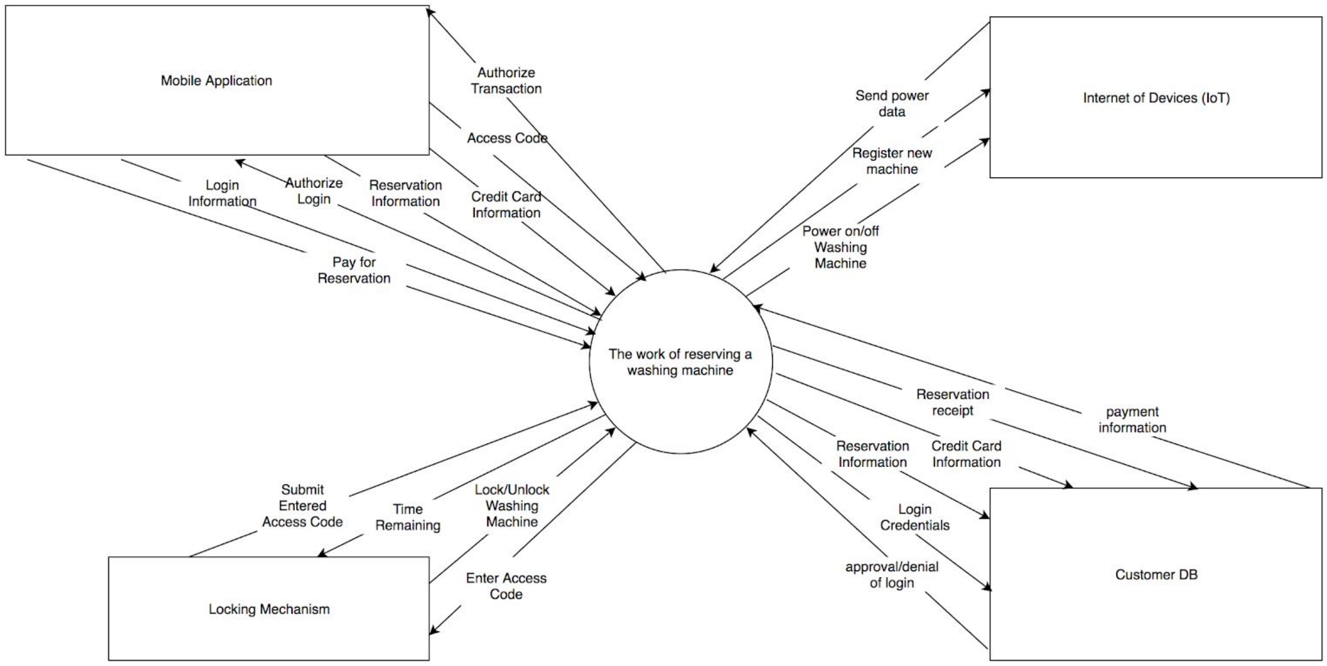


Figure 1: Context Diagram

Our project consists of four adjacent systems, a mobile application, a locking mechanism, an internet of things web service, and a customer database. The mobile application is responsible for sending user login and registration information, credit card data, and reservation input. The mobile application is responsible for receiving authorization to login and for receiving authorization to pay. The Locking mechanism is a hardware component that locks and unlocks a washing machine. The locking mechanism receives an access code entered by the user and submits the access code to the backend. The locking mechanism then receives the time left for the current reservation period. The customer database is responsible for storing reservation information, credit card information, login credentials. The customer database is also responsible for sending a reservation receipt and payment authorization. The internet of devices or internet of things is responsible for receiving power commands to the washing machine and for registering new washing machine and dryer devices. The internet of devices then sends power data. The context diagram also serves as a tool for finding business events and product events. The list of product events and their associated input and output flows for reserving a washing machine are shown below.

2.2 Product Use Cases

Product Event Name (Mobile Application)	**Input and Output**	**Summary of Product Use Case**
Mobile application sends login information	email and password (input)	Attempts to login the customer
Mobile application sends registration information	First name, last name, email, password, confirm password (input)	Registers a new user
Mobile application receives login authorization	Customer identifier (output)	Login information has been approved
Mobile application sends credit card information	Cardholder name, card number, card year, card month, card CVC (input)	Attempts to add/update payment information
Mobile application submits reservation information	Reservation date, reservation start time, reservation end time, number of washing machines, number of dryers, price (input)	Attempts to create a new reservation for the customer
Mobile application pays for reservation	Cardholder name, card number, card year, card month, card CVC (input), payment token (input)	Attempts to pay for the newly created reservation
Mobile application receives transaction authorization	Success message (output)	Reservation has been approved
Mobile application sends access code	4-digit access code (input)	Submits valid access code to unlock a washing machine

Table 1: Mobile Application Product Events

Product Event Name (Locking Mechanism)	**Input and Output**	**Summary of Product Use Case**
Locking Mechanism locks or unlocks Washing Machine	Power on/off command (input)	Turns the washing machine on/off
Locking Mechanism receives access code	4-digit access code (output)	Attempts to lock or unlock the washing machine
Locking Mechanism sends access code	4-digit access code (input)	Submits access code to database for verification
Locking Mechanism receives estimated time remaining	Time in milliseconds (output)	Receives estimated reservation time remaining for the washing machine

Table 2: Locking Mechanism Product Events

Product Event Name (Customer Database)	**Input and Output**	**Summary of Product Use Case**
Database receives login credentials	Email and password (output)	Receives login credentials to validate login
Database receives registration information	First name, last name, email, password, confirm password (output)	Registers user and stores information in the database
Database receives reservation information	Reservation date, reservation start time, reservation end time, number of washing machines, number of dryers, price (output)	Registers new reservation and stores information in the database
Database receives credit card information	Cardholder name, card number, card month, card year, car CVC (output)	Updates credit card information

Database receives payment	Price (output)	Receives payment information to validate payment
Database sends approval or denial of login	Customer Id or -1 (input)	Determines if the user can login
Database sends reservation receipt	Laundromat location, reservation start time, reservation end time, washing machine number, access code, price (input)	Sends reservation confirmation to the user
Database sends payment authorization	Payment confirmation message (input)	Payment information is validated and reservation is created successfully

Table 3: Customer Database Product Events

Product Event Name (Internet of Devices - IoT)	**Input and Output**	**Summary of Product Use Case**
IoT registers new machine	Microcontroller identifier	Registers new washing machine based on its attached microcontroller
IoT receives power command	Power command	Powers on washing machine via microcontroller
IoT sends power data	Power status	Washing Machine power status is sent to the locking mechanism

Table 4: AWS IoT Product Events

2.3 Functional Requirements

2.3.1 Ubiquitous Requirements

1. *Description:* The mobile application shall display local laundromats, ATMs, and banks within Ames ,IA

> *Fit Criterion:* The mobile application shall parse laundromat, ATM, and bank nodes from *openstreetmap.org* to obtain the name, address, latitude, and longitude data.
>
> *Rationale:* Each laundromat shall represent a location for which a reservation may be performed. If the user chooses to pay with coins, then it may be beneficial to the user if nearby ATMs and banks are displayed.

2. *Description:* The mobile application shall login the customer upon a successful login authorization.

> *Fit Criterion:* The mobile application shall store the customer's unique identification number using the singleton design pattern and then transition the user to the either a) credit card screen if a credit card has not been entered or b) the home screen if a valid credit card is on file
>
> *Rationale:* On a successful login attempt, the application needs to navigate the user to the next step in the sequence of reservation steps.

3. *Description:* The mobile application shall inform the user of a credit card charge upon a successful transaction authorization.

> *Fit Criterion:* The mobile application shall notify the user with a reservation receipt including the laundromat location, reservation date, reservation time frame, washing machine number, price, and reservation code upon a successful transaction authorization.
>
> *Rationale:* Once a reservation has been created, it is required by law to notify the user of a charge to their account and provide them with enough information to request a refund if-need-be.

2.3.2 Event-Driven Requirements

1. *Description:* When the customer clicks the login button, the mobile application shall send login data to the server

> *Fit Criterion:* When the customer clicks the login button, the mobile application shall attempt a login by sending the email address and password to the server for validation.
>
> *Rationale:* Since the application serves as a payment transaction platform, it is required that the user is provided a secure login method for an account tied to their personal information.

2. *Description:* When the customer clicks the reservation button, the mobile application shall send registration data to the server

> *Fit Criterion:* When the customer clicks the reservation button, the mobile application shall register a new user, given a valid first name, last name, email address, password, and confirmed password. Ideally two-factor authentication would be performed to validate the email address.
>
> *Rationale:* A customer must be given the ability to register a new account since we are required to record all transactions and to provide receipts of all reservations. Transactions must be recorded and paired to a customer account for company tax and refund disputes.

3. *Description:* When the customer clicks the credit card button, the mobile application shall send credit card information to the server

> *Fit Criterion:* When the customer clicks the credit card button, the mobile application shall validate and then securely send the cardholder name, card number, card expiration month, card expiration year, and card CVC number to the database.
>
> *Rationale:* A credit card must be on file in order to perform a reservation. It is a standard practice to ask for credit card information before allowing the user access to the entire application.

4. *Description:* When the customer clicks the reserve button, the mobile application shall create a new reservation.

> *Fit Criterion:* When the customer clicks the reserve, button, the mobile application shall attempt to create a new reservation by submitting the reservation date, reservation start time, reservation end time, number of washers, number of dryers, and price to the backend for validation. If the reservation criteria is valid and enough washers and dryers are available for the duration of the reservation, then a new reservation will be created.
>
> *Rationale:* A reservation form is required to create a new reservation that meets the time constraints requested by the customer.

5. *Description:* When the user enters a reservation code into the locking mechanism, the locking mechanism shall unlock the washing machine

> *Fit Criterion:* When the user enters a 4-digit reservation code generated upon the creation of a successful reservation, the locking mechanism shall validate the code by sending it to the server for confirmation. A successful validation shall unlock the washing machine.
>
> *Rationale:* The washing machine must remain locked until a valid reservation code has been entered by the correct user.

6. *Description:* When a new microcontroller has been purchased, the IoT web service shall register a new device.

> *Fit Criterion:* When a new microcontroller has been purchased by our clients, the device shall be registered to IoT web service via the AWS console.
>
> *Rationale:* The system must be scalable to allow for more washing machines to be added to IoT in the future.

7. *Description:* When the power command is received, then the power to the washing machine shall be enabled.

> *Fit Criterion:* When the power command is received from the locking mechanism, then the power to the washing machine shall be enabled
>
> *Rationale:* To lock/unlock the washing machine, the power source to the machine needs to be cut/enabled.

2.3.3 State-Driven Requirements

1. *Description:* While the washing machine is running, the locking mechanism shall receive the estimated time remaining

> *Fit Criterion:* While the washing machine is running, the locking mechanism shall receive the estimated time remaining in milliseconds for the current reservation.
>
> *Rationale:* The locking mechanism needs to know how much time is remaining so that when the time runs out, power can be cut to the washing machine.

2. *Description:* While the washing machine is running, the locking mechanism shall receive power status information

> *Fit Criterion:* While the washing machine is running, the locking mechanism shall receive an on/off message indicating the current status of the washing machine.
>
> *Rationale:* The locking mechanism needs to know the current status of the washing machine to prevent input from the keypad.

2.3.4 Unwanted Functional Requirements

1. *Description:* If the login credentials are invalid, then the mobile application shall invalidate the login and prevent them from navigation onward.

> *Fit Criterion:* If the web server returns a customer id of -1, then the login credentials were invalid and the login attempt shall be null and void. Both the email and password field shall display an error message indicating than an invalid email or password was entered.
>
> *Rationale:* It is imperative that we let the user know when an invalid login attempt occurred and what the cause may have been without giving away information to potential brute force hackers.

2. *Description:* If the registration credentials are invalid, then the customer shall not be registered as a new user.

> *Fit Criterion:* If the web server returns a registration error, then the registration input fields were not entered correctly and the registration shall be null and void. All fields including first name, last name, email, password, and confirm password shall display an error message indicating that an invalid email or password was entered.
>
> *Rationale:* For maintainability and security purposes, we must prevent invalid accounts from being added to our database.

3. *Description:* If the number of washers or number of dryers exceeds the available amount, then a warning shall be displayed.

> *Fit Criterion:* If the number of washers or number of dryers exceeds the available number of washers and dryers, then the reservation shall be canceled and a warning shall popup on the screen indicating the number of available appliances.
>
> *Rationale:* For maintainability and security purposes, we must prevent invalid reservations from being added to our database.

4. *Description:* If the customer's credit card is declined, then a warning shall be displayed.

> *Fit Criterion:* If the customer's credit card is invalid or has insufficient funds, then the reservation shall be canceled and a warning shall popup on the screen indicating that there was an issue with the customer's credit card and the reservation did not go through.
>
> *Rationale:* For maintainability and security purposes, we must prevent invalid transactions from going through our system.

5. *Description:* If the customer enters an invalid reservation access code, then the locking mechanism shall not unlock the washing machine.

> *Fit Criterion:* N/A
>
> *Rationale:* During a reservation time, the washing machine must only be available to the customer who reserved the appliance.

2.4 Non-Functional Requirements

2.4.1 Look and Feel Requirements

1. *Description:* The mobile application shall look profession by having a well-thought out and clean user interface design.

    Fit Criterion: At least 90% of test users shall report a positive user interface experience after demonstrating the mobile application.

2.4.2 Usability and Humanity Requirements

1. *Description:* The mobile application shall be simple to use

    Fit Criterion: At least 80% of customers shall complete the reservation process within the time constraints identified by the usability test found within the testing section of the report.

2.4.3 Performance Requirements

1. *Description:* Login validation with the backend shall take no longer than 3 seconds to complete.

2. *Description:* Reservation validation with the backend shall take no longer than 3 seconds to complete.

3. *Description:* Credit card validation and updates with the backend shall take no longer than 3 seconds to complete

4. *Description:* Reservation submission shall take no longer than 5 seconds to complete.

5. *Description:* Loading historical and future reservations shall take no longer than 2 seconds to complete.

6. *Description:* Unlocking of a machine shall take no longer than 2 seconds to complete after entering a valid reservation code.

7. *Description:* The locking of a machine shall take no longer than 1 second after the reservation period has ended (Based on System time).

2.4.4 Maintainability and Support Requirements

1. *Description:* The mobile application shall be portable to Android and iOS Operating Systems.

    Rationale: The mobile application needs to be able to reach the greatest number of potential customers if multiple operating systems are supported.

2.4.5 Security Requirements

1. *Description:* The mobile application shall have secure login and registration.

    Fit Criterion: The mobile application shall utilize Oauth2.0, Spring Security, and two-factor authentication.

2. *Description:* The mobile application shall have secure payment transactions.

    Fit Criterion: The mobile application shall use the *Stripe SDK* for secure payment transactions for both android and iOS devices.

3. *Description:* The mobile application shall have secure API endpoints.

    Fit Criterion: The mobile application shall use secure SSL encryption of HTTPS. Each API shall have an authenticated certificate appended to the api indicating that the call is coming from a valid user interface.

4. *Description:* The mobile application shall have encrypted data when sent via APIs.

    Fit Criterion: The mobile application shall utilize Spring Boot credit card encryption techniques to hide plain text APIs.

2.4.6 Cultural Requirements

1. *Description:* The mobile application shall be presented in English, but allow for scalability to multiple languages.

    Fit Criterion: The mobile application shall use localization techniques to easily translate text to support multiple languages.

2.5 Constraints

1. *Description:* The hardware appliances shall be limited to one appliance for the prototype, but allow for scalability.

    Fit Criterion: The hardware appliances shall be limited to a single washing machine for the prototype, but the architecture will be designed to allow for the addition of more appliances in the future.

2. *Description:* The mobile application shall not run on Windows Operating System.

    Rationale: Windows phones control the minority of mobile devices on the market and are no longer supported by Microsoft.

3. *Description:* The application shall not be built for web.

    Rationale: The goal of our prototype is to make a mobile reservation system that works on-the-go. A website would be for administrator data and analytics support which is out-of-scope for the project.

4. *Description:* The locking mechanism shall not modify the internal structure of a washing machine.

    Rationale: Modifying the wiring within a washing machine violates IEEE standards.

5 *Description:* Our prototype shall not exceed a budget of $500

6 *Description:* Our prototype shall utilize the Internet of Things concept as the management tool for the washing machine.

    Rationale: Requested by our clients

3. Development Process

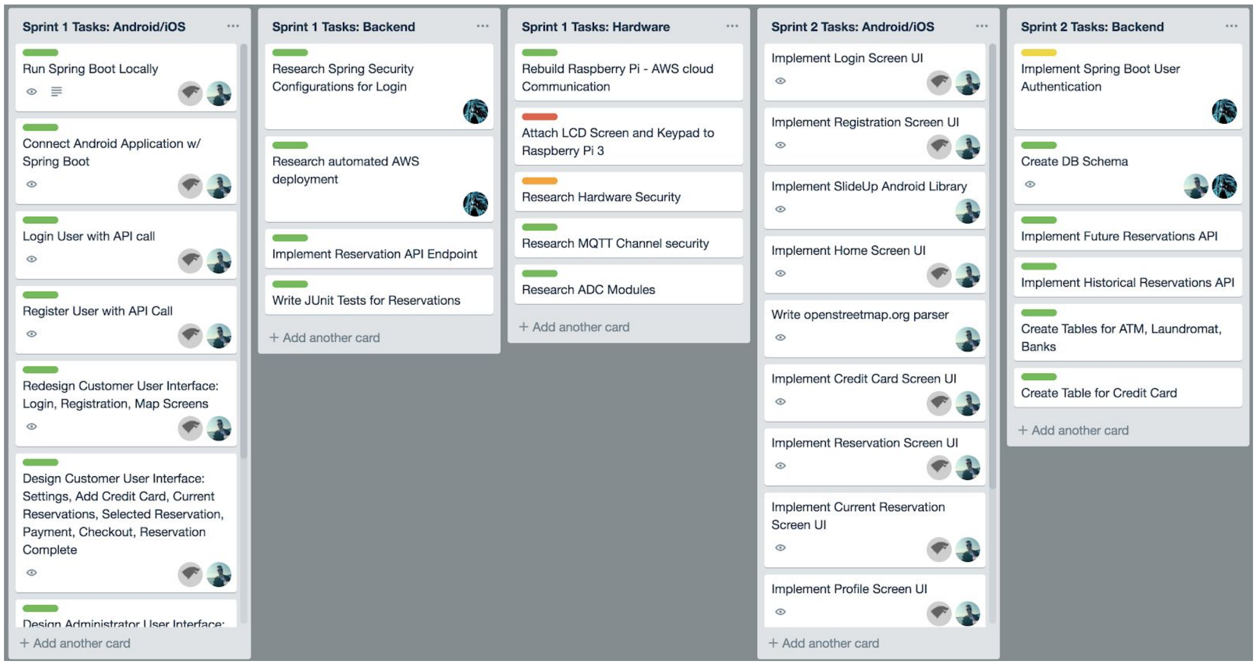
3.1 Agile Methodology

Agile is an iterative approach to software development that divides a projects timeline into a series of incremental steps. Agile places emphasis on the following management components: increments, sprints, storyboarding, and scrum. An increment is a series of sprints. A sprint is usually a one or two week development session. Therefore, an increment tends to contain 1-2 months worth of sprints, or approximately 8 sprints. Storyboarding is the process of identifying, assigning, and estimating the timeframe and difficulty of a set of tasks for each sprint. Each task is given a number, usually between 1 and 20, to specify how hard the task is to complete for the given sprint. At the end of each sprint, the values for all completed tasks is added up to equal a number called the *velocity.* The velocity is used as a comparator between sprints to see how much work is being completed.

3.2 Agile Scrum Board

Our team incorporated agile into our entire project planning and design process throughout the last two semesters. We have included our complete Scrum board for the second semester of Senior Design 492 below.

Sprint 1-2:



Sprint 1 Tasks: Android/iOS
- Run Spring Boot Locally
- Connect Android Application w/ Spring Boot
- Login User with API call
- Register User with API Call
- Redesign Customer User Interface: Login, Registration, Map Screens
- Design Customer User Interface: Settings, Add Credit Card, Current Reservations, Selected Reservation, Payment, Checkout, Reservation Complete
- Design Administrator User Interface:
- + Add another card

Sprint 1 Tasks: Backend
- Research Spring Security Configurations for Login
- Research automated AWS deployment
- Implement Reservation API Endpoint
- Write JUnit Tests for Reservations
- + Add another card

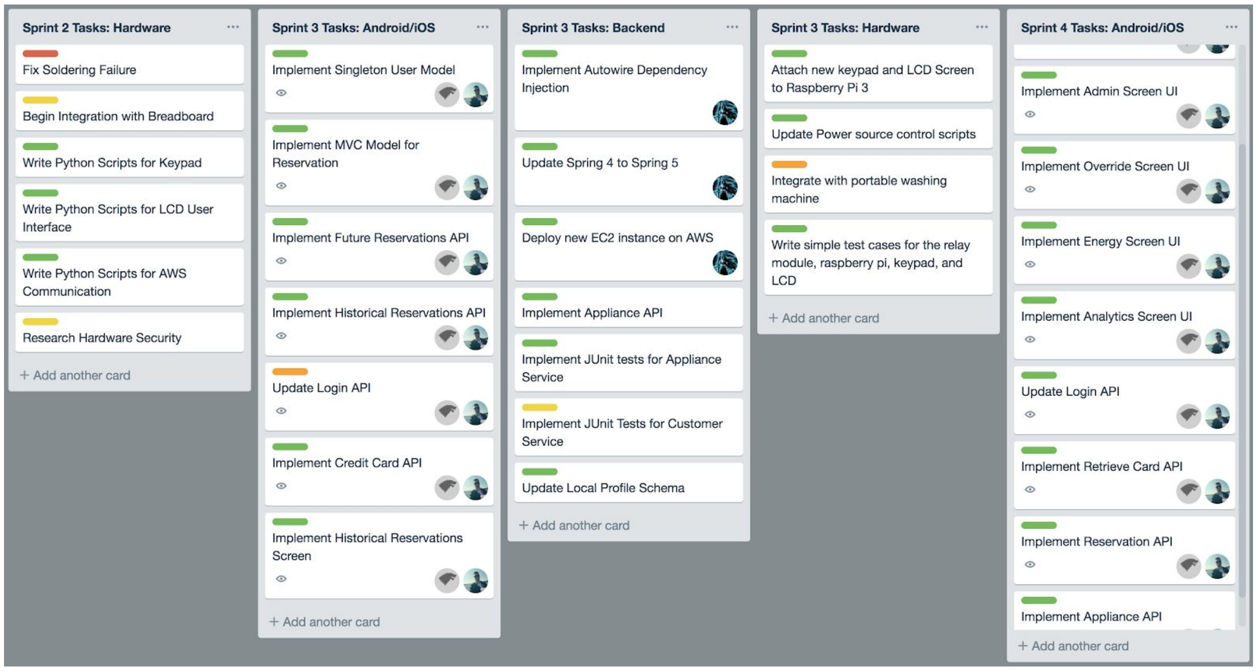
Sprint 1 Tasks: Hardware
- Rebuild Raspberry Pi - AWS cloud Communication
- Attach LCD Screen and Keypad to Raspberry Pi 3
- Research Hardware Security
- Research MQTT Channel security
- Research ADC Modules
- + Add another card

Sprint 2 Tasks: Android/iOS
- Implement Login Screen UI
- Implement Registration Screen UI
- Implement SlideUp Android Library
- Implement Home Screen UI
- Write openstreetmap.org parser
- Implement Credit Card Screen UI
- Implement Reservation Screen UI
- Implement Current Reservation Screen UI
- Implement Profile Screen UI
- + Add another card

Sprint 2 Tasks: Backend
- Implement Spring Boot User Authentication
- Create DB Schema
- Implement Future Reservations API
- Implement Historical Reservations API
- Create Tables for ATM, Laundromat, Banks
- Create Table for Credit Card
- + Add another card

Figure 2: Agile Sprints One and Two

Spring 2-4


Sprint 2 Tasks: Hardware
Fix Soldering Failure
Begin Integration with Breadboard
Write Python Scripts for Keypad
Write Python Scripts for LCD User Interface
Write Python Scripts for AWS Communication
Research Hardware Security
+ Add another card

Sprint 3 Tasks: Android/iOS
Implement Singleton User Model
Implement MVC Model for Reservation
Implement Future Reservations API
Implement Historical Reservations API
Update Login API
Implement Credit Card API
Implement Historical Reservations Screen
+ Add another card

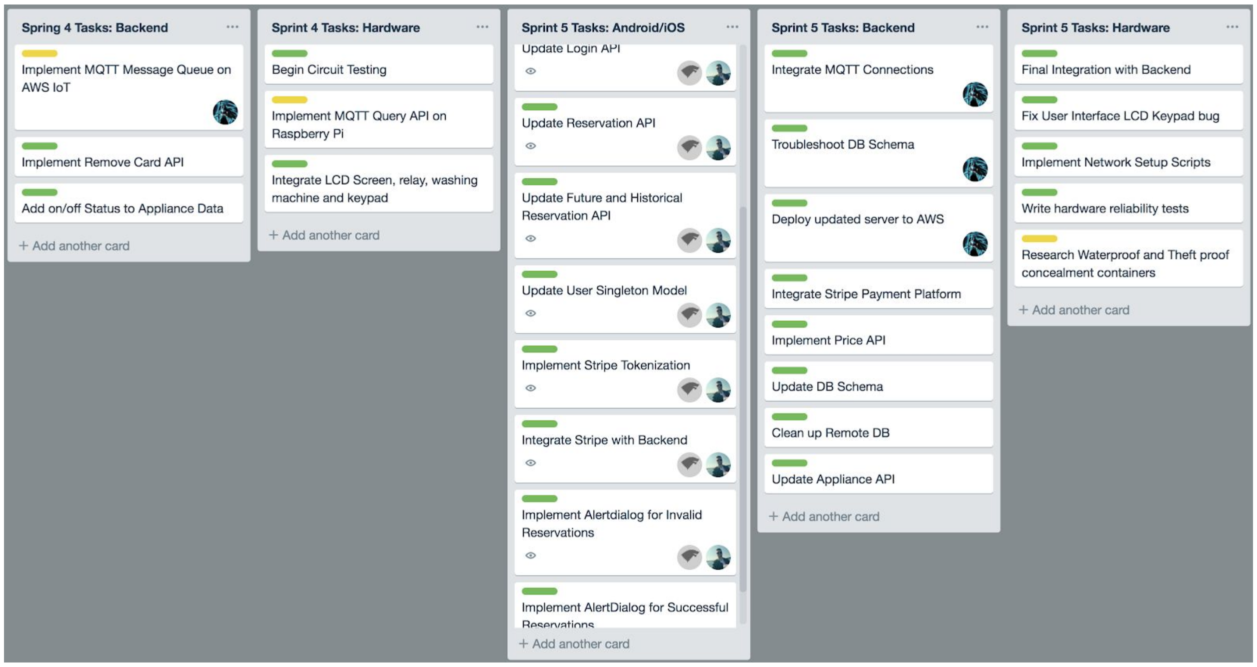
Sprint 3 Tasks: Backend
Implement Autowire Dependency Injection
Update Spring 4 to Spring 5
Deploy new EC2 instance on AWS
Implement Appliance API
Implement JUnit tests for Appliance Service
Implement JUnit Tests for Customer Service
Update Local Profile Schema
+ Add another card

Sprint 3 Tasks: Hardware
Attach new keypad and LCD Screen to Raspberry Pi 3
Update Power source control scripts
Integrate with portable washing machine
Write simple test cases for the relay module, raspberry pi, keypad, and LCD
+ Add another card

