ASHESI UNIVERSITY COLLEGE

A LIGHTWEIGHT IMPLEMENTATION OF THE INTERNET OF THINGS

APPLIED PROJECT

B.Sc. Computer Science

Michael Annor

2016

ASHESI UNIVERSITY COLLEGE

A Lightweight Implementation of the Internet Of Things

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Michael Annor

April 2016

Declaration

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

Acknowledgement

I am immensely thankful for all the support I have received from my supervisor, family and friends. Without their support I would not be able to complete this dissertation. I am grateful for the opportunity to broaden my understanding of the Internet of Things and related technologies.

Abstract

The Internet has undergone many changes in its 45-year-old history. The next wave for the Internet is the Internet of Things (IoT). IoT is is defined as “group of infrastructures interconnecting connected objects and allowing their management, data mining and the access to the data they generate” (Dorsemaine). This project implements a lightweight IoT system that can control devices in the home. This system can then be extended to other use cases in agriculture, healthcare and education.

Table of Contents

List of Tables

List of Figures

1. Chapter 1: Introduction

1.1 Background

The emergence and growth of the internet over the last five decades has led to a massive interconnection of people and devices across the globe, thus, facilitating broader spectrum of communication as well as the ability to work remotely. In recent years, the internet has grown past serving content on webpages to becoming more user-centric. It has become a participatory medium, allowing users to get more social and interactive by creating and sharing and collaborating on content (Castellani, Dissegna, Bui, & Zorzi, 2012). Gradually, the walls between the online and offline worlds are fading.

Today, the internet is undergoing a significant transformation. A "new class of users is establishing itself in the Internet landscape"- physical objects (Castellani, Dissegna, Bui, & Zorzi, 2012). It is predicted that there will soon be billions of these physical objects interconnected and liaising between the offline and online worlds (Dorsemaine, Gaulier, Wary, Kheir, & Urien, 2015). These connected physical objects will improve and redefine processes. They will also serve as a means for data collection for big data mining to enable better and more accurate predictions (Dorsemaine, Gaulier, Wary, Kheir, & Urien, 2015). Big data collected from physical objects is being used to gather intelligence with use cases across several industries including energy management and optimization, precision agriculture and renewable energy forecasting (Hamann, 2015). This is the basis of the Internet of Things (IoT).

The Internet of Things (IoT) paradigm has the potential to disrupt life as we know it. It would birth new commercial opportunities for businesses, evident from present investment from large multinational technology companies including, Google, Apple, IBM and Phillips. The innovative application of the Internet of Things in the household, out in the city and in industry would make life more convenient in these domains.

1.2 Problem

The predicted success of the Internet of Things (IoT) is not without obstacles. The Internet of Things (IoT) is lacking a set of widely accepted open standards for the interconnection of physical objects. At present this hinders the seamless interoperability of devices from varying vendors. There are however a few open source standards in IoT including "architectural frameworks, reference models and data-abstraction blueprints" (Logvinov, 2014).

The IoT could be described as the next big thing on the internet after social media. Fittingly, enjoyable user-experiences would be a key factor to the widespread acceptance of IoT as is the case with social media. A simple, well defined standard would be helpful for the growth of IoT.

1.3 Objective

This project would, taking cognizance of the potential opportunities and obstacles of the Internet of Things develop a model IoT-based system that allows interconnection and control of physical devices over the internet to investigate the directions the IoT paradigm should take for widespread adoption especially in the context of metropolitan Africa.

1.4 Overview of Remaining Chapters

The paper will next discuss the requirements of the system, followed by a chapter on the design considerations for the system. Thereafter the paper will discuss the implementation of the system. Next the paper will present the tests that were run on the developed system and the results. The paper will then conclude with a chapter on limitations and possible future work.

2. Chapter 2: Requirements

2.1 Overview

This section discusses the scope of this project and provides an overview of the requirement chapter.

2.1.1 Scope

This project designs and implements the Internet of Things in the context of a smart home, connecting physical objects with sensors to allow them automatically respond to changes in the environment and messages from other devices. For example, as shown in Figure 2.1, A proximity sensor may be used to sense a person's presence in a room, communicate this to a gateway which would in turn switch on a light bulb. In a similar manner, sensors can be used to operate other devices including fridges, washing machines, air conditioners, etc. The system should also allow a human user to remotely override the automated control of these devices. In addition, the system should keep the human user updated whenever changes occur for example, a user may receive an SMSwhen his front door is unlocked.

In this project, there is the option to either use a vendor-provided Internet of Things (IoT) Framework, or to independently build the framework that would interconnect the physical objects using available open standards.

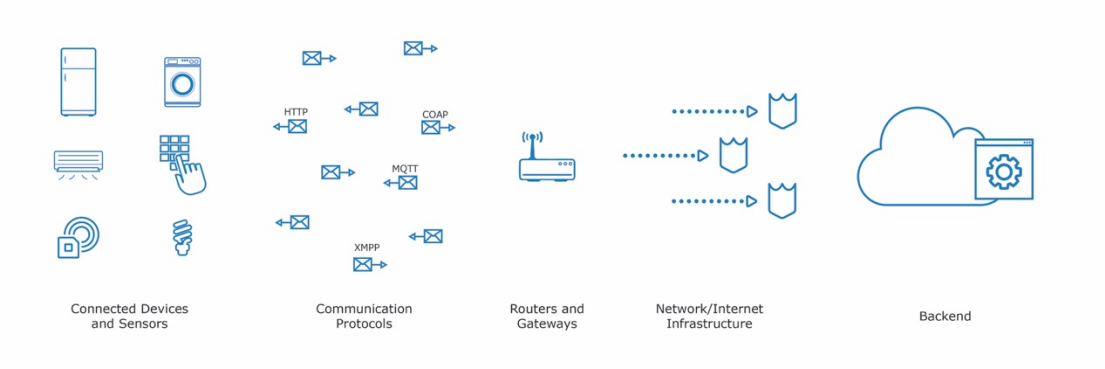

Figure 2.1: Building Blocks of the Internet of Things

A given IoT Framework will have common building blocks as shown in Figure 2.1. Connected Devices and Sensors could range from everyday home appliances to industry specific machinery, for example cookers, thermostats, unmanned forklifts, sprinklers, etc. These devices and sensors are equipped to connect to a computer network over various communication protocols. Common communication protocols in IoT include Constrained Application Protocol (CoAP), Message Queuing Telemetry Transport (MQTT), OMA LightweightM2M (LWM2M).

MQTT is a lightweight publish subscribe protocol for low bandwidth, high latency networks. This makes MQTT ideal for Machine to Machine communications and the Internet of Things. MQTT is convenient for use in constrained environments and has been widely used across industries since 1999. (MQTT, 2016). CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks (CoAP, 2016). LWM2M is a device management protocol suitable for sensor or cellular networks (Eclipse, 2016). These protocols connect the devices and sensors to a Gateway/Router that in turn channels the communication to a Backend Engine for processing.

Two common options for the vendor-provided frameworks are the Eclipse IoT Framework and the Windows 10 IoT Core.

The Eclipse Foundation manages a collection of open source IoT projects, one of which is the Kura Open Source framework for IoT. Windows 10 IoT Core is Microsoft's IoT

platform for its Windows platform, with editions for low cost devices, mobile and an Enterprise Edition for industry devices (Microsoft, 2016).

Initially the inclination was to develop parallel systems using the Kura Open Source framework for IoT on one hand and the Windows 10 IoT Core on another and there after run comparative tests to make recommendations on the more appropriate system. However, in light of the many threats to the Internet of Things (IoT) growing to the predicted scale such as challenges with interoperability, the absence of widely accepted standards, etc., it would be unfair to present a vendor's framework as ideal over it's competitors. Instead, this project would take a direction to develop a system using open technologies as a simple but representative model for a successful Internet of Things (IoT) era.

Unlike the Windows 10 IoT Core which is limited to Windows operating systems and Eclipse Kura framework which is complex with support for different communication technologies including Wi-Fi and Bluetooth, this project will be a lightweight and an interoperable implementation of an IoT solution.

In essence, the project would have different building blocks as shown below in the block diagram in Figure 2.2. These blocks would enable one to build an IoT framework with a minimal set of tools. With only a Hub, Mobile application and Connected Devices, a user can connect and control an entire home.

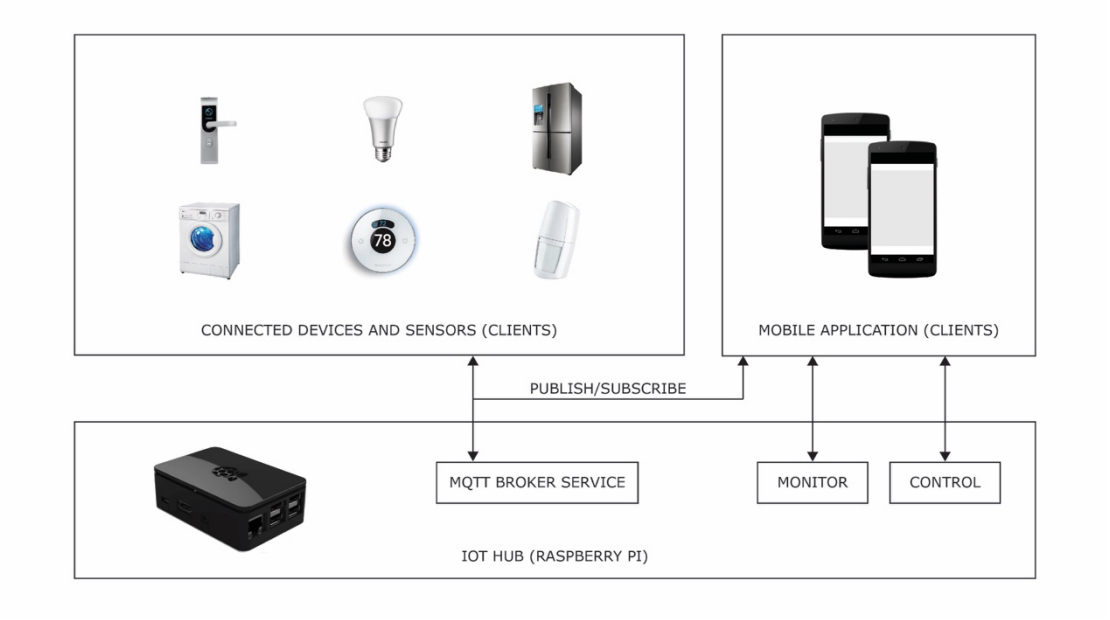

Figure 2.2: Block Diagram showing the components of the IoT system

The different components of the IoT system are explained below:

- **Internet of Things (IoT) Hub (Broker):** The Hub is the central server that will run all the services in the local environment. The main service that will run on the Hub is the MQTT Broker service. The Broker is a service that will handle the publishing and subscribing of messages using the MQTT protocols. It will connect and facilitate communication across clients. A client is a device or service that connects to the Broker and can send and receive messages. A publisher client is a client that is configured to send messages and a subscriber client is one, configured to receive messages. In some cases, a single device or service may be both a publisher and subscriber. For example, a light bulb can publish its state and subscribe to receive control instructions.

This IoT Broker software will run on a Raspberry Pi to interface the connected physical objects and a network. The Raspberry Pi is a portable low cost computer largely used for educational and prototyping purposes. It will be used instead of other alternatives (laptop computer, BeagleBone) because of its affordable price and computing power, making it a convenient choice. The Raspberry Pi 2 Model B has a 900MHz quad-core processor and 1GB RAM. As well as run services and process that will facilitate the automation and remote control.

The IoT Hub will connect devices via the MQTT protocol and will define how message passing is done between clients; publishers and subscribers. Had the project gone down the initial path, the IoT Hub (Broker) would be the host for the frameworks- Windows 10 IoT Core and Eclipse IoT Framework. There would be an internet accessible clone of the IoT Hub in the cloud to connect clients outside of the home.

- **Mobile Application:** The mobile application would be the user's interface to interact with the system and control the connected physical devices. The application would run on the Android Operating System and would connect to the devices through the IoT Hub (Broker) using the MQTT Protocol and Websockets. Websockets is a protocol that creates an interactive full duplex channel over a single TCP connection. In this project it would be used to create web MQTT clients.

  The mobile app would support interactions to view connected devices and their states, control these states and receive notifications when the states are changed. Additionally, the system could also allow the user to view reports on the system with respect to usage patterns, data usage and costs, etc.

- **Connected Physical Objects:** The system will be designed to control everyday electronic devices. Since a standard device (for example, light bulb or air

conditioner) does not come with connectivity features, the devices will be modified to include a Wi-Fi microcontroller module, to enable them communicate with the other parts of the system, and be controlled remotely. The inclusion of the Wi-Fi microcontroller module would thus make them connected physical objects/devices. These Wi-Fi enabled devices will run firmware that enable the remote control of the devices over a Wi-Fi network, communicating using the MQTT protocol as well.

2.1.2 Overview of Software Requirement Specification

The requirement specification in this chapter describes the software product into detail, specifying the functions of the product, the user classes and their characteristics. The latter parts of the chapter will also provide details on the operating platform for the software product comprising the hardware platform, operating systems and other required software components and applications. The chapter will also specify the software requirements to enable a software engineer design the application. The chapter will also specify the interfaces of the software as well as the non-functional requirements.

2.2 Overall Description

This section provides an overview of for the requirements which are defined in detail in later in the chapter. The context diagram in Figure 2.3 at a high level describes the inner workings of the system. Some sensors connected to the system will have readings relevant to some devices, thus would form the device's set of related sensors. The two main actors, the resident and home owner are also shown in Figure 2.3 and how they interact with the system. They use the application to control and manage the system respectively. All communication links to the Hub are publish and subscription messages. The Hub is the

central point where messages are processed. When the hub receives a message, it forwards it to the desired recipient.

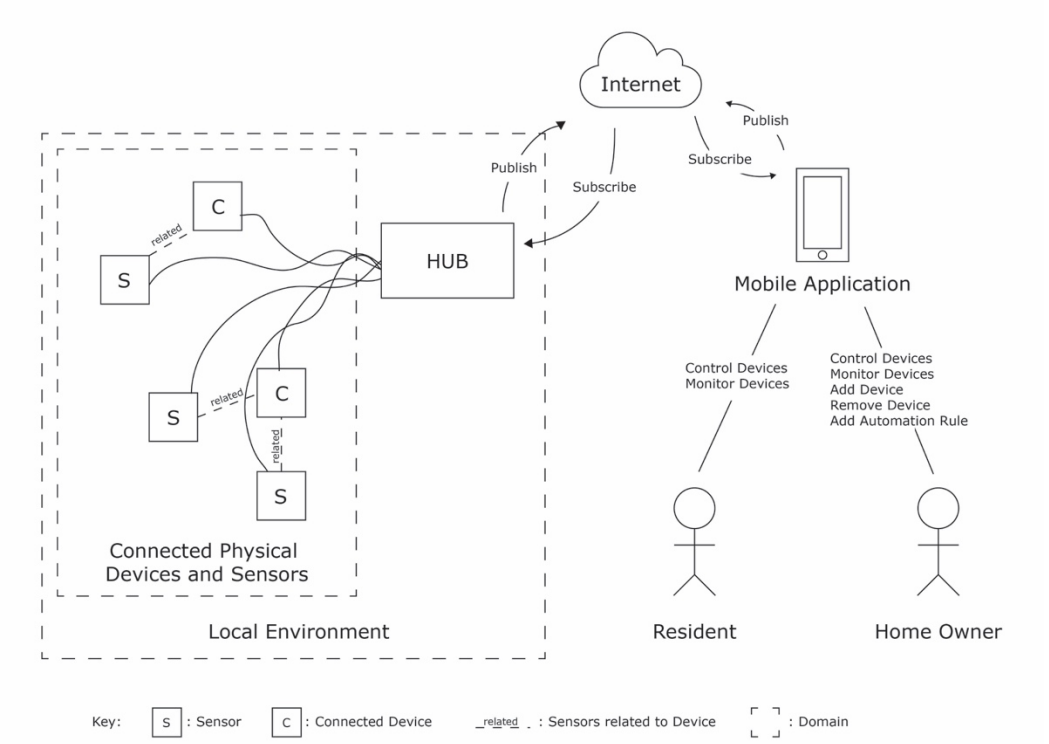

Figure 2.3: Context Diagram of System to be Developed

2.2.1 Product Perspective

As shown in the context diagram in Figure 2.3, the system will comprise three parts namely:

a) Internet of Things (IoT) Hub (Broker)

b) Mobile Application

c) Connected Physical Object

On the whole, the system will be largely self-contained and will run independent of other systems. However, to some extent, some of the parts listed above will interface with other external systems and services in order to achieve the goals of this system.

- **Internet of Things (IoT) Hub (Broker):** The Hub is the suite of services that will run on a Raspberry Pi in the local environment. This part of the system will be

independent of other systems. It will run its own database service and its own MQTT broker service. It will interface with other parts of the system, but no external systems. The broker will handle messaging between clients, and will bridge with another broker in the cloud for clients both in the local home environment and out on the internet.

- **Mobile Application:** This part will interface with the broker which in turn interfaces with the connected physical objects. The mobile application will run with other external systems to expand its features to include push notifications from Amazon Simple Notification Service and SMS messaging from SMSGH. This design of the mobile application software will help to satisfy the broad requirements of the overall system to enable the remote control of physical and otherwise not-connected objects. As well as monitor these objects remotely.
- **Connected Physical Objects:** Given that IoT is about bridging the connected world with the not-connected. A connected physical object is a device that can be accessed over a network. These may be electrical devices or sensors. A sensor is a connected physical object that sense changes in the physical world and report on the state. This part of the system will with the knowledge of electronics and an ESP8266 Wi-Fi module, interface un-connected, regular and un-automated physical objects with this IoT platform. The ESP8266 is a low cost microcontroller with Wi-Fi capabilities and the full TCP/IP stack and will be used to extend connectivity to traditional non-smart devices without network connectivity.

2.2.2 Product Functions

The principal functions of the software systems are as listed below:

- **Device Automation:** Based on clearly defined rules, this function enables the system to automate the control of physical devices. This functionality will particularly adjust the states of physical objects given real-life properties most likely from connected sensors.
- **Remote Device Control:** This function will enable an end-user to be able to override the programmed automation. A user should be able to remotely control individual objects at will.
- **Remote Device Monitoring:** The end-user should be able to view and monitor the current states of connected physical object. These may be the on or off state of two-state devices or the value of analog readings like temperature.

2.2.3 User Classes and Characteristics

As shown in the Context Diagram of the system (Figure 2.3) There are two main user classes of the system. One representing a home owner and the other, a resident.

- **Home Owner/Manager:** The home owner/manager could be any technological competent individual. Competence in this instance would mean that at the very least, the user should be able to operate applications on a smart phone with relative ease. In addition, the user may either be the owner of the home or be the person designated to manage the home. This would mean the user would have more permissions than a regular member of the home. Gender and educational level are not essential characteristics for this user class although at the very least the user should be able to read, in order to comfortably navigate and operate the application.
- **Home Resident:** This home resident should have demographics and characteristics very similar to the home owner/manager. The difference would arise in the levels of permissions given and the amount of control the user has over the home. Since the

resident has no managing responsibilities, he/she is expected to have fewer permissions than the home owner/manager.

2.2.4 Operating Environment

The different software modules of the system will run on different hardware platforms. The IoT Hub will run on a Raspberry Pi 2 Model B running the Raspbian Operating System because of its low cost, computing power and physical size. Other less ideal alternatives include be a PC, which is not as portable, or a mobile device which has a less powerful processor and fewer operating system options. The software to control and connect the Physical objects will run on the ESP8266 Wi-Fi module. And the End-user android application will run on a mobile phone running the Android operating system version 4.4 and upwards.

2.2.5 Design and Implementation Constraints

- The ESP8266 Wi-Fi module that would be used to give physical devices connectivity features has limitations on memory. For this project, the ESP8266s with 512KB flash memory were sourced. This allows firmware to be uploaded and run. However, in order for over-the-air updates (wireless) to firmware to be enabled, memory of double the program size (1MB) is required to hold the existing firmware and the replacement firmware. Thus, for this project, with the ESP8266s available, updates to firmware would have to be done using a serial connection.
- The ESP8266 is not designed to be 5V tolerant, thus all pins connected to the module must be stepped down to 3.3V. This would require either a 3.3V power supply unit or a more complex circuit design to do the conversion. A voltage beyond 3.3V will destroy the module.

- Communication using the MQTT protocol would openly transfer packets of data over the network connection. The system in granting access to devices over the internet requires that security measures are put in place to maintain the user's safety and privacy. Thus, communication may be encrypted at the expense of having a lightweight system because it increases the processing load. For this system, lightweight is a bigger priority than encryption,
- Another design consideration to make is the quality of service (QoS) to use for transferring packets over the network using the MQTT protocol. There are 3 levels Quality of Service with MQTT, which determine how reliable the broker or client handles messaging. "Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements" (Mosquitto, 2016). To meet the objective of being lightweight, the preferred QoS for the system is Level 1.

2.2.6 Assumptions and Dependencies

Using Wi-Fi as the communication channel for all devices in the system presents key assumptions and dependencies that must be met for the system to function as designed. These are listed below:

- A Wi-Fi network must be set-up where the system will be installed.
- The Wi-Fi network must have consistent access to the internet.
- All devices connected to the system must fully support the MQTT protocol.
- There is consistent electricity and back-up power supply in the case of a power cut for the system to work reliably, especially as a user may be remote.
- The user uses a mobile device running the Android operating system.

2.3 Specific Requirements

The functional requirements of the system have been organized according to the major product functions: Device Automation, Remote Device Control and Auxiliary Services.

2.3.1 Device Automation

This section describes the Device Automation requirement and provides use cases and a scenario. Detailed functional requirements are then listed and discussed.

2.3.1.1 Description and Priority

This feature allows the system to automate the control of devices based on some parameters. Its is a high priority feature, as the user would not be expected to manually control all the connected physical objects at all times. Automation is a primary objective of IoT. The automation of control will make living and management of the home more convenient. The use case diagram in Figure 2.4 describes a user's interactions with the system concerning this requirement.