Sprint 4 Tasks: Android/iOS
Implement Admin Screen UI
Implement Override Screen UI
Implement Energy Screen UI
Implement Analytics Screen UI
Update Login API
Implement Retrieve Card API
Implement Reservation API
Implement Appliance API
+ Add another card

Figure 3: Agile Sprints Two, Three, and Four

Spring 4-5


Spring 4 Tasks: Backend
Implement MQTT Message Queue on AWS IoT
Implement Remove Card API
Add on/off Status to Appliance Data
+ Add another card

Sprint 4 Tasks: Hardware
Begin Circuit Testing
Implement MQTT Query API on Raspberry Pi
Integrate LCD Screen, relay, washing machine and keypad
+ Add another card

Sprint 5 Tasks: Android/iOS
Update Login API
Update Reservation API
Update Future and Historical Reservation API
Update User Singleton Model
Implement Stripe Tokenization
Integrate Stripe with Backend
Implement Alertdialog for Invalid Reservations
Implement AlertDialog for Successful Reservations
+ Add another card

Sprint 5 Tasks: Backend
Integrate MQTT Connections
Troubleshoot DB Schema
Deploy updated server to AWS
Integrate Stripe Payment Platform
Implement Price API
Update DB Schema
Clean up Remote DB
Update Appliance API
+ Add another card

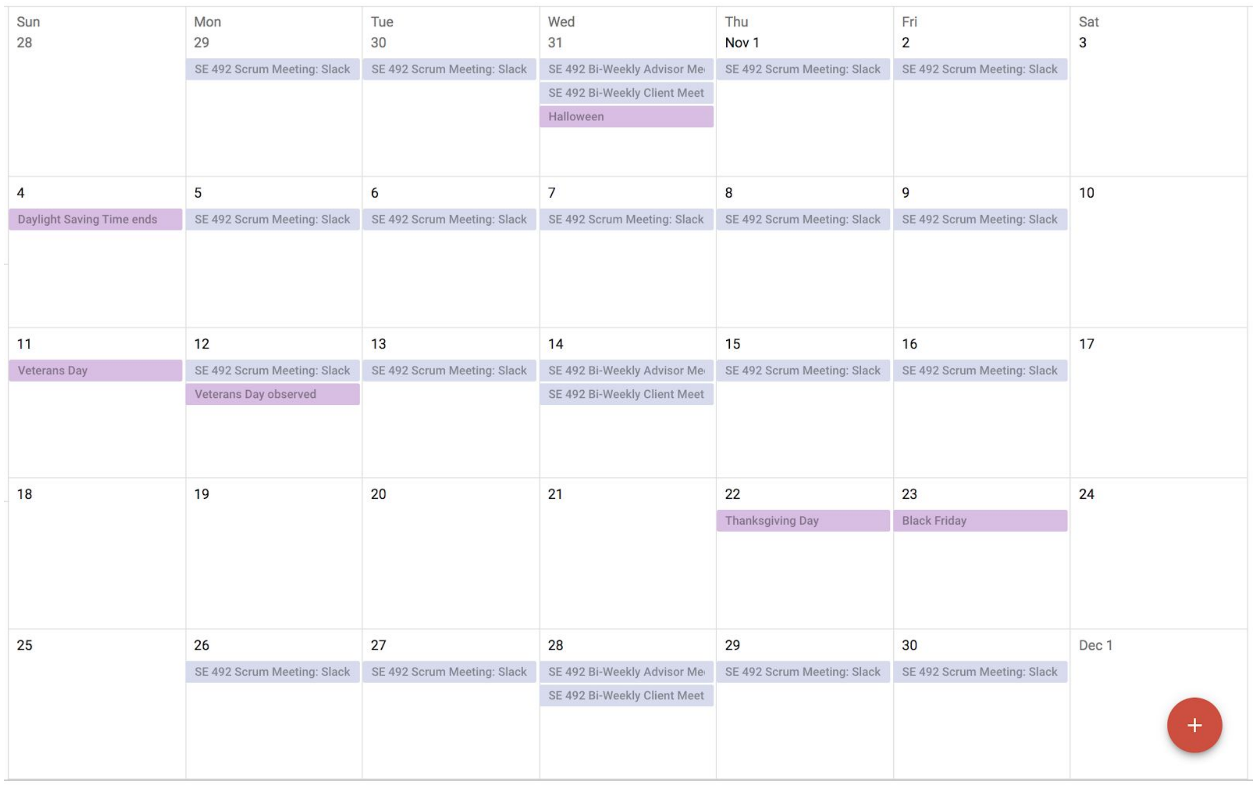
Sprint 5 Tasks: Hardware
Final Integration with Backend
Fix User Interface LCD Keypad bug
Implement Network Setup Scripts
Write hardware reliability tests
Research Waterproof and Theft proof concealment containers
+ Add another card

Figure 4: Agile Sprints Four and Five

3.3 Agile Scrum Meeting / Daily Standup

As mentioned above, agile methodology utilizes a concept call a *daily standup* or a *daily scrum* meeting where all of the developers get together and go around the room to discuss what they accomplished the previous day, what the plan on accomplishing during the current, and what roadblocks are in their way. Due to schedule conflicts, our team opted to use a team slack channel to communicate our plans daily. Every other Wednesday, our team would meet with both our clients and advisor to discuss future plans. A depiction of our agile calendar is shown below:

Calendar



Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29 SE 492 Scrum Meeting: Slack	30 SE 492 Scrum Meeting: Slack	31 SE 492 Bi-Weekly Advisor Me SE 492 Bi-Weekly Client Meet Halloween	Nov 1 SE 492 Scrum Meeting: Slack	2 SE 492 Scrum Meeting: Slack	3
4 Daylight Saving Time ends	5 SE 492 Scrum Meeting: Slack	6 SE 492 Scrum Meeting: Slack	7 SE 492 Scrum Meeting: Slack	8 SE 492 Scrum Meeting: Slack	9 SE 492 Scrum Meeting: Slack	10
11 Veterans Day	12 SE 492 Scrum Meeting: Slack Veterans Day observed	13 SE 492 Scrum Meeting: Slack	14 SE 492 Bi-Weekly Advisor Me SE 492 Bi-Weekly Client Meet	15 SE 492 Scrum Meeting: Slack	16 SE 492 Scrum Meeting: Slack	17
18	19	20	21	22 Thanksgiving Day	23 Black Friday	24
25	26 SE 492 Scrum Meeting: Slack	27 SE 492 Scrum Meeting: Slack	28 SE 492 Bi-Weekly Advisor Me SE 492 Bi-Weekly Client Meet	29 SE 492 Scrum Meeting: Slack	30 SE 492 Scrum Meeting: Slack	Dec 1

Figure 5: Agile Scrum Meeting

4. Design

4.1 Objective of the Task

Develop 1) a mobile application to reserve a set of washing machines and dryers 2) build a locking mechanism for a washing machine using a microcontroller, keypad and LCD display 3) utilize AWS IoT to register laundromat appliances and 4) use a web server to communicate between the mobile application, hardware components, and AWS.

4.2 Overview

Each individual component of our proposed design is explained in more detail in the upcoming sections. The following architecture diagram identifies the main communication channels between each module of our prototype. The mobile application serves as the user interface where a customer may create and pay for a reservation. Information sent from and received by the mobile application communicate with a Spring Boot Web Server via HTTP Rest APIs. Our Spring Boot Web Server queries a MYSQL database to retrieve reservation information related to the customer. AWS IoT is an Amazon Web Service for the Internet of Things. AWS IoT is where our microcontrollers are registered. The MQTT protocol is used to communicate between the microcontroller and AWS to share state changes such as power on/off status, or time remaining before a washing machine/dryer powers off. The commercial appliance refers to washing machines and dryers. For our prototype, we will be using a single washing machine. References made to commercial appliances will refer to our individual washing machine appliance. Spring Boot and AWS communicate via lambda functions to share washing machine related information such as reservation access codes.

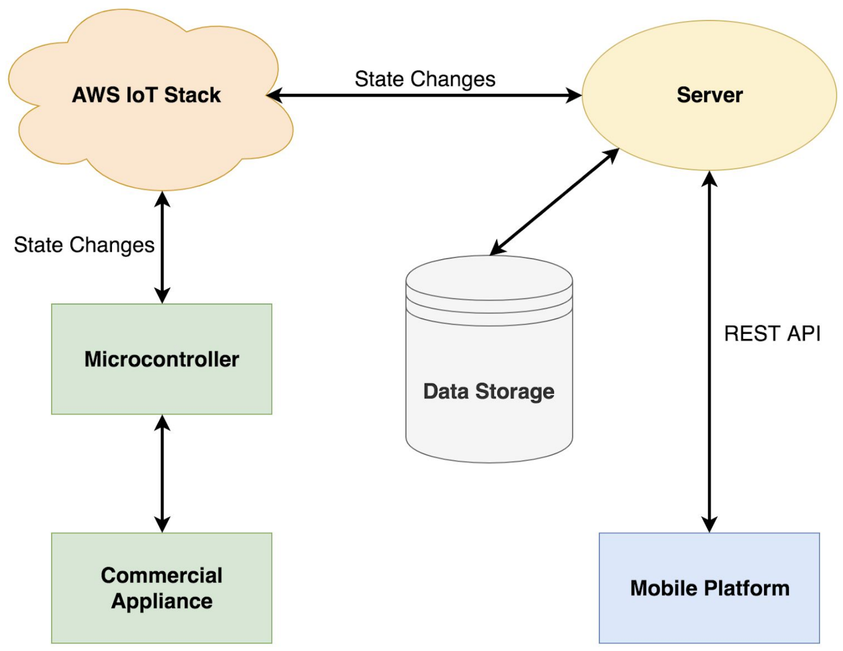


Figure 6: Architecture Diagram

4.3 Proposed Design - Mobile Application

Mobile Application

In order to create a mobile application that may run on multiple operating systems, one of two approaches may be taken: native development or cross-platform development. The native approach refers to the utilization of an operating system or IDE's core programming language. Native android applications use Java or Kotlin (a new programming language developed by Google) and native iOS applications use Swift. Cross-platform development refers to the use of specialized frameworks that can run on more than one operating system. Common cross-platform frameworks include Xamarin and React Native. Due to our team's prior experience in both native Android and iOS, the android mobile application will be developed with Java and the iOS mobile application will be developed with Swift.

4.3.1 Use Case Diagrams

Customer Use Case Diagram

At the request of our clients, the mobile application shall consist of two components: a customer scenario and an administrator scenario. The customer scenario involves a user being able to reserve a set of appliances for a fee. The administrator scenario involves an admin being able to view information about AWS IoT registered appliances. The first scenario can be seen below:

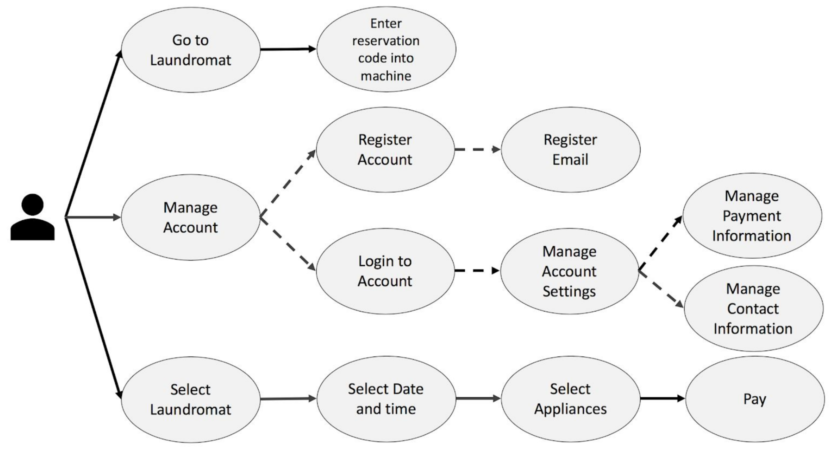


Figure 7: Customer Use Case Diagram

While using the mobile application, a customer shall be able to perform a set of actions including: Managing their account, creating a reservation, and using the appliance. Since our application requires a payment transaction service, each user must be logged in to a secure account. A valid email address is required for emailing reservation receipts. Payment information must be added to validate credit card information when attempting to pay the reservation fee. Contact information is needed for authentication and payment disputes. In order to create a reservation, the user must be able to select a laundromat location. Our client's currently own property for a single laundromat, but due to plans for expansion the mobile application will allow users to select from a list of local laundromats in Ames, IA. Once a location has been selected, a date, start time, and end time must be gathered to create a reservation. Typically, a single wash or dry cycle costs $1. The mobile application charges a convenience fee of $2 per cycle as requested by our clients. Once the reservation criteria has been added, the user must pay the fee through a secure payment transaction service. Once a reservation has been made, the user must travel to the selected laundromat location. To power on an appliance, the customer must enter a unique reservation code shown on the mobile application.

Administrator Use Case Diagram

While using the mobile application, an admin shall be able to view appliance analytics gathered from each laundromat location. Depending on the size and brand of the appliance, energy consumption may vary. The administrator shall be able to see what machines are using the largest percentage of energy. It is also likely that some machines will see more wear and tear than others due to their placement in the laundromat facility. The administrator shall be able to view what machines are active the most and what time of day sees the most customers. It has been brought to our attention that it is possible for users to create a reservation, but fail to show up at the laundromat during their reservation period. Under this exception case, an administrator must be able to override machines and allow for them to become available once again if the customer hasn't used their reservation code within a 15 minute grace period.

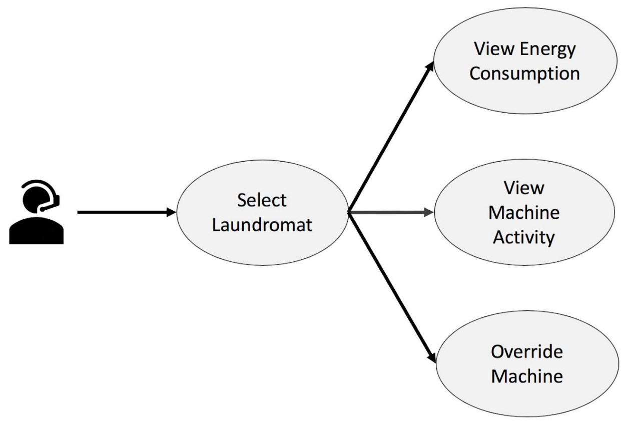


Figure 8: Administrator Use Case Diagram

4.3.2 Class Diagrams

Customer Class Diagram

To help document the software architecture of our proposed solution, our team developed a customer class diagram. The customer class diagram is to be used as a reference for the expected classes, attributes, operations, and relationships between screens. The diagram serves as a mapping of the structure for our reservation system. The proposed customer class diagram can be seen below:

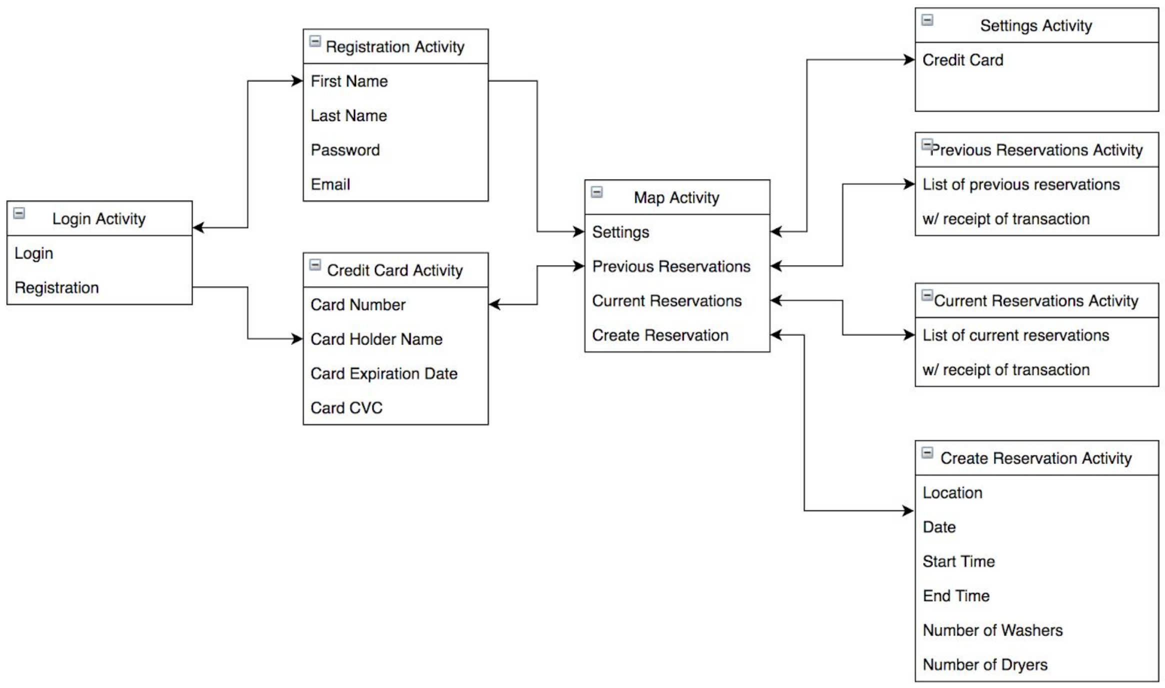


Figure 9: Customer Class Diagram

Login Activity

Mobile screen where the user may either create an account or login in with an existing account.

Registration

Mobile screen where the user creates an account by entering both identification and contact information.

Credit Card Activity

The first time a user logs in, it is common practice for the mobile application to request payment information before proceeding any further. Popular mobile applications including *Uber, Lyft, LimeBike,* and *Ofo* implement a similar structure.

Map Activity

Once a valid credit card has been submitted, the customer is taken to the home screen where they will be presented with a Google Maps screen. The screen allows for customers to search for and select laundromats near their current location to begin the reservation process. They may also navigate to their user settings where they made update contact and payment information. A recording of all current and previous reservations/transactions will be available for viewing.

Settings Activity

Mobile screen where the user may update their credit card and contact information.

Previous Reservations Activity

Mobile screen that displays a list of reservations that occurred in the past. All transactions and receipts must be available to the user.

Current Reservations Activity

Mobile screen that displays a list of reservations that are currently active. The transaction and receipt is available to the user. A cancellation fee shall be processed for the cancellation of a reservation.

Create Reservations Activity

Mobile screen that allows users to enter reservation criteria for submission. Upon submission, the customer shall be automatically billed a reservation fee and a unique reservation code will be presented to the user. The reservation code can be viewed by looking at current reservations.

Administrator Class Diagram

The administrator class diagram is also to be used as a reference for the expected classes, attributes, operations, and relationships been admin screens. The final iteration of our administrator class diagram blueprint can be seen below:

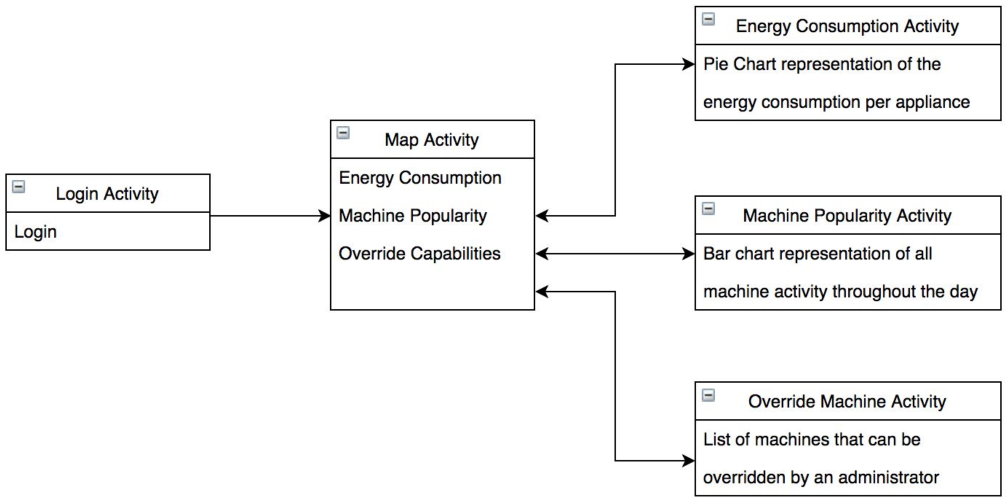

Energy Consumption Activity
Pie Chart representation of the energy consumption per appliance
Map Activity
Energy Consumption
Machine Popularity
Override Capabilities
Login Activity
Login
Machine Popularity Activity
Bar chart representation of all machine activity throughout the day
Override Machine Activity
List of machines that can be overridden by an administrator

Figure 10: Administrator Class Diagram

Login Activity

Mobile screen where our clients may login. A registration screen is not needed as an administrator shall have database read write permissions and should be added from an administrator portal. An administrator portal is out of scope for the project, but the design should reflect future expansion.

Map Activity

Once an administrator has logged into their account, they will be taken to the home screen where they will be presented with a Google Maps screen, similar to the customer home page. The screen allows for administrators to search for and select laundromats from which they own and operate. Once a laundromat has been selected, they will have the ability to view energy consumption data, machine activity data, and to override any given machine as specified in the *Terms of Service*.

Energy Consumption Activity

Mobile screen that displays a visual of the energy consumption per machine at the selected laundromat location.

Machine Popularity Activity

Mobile screen that displays a visual of the machine popularity throughout the day. For example, it may be beneficial to see that 7 machines were running at 1 pm while only 1 machine was running at 9 pm.

Override Machine Activity

Mobile screen that allows an administrator to override a machine by powering it on/off. This may be especially useful if a customer who made a reservation fails to show up for their “appointment” within the grace period identified in the *Terms of Service*.

Note: More information is given in the user interface section

4.3.3 User Interface Wireframes

Mobile Application Wireframes

A mobile application wireframe is a visual representation of the android and iOS user interface. Our team used *InVision Studio*, a professional screen design tool for developing high-end wireframes for a mobile application. Wireframes allow for us to better understand the screen-flow and use cases that our customers may take. In addition, they provide a clear visual of what the product shall look similar to when complete. The wireframes shown below will be used as guidance as we develop both the Android and iOS mobile applications.

Customer User Interface Wireframes

The customer user interface must allow for users to pay for a laundromat reservation. The first set of user interface wireframes for the customer are shown below:

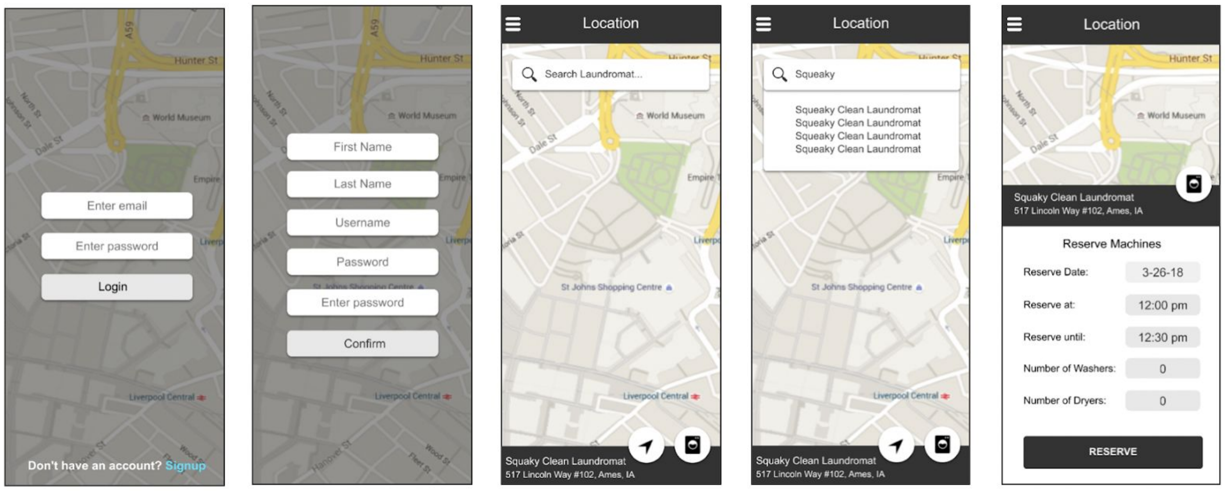


Figure 11: Customer Mobile Application Wireframes Set 1

Set 1

The first set of wireframes depict the user interface for login and reservation as outlined in the class diagram. Once a successful login attempt has been made, the customer will be taken to a homepage with Google Maps. The goal is to display markers on the map with the location of local laundromats in Ames, Iowa. The user may either select a laundromat marker or search for a laundromat to begin the reservation process. Once a laundromat has been selected, the user must provide a date and time frame for when the reservation should occur.

The second set of user interface wireframes for the customer are shown below:

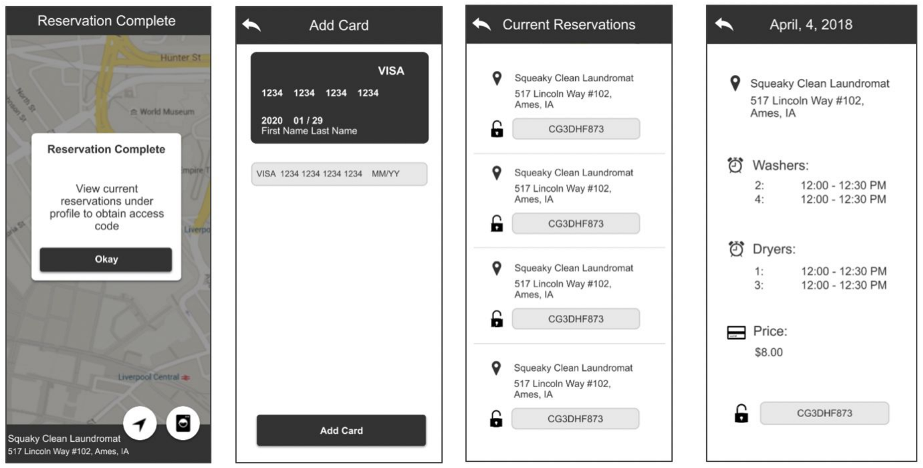


Figure 12: Customer Mobile Application Wireframes Set 2

Administrator User Interface Wireframes

The administrator user interface must allow for admins to view laundromat data analytics. The user interface wireframes for the administrator are shown below:

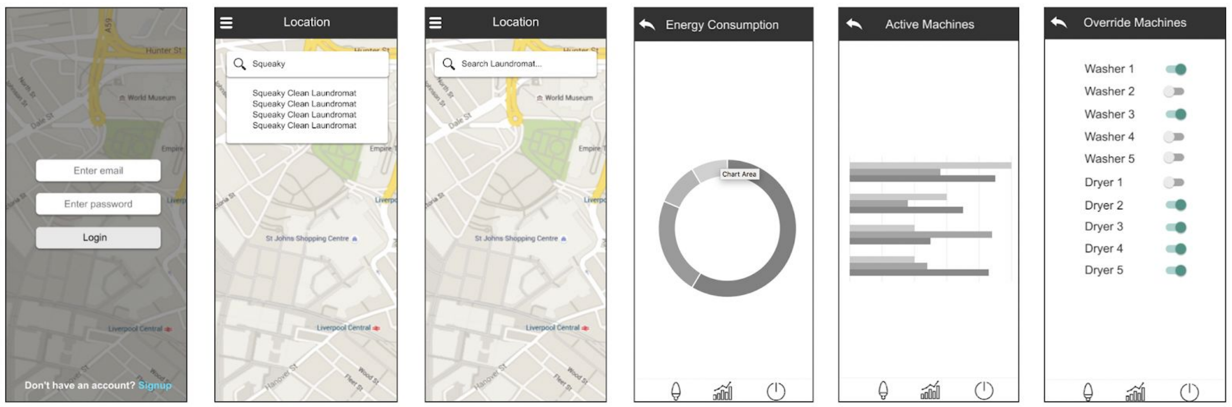


Figure 13: Administrator Mobile Application Wireframes

The administrator wireframes depict the user interface similar to that identified in the class diagram. Once the administrator has logged in successfully, the admin will be taken to a homepage with Google Maps, similar to the customer design-flow. The admin may select a laundromat marker or they may search for a laundromat. Once a laundromat has been selected, the administrator will be able to select one of three options: view energy consumption, view active machines, override machines.

4.4 Proposed Design - Backend

Overview

To both serve the many mobile application requests and connect the mobile applications to the hardware components, our team decided to use a distributed server. This server is actually a RESTful web application, and uses Spring Boot to connect to a remote database, handle unlocking requests from hardware components, and supply data to requesting mobile applications.

4.4.1 Web Application Server

The web application server was created as a standard Spring Boot application, utilizing Gradle for dependency management and Spring Data JPA with Hibernate to connect to databases. The web application server is divided into several different packages that separate out layers of the application. A ‘config’ package is where all of the connection and Spring bean configuration happens. The ‘controller’ package is where the controller layer sits, and is the starting point for all REST requests. The controller layer is the upper-most layer of the web application, and serves all incoming requests from the mobile applications. The ‘service’ package holds the service layer and sits directly below the controller layer. The service layer serves as the primary layer of business logic, modifying objects to store in the database and consolidating database information before returning it to the controller layer. The ‘repo’ package holds the repository layer, which sits directly between the service layer and the database. It accesses the database via Spring JPA and Hibernate, using standard SQL queries to look up data and return results to the service layer. The ‘entity’ package holds the database models that the rest of the application passes data around in. Each entity object defines a database table, and each entity also has its own segmented repository, service, and controller component. By segmenting the layers into components based on database models, the overall application architecture can remain clear and consistent while the web application expands throughout development.

4.4.2 AWS Cloud Deployment

For the prototype created with this project, only a single EC2 instance in AWS is necessary. By hosting the web application server on AWS, we are able to delegate most of the costs of hosting to an existing architecture and still have the web application server accessible for both mobile applications and the hardware implementation to remotely interact with. The prototype does not need to have high availability, time zone crossover, region redundancy, autoscaling, load balancers, etc. However, the prototype ha been designed in such a way that this is possible in the future, and can be implemented as described in the section on prototype commercialization.

4.4.3 Communication with Mobile Applications

The web application is primarily a RESTful web service that allows interaction with the mobile applications. It accepts many different REST requests in the controller layer, and uses standard JSON input formatting for data transference. When a REST request hits the controller layer, the data is passed through the server layer, relevant database calls are made in the repository layer, any data consolidation happens when the received data is returned to the service layer, and the final result is returned to the controller layer to send as a response to the mobile application that initiated the request.

4.4.4 Communication with Hardware

The web application uses a different method for interacting with the hardware components than it does with mobile applications. Due to the hardware being integrated with AWS IoT, it was decided that the communication between the hardware and the web application would be the default transfer method that connected the hardware with AWS IoT in the first place. It was initially thought that this transfer method was able to be easily converted into REST calls, however it was later decided that using MQTT queues made much more sense due to their integration with AWS IoT and their lightweight nature. The web application shall use an MQTT listener that is integrated with Spring Boot to listen to queues from the hardware. This MQTT listener shall be configured as beans from a file in the ‘config’ package.

4.4.5 Databases

There are two different database implementations that this project shall use for testing. The first implementation shall be an H2 embedded database that will automatically create itself each time the web application starts, and shall last only while the web application remains running. When this database starts, an SQL schema file shall be applied to set up the necessary database tables, and another SQL file shall be applied to insert test data. This H2 embedded database shall be used only for local testing, and is done for consistency and compartmentalization between developers. The second implementation shall be a MySQL database that is remotely hosted by Iowa State University. Upon connecting to this database, the SQL schema file shall be run to set up any database tables that are not already created. This allows some slight form of automation, because otherwise once a table is created then it must be updated manually if the schema ever changes in the future. This is only for the prototype, and should not be used once commercialized. For notes on commercialization, see the section on prototype commercialization.

4.5 Proposed Design - Hardware

Overview

In order to create a reservation system, our team must also implement a locking system that prevents a washing machine from powering on until the correct reservation code has been entered. Creating a locking system requires four components: an LCD-keypad user interface, a microcontroller a power source control unit, and a washing machine. The hardware system design concept is illustrated below:

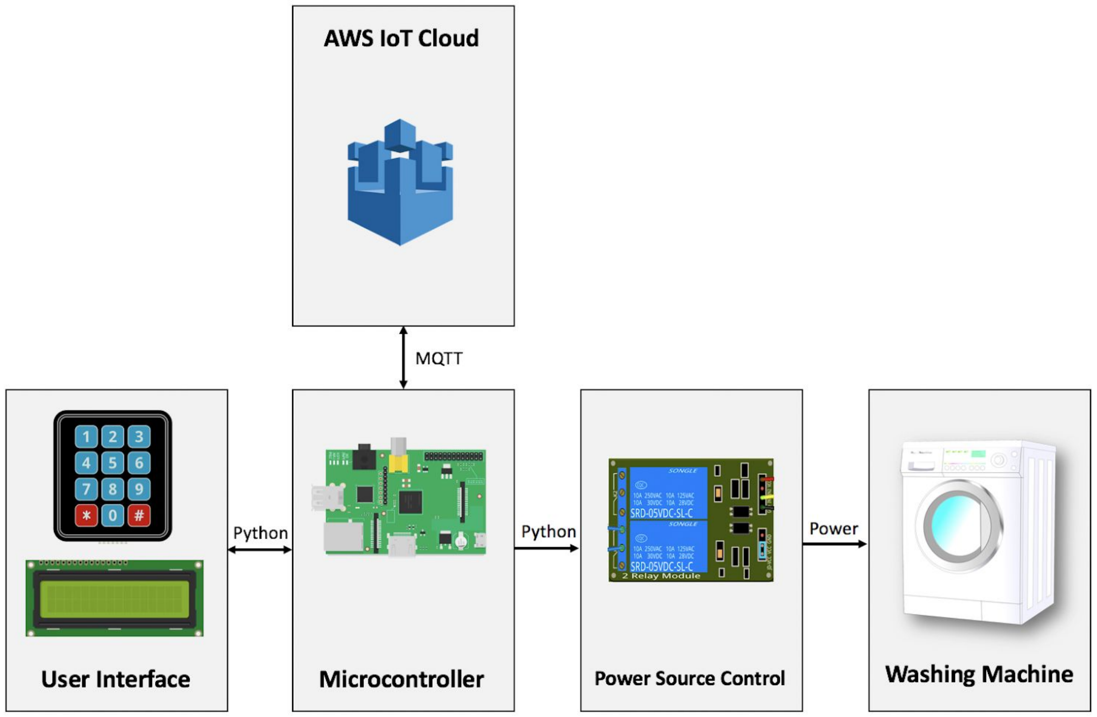


Figure 14: Hardware Concept diagram

4.5.1 Microcontroller

Before implementation can begin, our team must select a microcontroller or single-board computer that communicates with Amazon Web Services and is able to handle user input from a keypad/LCD screen user interface. Our client's recommended the following three microcontrollers: Arduino Uno, Arduino Yun, and the Raspberry Pi 3 Model B. The comparison for each microcontroller can be seen below:

Specs	**Arduino Uno**	**Arduino Yun**	**Raspberry Pi 3 Model B**
Processor	AVR ATmega328p	Atmega32u4 & Atheros AR9331	ARM1176JZF-S
Clock Speed	16 MHz	16 MHz & 400 MHz	700 MHz
RAM	2 KB	64 MB & 32 KB	512 MB
GPIO	20	20	8
Power	175 mW	N/A (~300 mW)	700 mW
WiFi & Ethernet	shield	built-in	built-in
OS & Language	none/C	Linino/C, python	Any Linux/Any
Price	$23.64	$89.95	$36.90

Table 5: Microcontroller Specifications

Arduino Uno

The arduino Uno was not selected for the following reasons:

- Lacks the computational power to support the MQTT protocol with Amazon Web Service.
- Does not come with a WiFi module on the board
- Must purchase a WiFi shield

The MQTT protocol is required to communicate with Amazon Web Service's Internet of Things as each microcontroller must be registered on the IoT service. WiFi is needed to connect with AWS to send commands, validate reservation codes, and receive status information.

Arduino Yun vs Raspberry Pi 3 Model B

Our team selected the Raspberry Pi 3 Model B for two reasons:

- Computational Power
- Budget

The Raspberry Pi 3 Model B offers a higher clock speed which means higher computational power. The higher computational power is not required for our prototype, but may provide benefit if expanded to multiple appliances. Since our clients plan on expanding the individual prototype to include a line of appliances, using a Raspberry Pi saves over 50% in costs for purchasing a stock of microcontrollers.

4.5.2 User Interface

To create a user interface that can be attached to a washing machine, our team will purchase a 9-digit keypad and an LCD screen. The 9-digit keypad will allow the user to enter a reservation code generated by the server and accessed via the android and iOS mobile application. Both components will be connected to a Raspberry Pi 3 microcontroller with jump wires. The keypad and LCD display need 23 pins and the Raspberry Pi board contains 40 GPIO pins which is sufficient for building the user interface. The 16x2 LCD screen uses 16 pins that will be separated into 3 power source pins, 9 data bus pins, and 4 control bit pins.

Fritzing

Fritzing is an open-source software tool for designing and editing circuits. Our team utilizing Fritzing to design a full-connected diagram of the hardware system as shown below:

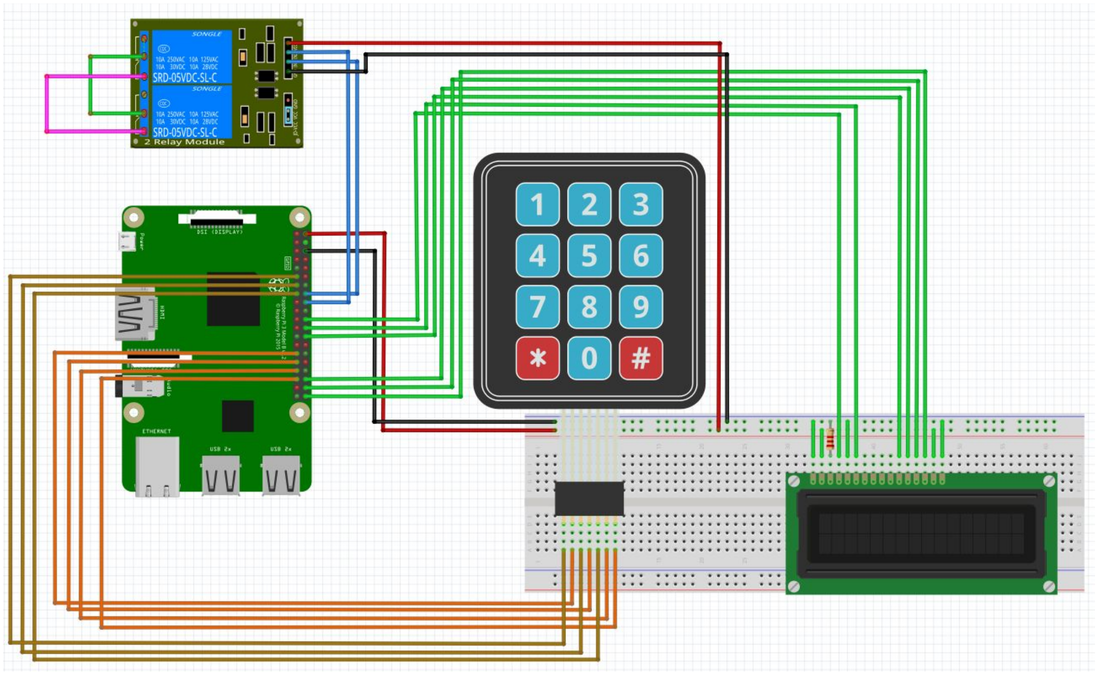


Figure 15 Full-connection electronic-hardware diagram

4.5.3 Power Source Control Unit

We have confirmed with our clients that building a device to collect information while the washing machine is running is out-of-scope for the proposed project. The primary goal is to control the power of the washing machine. To control the power to the washing machine, our team has selected to use a power relay in the circuit. There are several channel-relay modules available on the market including a 1 channel, 2 channel, 4 channel, and 8 channel relay module. It is important to note that we are only considering 250 V AC voltage / 10A current / 5V DC coil voltage. The price for the different channel-relay modules are shown below:

Channel Type	**Price**
1 Channel Relay Module	$5.80
2 Channel Relay Module	$6.79
4 Channel Relay Module	$7.86
8 Channel Relay Module	$8.98

Table 6: Channel Relay Module Cost

Cost

Since the cost for each relay module is within ~$1.00 from one another, budget was excluded from the final decision.

Decision

Implementation of our hardware system requires at minimum a 1 channel relay module. Our team selected a 4 channel relay module for two reasons:

- Safety
- Feedback Information

If a 1 channel relay module is used, a potential safety concern arises. A 1 channel relay module always has one wire branch connected with 100V AC voltage which could cause an electric shock if the circuit is not properly deployed. Therefore, it is recommended to follow the *"at least 2-channels"* rule. The 2-channels rule allows for high voltage to be detached from the circuit when not in use. The 4 channel relay module provides benefit over the 2 channel relay module as the 2 additional channels may be used to provide feedback information from the washing machine, if time allows. The remaining 2 channels will be set as "reserved channels". As a result, we have designed a simple circuit as shown below:

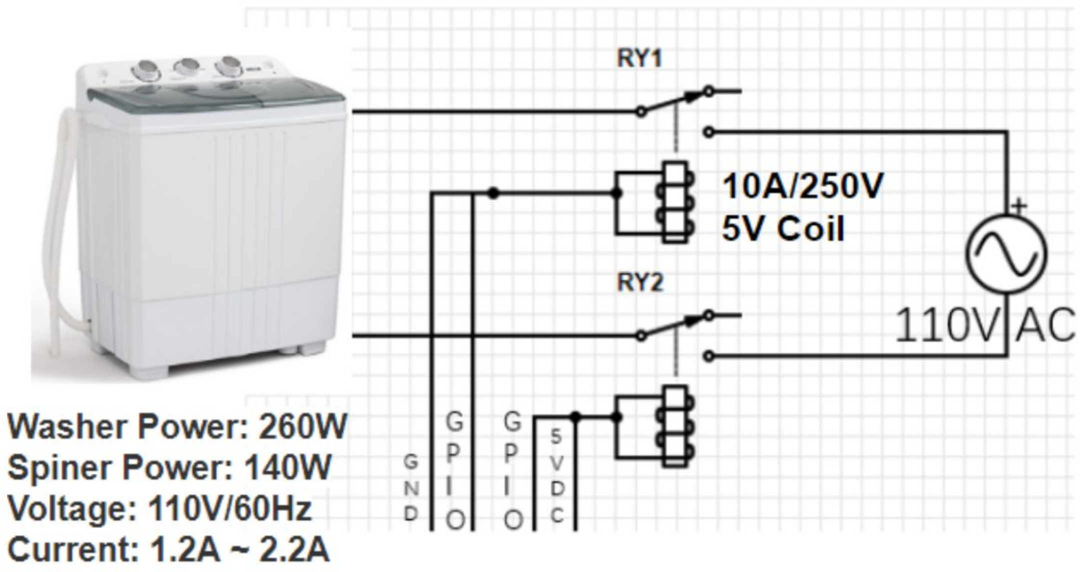


Figure 16: Circuit Diagram

4.5.4 Communication with AWS IoT

The communication between our local setup and AWS IoT is based on the Message Queuing Telemetry Transport messaging protocol. MQTT is a light-weight networking messaging protocol that has become popular recently for IoT communication. The concept behind MQTT is that devices may subscribe, publish, or listen to an AWS topic for transmitting commands. AWS IoT recommends using an MQTT implementation.

5. Implementation

5.1 Mobile Application Implementation

5.1.1 Completed Android Customer Class Diagram

As mentioned in the design section, a class diagram helps document the software architecture and serves as a mapping of the structure for our reservation system.The final version of the customer class diagram is shown below:

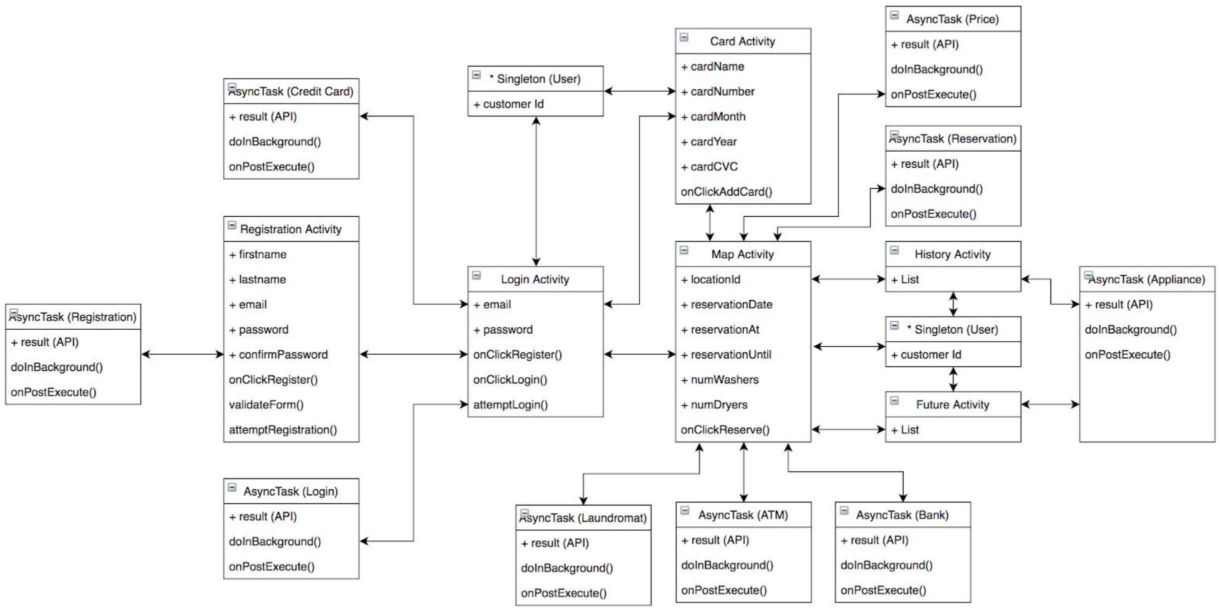


Figure 17: Complete Android Customer Class Diagram

Login Activity

The login activity requests an email address and password input from the customer in order to validate a login. When the user clicks the login button, an asynchronous task is created to make an API call to login. The asynchronous task prevents blocking on the main UI thread which could lead to an application crash. The onPostExecute() function is similar to a callback and is called when the API returns with data. On success, the API returns a valid customer ID for the logged in user or -1 on error. If the user fails to login, both input fields display an error message indicating an incorrect email or password. In addition to a valid customer ID being returned, a second asynchronous task is called to see if a credit card has been previously stored. If no credit card is on file, then the user is prompted to enter valid credit card information. Otherwise, they will be taken directly to the home page (Map Activity). If the user does not have an account, they may click the register button to navigate to a registration form.

Registration Activity

The registration activity requests identification information and contact information from the customer. Email validation occurs on the client-side. When the user clicks the register button, an asynchronous task is created to make an API to register the user. On success, the onPostExecute() function returns a valid customer Id and navigates back to the login screen.

Map Activity

The Map Activity is the home screen of the application. Users are provided with a screen displaying Google Maps from the *Google Maps Android SDK*. We used the open-source map database called *openstreetmap.org* to query for all Laundromat, Bank, and ATMs located in the Ames, IA region. Since the data is open-source, we are allowed to use the information so long as we credit *openstreetmap.org* for the data. Three separate asynchronous tasks were written to retrieve the laundromat, bank, and atm data from our database without blocking the main UI thread. To make a reservation, the user must click on a laundromat marker. A marker in Google Maps is a pin that appears at a given latitude longitude coordinate with an icon, title, and subtitle. In our case, the marker displays an icon of a laundromat with the name of the laundromat and the address of the laundromat. When the user clicks on the marker, a slider menu appears from the bottom of the screen that allows the user to make a reservation. The Map Activity also has a bottom navigation toolbar where the user may navigate to the settings activity, previous reservations activity, or current reservations activity.

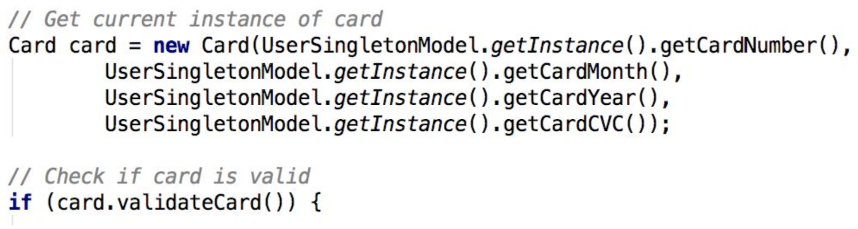
Reservation

Creating a reservation requires the following input information: reservation date, start time, end time, number of washing machines, and the number of dryers. As the user updates the number of washers and dryers, an asynchronous task is created to communicate with the Spring Boot Web Server to determine the current price based on the input parameters. When the user clicks the reserve button, a chain of callbacks happen. An asynchronous task is created to verify with the web server if enough appliances are available for the requested reservation. If so, the *Stripe SDK* is used to handle the payment in a secure manner. Otherwise, an alert displays on the screen to indicate that there are not enough appliances available.

Strike SDK - Payment Transactions

The *Stripe SDK* built by *Stripe* is a popular library that lets applications accept mobile payments and manage customer information from the stripe dashboard. By doing so, Stripe receives a small percentage of the total transaction. *Stripe* provides functionality for validating credit card input, securing credit card information, and creating payment charges.

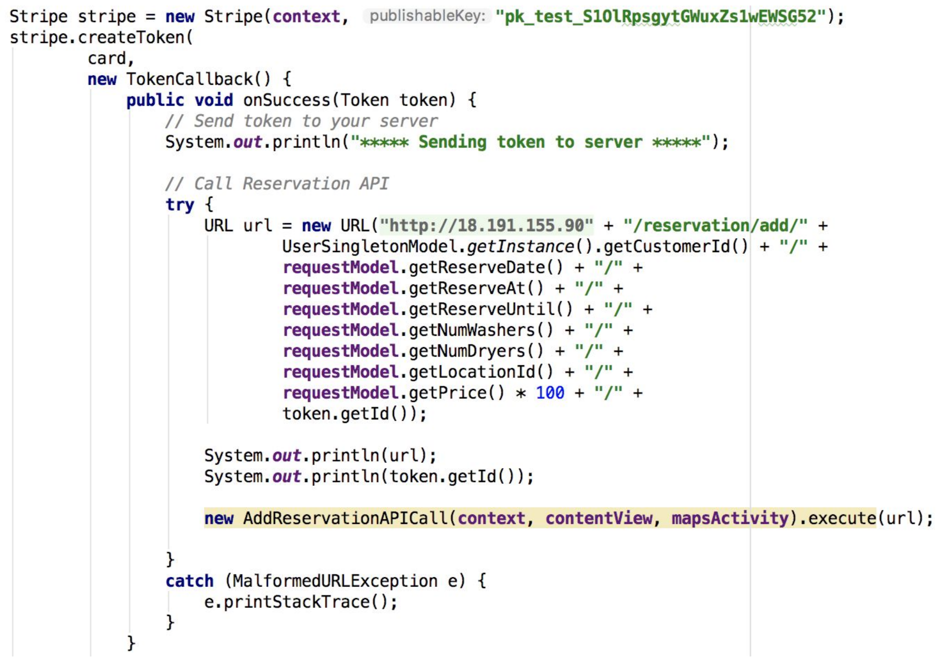
Credit Card Validation



```
// Get current instance of card
Card card = new Card(UserSingletonModel.getInstance().getCardNumber(),
        UserSingletonModel.getInstance().getCardMonth(),
        UserSingletonModel.getInstance().getCardYear(),
        UserSingletonModel.getInstance().getCardCVC());

// Check if card is valid
if (card.validateCard()) {
```

Figure 18: Stripe Credit Card Validation

Secure Payment Token



```
Stripe stripe = new Stripe(context,  publishableKey: "pk_test_S1OlRpsgytGWuxZs1wEWSG52");
stripe.createToken(
        card,
        new TokenCallback() {
            public void onSuccess(Token token) {
                // Send token to your server
                System.out.println("***** Sending token to server *****");

                // Call Reservation API
                try {
                    URL url = new URL("http://18.191.155.90" + "/reservation/add/" +
                            UserSingletonModel.getInstance().getCustomerId() + "/" +
                            requestModel.getReserveDate() + "/" +
                            requestModel.getReserveAt() + "/" +
                            requestModel.getReserveUntil() + "/" +
                            requestModel.getNumWashers() + "/" +
                            requestModel.getNumDryers() + "/" +
                            requestModel.getLocationId() + "/" +
                            requestModel.getPrice() * 100 + "/" +
                            token.getId());

                    System.out.println(url);
                    System.out.println(token.getId());

                    new AddReservationAPICall(context, contentView, mapsActivity).execute(url);

                }
                catch (MalformedURLException e) {
                    e.printStackTrace();
                }
            }
```

Figure 19: Stripe Payment Tokenizer

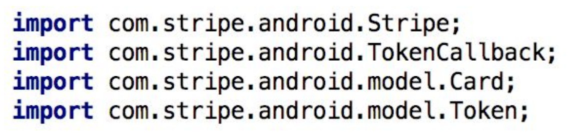
Reservation

As mentioned above, when the user clicks the reserve button, a Stripe payment token is created. To create a stripe payment token, a credit card must be supplied to the stripe tokenizer. The stripe tokenizer will create a token. If the creation of the token is successful then the onSuccess() function will execute and an API call is made to our Spring Boot Web Server to create the reservation and to create a charge line. The reservation API call is launched from an asynchronous task that doesn't block the main UI thread. On the server, we utilize stripe to create a charge line. A charge line is what executes the actual charge and and creates a receipt that can be viewed on the stripe dashboard.

Credit Card Activity

The credit card activity uses the Stripe credit card widget to handle user input. The stripe import for the credit card widget is as follows:

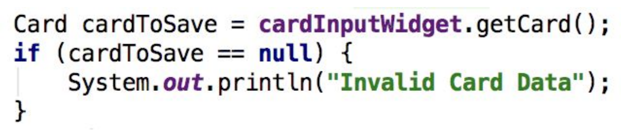
Stripe import



```
import com.stripe.android.Stripe;
import com.stripe.android.TokenCallback;
import com.stripe.android.model.Card;
import com.stripe.android.model.Token;
```

Figure 21: Stripe Imports

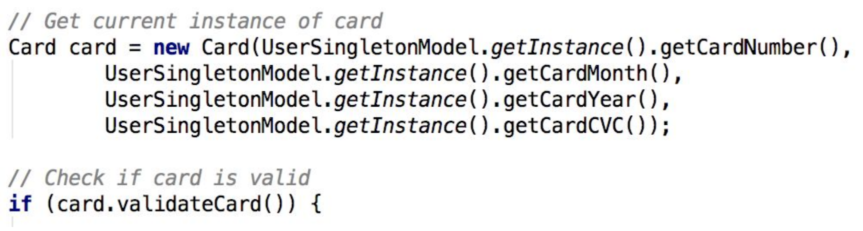
Stripe Card Widget



```
Card cardToSave = cardInputWidget.getCard();
if (cardToSave == null) {
    System.out.println("Invalid Card Data");
}
```

Figure 22: Stripe Card Widget

Stripe Card Validation



```
// Get current instance of card
Card card = new Card(UserSingletonModel.getInstance().getCardNumber(),
        UserSingletonModel.getInstance().getCardMonth(),
        UserSingletonModel.getInstance().getCardYear(),
        UserSingletonModel.getInstance().getCardCVC());

// Check if card is valid
if (card.validateCard()) {
```

Figure 23: Stripe Credit Card Validation

When the user clicks the Add Card button, an asynchronous task is created to send the valid credit card to the web server for storage.