2.3.1.2 Use Case: Device Automation

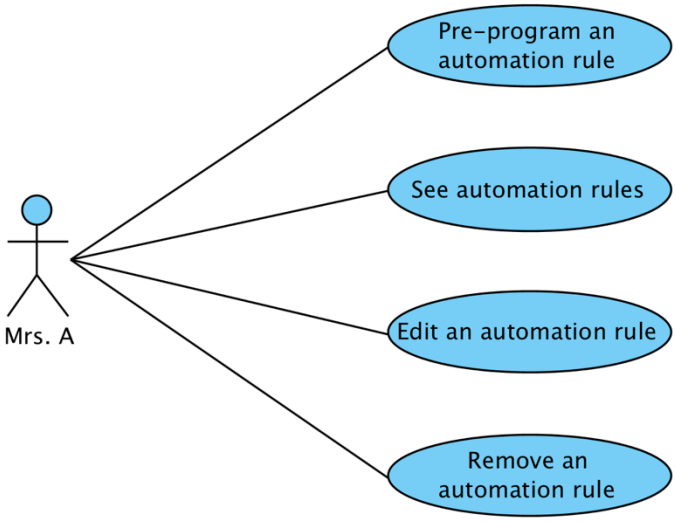

Figure 2.4: Use Case Diagram for Device Automation

2.3.1.3 Requirement Scenario: Device Automation

Mrs. A, a home owner acquires the system with connected devices and sensors. Being preoccupied with late meetings at the office, Mrs. A normally gets home very late in the evenings to a very dark compound. To deal with this problem, Mrs. A seeks to be able to program her outside security lights to come on once the sun has set. She sets up her connected light bulbs and connected light sensors and adds a new configuration to the system, to allow it control the lights as she desires

2.3.1.4 Functional Requirements

REQ-DA-1: A user should be able to add automation rules for control the connected devices

Table 2.1: Table of Requirements for REQ-DA-1

Description:	A home owner/manger would have an interface that allows him/her to create rules that the system would use to autonomously control the connected physical devices. An automation rule will be of the form: IF [SENSOR] HAS [SENSOR_READING] READING THEN CHANGE [PHYSICAL DEVICE] TO [TARGET STATE]
Inputs:	Select Sensor (e.g. daylight sensor)

	Trigger Sensor Reading (e.g. bright) Select Physical Device (e.g. front door light bulb) Target State of Physical Device (e.g. off)
Source:	The user would select the sensor and its trigger reading as well as the connected physical device and the state it should be changed to
Outputs:	The user received feedback on the success of the addition of the new rule
Destination:	The feedback will be viewed in the user's interface The rule will be delivered to the automation engine that deals with the autonomously controlling the connected devices
Action:	Using a diagram, select a combination of inputs would create a rule that would control the devices in the desired manner
Pre-condition:	Physical devices and sensors need to be already connected to the system
Post-condition:	The devices would be controlled with the rules put in place
Side effects:	Rules may conflict so the system would have to find a conflict resolution mechanism

REQ-DA-2: A user should be able to view all the automation rules setup in the system

Table 2.2: Table of Requirements for REQ-DA-2

Description	Once a rule has been configured, a should be able to view a list of all the rules that have been setup in the system
Inputs	None
Source	N/A
Outputs	The user views a list of the all the rules in the system
Destination	The listed rules will be viewed in the user's interface
Action	At the click or press of a button, a user should be able to request for a list of all the rules stored in the system
Pre-condition	Automation rules need to be already setup in the system
Post-condition	None
Side effects	None

REQ-DA-3: A user should be able to update automation rules for control the connected devices

Table 2.3: Table of Requirements for REQ-DA-3

Description	Once a rule has been configured, a should be able to conveniently modify the existing rules at will in any form possible
Inputs	The rule to be updated At least one of the following inputs is needed to modify the rule: Select Sensor (e.g. daylight sensor)

	Trigger Sensor Reading (e.g. bright) Select Physical Device (e.g. front door light bulb) Target State of Physical Device (e.g. off)
Source	The user would decide what changes should be made to the rule
Outputs	The user receives feedback on the success of the change
Destination	The feedback is sent to the user's interface The updated rule is delivered to the automation engine for the changes to take effect in the real world
Action	A user should be able to view a rule and modify any of the parts of the rule to suit any changes that need to be made in the real world
Pre-condition	The user needs to be able to view the rules that are setup in the system
Post-condition	The devices would be controlled with the updated rules put in place
Side effects	The updated rule may conflict with another rule so there will need to be tie breakers

REQ-DA-4: A user should be able to remove automation rules for control the connected devices.

Table 2.4: Table of Requirements for REQ-DA-4

Description	Once a rule is no longer needed to be enforced in the system, a user should be able to remove it, so that it has no effect on the connected physical devices
Inputs	The rule to be removed
Source	The user decides which rule no longer needs to be enforced
Outputs	Feedback on the success of the deletion
Destination	The feedback is viewed on the user's interface
Action	A user may at the click or press of a button, delete a rule from the set of rules in the system
Pre-condition	The rule needs to be in the system The user needs to be able to view the rule prior to deletion
Post-condition	The automation engine is updated to no longer enforce the deleted rule
Side effects	None

REQ-DA-5: The system should be able to autonomously control the connected devices using the programmed rules.

Table 2.5: Table of Requirements for REQ-DA-5

Description	The system would run an automation engine that monitors the rules and the parameters given in the rules and trigger an

	action when the parameter is met. The action triggered will be the change in the state of a connected physical device
Inputs	The automation rules in the system
Source	The system's database
Outputs	A change in the state of affected physical devices A notification sent to the user to keep him/her aware
Destination	The change in state is in the real world with physical devices The notification may be viewed on the user's interface
Action	The system will for each rule given, monitor the listed sensors. If a sensor reading matched the specified trigger reading, the corresponding connected physical device is controlled in accordance with the given rule
Pre-condition	The rule needs to be in the system The physical devices and sensors need to be connected to the system
Post-condition	The physical devices should change state in the real world
Side effects	None

2.3.2 Remote Device Control

This section describes the Remote Device Control requirement and provides use cases and a scenario. Detailed functional requirements are then listed and discussed.

2.3.2.1 Description and Priority

This feature allows the end-user to control all the physical objects connected to the system. It is of high priority as it is the only feature that is centered around the end-user. A user should be able to view the available options and select one, which will have an effect in the real world. This feature overrides automated control in Subsection 3.1, Device Automation. The use case diagram in Figure 2.5 describes a user's interactions with the system concerning this requirement.

2.3.2.2 Use Case: Remote Device Control

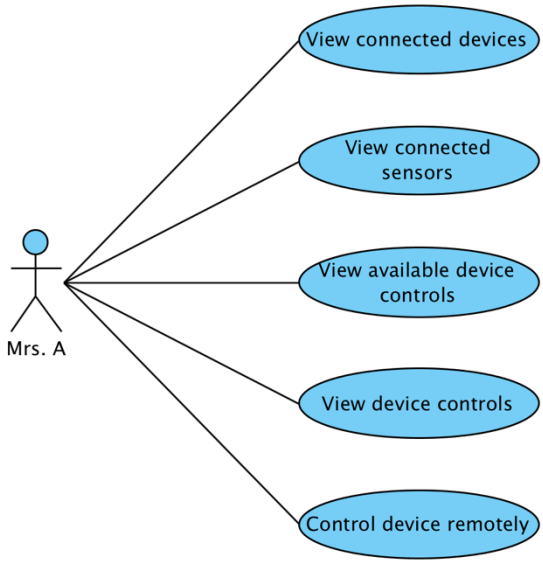

Figure 2.5: Use Case Diagram for Remote Device Control

2.3.2.3 Requirement Scenario: Remote Device Control

Mrs. A, a home owner with 3 children closes from work an hour after her children get home from school. She arranges for a driver to pick them off from school. When they get home, she wants to be able to unlock the front door so they can enter the house because they do not have keys. She installs a connected door lock along with the system and from her office, using her mobile phone, she can see if the door is locked or not, and can unlock the door for the children to enter. Throughout the day, when nobody is home, she can know if the door is locked and can be notified if the anyone unlocks it.

2.3.2.4 Functional Requirements

REQ-RDC-1: The user should be able to view all devices connected to the system

Table 2.6: Table of Requirements for REQ-RDC-1

Description	A user should be able to very easily view a list of all the devices in the system from anywhere in the world, at anytime
Inputs	None
Source	N/A
Outputs	A list of connected devices in the system
Destination	The list can be viewed on the user's interface

Action	The user can at the click or press of a button list all the connected devices in the system
Pre-condition	There needs to be connected devices in the system
Post-condition	None
Side effects	None

REQ-RDC-2: The user should be able to view each connected device's current status

Table 2.7: Table of Requirements for REQ-RDC-2

Description	The user should be able to view the current state of a connected device, whether it is on, off or whatever states a device is configured to have
Inputs	The device whose current state the user wants to view
Source	The user selects from a list in the the user's interface
Outputs	The current state of the connected device
Destination	The state will be displayed in the user interface
Action	A user at the click or press of a list item, may be able to request the current state a device
Pre-condition	The physical device needs to be connected
Post-condition	None
Side effects	None

REQ-RDC-3: The user should be able to view each connected device's set of related sensors

Table 2.8: Table of Requirements for REQ-RDC-3

Description	The user should be able to view the set of sensors which are directly related to a connected device
Inputs	The device whose current state the user wants to view
Source	The user selects from a list in the the user's interface
Outputs	A list of device-related sensors in the system
Destination	The sensors will be displayed in the user interface
Action	A user at the click or press of a list item, may be able to request the device's related sensors
Pre-condition	The physical device needs to be connected The associated sensors need to be connected
Post-condition	None
Side effects	None

REQ-RDC-4: The user should be able to view the controls available for each connected device

Table 2.9: Table of Requirements for REQ-RDC-4

Description	The user should be able to view the set of control options to change the state of a device
Inputs	The device whose controls the user wants to view
Source	The user selects from a list in the the user's interface
Outputs	The set of control options available for the specified device
Destination	The set of control options will be displayed in the user interface
Action	A user at the click or press of a list item, may be able to request for the device's available controls
Pre-condition	The physical device needs to be connected
Post-condition	None
Side effects	None

REQ-RDC-5: The user should be able to view all the sensors connected to the system

Table 2.10: Table of Requirements for REQ-RDC-5

Description	A user should be able to very easily view a list of all the sensors in the system from anywhere in the world, at anytime along with their current readings
Inputs	None
Source	N/A
Outputs	A list of connected sensors in the system and their current readings
Destination	The list can be viewed on the user's interface
Action	The user can at the click or press of a button list all the connected sensors in the system
Pre-condition	There needs to be connected sensors in the system
Post-condition	None
Side effects	None

REQ-RDC-6: The user should be able to control each device and change its state in the real world

Table 2.11: Table of Requirements for REQ-RDC-6

Description	The user should be able to override automation rules and at anytime and from anywhere, be able to change the state of any of the connected physical devices in the system
Inputs	The device to be controlled The desired control to change the device's state to the target state
Source	The user selects the device to be controlled from a list in the interface The user selects the controls from the interface

Outputs	A change in the state of affected physical devices Feedback on the success in changing the device's state
Destination	The change in state will occur in the real, physical world The feedback will be received in the user's interface
Action	A user may view a connected device along with it's related sensor readings and available controls. If the physical conditions in the home (got from the sensors) warrant a change in the device's state, the user may use any of the control options available to effect this change
Pre-condition	The user should be able to view connected devices, their current states and their current related sensor readings The user should be able to view the controls available for the selected device
Post-condition	The device should change state in the real world to the desired state
Side effects	The control may override conflicting automation rules. If a rule is being implemented whiles a user controls a device, the user's control will be used

2.3.3 Auxiliary Services

This section describes the requirement for Auxiliary Services and provides use cases and a scenario. Detailed functional requirements are then listed and discussed.

2.3.3.1 Description and Priority

This feature comprises all the services provided to support the main features (device automation and remote control). This includes customization features to suit the user's preferences and other features that enhance the usability of the system. Such services include adding new devices, accessing a log of past transactions and changing settings (e.g. network access point configurations). The use case diagram in Figure 2.6 describes a user's interactions with the system concerning this requirement.

2.3.3.2 Use Case: Auxiliary Services

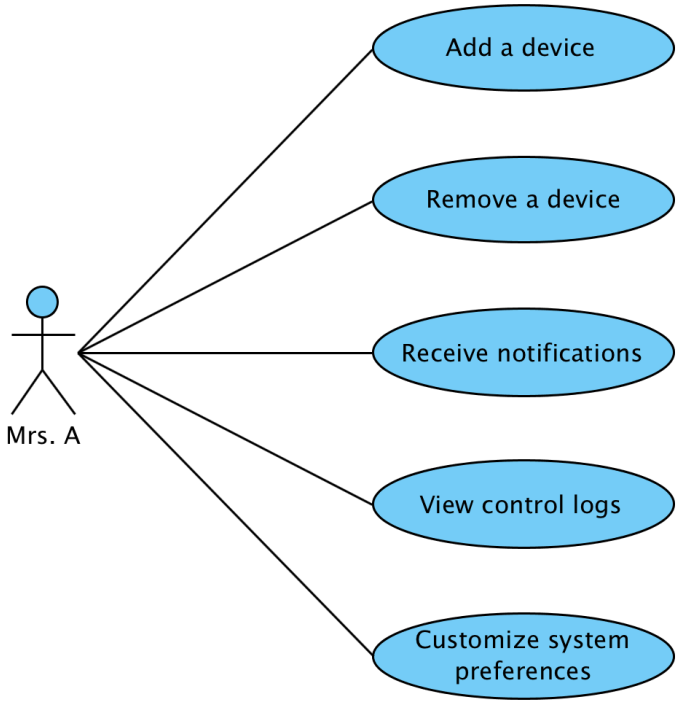

Figure 2.6: Use Case Diagram for Auxiliary Services

2.3.3.3 Functional Requirements

REQ-AS-1: The user should be able to add a device to the system

Table 2.12: Table of Requirements for REQ-AS-1

Description	A home owner/manager should be able to add a new device to the system and configure it accordingly
Inputs	Device ID Device Name Location Available Controls Set of Related Sensors
Source	The user doing the configuration would decide on these inputs
Outputs	Notification message on the success of the addition of a new device
Destination	The user will receive the notification on the user's interface
Action	A user should be able to conveniently add a device to the system and specify its name, location, available controls and related sensors through an intuitive interface. The user should be able to connect the physical device to the IoT Hub.
Pre-condition	The system is operational. The device is connected to the network.
Post-condition	The new device would be operable either via remote control or using the automation rules

Side effects	None

REQ-AS-2: The user should be able to remove a device from the system

Table 2.13: Table of Requirements for REQ-AS-2

Description	A home owner/manager may decide he/she no longer wants a particular device connected to the system. The user should be able to conveniently remove the device and restrict access to it through the system
Inputs	The device to be removed
Source	The user decides which device to remove from the system
Outputs	Feedback on the success of the deletion
Destination	The feedback may be viewed on the user's interface
Action	The home owner/manager should be able to view devices connected to the system and conveniently remove any of them at will. These connected device will no longer be functional within the context of the system
Pre-condition	The device must be connected to the system to be able to be removed
Post-condition	The device will no longer be involved in automation and controllable via the mobile app
Side effects	None

REQ-AS-3: The user should be notified whenever major changes are made to the system

Table 2.14: Table of Requirements for REQ-AS-3

Description	The Internet of Things enables physical devices to get connected and be accessible from anywhere in the world with an internet connection. Thus once the devices in the home are connected, they can be operated remotely. A home owner/manager would want to be kept updated on what happens in the home, irrespective of where he/she is
Inputs	Device Control Event
Source	System generated based on the occurrence
Outputs	Notification Message
Destination	SMS, Push Notification or Email
Action	The notification engine is able to connect to services that allow the sending of SMSs, Mobile Push Notifications and Emails from within the system whenever a notable change occurs. The user is thus promptly alerted
Pre-condition	A change occurs in the system
Post-condition	None
Side effects	None

REQ-AS-4: The user should be able to view a log containing a history of device control

Table 2.15: Table of Requirements for REQ-AS-4

Description	The system should keep a trace of all control transactions for auditing purposes. A user may want to monitor usage of the system over time as well as see what device was controlled, the timestamp of the control and the user that authorized the control transaction
Inputs	The time period for which transactions should be shown
Source	The user would specify the time period
Outputs	A list of transactions showing a date-time stamp, the authorizing-user, the control transaction, device information and sensor information
Destination	The transaction log will be viewed on the user's interface
Action	The user would have an interface that him/her to request the logs and filter them by date and time
Pre-condition	There would have to be transactions that have already occurred in the system
Post-condition	None
Side effects	None

REQ-AS-5: The user should be able to customize system preferences and notable network configurations

Table 2.16: Table of Requirements for REQ-AS-5

Description	The system would work with a local network and among other things it is highly crucial that when the local network details change, the system can very easily be reconfigured to work with new SSIDs, passwords, etc. Also the user should be able to modify other configurations to suit the user's preferences for the system
Inputs	The details to be modified changed. (e.g. Network SSID, Keys)
Source	The user would provide these replacement details
Outputs	Feedback on the success of the customization operation
Destination	The feedback can be viewed on the user's interface
Action	The user interface would provide forms that will allow users input changes and submit them to reflect on the system
Pre-condition	A change occurs in the network, or elsewhere in the physical environment that needs to be reflected on the system
Post-condition	The system now works with the new configuration and the old configuration details are no longer functional
Side effects	None

2.4 External Interface Requirements

This section discusses the system's interfaces. These comprise user interfaces, hardware and software interfaces, as well as communication interfaces.

2.4.1 User Interfaces

Different user interfaces would be needed for the different software components of the system. The three components with user interfaces are the mobile application, the IoT Hub (Broker) and the firmware that would operate each of the physical objects.

- **Mobile Application Interface:** Upon opening the application, an already existing user is shown a form to enter authentication details for access to the system. A new user may register to configure the system. The user would enter preferred login details and the IoT Hub's configuration details. Allowing a user to enter the Hub's configuration details allows the user to use the application with more than one hub by specifying the appropriate hostname and port number. Once the user has access to the application, the user may either view the connected physical objects and their respective available controls. As well as related sensor readings. The user may also view all sensor readings. And finally the user may adjust configurations and settings. A tabbed view may be used to make all these views easily accessible.
- **IoT Hub (Broker):** The user may access a Home Owner/Manager web application to exercise administrative rights and make changes to the Hub's configuration. Such changes may include system password changes, granting and revoking access to users, etc.
- **Firmware on Physical Objects:** Each physical device is configured to work with the setup IoT system. If any changes are made to the system's configuration, the updates need to be reflected on the physical devices. This process needs to be done

in a user-friendly and convenient manner. A simple web application would be developed to aid this. A user will be given a form to update network settings, authentication details, etc.

2.4.2 Hardware Interfaces

The IoT Hub (Broker) and mobile applications would run on any Linux operating computer and any android enabled mobile phone respectively. Thus, these two software components do not have designated hardware interfaces. These however would need to be connected to one another through a Wi-Fi (802.11) network. Wireless was chosen over cable because of the convenience of setup. Also, Wi-Fi was conveniently chosen over other Wireless networks (e.g. Bluetooth) because of its speed, transmission range and availability on several devices. Hardware network interfaces needed for these components include a wireless access point, etc.

The physical objects would need to be connected to the network as well. The hardware interface for this connection is the ESP-8266 (ESP-01) Wi-Fi module. This would connect to the wireless access point. The ESP-8266 module has an on-board processor for controlling the physical object.

The communication over the network between the components will primarily be done using the MQTT (MQ Telemetry Transport) protocol, “a machine-to-machine (M2M)/IoT connectivity protocol” (MQTT, 2016).

2.4.3 Software Interfaces

The system’s software would interface with a backend database using a REST Application Programming Interface (API). The REST API would give applications within the system privileges to access the database as well as save and modify records. The API will return data from the database (e.g. device control history) in JSON format, or Result

Codes for other functions that do not access data. The software will interface with an SMS service provider's API (SMSGH) to be able to send SMSs for notification. This interfacing will transfer the SMS service keys for authentication as well as the messages to be delivered. The mobile application would interface with Amazon AWS Simple Notifications Service (SNS) for the delivery of push notifications to the user. The interface with Amazon SNS will also transfer the AWS keys for authentication as well as the contents of the notification message.

2.4.4 Communications Interfaces

The nature and essence of the system being developed makes it a very communication heavy system. The different system modules (devices, hub, end-user application) would communicate with one another. It is expected that messages sent would be delivered within a specified acceptable time period. It is also expected that messages are delivered correctly. Messages to change the state of a device in the physical world in particular need to be reliably delivered accurately because they have security and safety implications. All communications would have to be secure and only accessible with authorization.

The following protocols will be used throughout the system:

- Hypertext Transfer Protocol (HTTP)
- MQTT
- Websockets

2.5 Nonfunctional Requirements

The non-functional requirements for the system have been classified into 5 main groupings and explained below:

2.5.1 Performance requirements

The system should be lightweight in nature. Lightweight processing and communications are necessary in order for the system to scale efficiently. The Internet of System deals with an ecosystem of a large number of devices. Thus, adding more devices should not take a toll on the system.

2.5.2 Standards compliance

The system should comply with open standards for the Internet of Things to maintain interoperability with a wide range of physical devices, sensors, etc. In order for the system to be relevant as an Internet of Things solution, interoperability is key, given the vast array of devices.

2.5.3 Reliability

All communication between devices, the hub or the end-user application meant to control a device and change its state should be done with reliability. In the system, devices may be controlled autonomously or remotely. In order for the user to trust that the system works, all control instructions must be delivered with the highest reliability. However, communications publishing the the state of devices or sensor readings may have reduced reliability because of the continuous publishing of these states. Because the state of a device will be resent in after an interval, such messages do not need to be delivered with the highest reliability.

2.5.4 Availability

A user may control the devices in the home from anywhere in the world at anytime over the internet. Thus, the system should always be available. And should be able to

recover on its own for some foreseeable problems (e.g. power outage). The system should inform the user whenever there's downtime in the local environment and when the problem is resolved.

2.5.5 Security

The system grants a user access to devices in the home over the internet. This means that only authorized persons are granted this access over the internet. The system should be secure and prone to snooping. Different classes of users should only have access to aspects of the system for which they have permission.

3. Chapter 3: Architecture and Design

3.1 Introduction to Architecture and Design

This section on architecture and design introduces the chapter with the purpose and a brief summary.

3.1.1 Purpose:

This chapter defines how the system described in the requirements section is to be implemented. This design gives an overarching view of the system architecture to enable a developer build the system according to specification. The document will include a system overview, detailing the context. It will also include detailed Data Design as well as a snapshot and explanation of the human user interfaces.

3.1.2 Summary and overview of chapter:

The chapter will detail the specifics of the system architecture using a deployment diagram, sequence diagrams, and user interfaces. The document will also discuss design considerations and briefly describe alternative architectures that were thought of in the designing of this system.

3.2 Design Considerations

The subsequent sections discuss the system architecture, and discusses alternative architectural designs.

3.2.1 System Architecture

Figure 3.1 is a block diagram showing major subsystems and the interconnections among them. The architectural diagram shows the different software components for example, the MQTT Broker and Automation Engine. The diagram shows which hardware component the software would be deployed to. In addition the diagram shows the interfaces between the components. These interfaces include various APIs for connected services as well as communication protocols. The outermost box at the top represents the host, where the various engines and services will run. The hardware component for the host is a Raspberry Pi and will communicate over Wi-Fi using MQTT.