History Activity

The history activity is used to display a list of all previous reservations. A specific reservation in the list may be selected to view additional details. The initial details that are visible in the list of reservations include: laundromat name, laundromat address, and reservation code. The expanded reservation displays additional information including: appliance number, start time, end time, and a receipt with the price. An asynchronous task is created to retrieve all previous reservations, given a customer id.

Future Activity

The future activity is used to display a list of all current reservations. A specific reservation in the list may be selected to view additional details. The initial details that are visible in the list of reservations include: laundromat name, laundromat address, and reservation code. The expanded reservation displays additional information including: appliance number, start time, end time, and a receipt with the price. An asynchronous task is created to retrieve all current reservations, given a customer id.

5.1.2 Completed Android Administrator Class Diagram

The final version of the administrator class diagram is shown below:

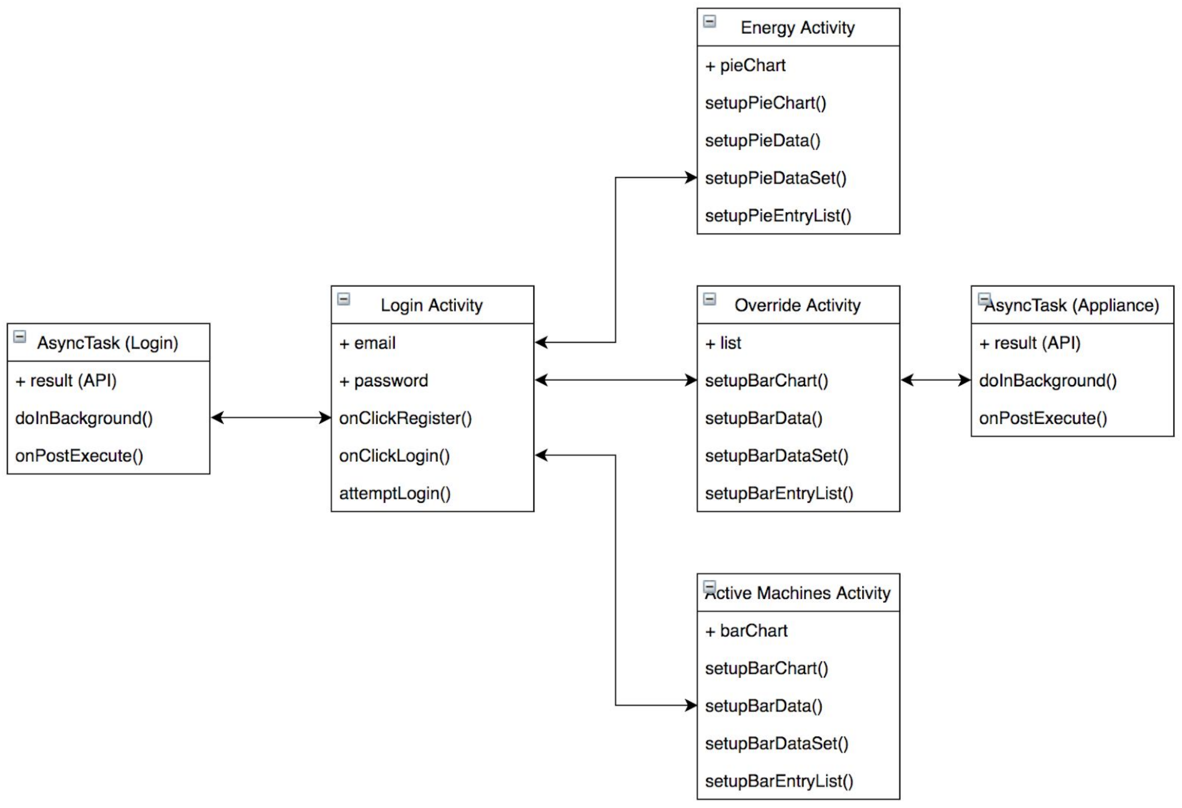


Figure 24: Complete Administrator Class Diagram

Login Activity

The login activity requests an email address and pass input form the administrator in order to validate a login. When the admin clicks the login button, an asynchronous task is created to make an API call to login. On success, the API returns a valid administrator ID or -1 on error. If the administrator fails to login, both input fields display an error message indicating an incorrect email or password. Otherwise they will be taken to the home page (Map Activity). An administrator account is not made through the mobile application. An administrator account must be added directly to the database. With future expansion, a website administrator dashboard will be used manage admin accounts.

Map Activity

The Map Activity is the home screen of the application. Similar to the customer flow, administrators are also provided with a screen displaying Google Maps along with a set of laundromat markers. The administrator may click on a laundromat marker to see a toolbar of analytics options for the laundromat. Those include: energy data, active data, and override capabilities.

Energy Activity

The Energy Activity uses a pie chart to display how much energy each appliances uses with respect to the total energy consumption. The scope of our project does not include building hardware to calculate the energy consumption per machine. Rather, we were asked to develop a user interface proof of concept that could be used with future expansion.

Active Machines Activity

The Active Machines activity uses a horizontal bar chart to display how many appliances are active for each hour of the day. For example, the graph will display if 7 appliances were reserved at 1:00 pm, but only 2 appliances were reserved at 9:00 pm. Similar to the energy activity, the scope of our project does not include registering more than one device. Instead, we use the reservation data as a reference for populating the bar chart. With future expansion, both graphs may be of benefit to our clients.

MPAndroidChart Library

The MPAndroidChart library is an android and iOS library built for implementing professional and high-quality visual data displays. Our team used the library to design both the pie chart and horizontal bar chart for the energy activity and active machines activity.

5.1.3 Completed Android Customer Interface

The complete customer android interface is shown below. For a more detailed view of each screen, please refer to the operation manual in section 6.

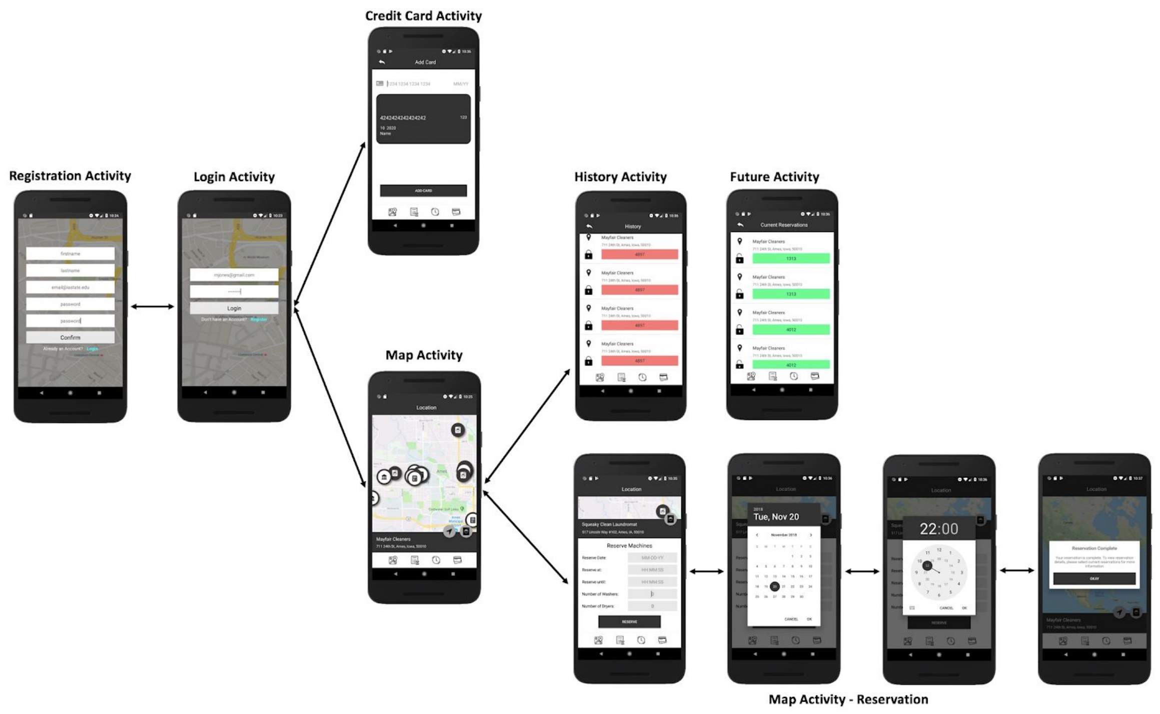


Figure 25: Complete Customer UI

5.1.4 Completed Android Administrator Interface

The complete customer android interface is shown below. For a more detailed view of each screen, please refer to the operation manual in section 6.

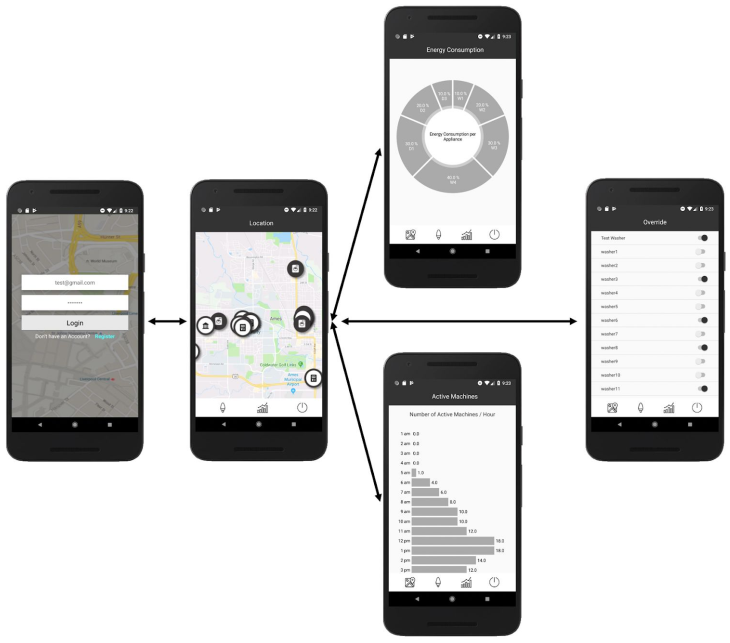


Figure 26: Complete Administrator UI

5.1.5 Completed iOS Customer Class Diagram

As mentioned in the design section, a class diagram helps document the software architecture and serves as a mapping of the structure for our reservation system.The final version of the customer class diagram is shown below:

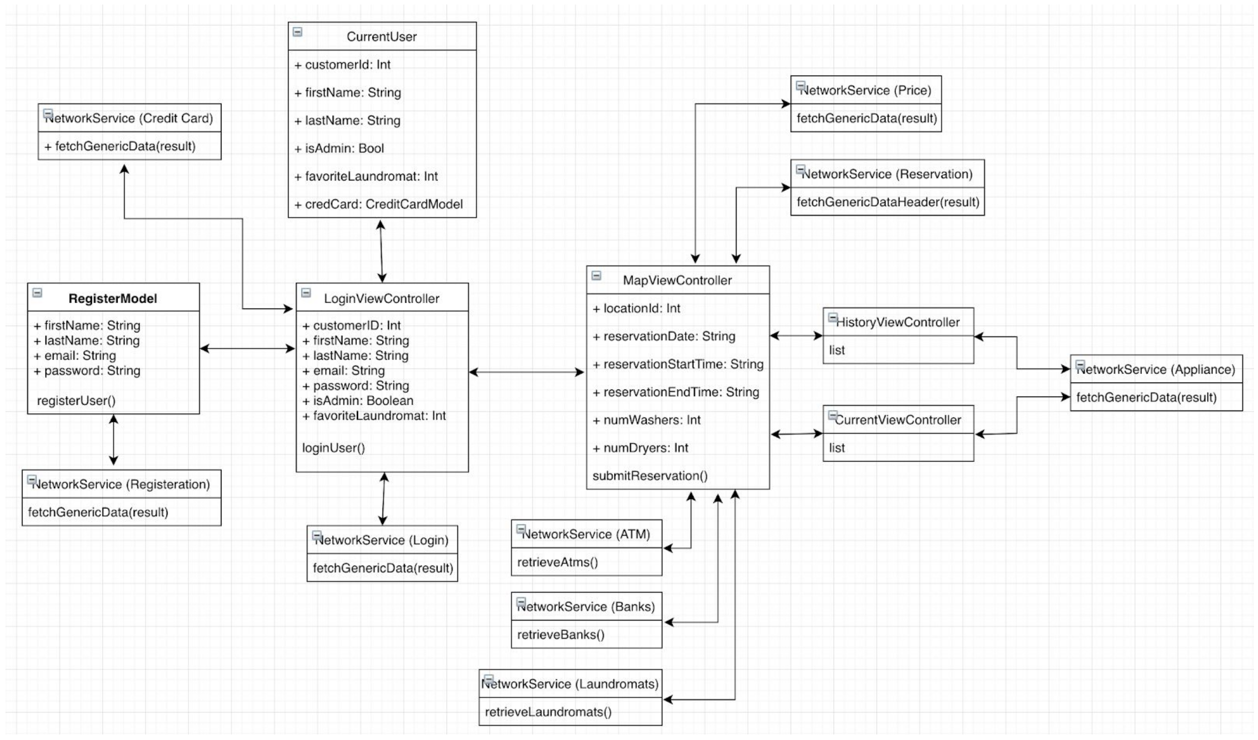


Figure 27: Complete iOS Customer Class Diagram

Login View Controller

The login view controller presents a form for a user to login. The login form uses an email and password to validate who the user is. When the user submits a user and password they are sent to the server using a URLSession. If the login was successful the user's first name, last name, customer id, whether they are an admin and their favorite laundromat location if they have one. In case that the login fails because either the user doesn't exist of the user entered a wrong password the user will be prompted to retry their email and password. Once a successful user has logged in the user will be presented the map view controller.

If a new user has downloaded the application they can register. By tapping the "SignUp" button in the login view the user will be presented a form that accepts values such as first name, last name, email, and password. After the user enters the required information it will be send to the service using a URLSession. After a user is successfully create on the server the same information if returned as a login and the user to presented the map view controller.

Map View Controller

The map view controller is the main screen that the user will interact with. The user will be presented with a screen showing the google maps sdk. The map is populated with banks, atms, and laundromats in the users area and the location of the banks, atms, and laundromats are all stored on the backend. We use openstreetmap.org to fetch all these locations. To make sure that the fetching of all these locations don't block the main thread a URLSession is used to obtain the required information to display the icons on the map. For a user to make a reservation they have to select laundromat location which is represented on the map by a washing machine. Tapping a laundromat will popup a bar on the bottom of the map and in that bar is another button with a washing machine as the icon and this will take the user to the reservation form

Reservation

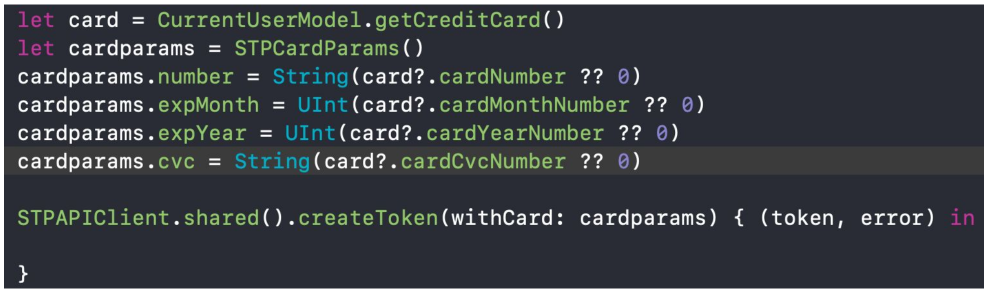
The required information that is needed for a user to create a reservation are the following: the date of the reservation, the start and end time of the reservation, number of washers, and the number of dryers. As the user is inputting this information an URLSession will be created to retrieve a live value that shows the user the current price of the reservation. This price amount will be updated whenever the user changes the amount of washers or dryers and the length of the desired reservation is used to make the price calculation. Once the user finishes inputting the required information they can submit the reservation which is sent to the backend for the payment to authenticated by Stripe and the reservation to be stored in the database.

Stripe Payments

The Stripe SDK is used to authenticate a payment for a reservation submitted by the user. For a more detailed summary on why we use the Stripe SDK reference android customer class diagram section of the paper.

Token Creation

Stripe provides a function to create a token given a credit card. The implementation is below. In the code below a token is returned from the provided Stripe function. After a token is obtained a reservation can be sent to the backend to be processed and authenticated.



```swift
let card = CurrentUserModel.getCreditCard()
let cardparams = STPCardParams()
cardparams.number = String(card?.cardNumber ?? 0)
cardparams.expMonth = UInt(card?.cardMonthNumber ?? 0)
cardparams.expYear = UInt(card?.cardYearNumber ?? 0)
cardparams.cvc = String(card?.cardCvcNumber ?? 0)

STPAPIClient.shared().createToken(withCard: cardparams) { (token, error) in

}
```

Figure 28: iOS Stripe Tokenizer

History View Controller

The history view controller displays a list of past reservations made by the current user user logged in. For each reservation in the list the start time, end time, reservation date, appliance number, and the code that was required to use the machine. The list of past reservations is fetched from the server using a URLSession so it doesn't block main thread.

Current View Controller

The current view controller displays a list of current reservation made the current user logged in. These reservations that have start dates and start time in the future. For each reservation in the list the start time, end time, reservation date, appliance number, and the code required to use the machine. The list of current reservations is fetched from the server using a URLSession so it doesn't block the main thread.

5.1.6 Completed iOS Administrator Class Diagram

The final version of the administrator class diagram is shown below:

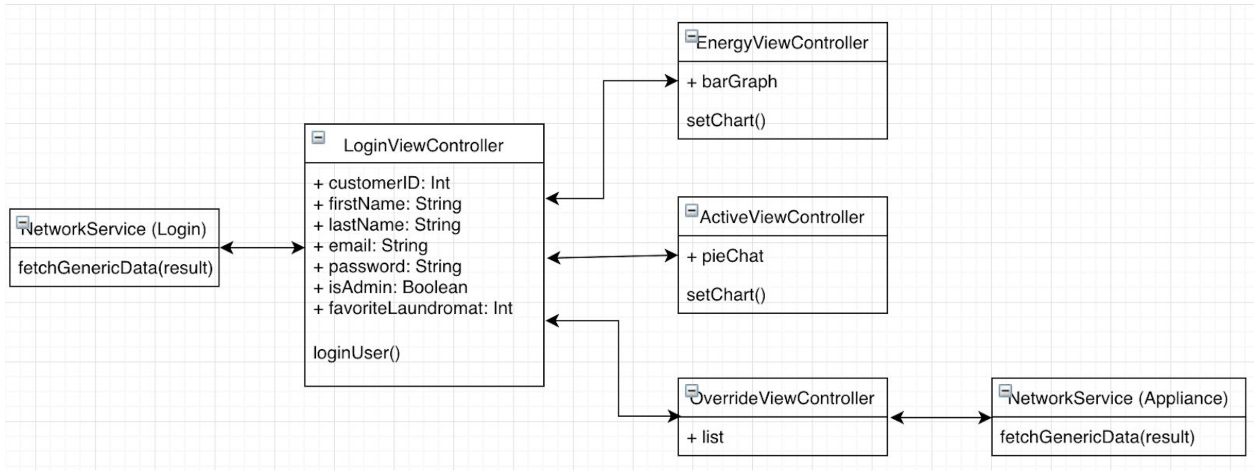


Figure 29: Complete iOS Administrator Class Diagram

Login View Controller

The login view controller presents a form for a user to login. The login form uses an email and password to validate who the user is. When the user submits a user and password they are sent to the server using a URLSession. If the login was successful the user's first name, last name, customer id, whether they are an admin and their favorite laundromat location if they have one. In case that the login fails because either the user doesn't exist of the user entered a wrong password the user will be prompted to retry their email and password. Once a successful user has logged in the user will be presented the map view controller.

Admin Map View Controller

The admin map view controller is the almost the same as the map view controller except the admin is only shown the laundromats. When the admin selects a location they will be able to look at statistics about the machines at that location such as energy used by each machine, how long each machine has been active, and override capabilities.

Active View Controller

The active view controller shows a bar graph using Charts to show how many machines are active for each hour of the day. Right now in the application the date that is shown in the bar graph is just placeholder. In a real product environment the data would be from a real machine and pulled from the backend.

Energy View Controller

The energy view controller shows a pie chart using Charts to show the amount of energy used by each machine in the laundromat. Same as for the active view controller the data being displayed is just a placeholder. In a real product environment the data would be from real machines and pulled from the backend.

Override View Controller

The override view controller is a list of machines at the selected laundromat. The administrator is able to override a machine by shutting the power off or on.

Charts

Charts if the library used to create the bar graph and pie charts for the active view controller and the energy view controller. This library is the same as the android MPAndroidChart library just written in Swift for the iOS platform.

5.1.7 API Table

To send and receive information on the Android and iOS mobile application, several APIs were implemented to communicate with the Web Server. The following APIs were used in the final version of our prototype:

API	Description
http://18.191.155.90/login/ username/password	Sends a username and password to the Spring Boot Web Server to check for validation. If valid, the API returns a valid customer Id or -1
http://18.191.155.90/cardInfo/customerId	Sends a customer ID to the Spring Boot Web Server to retrieve credit card information. If valid, the API returns card information
http://18.191.155.90/signup/ firstname/lastname/email/ password	Sends a first name, last name, email, and password to the Spring Boot Web Server to register a user.
http://18.191.155.90/ reservation/price/ reservationDate/reservationAt/ reservationUntil/numWashers/ numDryers/locationId	Sends reservation input data to the Spring Boot Web Server to determine the reservation price
http://18.191.155.90/ appliance/locationId/ reservationDate/ reservationAt/reservationUntil	Sends a location, reservate date and reservation time to the Spring Boot Web Server to retrieve a list of appliances
http://18.191.155.90/laundromat/list	Retrieves a list of laundromats in Ames, IA
http://18.191.155.90/bank/list	Retrieves a list of banks in Ames, IA

http://18.191.155.90/atm/list	Retrieves a list of ATMs in Ames, IA
http://18.191.155.90/reservation/history	Retrieves a list of previous reservations
http://18.191.155.90/reservation/future	Retrieves a list of current (future) reservations
http://18.191.155.90/ reservation/add/customerId/ reservationDate/ reservationAt/ reservationUntil/ numWashers/ numDryers/locationId/price	Sends reservation input data to the Spring Boot Web Server to create a new reservation

Table 7: API List

5.2 Backend Implementation

5.2.1 Server Layers

The back end development team has put together a server that serves 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 server layer architecture diagram is shown below:

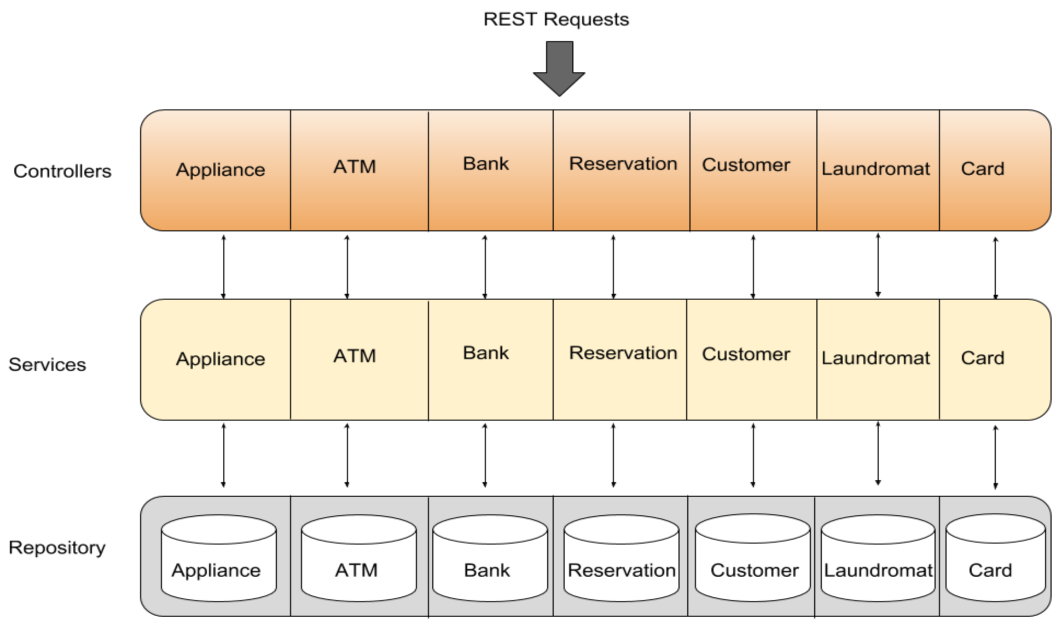


Figure 30: Server Layer Architecture Diagram

5.2.2 Controller Layer

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 calls the next layer, the Service layer, to serve responses back to the client.

5.2.3 Service Layer

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, create or insert 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.

5.2.4 Repository 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.

5.2.5 Databases

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 with test data. The Database schema is shown below:

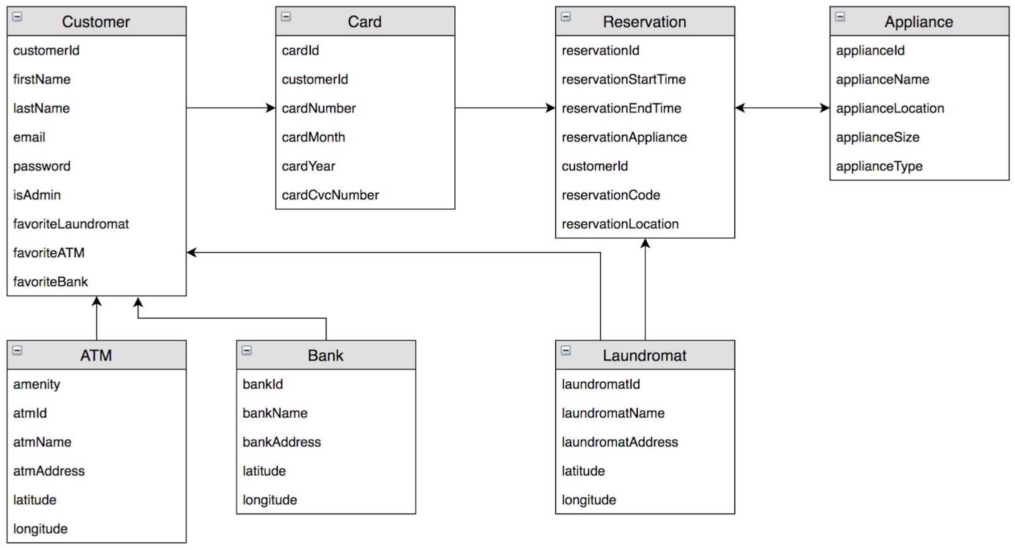


Figure 31: Database Schema

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 is used when the back end is set up on a remote server and allows 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.

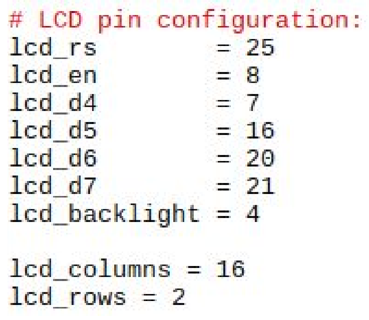
5.2.6 Spring Security

The final portion of the back end is Spring Security. Spring Security allows 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 is integrated with the Controller layer, but it sits separately from the other layers and is integrated with the application’s custom user storage mechanism.

5.3 Hardware Implementation

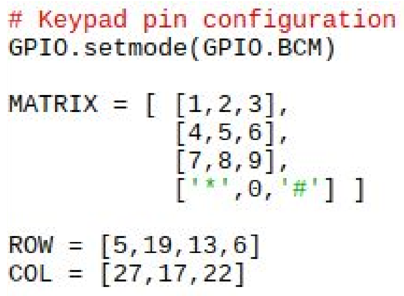
5.3.1 GPIO Pins

To implement a locking mechanism with a Raspberry Pi 3 microcontroller, a 16x2 LCD Screen and a 9-digit keypad, GPIO pins needed to be defined. A GPIO pin is a general-purpose input/output pin that is part of the Raspberry Pi 3. In fact, the Raspberry Pi 3 contains a 40-pin GPIO header that may be configured. For the LCD pin to work, we used the following LCD pin configuration:



```
# LCD pin configuration:
lcd_rs        = 25
lcd_en        = 8
lcd_d4        = 7
lcd_d5        = 16
lcd_d6        = 20
lcd_d7        = 21
lcd_backlight = 4

lcd_columns = 16
lcd_rows = 2
```

Figure 32: LCD Pin configuration



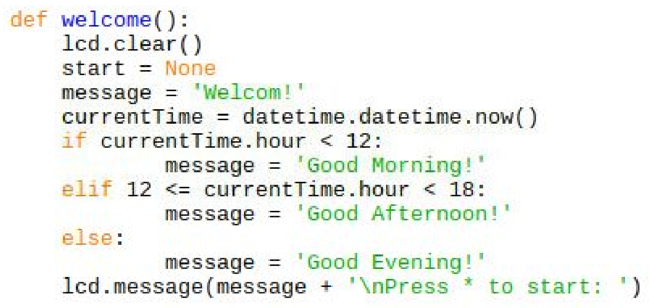
```
# Keypad pin configuration
GPIO.setmode(GPIO.BCM)

MATRIX = [ [1,2,3],
           [4,5,6],
           [7,8,9],
           ['*',0,'#'] ]

ROW = [5,19,13,6]
COL = [27,17,22]
```

Figure 33: Keypad Pin Configuration

The keypad matrix represents the possible input values that can be obtained from the keypad. Our team purchased a 9-digit keypad with two special characters: *, #. The keypad contains four rows consisting of 3 buttons per row. Therefore the keypad matrix is: [[1, 2, 3], [4, 5, 6], [7, 8, 9], ["*", 0, "#"]].

5.3.2 LCD Screen

The 16x2 LCD Screen is used to display greeting messages and to display each reservation code digit entered by the customer. We wanted to display a welcoming message to the user based on the current time of day. The python script for the LCD screen is shown below.

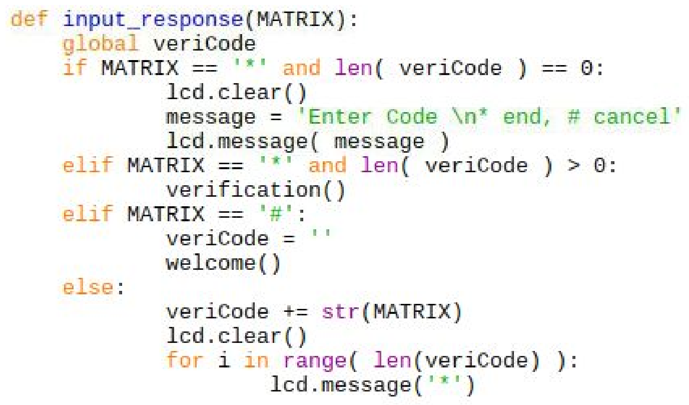


```
def welcome():
    lcd.clear()
    start = None
    message = 'Welcom!'
    currentTime = datetime.datetime.now()
    if currentTime.hour < 12:
            message = 'Good Morning!'
    elif 12 <= currentTime.hour < 18:
            message = 'Good Afternoon!'
    else:
            message = 'Good Evening!'
    lcd.message(message + '\nPress * to start: ')
```

Figure 34: LCD Welcome Script

5.3.3 Keypad Response

To begin entering a reservation code, the user is prompted by the LCD screen with instructions for beginning the process. The function tied to this is the input_response() function. The input_response() function is used to determine what interaction the customer would like to take. For example, the user may choose to enter a reservation code, they may choose to cancel the current input, or they may choose to submit the reservation code input. The python script for the input_response() function is shown below:

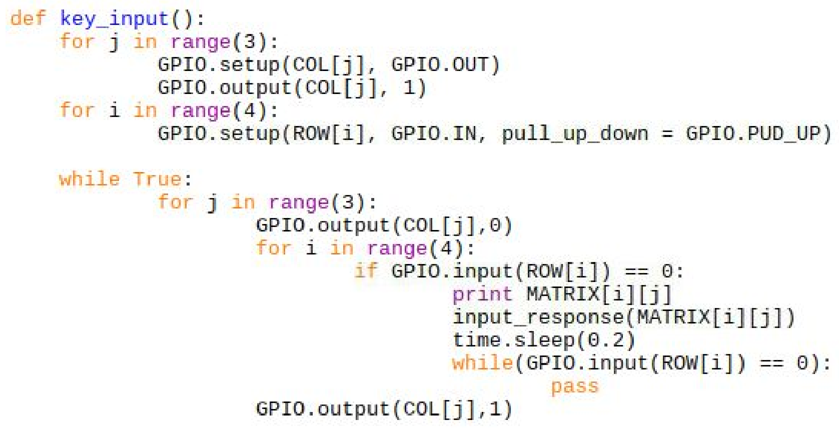


```
def input_response(MATRIX):
    global veriCode
    if MATRIX == '*' and len( veriCode ) == 0:
            lcd.clear()
            message = 'Enter Code \n* end, # cancel'
            lcd.message( message )
    elif MATRIX == '*' and len( veriCode ) > 0:
            verification()
    elif MATRIX == '#':
            veriCode = ''
            welcome()
    else:
            veriCode += str(MATRIX)
            lcd.clear()
            for i in range( len(veriCode) ):
                    lcd.message('*')
```

Figure 35: Keypad Prompt Script

5.3.4 Keypad Input

As the customer enters the reservation code, we must obtain each input value. To receive keypad input from the customer, the key_input() function is required. The key_input() function is shown below:

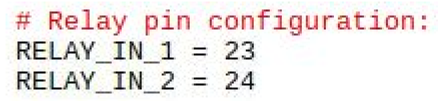


```
def key_input():
    for j in range(3):
            GPIO.setup(COL[j], GPIO.OUT)
            GPIO.output(COL[j], 1)
    for i in range(4):
            GPIO.setup(ROW[i], GPIO.IN, pull_up_down = GPIO.PUD_UP)

    while True:
            for j in range(3):
                    GPIO.output(COL[j],0)
                    for i in range(4):
                            if GPIO.input(ROW[i]) == 0:
                                    print MATRIX[i][j]
                                    input_response(MATRIX[i][j])
                                    time.sleep(0.2)
                                    while(GPIO.input(ROW[i]) == 0):
                                            pass
                    GPIO.output(COL[j],1)
```

Figure 36: Keypad Input Script

5.3.5 Relay Module

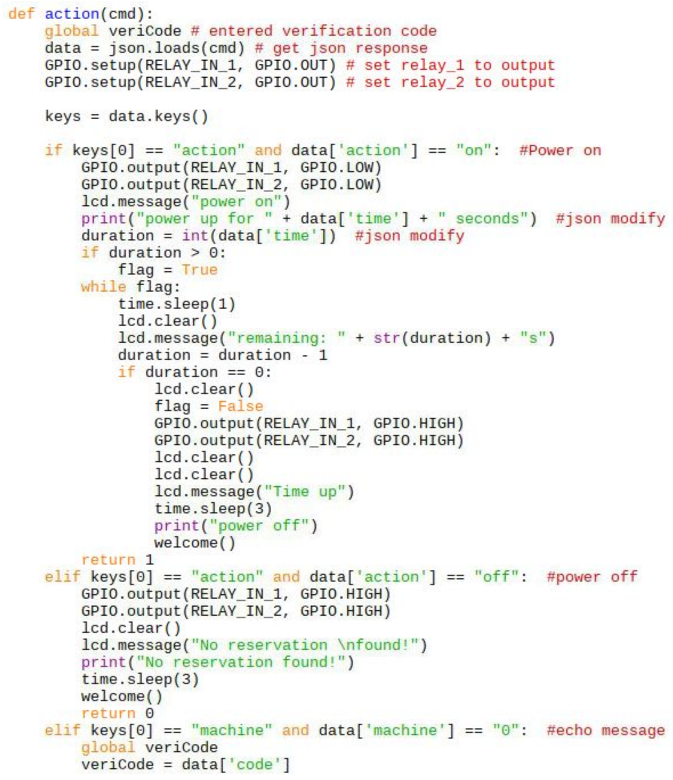
To complete the hardware module of our project, a 4-channel relay module needed to be implemented with the Raspberry Pi 3. Configuration for the 4-channel relay module only required to relay module pins. This is because, only two commands were needed for RELAY_IN_x. The RELAY_IN_x represents the RY1 and RY2 pins shown shown.



```
# Relay pin configuration:
RELAY_IN_1 = 23
RELAY_IN_2 = 24
```

Figure 37: Relay Module Pins

As mentioned earlier, the Raspberry Pi 3 microcontroller communicates with AWS IoT to send and receive commands. The microcontroller receives time information from AWS identifying the time remaining on the washing machine before it powers off. The microcontroller must also send on/off status to AWS. A function called action() was written to handle the parsing of different inputs. A JSON formatted command is parsed by the function to retrieve the on/off status of the washing machine. The entire action() function is shown below:



```
def action(cmd):
    global veriCode # entered verification code
    data = json.loads(cmd) # get json response
    GPIO.setup(RELAY_IN_1, GPIO.OUT) # set relay_1 to output
    GPIO.setup(RELAY_IN_2, GPIO.OUT) # set relay_2 to output

    keys = data.keys()

    if keys[0] == "action" and data['action'] == "on":  #Power on
        GPIO.output(RELAY_IN_1, GPIO.LOW)
        GPIO.output(RELAY_IN_2, GPIO.LOW)
        lcd.message("power on")
        print("power up for " + data['time'] + " seconds")  #json modify
        duration = int(data['time'])  #json modify
        if duration > 0:
            flag = True
        while flag:
            time.sleep(1)
            lcd.clear()
            lcd.message("remaining: " + str(duration) + "s")
            duration = duration - 1
            if duration == 0:
                lcd.clear()
                flag = False
                GPIO.output(RELAY_IN_1, GPIO.HIGH)
                GPIO.output(RELAY_IN_2, GPIO.HIGH)
                lcd.clear()
                lcd.clear()
                lcd.message("Time up")
                time.sleep(3)
                print("power off")
                welcome()
        return 1
    elif keys[0] == "action" and data['action'] == "off":  #power off
        GPIO.output(RELAY_IN_1, GPIO.HIGH)
        GPIO.output(RELAY_IN_2, GPIO.HIGH)
        lcd.clear()
        lcd.message("No reservation \nfound!")
        print("No reservation found!")
        time.sleep(3)
        welcome()
        return 0
    elif keys[0] == "machine" and data['machine'] == "0":  #echo message
        global veriCode
        veriCode = data['code']
```

Figure 38: Action Function

5.3.6 Completed Locking Mechanism

The completed locking mechanism is shown below. The final prototype version consists of a 9-digit keypad, a 16x2 LCD Screen, a Raspberry Pi 3, and a 4-channel relay.

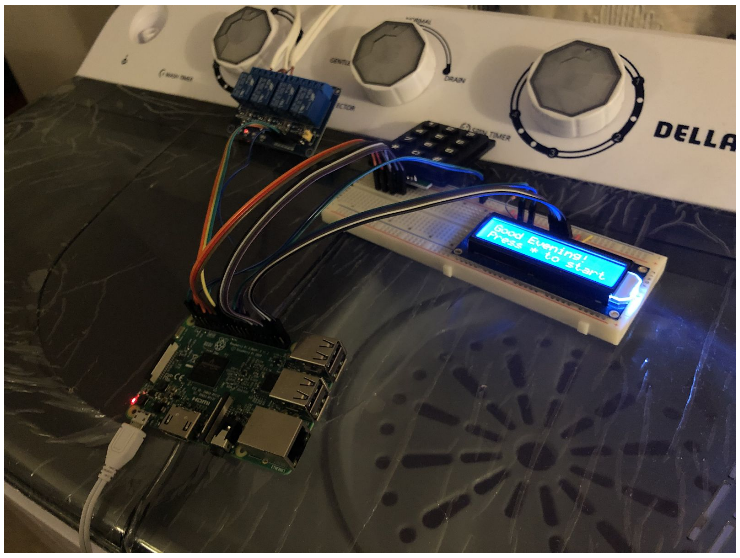


Figure 39: Completed Locking Mechanism

6. Testing Plan, Process and Results

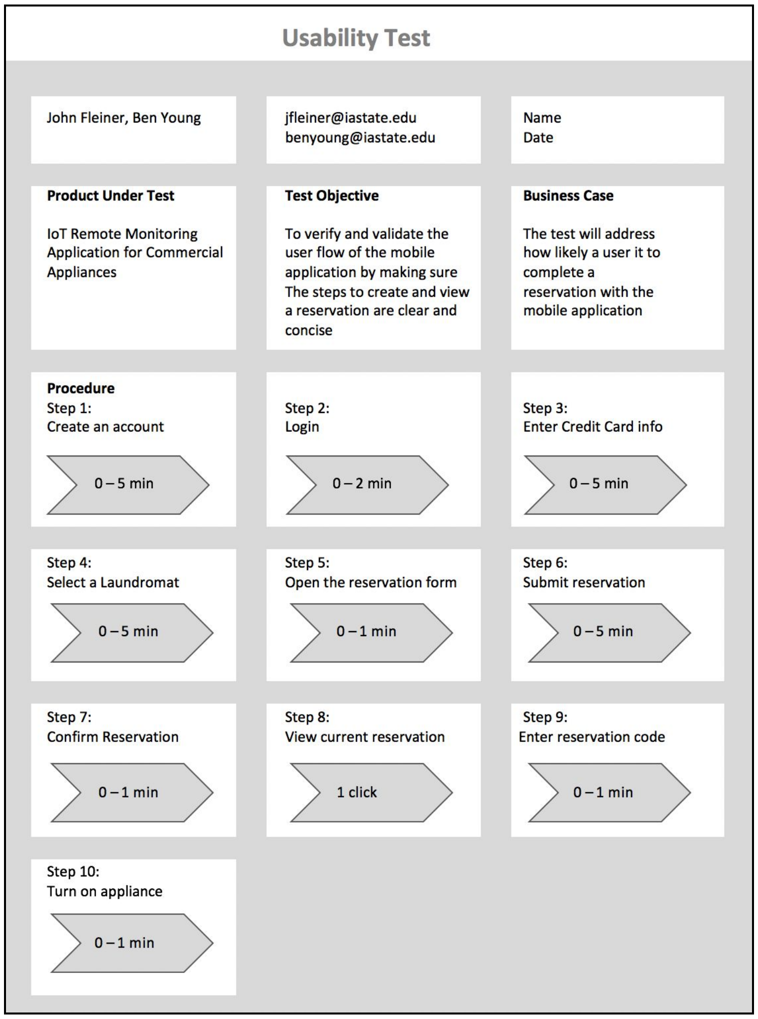
6.1 Mobile Application Usability Testing and Validation

6.1.1 Usability Testing

Usability testing is a common technique used to test how well a task can be performed by real customers. To implement a usability test, two items are required: A usability test document and a user.

Usability Test Document

A usability test document is a document that identifies the steps or actions that are to be completed by a user. Each step is marked with criteria that can either be passed or failed. For example, a step must be completed within 1 minute or a step must be completed in under 2 clicks. Our team put together the following usability test document:



Usability Test

John Fleiner, Ben Young	jfleiner@iastate.edu benyoung@iastate.edu	Name Date
Product Under Test IoT Remote Monitoring Application for Commercial Appliances	**Test Objective** To verify and validate the user flow of the mobile application by making sure The steps to create and view a reservation are clear and concise	**Business Case** The test will address how likely a user it to complete a reservation with the mobile application
Procedure Step 1: Create an account 0 – 5 min	Step 2: Login 0 – 2 min	Step 3: Enter Credit Card info 0 – 5 min
Step 4: Select a Laundromat 0 – 5 min	Step 5: Open the reservation form 0 – 1 min	Step 6: Submit reservation 0 – 5 min
Step 7: Confirm Reservation 0 – 1 min	Step 8: View current reservation 1 click	Step 9: Enter reservation code 0 – 1 min
Step 10: Turn on appliance 0 – 1 min		

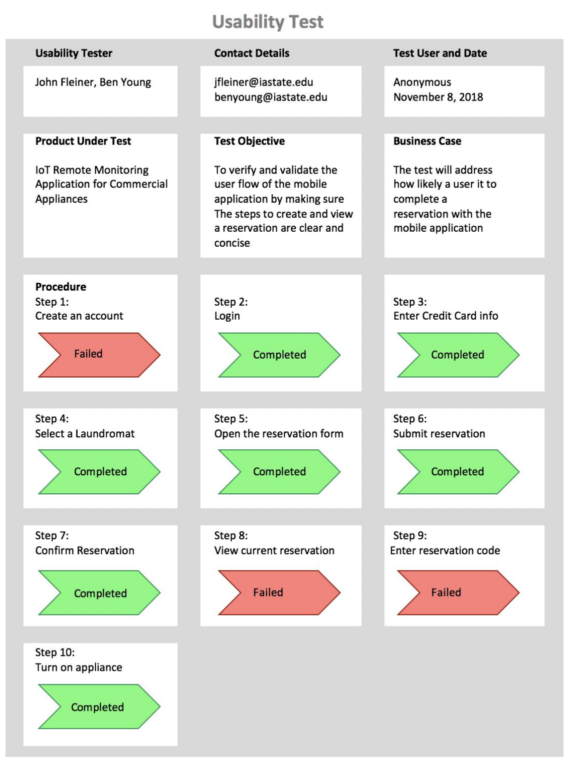
Figure 40: Usability Test Template

Usability Test Template

Our usability test template identifies 10 steps that must be completed by a real user. The 10 steps reflect actions that are to be performed on both the android and iOS mobile applications. Each step identifies pass fail criteria. It is expected that *step 1* be completed within 5 minutes. Creating an account involves registering a first name, last name, email, and password. The Login step should be completed within 2 minutes. We gave extra time to account for autocorrect and grammar issues when using a mobile keyboard. It is expected that the credit card information be added within 5 minutes. There are 5 inputs required and we provided a fake card as shown below. Selecting a laundromat should be completed within 5 minutes, including time for zooming in on google maps and selecting a laundromat marker. Step 6 shall take less than 5 minutes to complete which includes submitting a reservation date, start time, end time, and number of appliances. Step 7 should take less than a minute as it requires the user to exit out of the confirmation popup. The 8 shall take less than one click to view. If the user doesn't click the correct icon, then our user interface needs to be updated. Step 9 should take less than 1 minute as the user only needs to enter the reservation code into the keypad. The final step should take less than 1 minute since the only requirement is to turn on the appliance.

Usability Testers

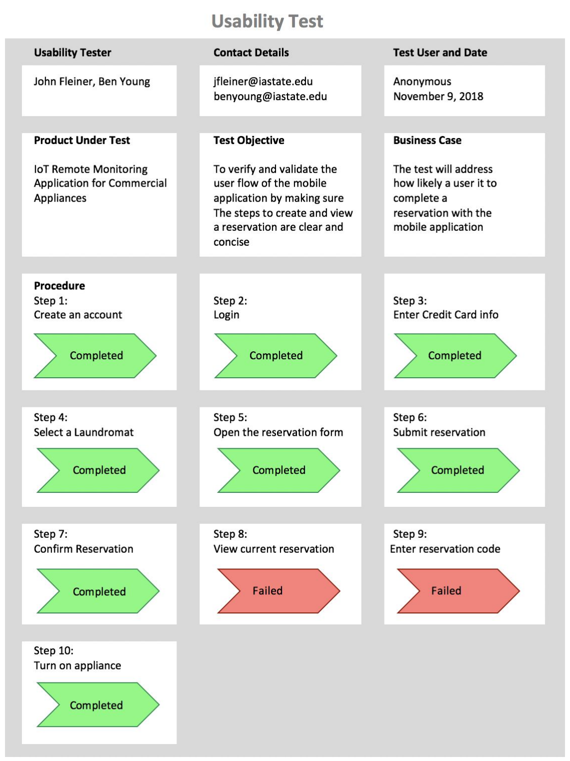
Over the course of two weeks, our team tested several potential users including friends and family. The results from the usability tests can be seen as follows.



Usability Test

Usability Tester	Contact Details	Test User and Date
John Fleiner, Ben Young	jfleiner@iastate.edu benyoung@iastate.edu	Anonymous November 8, 2018
Product Under Test IoT Remote Monitoring Application for Commercial Appliances	**Test Objective** To verify and validate the user flow of the mobile application by making sure The steps to create and view a reservation are clear and concise	**Business Case** The test will address how likely a user it to complete a reservation with the mobile application
Procedure Step 1: Create an account — Failed	Step 2: Login — Completed	Step 3: Enter Credit Card info — Completed
Step 4: Select a Laundromat — Completed	Step 5: Open the reservation form — Completed	Step 6: Submit reservation — Completed
Step 7: Confirm Reservation — Completed	Step 8: View current reservation — Failed	Step 9: Enter reservation code — Failed
Step 10: Turn on appliance — Completed		

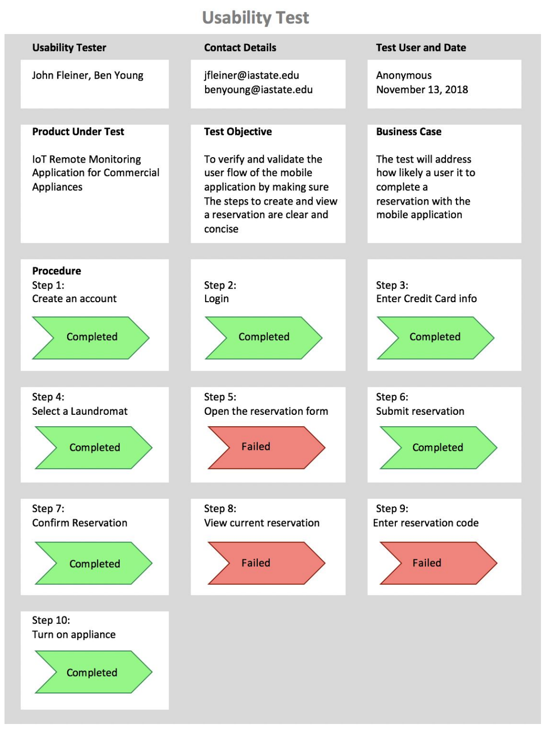
Figure 41: Usability Test 1



Usability Test

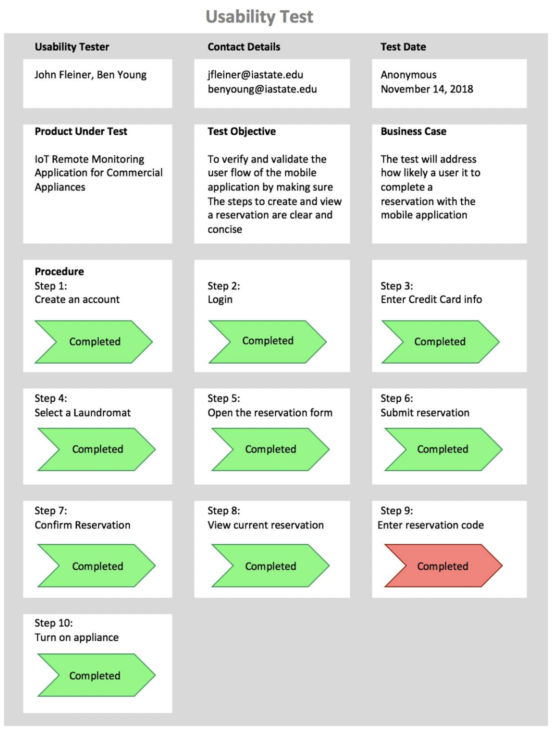
Usability Tester	Contact Details	Test User and Date
John Fleiner, Ben Young	jfleiner@iastate.edu benyoung@iastate.edu	Anonymous November 9, 2018
Product Under Test IoT Remote Monitoring Application for Commercial Appliances	**Test Objective** To verify and validate the user flow of the mobile application by making sure The steps to create and view a reservation are clear and concise	**Business Case** The test will address how likely a user it to complete a reservation with the mobile application
Procedure Step 1: Create an account — Completed	Step 2: Login — Completed	Step 3: Enter Credit Card info — Completed
Step 4: Select a Laundromat — Completed	Step 5: Open the reservation form — Completed	Step 6: Submit reservation — Completed
Step 7: Confirm Reservation — Completed	Step 8: View current reservation — Failed	Step 9: Enter reservation code — Failed
Step 10: Turn on appliance — Completed		

Figure 42: Usability Test 2



Usability Test

Usability Tester John Fleiner, Ben Young	**Contact Details** jfleiner@iastate.edu benyoung@iastate.edu	**Test User and Date** Anonymous November 13, 2018
Product Under Test IoT Remote Monitoring Application for Commercial Appliances	**Test Objective** To verify and validate the user flow of the mobile application by making sure The steps to create and view a reservation are clear and concise	**Business Case** The test will address how likely a user it to complete a reservation with the mobile application
Procedure Step 1: Create an account — Completed	Step 2: Login — Completed	Step 3: Enter Credit Card info — Completed
Step 4: Select a Laundromat — Completed	Step 5: Open the reservation form — Failed	Step 6: Submit reservation — Completed
Step 7: Confirm Reservation — Completed	Step 8: View current reservation — Failed	Step 9: Enter reservation code — Failed
Step 10: Turn on appliance — Completed		



Usability Test

Usability Tester John Fleiner, Ben Young	**Contact Details** jfleiner@iastate.edu benyoung@iastate.edu	**Test Date** Anonymous November 14, 2018
Product Under Test IoT Remote Monitoring Application for Commercial Appliances	**Test Objective** To verify and validate the user flow of the mobile application by making sure The steps to create and view a reservation are clear and concise	**Business Case** The test will address how likely a user it to complete a reservation with the mobile application
Procedure Step 1: Create an account — Completed	Step 2: Login — Completed	Step 3: Enter Credit Card info — Completed
Step 4: Select a Laundromat — Completed	Step 5: Open the reservation form — Completed	Step 6: Submit reservation — Completed
Step 7: Confirm Reservation — Completed	Step 8: View current reservation — Completed	Step 9: Enter reservation code — Completed
Step 10: Turn on appliance — Completed		

Figure 43: Usability Test 3

Figure 44: Usability Test 4

6.1.2 Usability Testing Results

Interview

After each test came to end, our team conducted a 20 minute interview to identify why different steps failed or took longer than expected. The results from those interviews are explained below.

Test 1: The first test resulted in 3 failed steps. For step 1, the android application didn't have a working button that would navigate the user back to the login screen which through the user off. Step 8 asks the user to click the current reservation button on the bottom navigation bar of the mobile application. The previous and current reservation button icons were a bit confusing and caused the wrong reservation list to load. When the user enters a reservation code, they must enter * (star) # (digit) # (digit) # (digit) # (digit) * (star) into the keypad. An example is: *1234*. The user was unaware that they needed to enter a * (star).