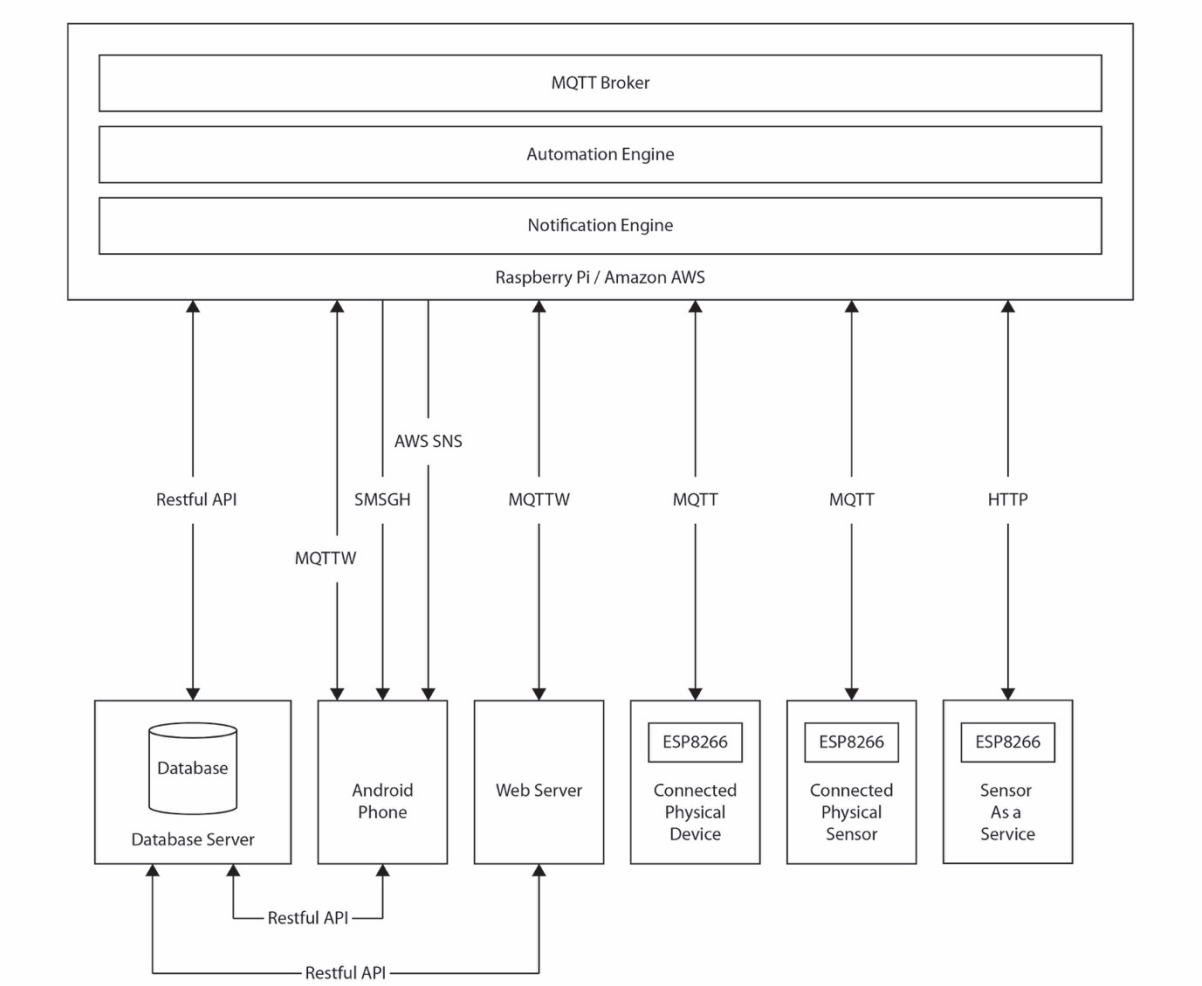

Figure 3.1: A Deployment Block Diagram Showing the Subsystems of the System

3.2.2 Component Decomposition: Sequence Diagram.

Figure 3.2 shows a sequence diagram describing the interactions between a home owner/manager and the different software components of the system to create, view and edit automation rules. The sequence diagram shows the requests that will be made to the backend API and the respective database query results.

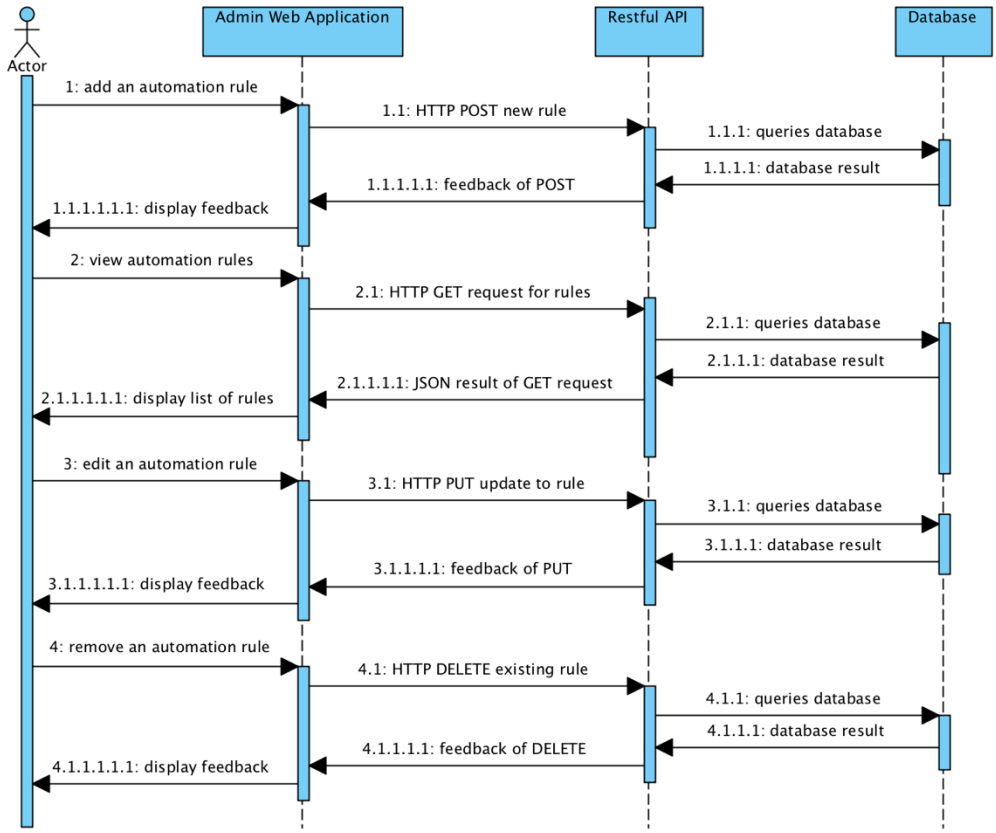

Figure 3.2: A Sequence Diagram for Device Automation

The sequence diagram in Figure 3.3 describes an end user's interactions with the software components to successfully control a connected object remotely. The user interacts with an Android application which connects with a RESTful API to make database calls. The user may also use the application to directly interact with the MQTT Broker to control devices and view their current states.

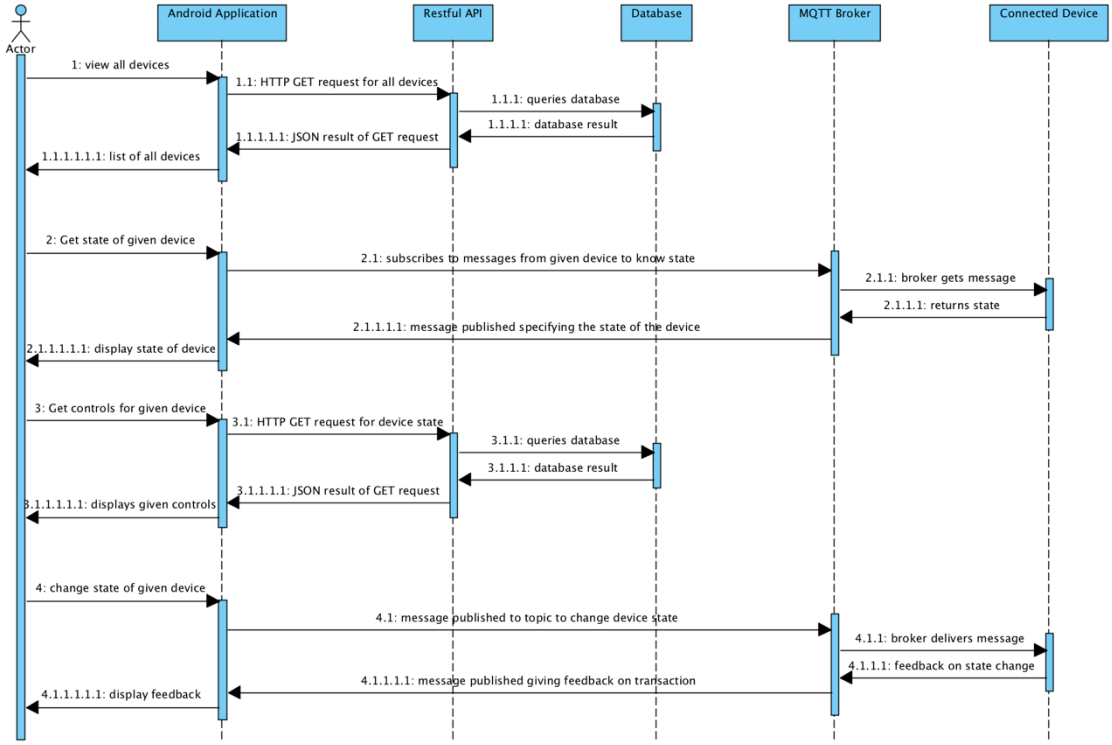

Figure 3.3: A Sequence Diagram for Remote Device Control

Figure 3.4 is a sequence diagram for a user's interactions with the software components while using the system's auxiliary services. The diagram shows the actions a user takes and shows the messages that are sent back to the user in response. A user can add or remove a device as well as customize the system's preferences. For all the interactions, the user receives feedback from the system, viewed on the user interface.

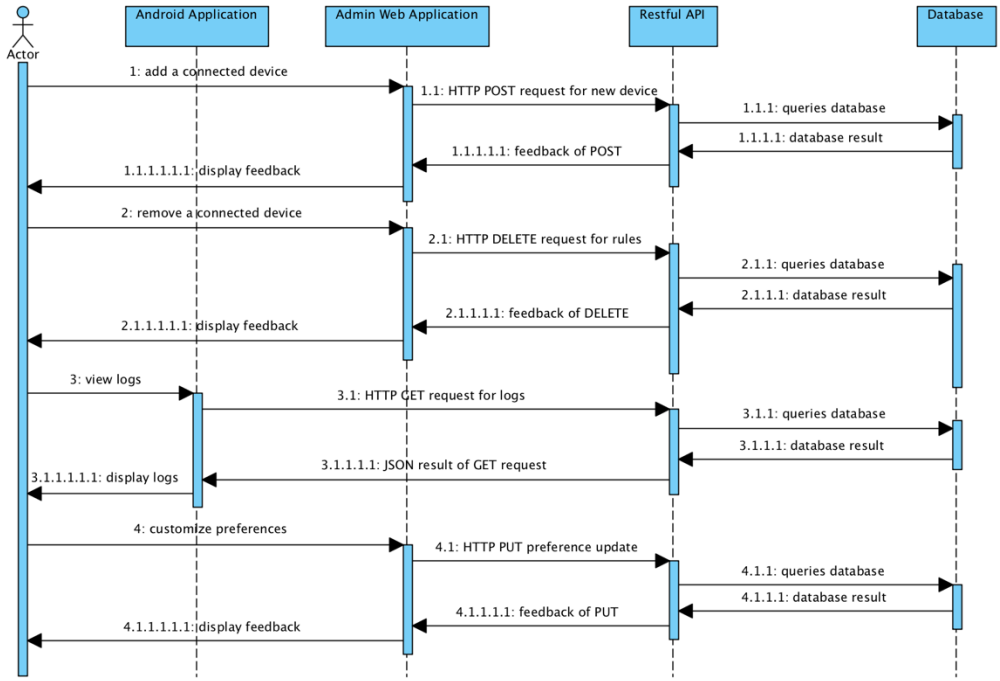

Figure 3.4: A Sequence Diagram for Auxiliary Services

3.2.3 Architectural Alternatives

Figure 3.5 Shows an alternative architecture that was created in designing the system. The design shows the hardware components (Raspberry Pi and Connected Object). The design also specifies MQTT as the protocol for communicating between components.

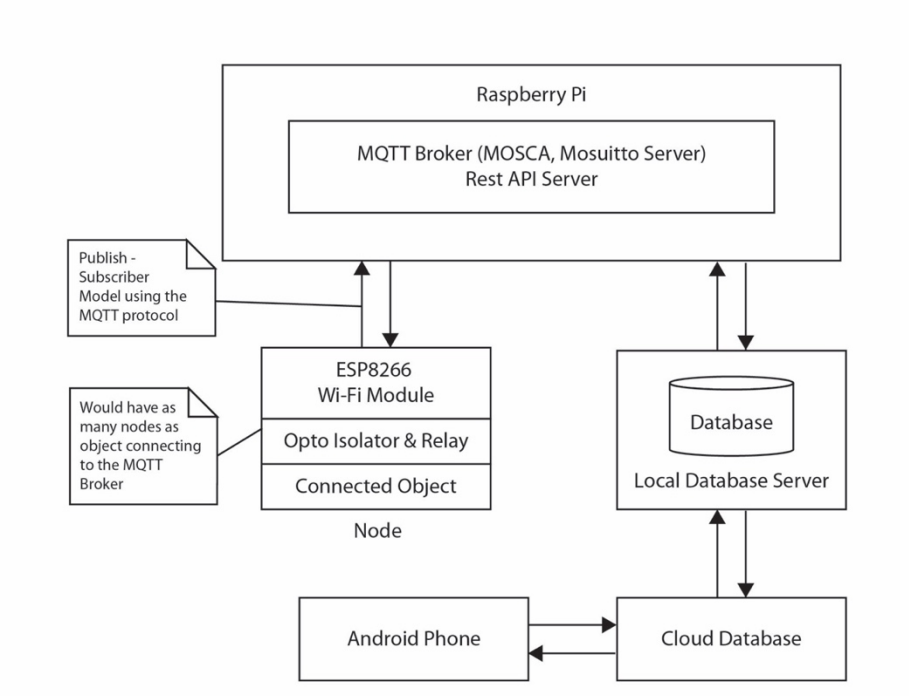

Figure 3.5: An Alternative Architectural Design for the System

3.2.4 Design Rationale

The difference with this system (Figure 3.5) is that the android application does not directly communicate with the MQTT broker. Instead it communicates with a database. Having to go though a database would make the application unresponsive to real-time changes to the states of connected devices in the physical devices. Also the design uses two databases. Maintaining synchronized databases places an unnecessary processing overhead on the system. This impedes the goal for the IoT system to be lightweight. Thus the architectural design was further developed into the architecture in the design in Figure 3.6 which extends the MQTT protocol to the Android phone and the cloud. Figure 3.6 is an update on Figure 3.1.

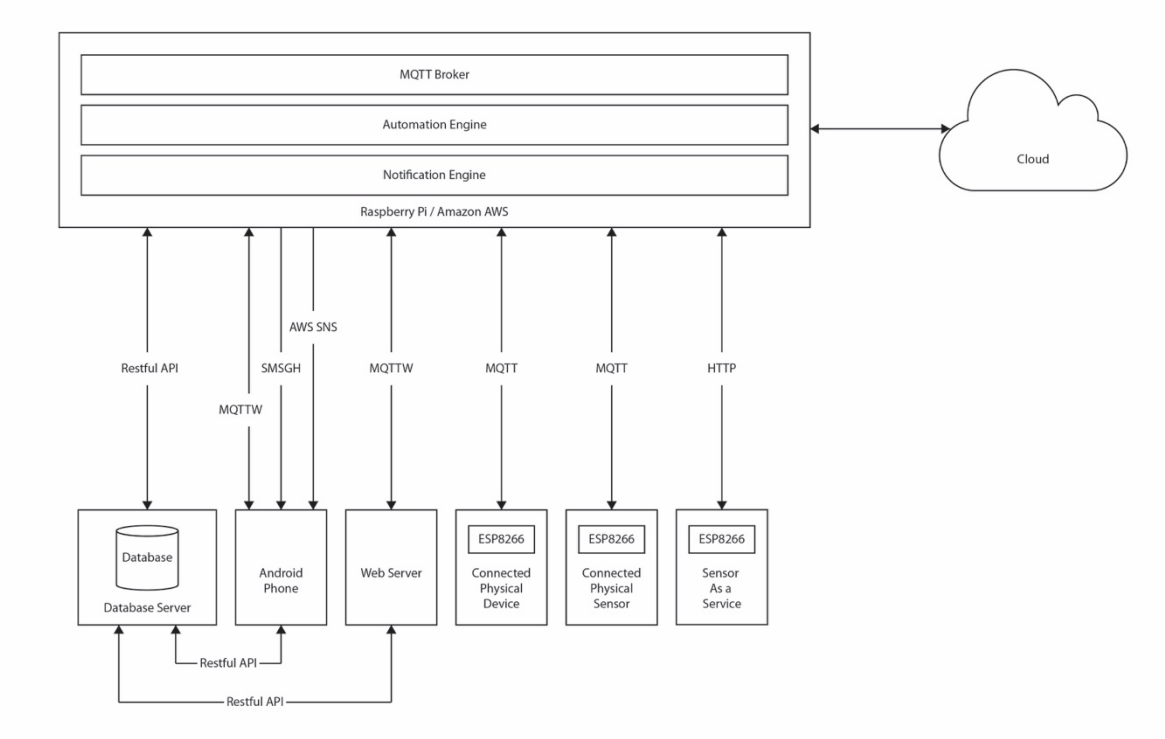

Figure 3.6: Selected Architecture with Cloud

3.3 Data Design

This section describes how the data in the system is stored and organized.

3.3.1 Data Description

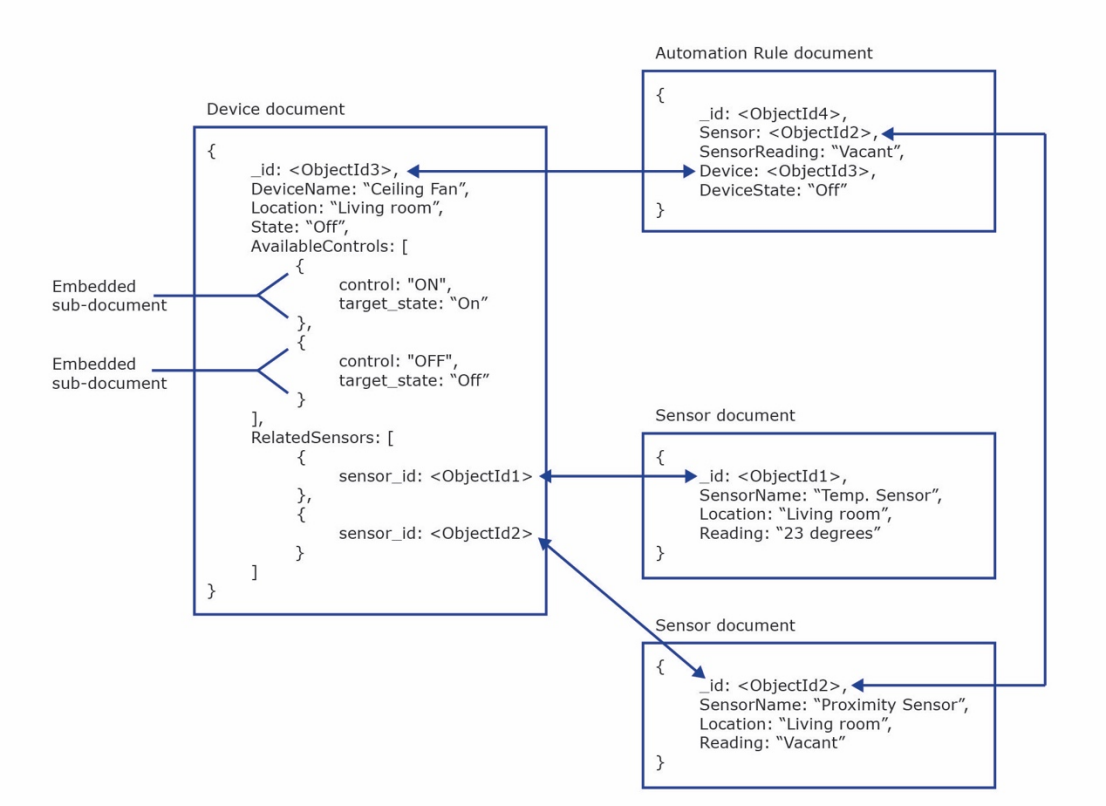

Figure 3.7: A Data Model Diagram for the System

The system uses a non-relational schema-less document database. Because of the schema-less nature, each record does not have to strictly follow the data model described in Figure 3.7. The model is a guide for all records. Document databases store records as key-value pairs. Thus, as the IoT system scales, there will be a lower overhead on database query operations. Given that the document database is non-relational, for this system embedded sub-documents are used to model relations. The data model is further documented in the data dictionary below (Table 3.1), showing data types and descriptions.

3.3.2 Data Dictionary

Table 3.1: Table of Database Fields and Descriptions

Document Name	Field Name	Data Type	Allow Nulls	Field Description
Sensor	_ID	Object Id	No	This field uniquely identifies a connected sensor

Document Name	Field Name	Data Type	Allow Nulls	Field Description
	Sensor Name	String	No	This field identifies a connected sensor by a recognizable name
	Location	String	Yes	This field describes where the connected sensor is installed
	Reading	String	Yes	This filed contains the current reading of the connected sensor
Device	_ID	Object Id	No	This field uniquely identifies a connected physical device
	Device Name	String	No	This field identifies a connected physical device by a recognizable name
	Location	String		This field describes where the connected physical device is installed
	State	String	No	This filed contains the current state of the connected physical device
	Availa-ble Controls	Array	No	This field contains an array of controls available to the connected physical device
	Related Sensors	Array	Yes	This field contains an array of Sensors IDs representing the set of sensors tied to a given connected physical device
Automa-tion Rule	_ID	Object Id	No	This field uniquely identifies an automation rule
	Sensor	String	No	This field contains a reference to the sensor that will trigger the action in the rule
	Sensor Reading	String	No	This field contains the sensor reading that will trigger the action in the rule
	Device	String	No	This field contains a reference to the device that will be automatically controlled once the trigger occurs
	Device State	String	No	This field contains the state that the physical device should be changed to

3.4 Human Interface Design (Screens)

This section discusses the screens of the mobile app and the design considerations that were made.

3.4.1 Overview of the User Interface

The interface of the mobile application will primarily allow a user to see the connected devices and their related sensors. The application should also allow the user to

very easily monitor the controls of the objects and override the automation controls if they desire. The user would on opening the application see a selectable list of objects. On selecting the object from the list the user can then view the details and controls specific to the object. From the home view, the user can select from the navigation, options to view the list of sensors and their readings, as well as the log of past controls across all devices.

Two design alternatives were created and evaluated. In the screens shown in section 3.4.2, there are two options for the main navigation view. Option A (Figure 3.8) uses a tabbed layout and option B (Figure 3.9) uses a navigation drawer layout. Figures *Figure 3.8* and *Figure 3.9* show alternative ways to navigate the menu options.