Test 2: After the first test, our team fixed the login issue as that was a programming error rather than a usability issue. The second test resulted in 2 failed steps. Similar to the first user, step 8 and step 9 failed due to a lack of clarity with both the reservation icons and required keypad input.

Test 3: The third test resulted in 3 failed steps. Unlike the previous users, our third tester failed step 5. Step 5 asks that the user click the laundromat icon on the bottom slider drawer after selecting a laundromat location. It wasn't clear to the user that the laundromat icon needed to be clicked to open the form.

Test 4: The fourth test resulted in 1 failed step. Similar to the first two tests, the user was not aware that the keypad input needed a * (star) before and after the reservation code numbers.

After each interview was conducted, a correct demonstration of all steps were provided. Once the correct demonstration was given, each tester was asked to re-evaluate the steps that were done incorrectly. Positive feedback was given once the steps were made clear.

Validation

Validation is the process of evaluating a product with tests and use cases to determine if the product meets stakeholder requirements. In our case, mobile validation is the process of evaluating the android and iOS mobile applications to determine if the product meets our clients requirements. The results from the usability test indicate that there are in fact several areas of misunderstanding or confusion while using the mobile application. The primary areas for concern include: viewing the current and previous reservation information and entering the numpad. To account for the areas of concern, new icons were created to differentiate between history and current reservations. Due to the current prototype implementation, changes will not be made to the numpad. However, future implementations and expansions shall include a keypad with an "enter" button to eliminate confusion with special characters.

6.2 Mobile Application Unit Testing and Validation

6.2.1 Unit Testing

To test the accuracy of our mobile applications, android and iOS unit tests were written to cover login, registration, map display, and reservations. The unit tests and results are shown in the table below:

Unit Test	Description	Expected	Actual
is_login_admin()	Given a valid administrator account, verify that the API returns a valid administrator ID	True	True
is_login_customer()	Given a valid customer account, verify that the API returns a valid customer ID	True	True

is_login_invalid()	Given an invalid account, verify that the API returns an error code of -1	-1	Null
is_registration_valid()	Given a valid registration entry form, verify that the API creates the user account by returning a customer JSON	{“firstname”: “michael”, “lastname”: “jones”, “email”: mjones@gmail.com”, password: “password”}	{“firstname”: “michael”, “lastname”: “jones”, “email”: mjones@gmail.com”, password: “password”}
is_registration_valid_email()	Given an invalid email address, verify that the application displays an error	False	True
is_registration_valid_firstname	Given an invalid first name with alphanumerics, verify that the application displays an error	False	True
is_registration_valid_lastname	Given an invalid last name with alphanumerics, verify that the application displays an error	False	True
is_registration_valid_empty_firstname()	Given an empty first name, verify that the application displays an error	False	False
is_registration_valid_empty_lastname()	Given an empty last name, verify that the application displays an error	False	False
is_registration_valid_empty_email()	Given an empty email, verify that the application displays an error	False	False
is_registration_valid_empty_password()	Given an empty password, verify that the application displays an error	False	False

is_registration_valid_password_length	Given a password of length 6 (or higher), verify that the application approves	True	True
is_registration_invalid_password_length	Given a password of length 1, verify that the application displays an error	False	True
is_card_valid	Given a valid test Stripe credit card, verify that the card validation approves	True	True
is_card_invalid	Given an invalid credit card, verify that the card validation disapproves	True	True
is_laundromats_received()	Calling the laundromat API should return a json array of length 3	3	3
is_atms_received()	Calling the atm API should return a json array of length 9	9	9
is_banks_received()	Calling the bank API should return a json array of length 6	6	6
is__reservation_price_valid()	Given a valid reservation of 2 machines for 1 hour, verify that the price is $4	4	4
is_reservation_calendar_valid()	Given a calendar date in the past, verify that the application displays an error	False	True
is_reservation_time_valid()	Given a reservation end time that is before the reservation start time, verify that the application displays an error	False	True
is_reservation_num_appliances_valid()	Given a request for 1000 machines, verify that the applications displays a warning	False	False

	and prevents a reservation from being created		
is_reservation_valid_card()	Given a valid reservation, but an invalid credit card, verify that the applications displays a warning and prevents a reservation from being created	False	False
is_reservation_valid()	Given a valid reservation and a valid card, verify that a reservation is made and that a charge has been created on the stripe dashboard	True	True
is_many_historical_reservations_received()	Calling the history API on a user with >= 1 previous reservations should return a json array greater than 0	True	True
is_no_historical_reservations_received()	Calling the history API on a user with no previous reservations should return a json array of 0	True	True
is_many_current_reservations_received()	Calling the future API on a user with >1 current reservations should return a json array greater than 0	True	True
is_no_current_reservations_received()	Calling the future API on a user with no current reservations should return a json array of 0	True	True

Table 8: Mobile Unit Test Results

6.2.2 Unit Testing Results

Out of the 28 unit tests that were run, 7 unit tests failed and 21 unit tests passed. The success rate of our unit tests was 75% and the failure rate of our unit tests was 25%.

Validation

The results from the unit test indicate that the core functionality of the application works as expected: A user is able to create an account, sign in, add a credit card, and make a reservation. The unit tests do however identify several boundary exceptions that must be handled before a production release can be launched. Several exceptions occurred with the login and registration portion of the application, primarily with invalid user input. The email address is a core requirement for our mobile application as it is the primary contact information for the user and is where billing information will be emailed to. Currently, the email address isn't being validated, despite the implementation of an email-only input field. Likewise, the first name and last name input fields aren't being validated for real names. When running the mobile application on a real device, a keyboard appears with only valid characters. When running the mobile application on an emulator or test device, additional characters may be entered. Safety checks should be implemented to handle these errors. The second set of errors involve the reservation fields. Since the mobile application allows for only one date to be entered, there is no date rollover if the end time is earlier than the start time. This means that if a user enters a start time at 5 pm and an end time at 3 pm on January 1st, the reservation will be made from 5 pm to 3 pm on January which isn't possible. The correct reservation should either be from 5 pm January 1st to 3 pm January 2nd or display an invalid input message. Currently, it is also possible to select a date that has happened in the past. A user should not be allowed to select a date that has already occurred when making a reservation. Once these changes have been made, the application's validation will be at 100% based on our test criteria.

6.3 Backend Server Testing and Validation

The test plan for our server was tightly coupled to the design due to our foresight in ensuring a modular structure. We partitioned our server into three components: controller, service, and data access layers. We primarily placed our logic and heavy lifting in the service layer, allowing our controllers to simply call on the service layer to access, manipulate, and return the desired data. Thus, our testing plan revolved around two main components: unit testing each method in the service layer as well as each customer query/method in the data access layer. We only had to test custom queries in the data-access layer as we integrated Java's JPA API to handle basic data accesses, such as finding records by ID in the database. This allowed our data-access layer to be lightweight. Due to time constraints, the majority of our server testing followed 'happy path' ideologies to ensure core functionality before handing off the product to our clients.

6.3.1 Service Layer

As previously mentioned, most of the heavy lifting was placed into the service layer by design. Thus, this was the focal point of our backend testing and took up the majority of time allotted to testing for the backend team. Each layer of our service was partitioned into components corresponding directly to the tables in the database.

Due to the design of our server, we decided to unit test each method in each class using mockito. Mockito allowed us to isolate the SUT (system under test) by eliminating each outside dependency and test only the functionality within each method. We mocked the data access layer in each method and, using mockito, knew exactly what would be returned by the data access layer in each method call. This is equivalent to hardcoding the results of the queries being returned to the calling entity, in this case the methods in the service layer. By doing this, we ensured we were simply testing functionality in the service methods and not relying on outside dependencies.

To automate our testing and simplify continuous integration and testing, we used the JUnit testing framework. Each method in our service layer classes corresponds to at least -- and in some cases, multiple -- JUnit test cases. Implementing JUnit allowed us to write test cases with expected values. Due to the automated running nature of the framework, we could make changes to the methods at the request of our frontend team and run the JUnit tests for the specific class at the click of a button and confirm the integrity of the methods instantly.

6.3.2 Data-Access Layer

To test our data-access layer, we utilized Spring Boot’s TestEntityManager as well as JUnit to automate the running of our unit tests. The methods in our data-access layer are essentially endpoints for custom queries in our database. As previously mentioned, JPA provides you with basic queries, such as searching for records by their ID in the database. However, many custom queries were essential to provide needed functionality in the service layer. Thus, it was essential to put these queries under test independent of the outer dependencies.

Rather than make use of our external database or embedded database, we used a TestEntityManager to simulate the use of an external database. The external database was infeasible for unit testing as the client eventually will not have access to that, making the tests irrelevant. We could have used our embedded H2 database, but any changes the data could have broken our tests. By using the TestEntityManager, you are able to define sets of data that are relevant to the queries under test and persist them, giving you full control of the data being accessed by your autowired repo layer. Thus, using java code, you can create and insert data, and then call your repo methods which will operate with the data persisted by your code in that class. This made it rather trivial to define expected results, call the method under test, and test for equality using JUnit.

6.4 Hardware Component Testing and Validation

The hardware testing procedure was focused on two aspects in terms of the hardware implementation: 1. Onboard scripts testing and 2. Circuit functionality testing.

6.4.1 Onboard Scripts Testing

As previously mentioned in the implementation sections, a Raspberry Pi 3 Model B is used for the monitoring feature to the washing machine. In order to get a robust and reliable working prototype, we have performed several tests to the embedded scripts which are running on the Raspberry Pi.

The GPIO and Networking scripts are written with Python 3.6, and the “calling-methods” along with configuration setups are written with Shell scripts on the Unix-based operating system, Raspbian. The testing procedure was separated into two phases where the first comes with Python scripts testing and second as Shell scripts testing.

Python Scripts Testing:

There are a few of testing frameworks which we could adapt into our testing implementation. The one we used was simply the “unittest” which comes along with the Python application. Recall that the GPIO and Networking scripts was written in Python, we performed unit tests to these two “units” of the script.

The GPIO functionalities were tested by firstly creating mock Python modules to perform simple LED on/off testing as the fundamental setup of our test. Then we follow the similar strategy (by creating mock modules) to test the “relay” module and LCD/Keypad module. The mock modules would make sure that each module (relay, LCD, Keypad) has the functionality and reliability as we intended. After several rounds of testing with different parameters, the result showed that the prototype is able to perform as what we defined in the implementation section in terms of GPIO inputs/outputs.

The Networking scripts were tested by associating with the test-panel provided by Amazon Web Services. Since the MQTT queue implementation was the core part of the Networking scripts, AWS provides a test-panel on the IoT section which we could take advantage of in our testing phase. The test-panel allows us to subscribe to the same “topic” (channel in MQTT communication) via their web interface, in addition to that, we can manually publish/receive messages to/from the topic. Thus, we have tested our Networking scripts by creating “testing messages” through the test-panel. The result showed that our Networking scripts are working smoothly as we intended.

Shell Scripts Testing:

There are also some unit testing frameworks out there which we could taking into our Shell script testing steps. However, since the only usage of Shell scripts in our prototype was to provide an easier start-up of the Python scripts and “one-time” configurations, it is enough to test the Shell scripts via several test runs to make sure that the scripts are giving successful calls to our Python scripts. The results showed that the Shell scripts can give us 100% successful calls to the Python scripts.

6.4.2 Circuit Functionality Testing

The circuit functionality testing is based on the breadboard circuits. Since the prototype is still in the implementation phase, informal tests were performed instead of full product usability testing. As a recall from the previous introductions, the circuit consist of three major modules as well: relay component, 16*2 LCD screen, and a 3*4 matrix keypad. The tests were separated into three steps respectively to the modules.

Relay Component

This is the key part of the prototype since it controls the power source of the washing machine by toggling a 110V household power via magnetic circuit which is under control of a 5V DC voltage. The Raspberry Pi is in its order to output the 0/5V DC voltage to the relay as above. The tests are focused on getting a consistent result from the relay module so that we get the fully control over the power source of the washing machine. To accomplish this, a function generator and an oscilloscope were used for the test. For the reason of safety during the testing phase, the function generator generates a relatively low AC power (~10V sinusoidal waveform) which feeds into the relay module as mentioned in previous sections. The oscilloscope probes were placed at the other end of the relay to show the results. The Raspberry Pi outputs 5V DC toggle voltages to the relay module via its GPIO ports. As the Raspberry Pi toggles the relay module, oscilloscope shall present us the on/off power of the circuit. As a proof of correctness, we performed such test under different voltages and time intervals in several rounds. The results showed that the circuit is robust and reliable.

LCD Screen

The testing phase of this part was to ensure that the LCD screen has the ability to present the intended messages. We simply wrote up a Python module to test different messages and check the output from the screen. The tests also included some “extreme” cases such as in a cold/hot environment (roughly between -10 to 40 Celsius degrees). In addition to these, we tested the contrast adjusting under different brightness. The result showed that, the LCD would work as intended when the temperature maintains above 0 degree as bottom line. We were also able to adjust the contrast of the screen to fit an indoor environment.

Matrix Keypad

The keypad testing was all about the input correctness testing which includes single key input correctness and multiple key input correctness. To test this, a testing Python module was created to read keypad inputs and output the results in the console. As the result showed, the keypad was able to give precise input readings and distinguish consecutive inputs as intended.

6.5 Requirements Verification and Validation

Functional Requirement	Verification	Validation

The mobile application shall display local laundromats, ATMs, and banks within Ames ,IA	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_laundromats_received()
		
		is_atms_received()
		
		is_banks_received()

The mobile application shall login the customer upon a successful login authorization.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_login_customer()
		
		is_login_invalid()

The mobile application shall inform the user of a credit card charge upon a successful transaction authorization.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_reservation_valid()
		
		is_card_valid()
		
		is_card_invalid()
		
		is_reservation_valid_card()

When the customer clicks the login button, the mobile application shall send login data to the server	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_login_customer()
		
		is_login_invalid()

When the customer clicks the reservation button, the mobile application shall send registration data to the server	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_reservation_price_valid()
		
		Is_reservation_calendar
		
		_valid()
		
		is_reservation_time_valid()
		
		Is_reservation_num_washers
		
		_valid()

		is_reservation_valid_card()
		
		is_reservation_valid()

When the customer clicks the credit card button, the mobile application shall send credit card information to the server	Usability Test
	
	Code Inspection
		Unit Tests:
		
		Is_card_valid()
		
		is_card_invalid()

When the customer clicks the reserve button, the mobile application shall create a new reservation.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_reservation_price_valid()
		
		Is_reservation_calendar
		
		_valid()
		
		is_reservation_time_valid()
		
		Is_reservation_num_washers
		
		_valid()
		
		is_reservation_valid_card()
		
		is_reservation_valid()

If the login credentials are invalid, then the mobile application shall invalidate the login and prevent them from navigation onward.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_login_customer()
		
		is_login_invalid()

If the registration credentials are invalid, then the customer shall not be registered as a new user.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_registration_valid()
		
		is_registration_valid_email()
		
		Is_registration_valid
		
		_firstname()
		
		is_registration_valid
		
		_lastname()
		
		is_registration_valid_empty

		_email()
		
		is_registration_valid_empty_password()
		
		is_registration_valid
		
		_password_length()
		
		is_registration_invalid_password_length()

If the number of washers or number of dryers exceeds the available amount, then a warning shall be displayed.	Usability Test Code Inspection	Unit Tests: is_reservation_num_appliances_valid()

If the customer's credit card is declined, then a warning shall be displayed.	Usability Test
	
	Code Inspection
		Unit Tests:
		
		is_reservation_valid_card()
		
		is_reservation_valid()

When the user enters a reservation code into the locking mechanism, the locking mechanism shall unlock the washing machine	Usability Test Code Inspection	onboard_python_keypad_test()
When a new microcontroller has been purchased, the IoT web service shall register a new device.	Usability Test Code Inspection	N/A

When the power command is received, then the power to the washing machine shall be enabled.	Usability Test
	
	Code Inspection
		onboard_python_power_on_test()
		
		onboard_python_power_off_test()

Table 9: Requirements Verification and Validation

7. Security

Several security concerns were considered during the building of this prototype, namely the need to be able to expand from normal HTTP to more secure HTTPS when communicating between the phone applications and the web application. By using Spring Boot and Spring Security, HTTPS is supported with several configuration changes. Since Spring Boot is also built and served on a Tomcat server, that server can also be further configured to provide deeper levels of encryption, specifically enabling specific versions of encryption protocols and adding the potential to have a specified keystore for encrypting and decrypting communications to enable HTTPS. By specifically enabling different encryption protocols, the client will have more control over what is actually used when communicating, and unsecure protocols such as SSLv3 or even TLSv1.0 to stay compliant with the PCI Data Security Standard.

The most prominent implementation of security in the prototype application is the inclusion of Spring Security inside of the web application. By including Spring Security, the application is configurable such that each and every method may have custom authentication on it if necessary, and the application as a whole can support a customized authentication using self-defined user roles and a managed database of custom users. While the implementation of Spring Security was not finished for the prototype in regards to a customized implementation of user authentication, it was added and initially set up, so all that needs to be done to complete the authentication is to add the customization and define user roles. With this customization, the web server prototype will be able to have different functionality enabled for administrative vs normal users, locking down the web application's REST endpoints both from a normal user trying to access administrative controls and from a non-registered user trying to access data from the endpoints.

While the main security increases between phone applications and the server exist from implementing customized Spring Security authentication and sending REST requests over HTTPS instead of HTTP, because credit card information is being sent inside some of the REST requests it may be a good idea to hash and encrypt this personal information. There are several ways of doing this, however as long as the phone applications and the web application server both agree on the method of encryption or hashing then any form of further encryption and obfuscation of credit card numbers may be utilized. This would be especially useful when storing credit cards in the database, as databases are often very important targets for theft and hacking and can yield concentrated amounts of personally identifiable data.

Since Stripe is used for appliance payments via credit card, part of the security Stripe enforces is that mobile devices may not directly make payments to Stripe. Therefore, payments are done by the application server, negating the need to send credit card information over the network to a mobile device to necessitate a payment. This not only consolidates the times credit card information is sent over the network to only when the credit card is updated, but also compartmentalizes the information such that if a user's mobile application or phone becomes compromised, then they may still use the device without credit card information being leaked as easily, slightly lessening the risk of credit card fraud.

The final security concern covered by the prototype is how the web application server communicates with the specific washers and dryers. The prototype implements communication via MQTT queues hosted in AWS, with each arduino device having its own MQTT queue and the web application listening on all MQTT queues for authentication information. By using AWS' implementation of MQTT and AWS IoT connections, all MQTT traffic is secured by the TSLv1.2 encryption protocol, making it currently the most secure part of the prototype.

8. Project Management

8.1 Roles and Responsibilities

Name	Role(s)	Responsibilities
John Fleiner	1. Team Lead 2. Android Mobile Developer 3. Scrum Master 4. Requirements Specialist	● User Interface Design ● Android Implementation and Development ● Android Usability Testing ● Android Unit Testing ● Integration with server team ● Plan Agile Sprints and Team Meetings ● Write Project Requirements Specifications
Ben Young	1. iOS Mobile Developer	● User Interface Design ● iOS Implementation and Development ● iOS Usability Testing ● iOS Unit Testing
Thomas Stackhouse	1. Backend Lead 2. AWS IoT Backend Developer	● Spring Boot Gradle Build ● Spring Boot Security ● AWS IoT MQTT Implementation

Casey Gehling	1. API Lead 2. Spring Boot Backend Developer 3. Meeting Scribe 4. Mobile Server Integration Lead	● Write APIs + queries for mobile team ● Server Layer Testing ● Record documentation from advisor, client, team, and scrum meetings
Hongyi Bian	1. Hardware Lead	● Raspberry Pi 3 Microcontroller Implementation ● LCD Keypad Implementation ● 4-Channel Relay Module Implementation ● AWS IoT Integration ● Hardware Circuit Design
Yuanbo Zheng	1. Hardware Engineer	● Raspberry Pi 3 Microcontroller Implementation ● LCD Keypad ● Hardware Onboard Control Testing ● Hardware Circuit Testing ● Hardware Security

Table 10: Roles and Responsibilities

Mobile Development

John Fleiner and Ben Young led development efforts for the native android and iOS mobile applications. Fleiner served as the team lead, handing communication, planning, and integration efforts for each of our sprints identified in section 2. He was responsible for ensuring that all team deadlines were met and that team contributions were evenly distributed among the team. Fleiner performed development work for the android version of the mobile application, including user interface design and implementation. Ben Young was head of iOS mobile development efforts. Young's efforts include the mobile user interface design and implementation.

Backend Development

Casey Gehling and Thomas Stackhouse led development efforts for the Backend implementation of our prototype. Thomas Stackhouse was responsible for setting up our Spring Boot Web Server and AWS E2C instance. Stackhouse implemented automated deployment to AWS and worked towards implementing Spring security. Casey Gehling led integration efforts between the mobile application and Spring Boot Web Server. Gehling was responsible for creating our teams database schema and for writing all APIs needed by the mobile team for accessing information security from our database.

Hardware Development

Hongyi Bian and Yuanbo Zheng worked together to implement the locking mechanism for our team's washing machine. Hongyi Bian led hardware development efforts for the hardware circuit design that was used to attach our Raspberry Pi 3 microcontroller, LCD screen, keypad, 4-channel relay module, and washing machine. Yuanbo Zheng led hardware testing efforts for both the onboard script testing and circuit functionality testing. Zheng also helped with the hardware circuit design and implementation.

8.2 Projected Timeline

During the first semester of Senior Design, our team attempted to plan our project to accurately convey work requirements over the course of two semesters. Our team initially proposed a timeline that is shown below:

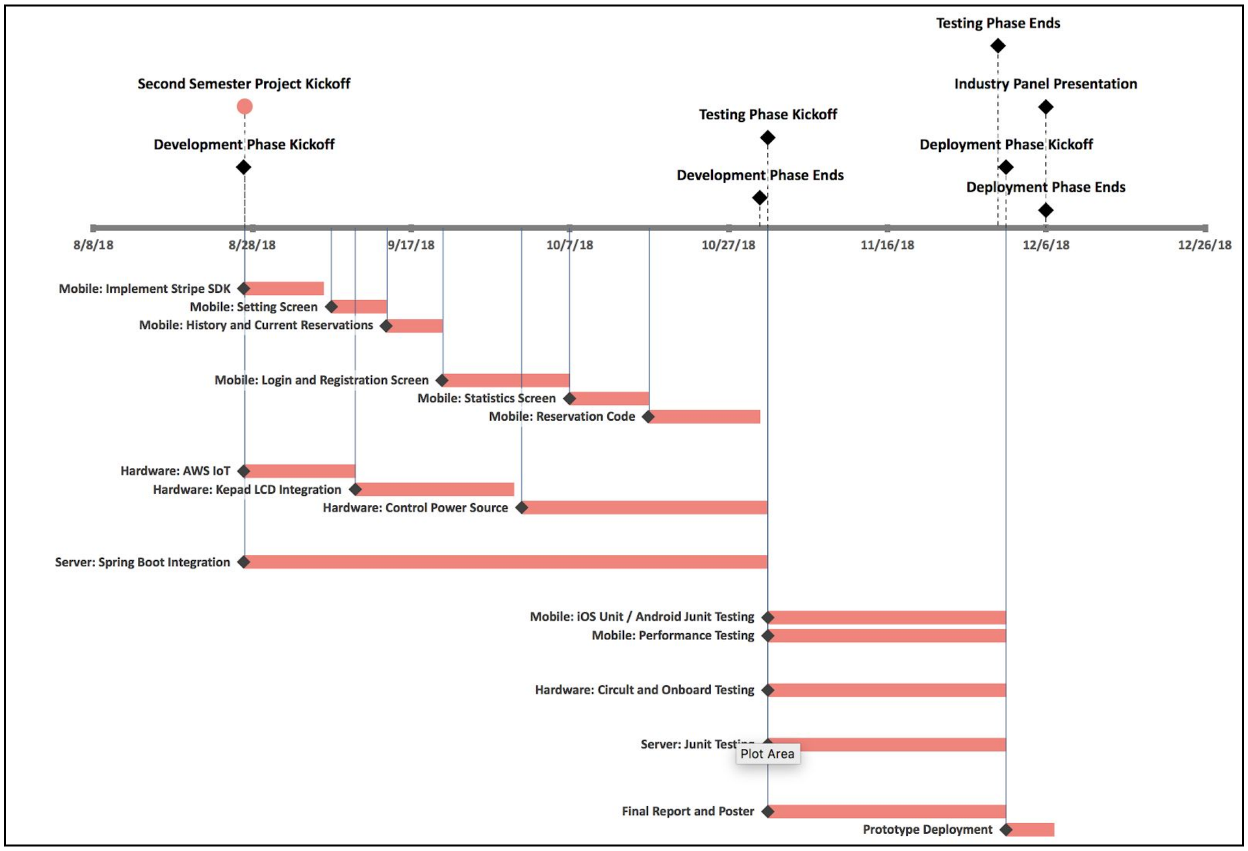


Figure 45: Projected Timeline

Development Phase

We initially planned on having 3 major second semester phases: Development phase, Testing phase, and Deployment phase. The testing phases requires development efforts from all three teams: mobile, hardware, and backend. When the timeline was created, our team didn't plan on redesigning the mobile application's user interface. Therefore, it was expected that additional time could be allocated to the development of each screen. The timeline planned for approximately 1 week per screen. Like, at the time of the proposal, the specific details for the locking mechanism hadn't been hashed out. We knew that keypad and LCD integration and power source control was needed, but the information related to scripting and soldering was absent. The spring boot integration left out technical details as API documentation hadn't been written by our team to identify what data must be sent and received by the mobile application.

Testing Phase

Our team wanted to have development completed by the end of October so that a month could be given to testing all three components of our project. 30 days were allocated to mobile unit and performance testing, hardware circuit and onboard testing, and server unit testing.

Issues with the Proposed Timeline

The development phase described above has many inconsistencies and unknown variables that affected how well we would be able to execute everything. First, our team opted to redesign the user interface so that we could present a professional looking application. Prior to this decision, the application didn't meet our non-functional look and feel requirement. Due to the redesign, our timeline had to be re-thought out. Furthermore, our team opted to utilize the Agile development process for second semester to mitigate risks and issues associated with communication and management from first semester. By implementing Agile, we were able to identify with much better accuracy and consistency what tasks need to be completed in order to meet our incremental planning goals. The mobile application redesign allowed our team to think about the APIs and database schema needed to make our product work. These decisions gave us the ability to create a brand new timeline to help us deliver a final working prototype with professional-like qualities.