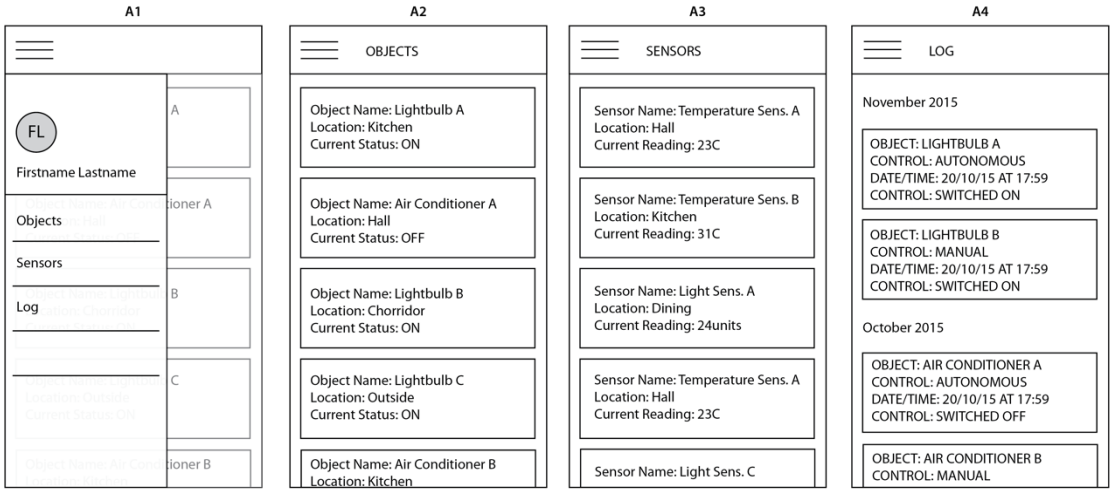

Figure 3.8: User Interface Design (Option A)

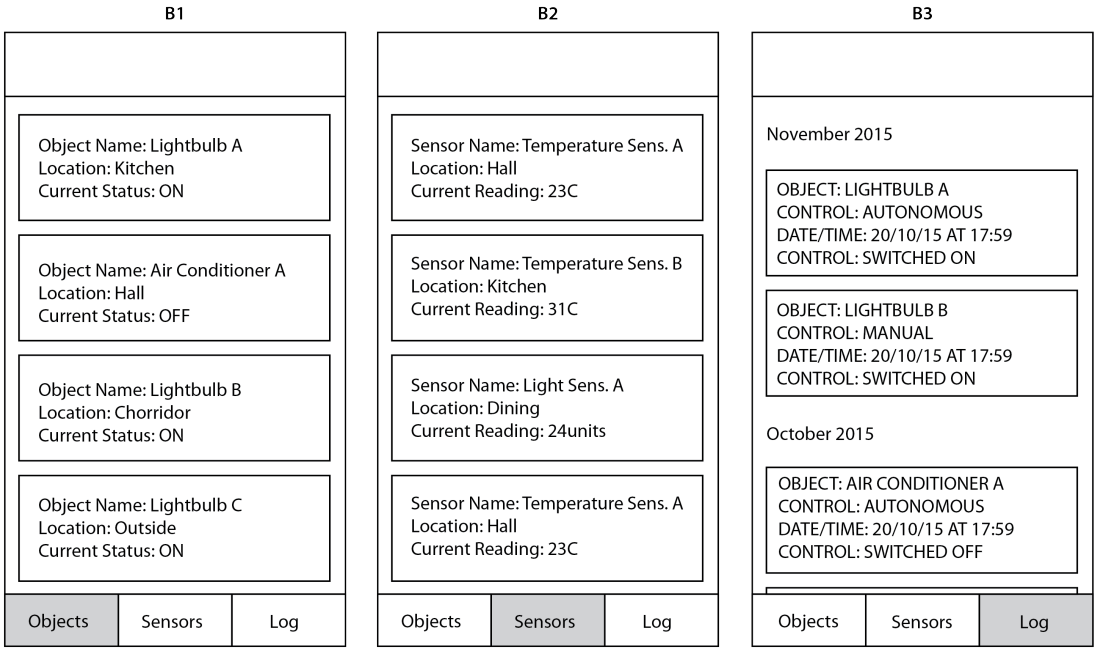

Figure 3.9: User Interface Design (Option B)

On selecting a device, there are three options C (Figure 3.10), D (Figure 3.11) and E (Figure 3.12). C displays all the objects details on one screen. D splits the views for the controls, related sensor readings and logs across three tabs and option E is a hybrid of C and D which merges the sensor and control tabs in one. E was developed because viewing sensors whilst adjusting controls avoids continuous switching between tabs when controlling an object.

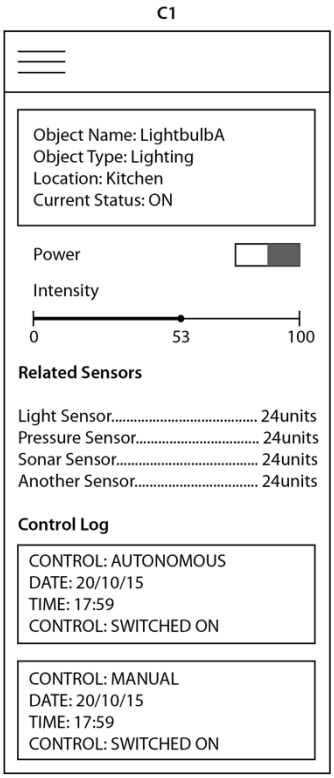

Figure 3.10: User Interface Design (Option C)

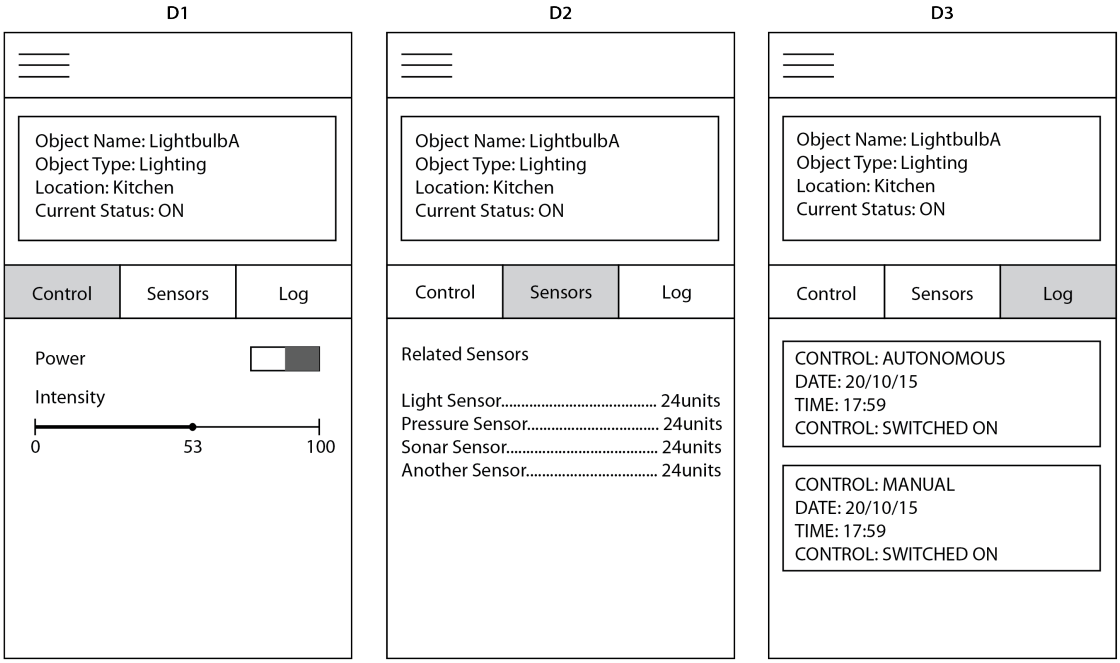

Figure 3.11: User Interface Design (Option D)

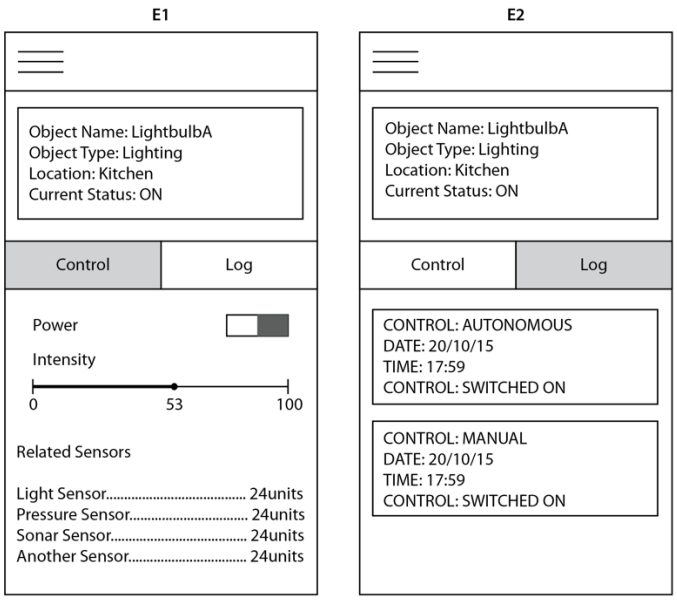

Figure 3.12: User Interface Design (Option E)

3.4.2 Screen Images, Objects and Actions

- Option A (Figure 3.8): This is one of two options for the landing view of the application. The first screen is the list of connected objects with identifiable details. This option uses the navigation drawer layout. Clicking the Objects, Sensor and Log menu items in A1 takes the user to A2, A3 and A4 respectively.
- Option B (Figure 3.9): This option, using the tab layout is the second of two options for the landing view of the application. Clicking the tabs at the bottom take the user to the respective views with the shaded menu items.
- Once an object is selected from the list, options C to E depict three interface options to view the object details and control it.
- Option C (Figure 3.10): Option C uses a single page view, where all the information is displayed on one screen

- Option D (Figure 3.11): Option D uses a tabbed view to display the object details. The tabs control, sensors and logs show the available controls, related sensors and the control log respectively.
- Option E (Figure 3.12): Option E is a hybrid of Options C and E. E still uses a tabbed view but combines the control and sensor views into one tab. With this view, whilst controlling devices a user can monitor the related sensor readings.

Hardware Design

Most physical devices are not manufactured with network connectivity. Therefore, for this system to work with a wider range of physical devices, a Wi-Fi module was connected to them. Figure 3.13 shows a schematic diagram for transforming disconnected devices into connected physical devices.

The following considerations were made in designing and implementing this hardware component of the system:

- The 3.3V Regulator is used to step down the input voltage from the power supply to 3.3V which the ESP8266 Wi-Fi module requires.
- A transistor was used with a relay as a switch to use the small current from the ESP8266 microcontroller to control devices that require a larger current. And to prevent the high current from affecting the ESP8266.