8.3 Actual Timeline

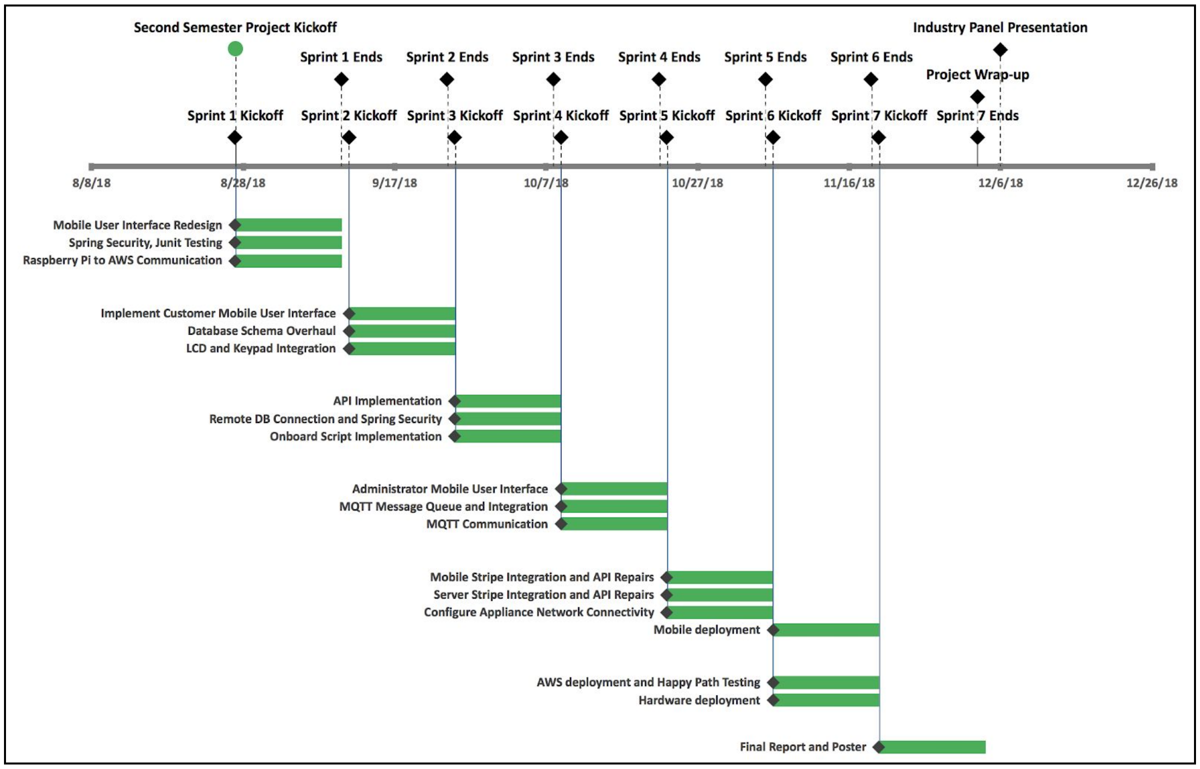


Figure 46: Actual Timeline

Depicted above, is our complete Agile-based Gantt Chart for the Senior Design 492. Our team planned for two week sprints covering 3 increments. Increment 1 was divided into sprint 1, sprint 2, and sprint 3 where the primary goal was to complete all customer related requirements . Increment 2 was divided into sprint 4 and sprint 5 where the primary goals were to implement the administrator related requirements and to integrate the Stripe payment transaction platform for both mobile and server. Increment 3 was divided into sprint 6 and sprint 7 where the primary goal was testing and deployment.

8.4 Lessons Learned

Many lessons were learned throughout senior design these last two semesters, but probably the largest was to pay attention to the EC2 container base image when provisioning an EC2 instance on Amazon Web Services. Since the client had set up the AWS account, and it was on a free trial for the first semester, our team's EC2 instance was covered in the free tier. However, during the second semester the free tier expired, and it was discovered that the Red Hat Enterprise instance that we had been using was very expensive, up to the price of $50 a month for a micro-sized instance. When the client caught this error and reported it to our team, we did a quick price calculation and switched to using an Amazon Linux instance instead, saving over $40 a month in instance costs. In the future, the team's acting solution architect that was working with AWS made sure to check estimated costs with a price calculation tool when working with different AWS functionality so that the cost to the clients could be minimized.

Another lesson learned is that when learning new skills, such as soldering, our team should plan for accidental damages as an anticipated risk. Neither of the hardware team members had soldered before, and during the learning process the first keypad was accidentally damaged such that the middle row of buttons did not work. Because of this accident, the client ordered two replacement keypads so that if it happened again there would be a backup on hand. If this had been thought of as a risk ahead of time, our team would have been able to be more productive during the two weeks that we waited for the replacement keypad to arrive.

A third thing that our team learned while doing this project was the importance of defining all portions of the overall architectural structure before implementation. By the second or third week into last semester, the entire architectural structure for the project and how every portion was going to communicate with each other was defined and known to work, with the exception of connecting the hardware to the web application server. Originally it was thought that the use of lambda expressions would work well such that the hardware would connect with AWS IoT and would trigger lambda expressions to send REST requests to the application server to check whether the washer or dryer would turn on or not. During the second semester, when implementing the hardware portion it was discovered that using lambda was not going to work the way our team wanted. Therefore, we instead implemented MQTT queue listeners on the web application server so that the web application and the hardware devices could talk to each other with two-way communication. However several weeks were spent implementing this after it was found that this was the best way to communicate from the web application to the hardware, and this threw off the schedule of other things that were planned on being done, namely implementing customized Spring Security.

On the mobile application side of development, another lesson was learned. Constant user interface redesigns due to continual improvement and feedback heavily increased development time, especially since both the Android and iOS applications were developed separately with different user interfaces but similar screen flows. This was done intentionally so that a better design overall was able to be achieved, however because each application was developed separately and they were later combined, a lot of time was spent re-working screen layouts and user interfaces. It would have saved a lot more time if the full user interface was worked out before the screens were fully implemented, and then the mobile developers could have worked together to implement the design in both platforms and then worked from that base to develop the final screens. If this had been implemented, potentially 50+ hours of work may have been saved from our mobile developers.

One defect that was found when testing the full system before the final client demo was that the web server used the local time of the AWS instance when it was calculating if a reservation was scheduled to turn on the washing machine or dryer. While this is not a problem when testing locally on a laptop, or when run on an AWS EC2 instance where the time zone on the machine has been set to the same time zone as the locations it covers, this is a problem if a single instance or a single ECS cluster is servicing multiple different time zones. While this is a relatively simple design defect to fix, it is a lesson learned when designing a high availability system that has the potential to span multiple time zones.

The final lesson that was learned throughout this senior design process was that during development remote schema management is very important. In this project any data that was being used was test data and could easily be wiped out if necessary to update the remote schema. However, as the schema evolved on the automatically spun up local instance that was being used for testing by the backend development team, the remote database schema was not updated in tandem. This resulted in unexpected errors when the finished web application server was placed on the AWS EC2 instance, connected to the remote database, and the mobile applications attempted to make calls to the server to create reservations. This could have been solved by using flyway or another database schema versioning tool, by having the data models fully fleshed out before implementation, or by testing the mobile applications with the server running locally but connected to the remote database and manually keeping the remote database up to date as the database schema evolved.

9. Operation Manual

9.1 Overview

The operation guide contains the instructions needed to operate the *Iot Remote Reservation application for Laundromat Appliances* mobile application to create a reservation for our installed washing machine. The operation manual includes both an installation guide and a user guide. Setup instructions are given for running our project. Step-by-step customer instructions are provided for creating an account, for logging into the mobile application, for storing billing information, for selecting a laundromat, for creating a reservation, for viewing reservation receipts, and for obtaining the reservation access. Step-by-step administrator instructions are also provided for logging into the mobile application, for selecting a laundromat, for viewing energy consumption analytics, for viewing machine activity analytics, and for overriding active machines.

9.2 Intended Users

The operation guide is intended to be used by laundromat customers of of clients *Greiner Jennings Holdings, LLC.* The information found in this guide also applies to our teaching assistants, professors, and panel members interested for demonstration purposes.

9.3 System Configuration

The *IoT Remote Reservation Application for Laundromat Appliances* mobile application runs on both the the Android and iOS Operating systems. The android application is compatible with Android 8.0 Oreo API 26. The iOS application is compatible with iOS 11.

9.4 Contingencies

Both the Android and iOS mobile devices require internet connectivity when in use. If the product is to be released in the future, it is required that the administrators create a *Stripe* account to obtain a new *Stripe* API key for payment transactions. It is also required that the administrators create a *Google Gmail Developer* account to obtain a new *Google Maps* API key for access to the map data.

9.5 Getting Started

The getting started section explains how to setup and install *Iot Remote Reservation application for Laundromat Appliances*

9.5.1 Prerequisites

Since the mobile application is a proof of concept and several important security features have been left out of the implementation due to time constraints, an APK file and an IPA file have not been generated. An APK file is the file format generated by Android to be submitted to the *Google Playstore*. An IPA file is the file format generated by iOS to be submitted to the *App store*. Since an APK and IPA file have not been generated, running our project requires both Android Studio and Xcode IDEs to be installed. Xcode is the official IDE for the Mac OS and requires an application laptop or desktop to run.

9.5.2 Application Installation

To download the source code for our project, please visit https://git.ece.iastate.edu/sd/sddec18-17 to download a .zip file. Unzip the compressed file. From the Android Studio Launch screen, click open project and navigate to the directory where the unzipped project folder is located. Select the folder and open the project. From the Xcode Launch Screen, click open project and navigate to the directory where the unzipped project folder is located. Select the .xcodeproj file and open the project.

9.5.3 Customer Steps

The following instructions include step-by-step processes for customers who are looking to reservation a washing machine or dryer from a laundromat.

9.5.4 Create an Account

On application launch, the user is directed to a login screen. To create an account, click on the cyan button labeled “Register”. You will be directed to a screen shown on the following page. The screen has several input fields that are required to be filled out. Those fields include: First name, Last name, Email, Password, and confirm password. Once all of the fields are entered, the user must click the “Confirm” button. Upon a successful registration, the user will be redirected to the login screen. Otherwise, the invalid fields display an error message.

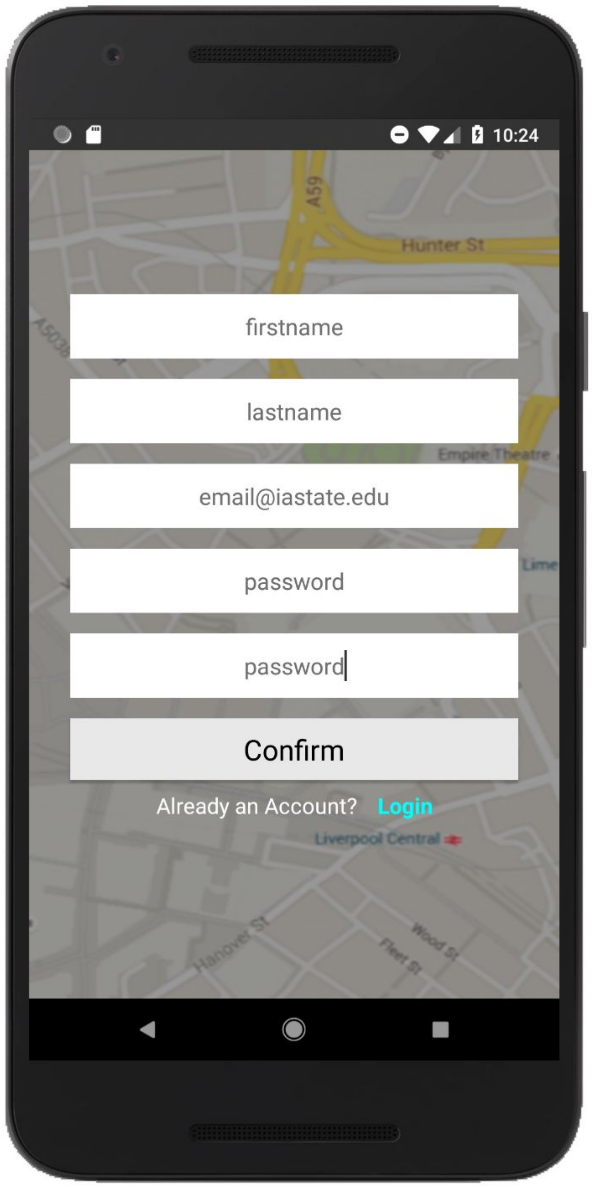


Figure 47: Registration Screen

9.5.5 Login to Account

Once a new account has been registered, the user may login to the application. To login, enter both the email address and password used when registering for an account. An example is shown below:

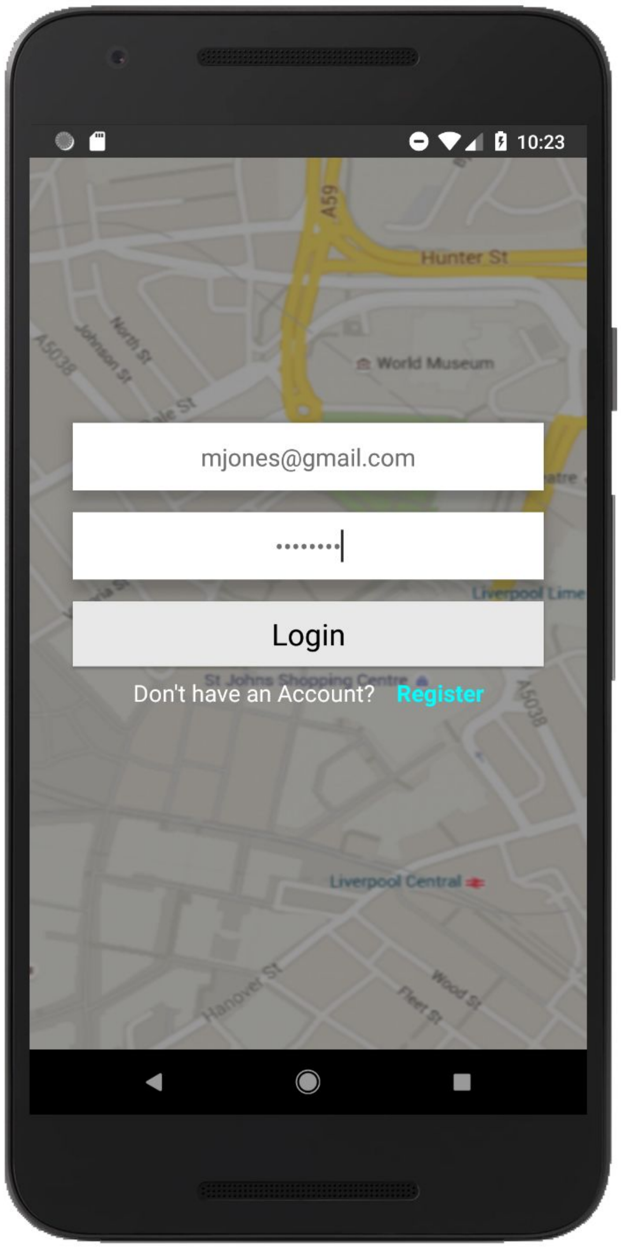


Figure 48: Login Screen

9.5.6 Credit Card Information

After logging into the application, if it's the users first time, then they will be prompted to enter their credit card information. The information required includes: card number, card expiration month, card expiration year, card CVC. The Stripe payment SDK is used to validate the card information.

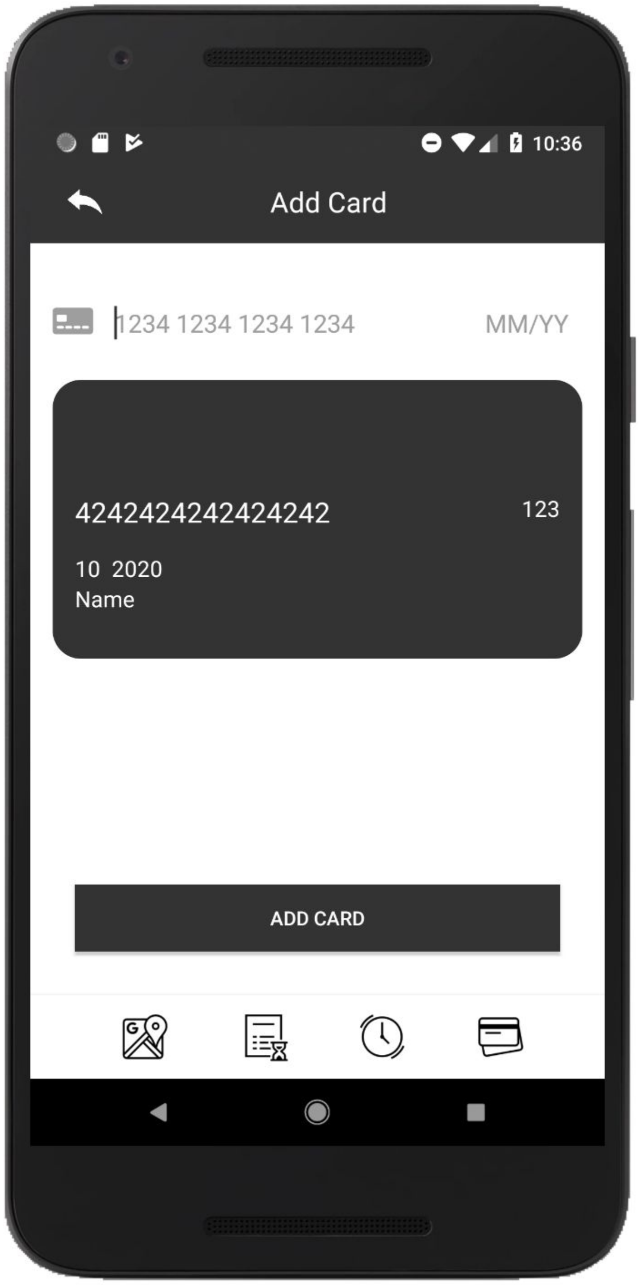


Figure 49: Credit Card Screen

9.5.7 Home Screen

Once the credit card information has been validated and saved, the user will be directed to the home screen. The home screen utilizes Google Maps to display nearby laundromats, ATM machines and Banks. To begin a reservation process, click on one of the dark gray laundromat markers shown on the map. Clicking on a laundromat marker will display a dark gray bottom bar with a laundromat icon. Click the laundromat icon.

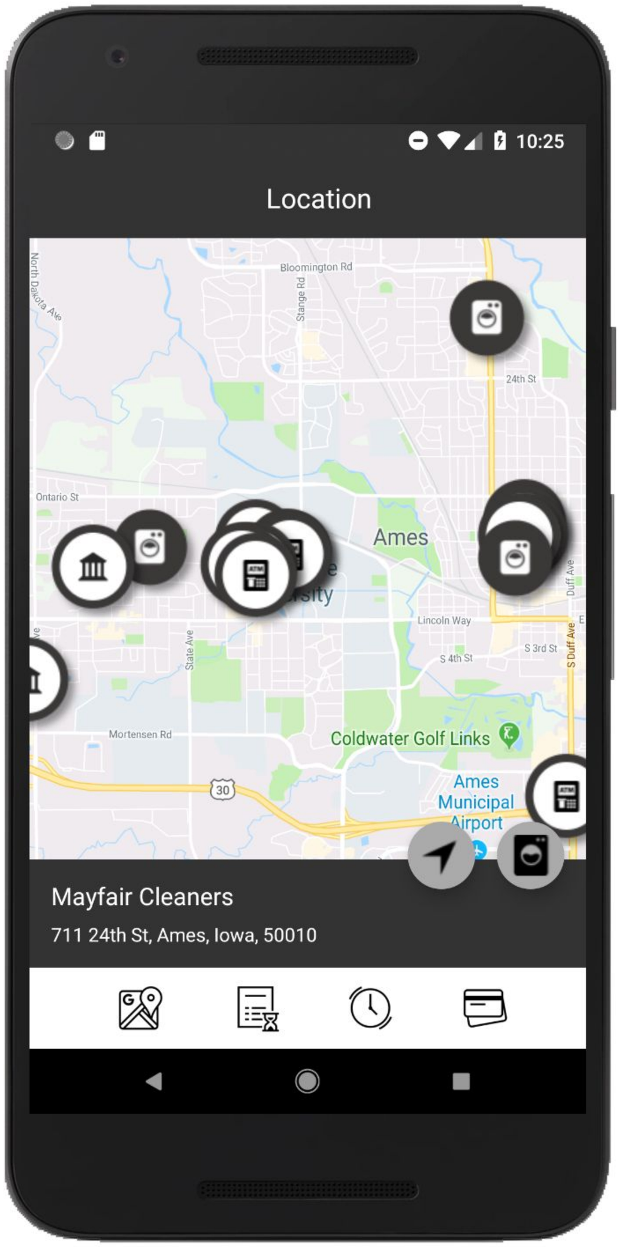
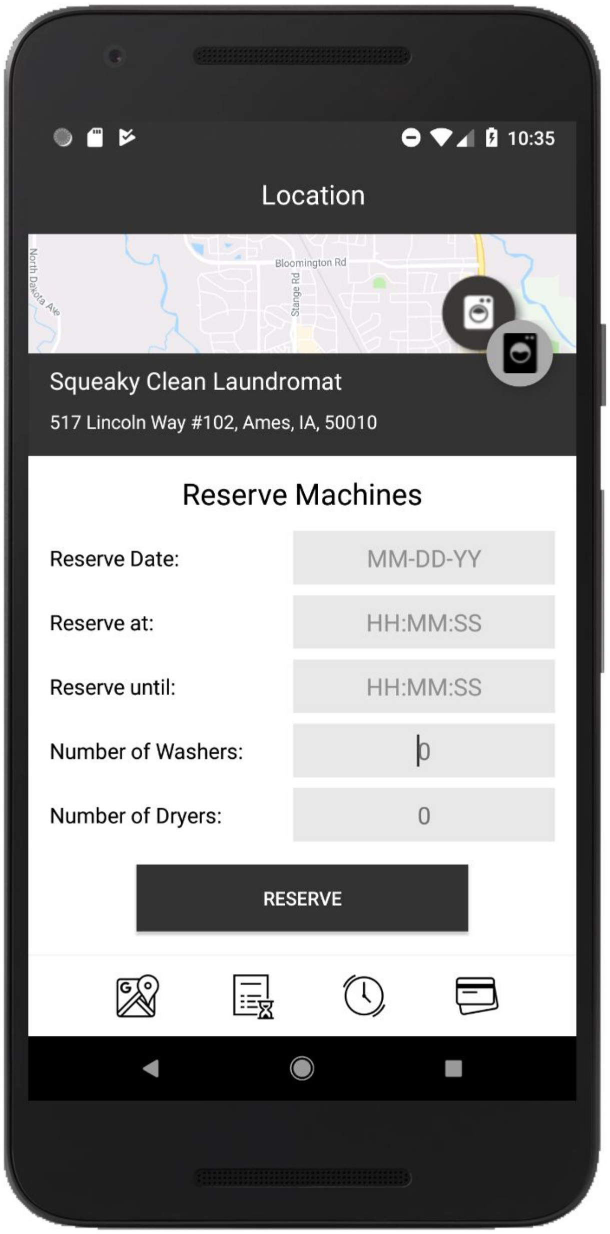


Figure 50: Home Screen

9.5.8 Reservation Screen

Once the laundromat icon has been clicked from the bottom dark gray bar, a reservation form will appear. The reservation form requires the following information: the date for the reservation, the start time for the reservation, the end time for the reservation, the number of washing machines needed, and the number of dryers needed. The price for a reservation is $2 per hour per machine. The total will be displayed at the bottom. After the form has been completed, click the “Reserve button”.



Squeaky Clean Laundromat

517 Lincoln Way #102, Ames, IA, 50010

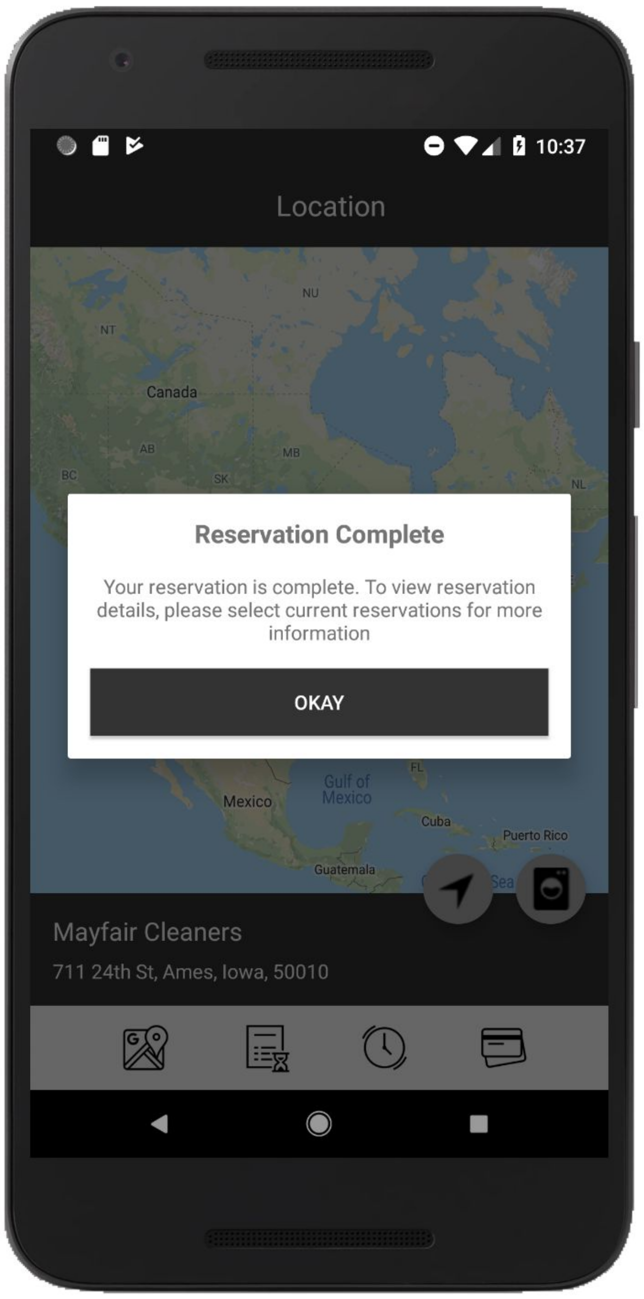
Reserve Machines

Reserve Date:	MM-DD-YY
Reserve at:	HH:MM:SS
Reserve until:	HH:MM:SS
Number of Washers:	0
Number of Dryers:	0

Figure 51: Reservation Screen

9.5.9 Confirmation Screen

Once the reservation has been submitted successfully, the user will see a confirmation popup on-screen. Otherwise, a popup will display a different error message depending on the error, including if the credit card was declined or if the number of washing machines or dryers is unavailable during the reservation period.

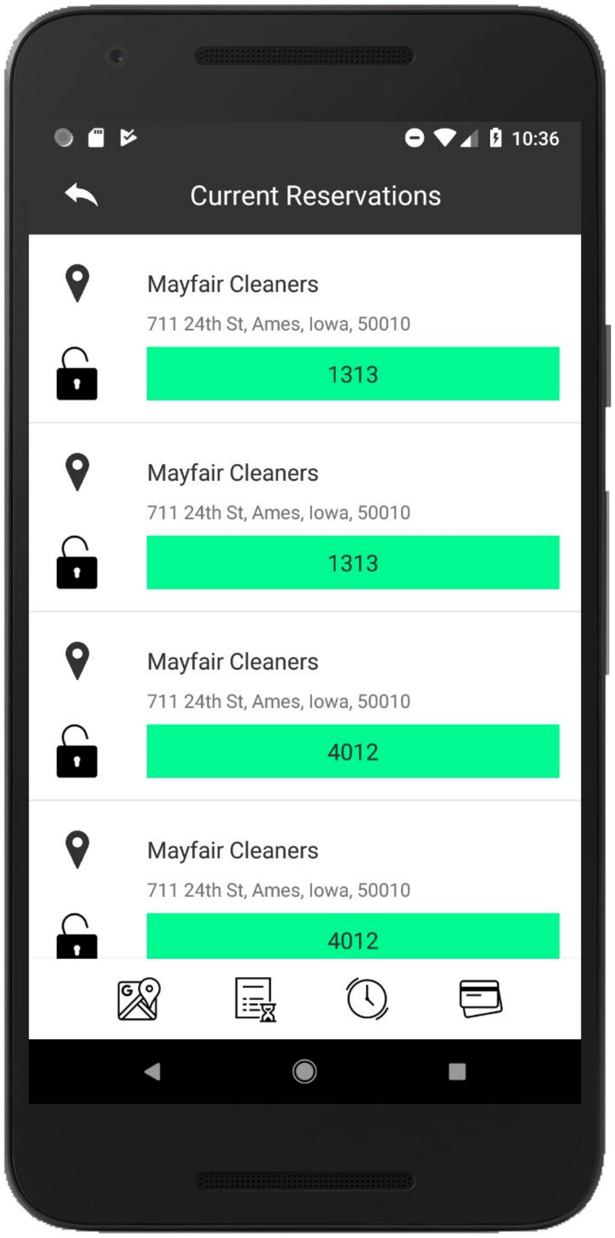


Your reservation is complete. To view reservation details, please select current reservations for more information

Figure 52: Confirmation Screen

9.5.10 Current Reservations Screen

Once a reservation has been submitted, you may view all current reservation details by clicking the third button on the bottom white toolbar. The third button looks like a watch or clock. Clicking on the button navigates the user to a screen that displays a list of reservations. The visible information includes the laundromat location, address, and reservation access code. The reservation receipt can be viewed by clicking on one of the rows in the list.



Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
1313

Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
1313

Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4012

Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4012

Figure 53: Current Reservations Screen

9.5.11 Previous Reservations Screen

After a reservation time has closed, the reservation will no longer be visible from the list of current reservations. Instead, the reservation will be moved to the previous reservations screen. Previous reservations may be viewed by clicking the second button on the bottom white toolbar. Clicking on the button navigates the user to the list of previous reservations. The visible information includes the laundromat location, address, and reservation access code. The reservation receipt can be viewed by clicking on one of the rows in the list.

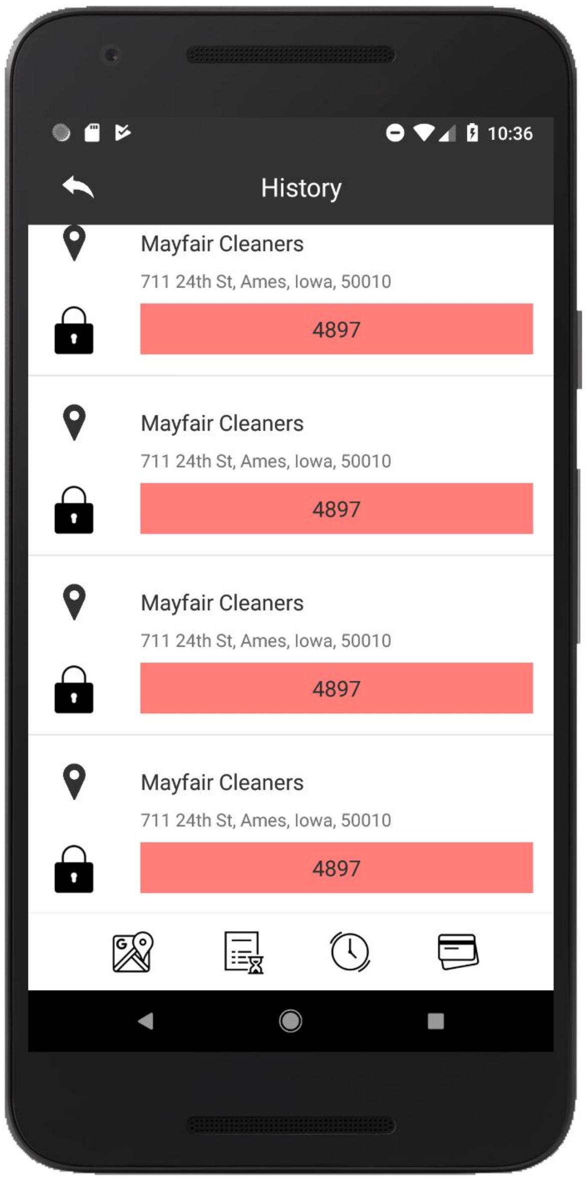
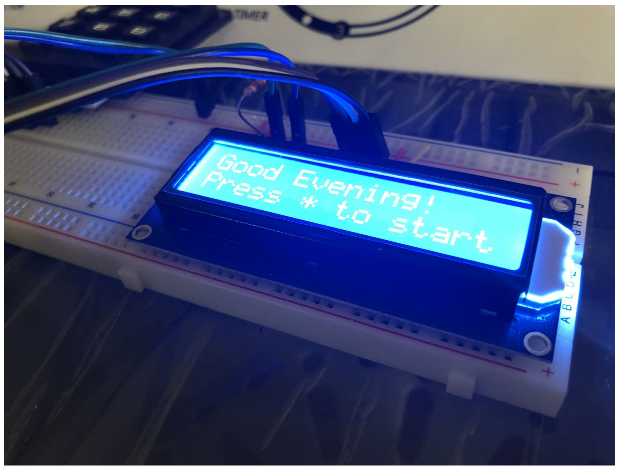

10:36
History
Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4897
Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4897
Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4897
Mayfair Cleaners
711 24th St, Ames, Iowa, 50010
4897

Figure 54: Previous Reservations Screen

9.5.12 Access Code

Once the access code has been obtained by viewing the current reservation, enter the reservation code into the keypad attached to the appliance.



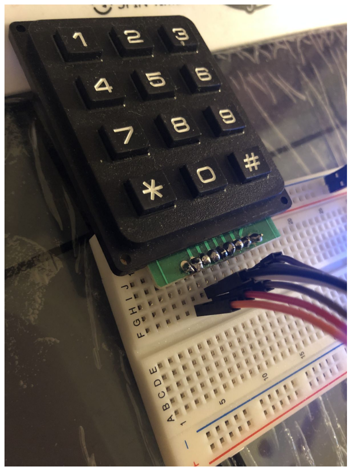


Figure 55: LCD Screen **Figure 56: Keypad**

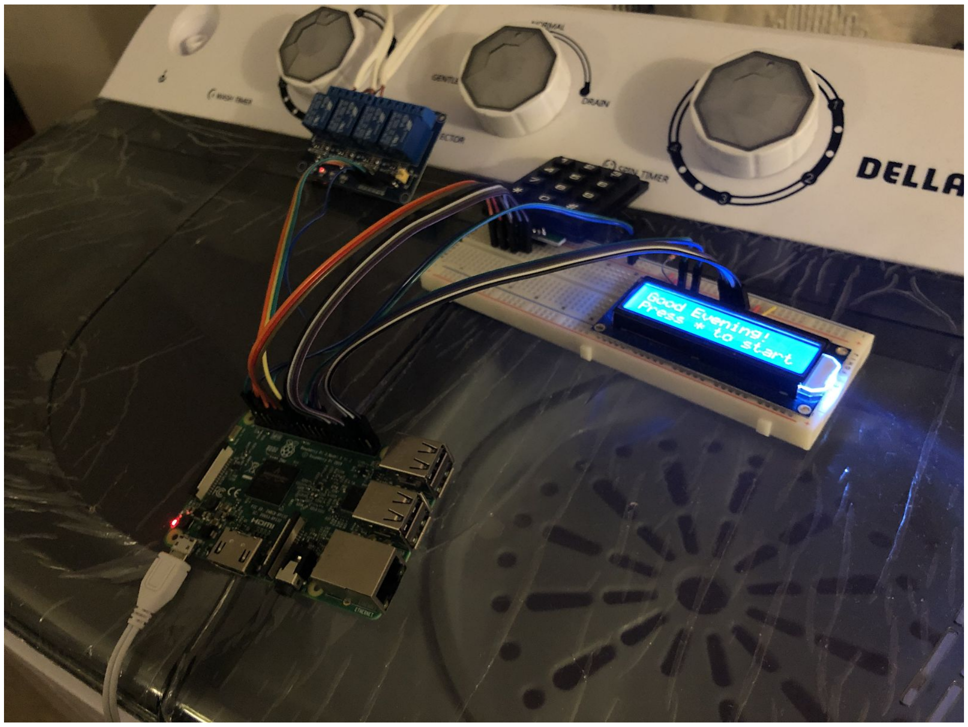


Figure 57: Washing Machine Locking Mechanism

9.5.13 Administrator Steps

The following instructions include step-by-step processes for administrators who need to either view data analytics obtained from their laundromat or override appliances.

9.5.14 Administrator Login

An administrator does not need to register for an account via the mobile application. Rather, the long-term plan is for administrator accounts to be created through a developer portal on an administrator website. The website implementation was out-of-scope for the project and was not requested by our clients. An administrator account is currently created by directed appending a user via the MySQL dashboard. Using administrator login and password information, the admin may login to the mobile application.

9.5.15 Administrator Home Screen

After logging into an account, the administrator is directed to a home screen similar to the customer home screen. The admin may view nearby laundromats owned and operated under their name. Clicking on a laundromat marker displays a bottom navigation bar for viewing analytics for the selected laundromat.

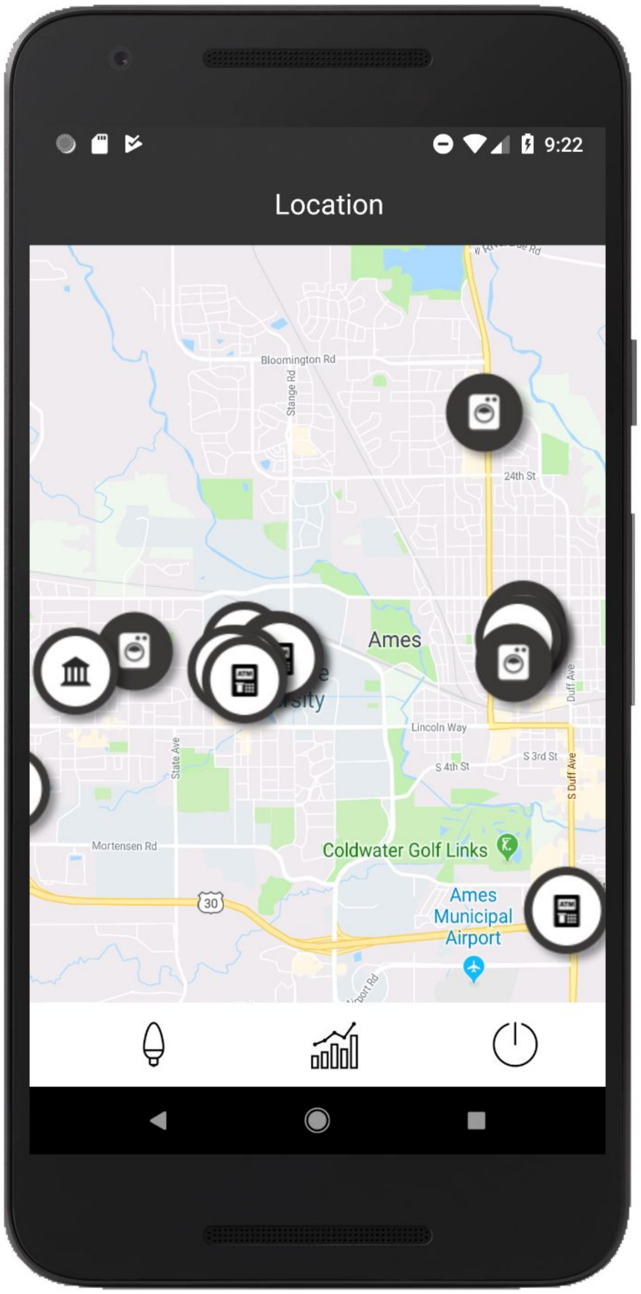


Figure 58: Administrator Home Screen

9.5.16 Energy Consumption Screen

Once a laundromat has been selected, the administrator may view different characteristics about the laundromat. To view energy consumption statistics, click on the light bulb button. The light bulb button directs the user to the energy consumption screen. The energy consumption screen shows a pie chart that displays how much energy each appliance uses relative to the total energy consumption of all appliances.

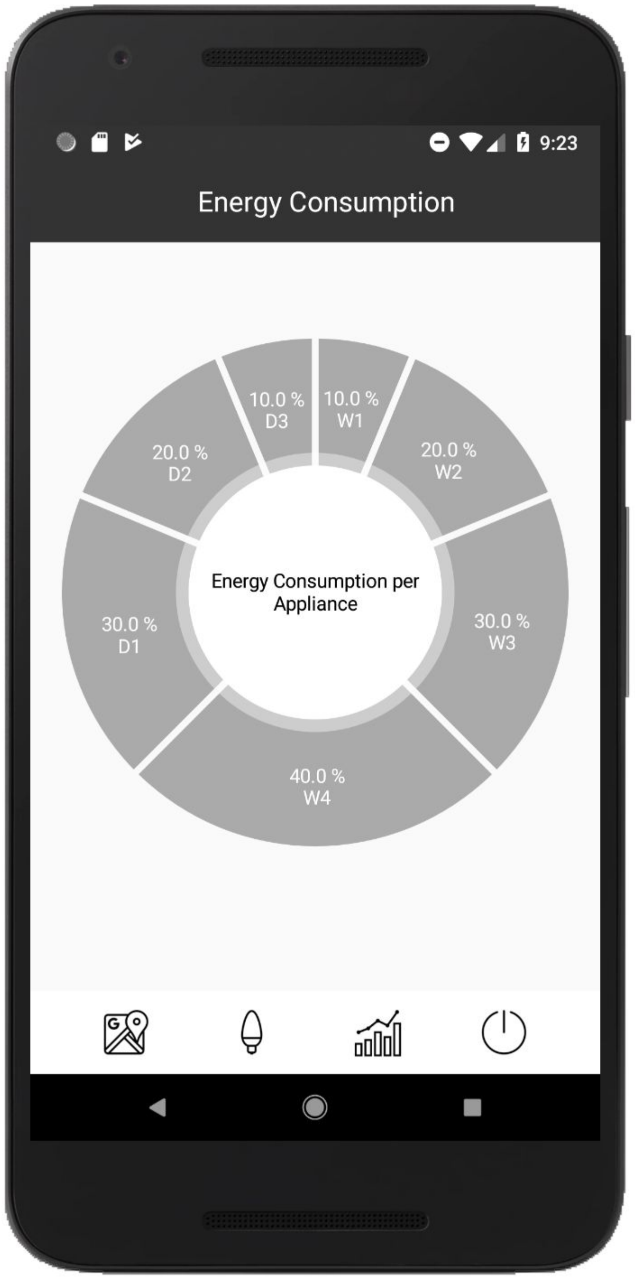


Figure 59: Energy Consumption Screen

9.5.17 Machine Activity Screen

The administrator may also view statistics about how many appliances are running per hour throughout the day. The information can be used to determine what times of the day are the busiest and can be used in the future for price surging.

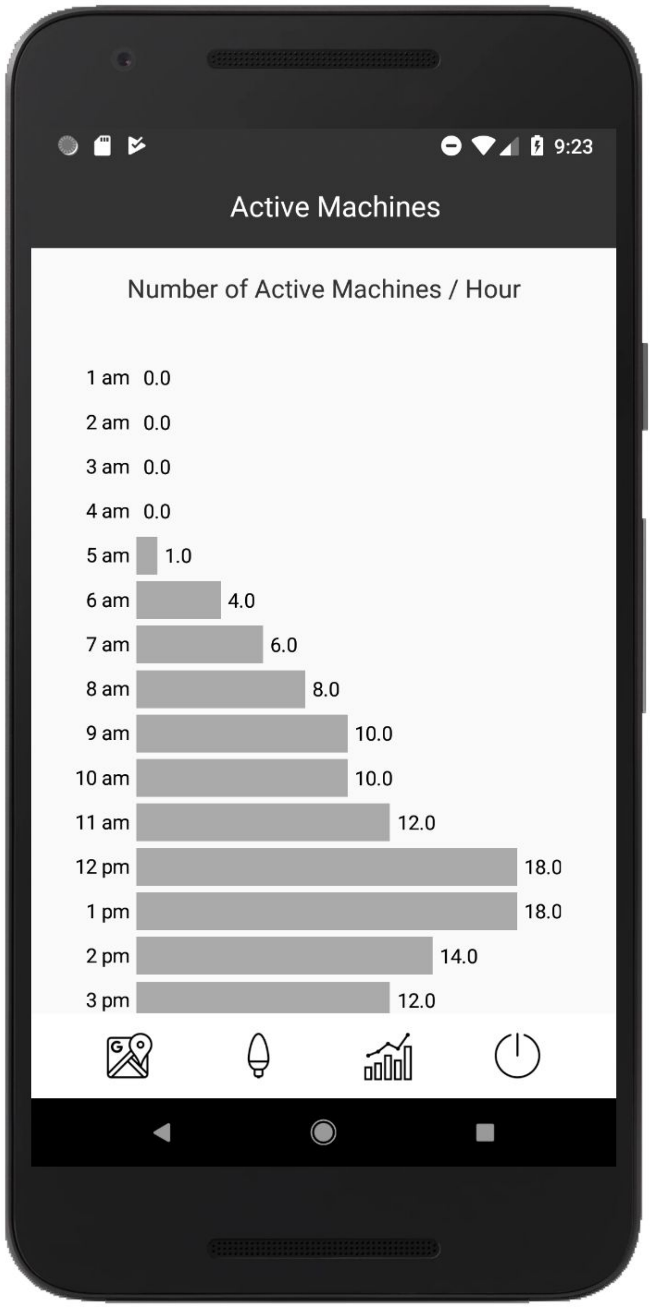
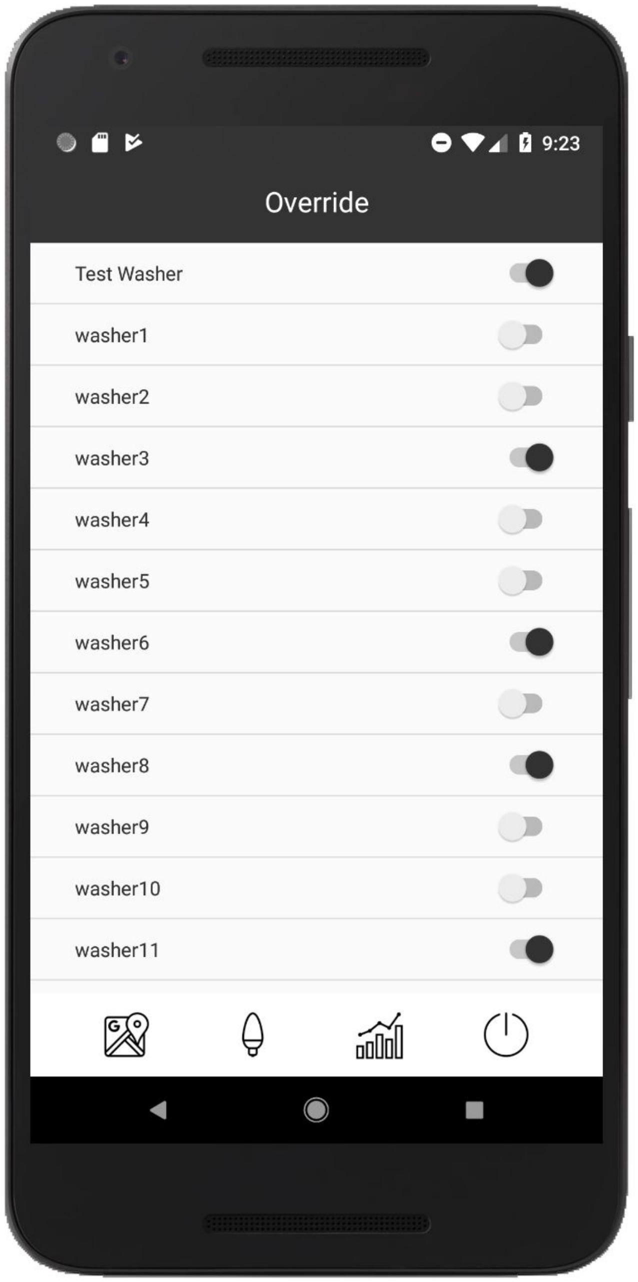


Figure 60: Machine Activity Screen

9.5.18 Machine Activity Screen

The administrator is also given the privilege to override machines, depending on the circumstance. For instance, if a customer does not activate their reservation code within the first 15 minutes of their reservation, the appliance may forfeited to another customer, as per the *Terms of Service*. To navigate to the override screen, the administrator must click on the power button.



Test Washer	●
washer1	○
washer2	○
washer3	●
washer4	○
washer5	○
washer6	●
washer7	○
washer8	●
washer9	○
washer10	○
washer11	●

Figure 61: Override Screen

10. Previous Work and Literature

Our project *IoT Remote Reservation Application for Washers and Dryers* strictly relates to the general concept of the “Internet of Things” or “IoT”. The Internet of Things refers to a network connection of physical devices such as vehicles, home appliances, and other commercial and industrial products embedded with software, electronics, and sensors. In 1999, Kevin Ashton from Auto-ID Labs at MIT coined the term “The Internet of Things”. Even though the term was coined in 1999, the idea behind IoT was established well before that. In fact, Carnegie Mellon University created one of the first examples of the Internet of Things when they built a Coca Cola Machine that was connected to the internet[1]. Since 1999, the Internet of Things has grown exponentially. According to a news post by IEEE SPECTRUM, there will be nearly 50 billion individual devices connected as IoT network in 2020[2]. The schema we have planned for our project is a terminal-cloud-terminal model of internet of things. As mobile application and cloud computing techniques have advanced, the internet of things has transformed towards mobile devices and web services like AWS and BlueMix. This way, we can build a full-lifecycle, spatial distributed IoT project which is similar to but not exactly like the ‘ancient drink machines’.

10.1 Market Survey: IoT Pay-Per-Wash Industry

The IoT Pay-Per-Wash Industry saw a spark in 2014 when the company *Bundles* headquartered in the UK launched a clean clothes pay-per-wash business model that utilizes the Internet of Things technology. The *Bundles* business model serves as a leasing scheme were customers may lease a set of laundry appliances and pay a monthly fee based on their appliance usage. The business model has been adapted by UW Huismeester, a company based in the Netherland’s to their home caretaker service[4].

10.2 Market Survey: Samsung Electronics Laundry Innovation

On January 7, 2018, Samsung Electronics announced their expanded laundry line up with a new Premium Iot Compact Washer. The WW6850N washing machine uses IoT to connect to Samsung’s SmartThings ecosystem. The washing machine collects usage data and implements sensors to provide recommendations for the most optimal wash cycles based on color and fabric type. The laundry planner feature gives customers the ability to manage how long a wash cycle will take and they may further monitor the machine for performance data[5].

10.3 Market Survey: Berendsen Microsoft Azure and IoT Hotel Laundry Service

Last April, Microsoft released an article highlighting an interesting use by Berendsen of their Microsoft Azure Cloud solution to develop an Internet of Linens. Berendsen implemented Microsoft Azure IoT to track linen ID tags throughout the cleaning process of 1 million pieces of linen that they launder and return to facilities everyday throughout Europe[6].

11. Prototype Commercialization

One of the first things for commercializing the prototype is AWS scalability. Currently, the prototype exists on a single T2.micro EC2 instance running Amazon Linux. In a fully commercialized application, it would be beneficial to run all of the instances in an ECS cluster for scalability. A benefit of running the web application this way would be that auto scaling could be enabled to spin up instances when they are needed and take down extra instances when they are not. ECS clusters also have their own load balancer in front of all of the instances that allows the scalability of dynamically allocated instances to work effectively without having to change specific URLs in the mobile applications.

Another thing that the prototype needs polished is that currently the reservations use the time on the machine the application is hosted on to actually check if it is time for the reservations or not. This becomes a problem when the EC2 instances are not set to having the same time zone as the laundromats that they are serving. To fix this temporarily, the client may manually set the time zones on each of the EC2 instances that they set up. However, for long-term scalability and growth, it would be more beneficial to refactor the method that checks if it is currently time for a reservation to be active to a method that takes a time input and then checks the reservation against that. Then the localized time for the washer or dryer the reservation is for can be calculated via a time zone element that can be stored with each appliance in the database, and then is checked against the reservation time for the appliance. This would allow the web application to exist outside the need to be in the same region as the appliances, allowing cross-regional support and redundancy.

Deploying to AWS is also slightly different from just running the web application normally. First, the path from locally finding the AWS IoT keystore needs to be changed to the full path the keystore will be located on the AWS instance. Then web application needs to be built into a jar, and uploaded onto an EC2 instance in AWS. Finally, the team suggests running the web application using ‘nohup’ to allow logging, and running it in the background so that when the terminal logged into the EC2 instance loses connection the web application does not terminate. To make this process easier, it would be beneficial to fix how the web application dynamically loads the keystore so it may be bundled as a resource to the application, and look into automated deployment solutions such as docker and terraform.

To keep the data that is sent to and from the web application secure, HTTPS will need to be enabled both on the mobile applications and on the web application server. This may be done on the server side via setting up the Tomcat instance that Spring Boot relies on to run, and the mobile side can implement the connection to validate with the Tomcat instances running on AWS. This will most likely need to be done using a common keystore that is shared among all of the deployed AWS instances. The actual connection protocol that HTTPS will use should be either TLSv1.1, TLSv1.2, or even TLSv1.3, as TLSv1.0 and SSLv3 are both considered insecure.

Implementing HTTPS goes a long way towards keeping personally identifiable data secure, but it may be a good idea to further encrypt the credit card information. This could be done using almost any existing encryption strategy, as long as the mobile applications and the web application server agree on the implementation. Credit card data should also be encrypted before it is stored in the database, as currently it is not and that is a definite security concern.

One of the many benefits of having a web application using Spring Boot is the ability to use Spring Security to both secure endpoints on the application and to have customized authentication. Currently Spring Boot is enabled on the web server, however customized authentication and role management has not been implemented yet. By implementing this, the commercialized application will be able to handle its own authentication not only on specific methods but also on a blanket API bases across the application. This will help secure the application by not allowing normal users to do admin specific privileges.

In regards to administrative roles being needed for specific API endpoints, the administrative side of the web application also needs to be fleshed out when the prototype is commercialized. Both of the mobile applications have administrative screens mocked up, but the data that fills the screens is hard coded. This data needs to be pulled in from endpoints added to the web application server, and any endpoints that do not currently exist that are necessary to support administrative functionality will need to be added.

Speaking of functionality that needs to be added for commercial usage, the current prototype uses a single MQTT queue to talk to a washer. To make the prototype commercially scalable, there are several options depending on how the MQTT connections are decided to be configured. Currently, the prototype uses a single MQTT queue per machine. However, it is possible to have multiple machines using the same MQTT queue, and just include a machine number in the message so each machine may sort through the different messages in the queue. It is also possible to dynamically add and remove queues with the MQTT client chosen for the prototype. A way of making sure a message is not picked up by two different web application instances on the same MQTT queue also needs to be looked into, so as not to duplicate the check on multiple web application instances.

The final improvement to transform the prototype into a commercially scalable product is to update the arduino attached to the washers and dryers to be able to work with multiple appliances at once. By having a one to one mapping from arduino to appliance, the scalability is severely limited when it is possible to just have one arduino controlling several washers and dryers. This could be implemented by entering ‘*’ [machine number] ‘*’ [unlocking code] ‘*’ with using ‘#’ to clear input, or some similar method of expansion.

12. References

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

2. 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.

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

4. “Internet Enabled Pay-per-Wash: a Model Offering Multiple Benefits.” *Ellen MacArthur Foundation*, www.ellenmacarthurfoundation.org/case-studies/internet-enabled-pay-per-wash-a-model-offering-multiple-benefits.

5. Samsung U.S. Newsroom. “Samsung Expands Laundry Line Up with New Premium Compact Washer.” *Samsung Global Newsroom*, Samsung Newsroom US, 8 Jan. 2018, news.samsung.com/us/samsung-quickdrive-WW6850N-washing-machine-laundry-CES2018/.

6. “Transforming Hotel Laundry Service with the Internet of Linens.” *Microsoft Green Blog*, 17 May 2017, blogs.microsoft.com/iot/2017/04/13/transforming-hotel-laundry-service-with-the-internet-of-linens/.