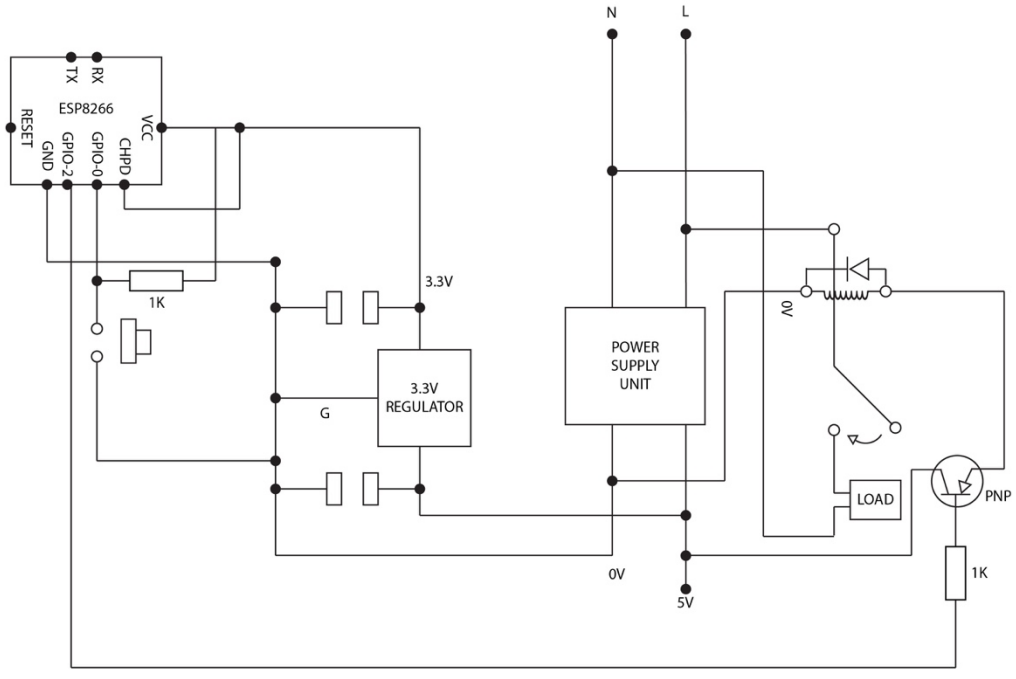

Figure 3.13: Schematic Diagram for Connecting Physical Devices to ESP8266

3.5 Traceability Requirement Matrix:

The matrix below in Table 3.2 maps requirements for the system to components. This demonstrates how the design in this section addresses the requirements. A dot in the table signifies that the requirement was in part fulfilled using the corresponding system component.

System Components:

1. MQTT Broker (MB)
2. Automation Engine (AE)
3. Notification Engine (NE)
4. Raspberry Pi (RPi)
5. Restful API (RAPI)
6. Amazon AWS Simple Notifications Service (SNS)
7. SMS GH (SMS)
8. Database Server (DB)
9. Web Server (WS)
10. Android Application (MOB)
11. Connected Devices (CD)
12. Connected Sensors (CS)

Table 3.2: Matrix Mapping Requirements to System Components

Requirement Code	**MB**	**AE**	**NE**	**RPi**	**RAPI**	**SNS**	**SMS**	**DB**	**WS**	**MOB**	**CD**	**CS**
REQ-DA-1		•			•		•	•	•		•	•
REQ-DA-2		•			•		•	•	•		•	•
REQ-DA-3		•			•		•	•	•		•	•
REQ-DA-4		•			•		•	•	•		•	•
REQ-DA-5	•	•	•	•	•	•	•	•			•	•

Requirement Code	MB	AE	NE	RPi	RAPI	SNS	SMS	DB	WS	MOB	CD	CS
REQ-RDC-1	•			•	•			•		•	•	•
REQ-RDC-2	•			•	•			•		•	•	•
REQ-RDC-3	•			•	•			•		•	•	•
REQ-RDC-4	•			•	•			•		•	•	•
REQ-RDC-5	•			•	•			•		•	•	•
REQ-RDC-6	•			•	•			•		•	•	•
REQ-AS-1	•		•	•	•	•	•	•	•		•	•
REQ-AS-2	•		•	•	•	•	•	•	•		•	•
REQ-AS-3	•		•			•	•					
REQ-AS-4					•			•		•		
REQ-AS-5	•			•	•			•	•			

4. Chapter 4: Implementation

4.1 Tools, Libraries and Frameworks Employed

- **Raspberry Pi 2 Model B:** The Raspberry Pi is a portable low cost computer largely used for educational and prototyping purposes. It is able to run a desktop operating system and has 4 USB ports, an Ethernet (802.3) port and an HDMI port. The Raspberry Pi 2 has 40 programmable General Purpose Input Output (GPIO) pins. Considering costs, computing power, network capabilities and physical size, the Raspberry Pi makes an ideal dedicated MQTT server in the home environment. Since the Raspberry Pi 2 Model B does not have inbuilt Wi-Fi (802.11) capabilities, a USB Wi-Fi dongle will be used to supplement its functionality.
- **ESP8266:** The ESP8266 is a very low cost microcontroller with Wi-Fi (802.11) capabilities and the full TCP/IP stack. For this project, the ESP is configured to run Arduino code and is connected in a circuit to control and give existing physical devices Wi-Fi capabilities. The ESP8266s used in this project are the 512KB ESP-01 which have 2 General Purpose Input Output (GPIO) pins for control.
- **PlatformIO:** PlatformIO is an open source ecosystem for professional IoT development. It provides a tool chain for programming a wide range of embedded boards. The development for the ESP8266 initially was started with the Arduino IDE, but using the OSX operating system, the IDE was unable to upload to the ESP8266. PlatformIO was a functioning alternative with support for several libraries ESP8266 development.
- **Mosquitto Broker:** Mosquitto is an open source lightweight server implementation of the MQTT protocol. The server implementation allows several clients to connect and publish and subscribe to messages on topics. In the system, Mosquitto is

installed on the Raspbian distribution of Linux, with support for WebSockets. This means that clients connecting to the broker may publish and subscribe using both MQTT and WebSockets.

- **PubSubClient:** PubSubClient is an open source Arduino library for creating MQTT clients for publishing and subscribing to topics. The library is compatible with the ESP8266 and is used to control the physical devices and to know their states.
- **ESP8266WiFi:** This is a part of the ESP8266 Arduino Core and provides functions to control the WiFi capabilities of the Wi-Fi module for example setting up a web server, getting the MAC Address or IP Address, etc.
- **Eclipse Paho JavaScript Client:** Paho is an open source JavaScript library to create web MQTT clients that can publish and subscribe to the broker using WebSockets. This allows web applications to monitor and control physical devices in the system.
- **Ionic Framework:** Ionic is a cross-platform mobile application framework based on Node JS, Angular JS and Apache Cordova. Ionic allows the building of mobile applications using web technologies and has support for a large number of libraries. Apache Cordova allows you to build for multiple platforms (Android, IOS, Windows Mobile) using one code base. Using the Ionic Framework would enable this project extend to multiple mobile platforms with little costs to the project timeline.
- **MongoDB:** MongoDB is a scalable open source document database system. MongoDB is non-relational and uses key-value pairs to store complex JSON style objects. Because of the multi faceted nature of the Internet of Things (IoT), it will be difficult to design a schema that suits all objects (actuators, sensors, devices, etc.) which have non-uniform properties.

- **SMSGH API:** The SMSGH API is an interface to send SMSs to phone numbers from within an application. It uses a token-based authentication approach and is linked with an SMSGH service account for billing.
- **Amazon Simple Notification Service (SNS):** Amazon SNS is a web-based messaging service for delivering messages to mobile devices as push notifications from the cloud.

4.2 Implementation Techniques

Given the multifaceted nature of the internet of things (IoT), different implementation techniques were employed for implementing the different components of the system.

4.2.1 Host Target Development

- **Physical Devices:** Given the computing constraints that connected physical devices have, a Host-target development approach was used for developing firmware for the ESP-8266 Wi-Fi module that gave the physical devices connection capabilities. The ESP-8266 firmware was developed and deployed using PlatformIO.
- **Mobile Phones:** The Host Target development approach was also used for the development of the mobile application. The mobile application was developed using the Ionic Framework whose software development kit is PC-based. Once the code was written and tested, it was then deployed to an android mobile phone for further testing and use. The mobile application is designed specifically to run on the android operating which in turn runs on android-enabled devices.

4.2.2 Component Reuse

For other components, the Component Reuse model was employed to increase dependability, to ensure compliance with protocol standards and to accelerate development.

- **MQTT Broker:** With setting up the MQTT broker to implement the protocol to support publishing and subscription of messages over topics. A topic is a string the broker uses to filter messages to clients. The Mosquitto MQTT broker was selected from the many options available because it is an open-sourced project with the Eclipse Public License (EPL). This affords Mosquitto a reliable community to maintain the project and keep the codebase updated.
- **Reasons for reusing Mosquitto:**
    - Mosquitto is written in C, enabling it to run on many devices including embedded systems
    - Mosquitto is a very lightweight implementation of the MQTT protocol and test results show that it consumes about 3MB RAM with 1000 clients connected
    - Mosquitto supports the MQTT and Websockets protocols
    - Mosquitto has a bridge that supports connection to another broker. In this case, two brokers (one in the cloud and another on the local network) may be conveniently connected together
    - Mosquitto is very configurable
    - Mosquitto is well developed, standards compliant and maintained by the Eclipse foundation. Having to develop a broker of this nature would require unavailable resources and derail the project's timeline

4.3 How the System Works

To satisfy the system's requirements, the two main ways of controlling the devices were implemented. These are the Automation Engine and the Remote Control Engine. Pseudo code showing how they were implemented are shown below. Both pieces of code run as MQTT clients on the broker and can trigger and respond to events as well.

4.3.1 Automation Engine

Once this is implemented, without human intervention physical devices can be controlled using automation rules in the database. This piece of code will run on the Hub.

Listing 4.1: Pseudo code for Device Automation

```
/**
 * Sensors publish messages to broker containing sensor readings
 * Devices publish messages to broker containing current states
**/
Data structures:
SensorEvent {
      Sensor; Reading;
}
Automation Rule {
      Device; Sensor; SensorReading; DeviceState;
}
Event {
      Message from sensor;
}
Helper Functions:
/**
 * This function responds to a sensor event, and based on stored
 * rules, changes the device state
**/
function Automate (Event E) {
      GET automation rules with given sensor that triggered event E
      For each (Rule) {
            Sensor s = E.Sensor;
            Device d = Rule.Device;
            SensorReading sr = Rule.SensorReading;
            DeviceState ds = Rule.DeviceState;
//Check if SensorReading corresponds with the rule's trigger reading
            if E.reading is equal to sr {
                  //Check if device mode is set to automatic control
                  if d.mode is equal to automatic {
                        d.state = ds;
                  }
            }
      }
```

```
}

/**
 * When Event e is triggered the automate function is called
**/
on event(e){
      automate(e);
}
```

4.3.2 Remote Control Engine

This module allows a user to override the device automation and change the state of a device. This piece of code will run on the Android application.

Listing 4.2: Pseudo code for Remote Control

```
/**
  * A user monitors the connected devices and decides to change the
  * state of a given device
**/
Data structures:
Device {
      Name; State; Location; Controls; Mode;
}
Helper Functions:
/**
 * This helper function changes a device control mode to either manual
 * or automatic.
**/
function ChangeDeviceMode(Device d, Mode m){
//Mode may be manual or automatic. Determines which control mode is
//used
      Device.Mode = m;
}
/**
 * This helper function changes the physical state of a device
**/
function ChangeDeviceState (Device d, State Control){
//Changes the device mode to manual control
      ChangeDeviceMode(Device, Manual)
//Control may be any of the available states a device can be changed to
      d.State = Control;
}
/**
 * The user uses this function to remotely control devices from the app
**/
function RemoteControl(Device) {
//User requests to see available controls for the device
      Device.controls = GET DeviceControls;
//User requests for the device's current state and makes a decision
      State = GET DeviceState;
//User remotely controls device
      ChangeDeviceState(Device, Control)
}
```

4.4 Evidence of Implementation

This section comprises pictures that demonstrate the implemented system. These include hardware and software implementations.

4.4.1 Hardware

Figures Figure *4.1* and Figure *4.2* show connected physical devices. Figure 4.1 is an initial implementation on a breadboard to control a red LED. Once this was tested and working, the soldered board in Figure 4.2 was designed and built to control a light bulb. From the mobile application, both connected devices can be controlled.

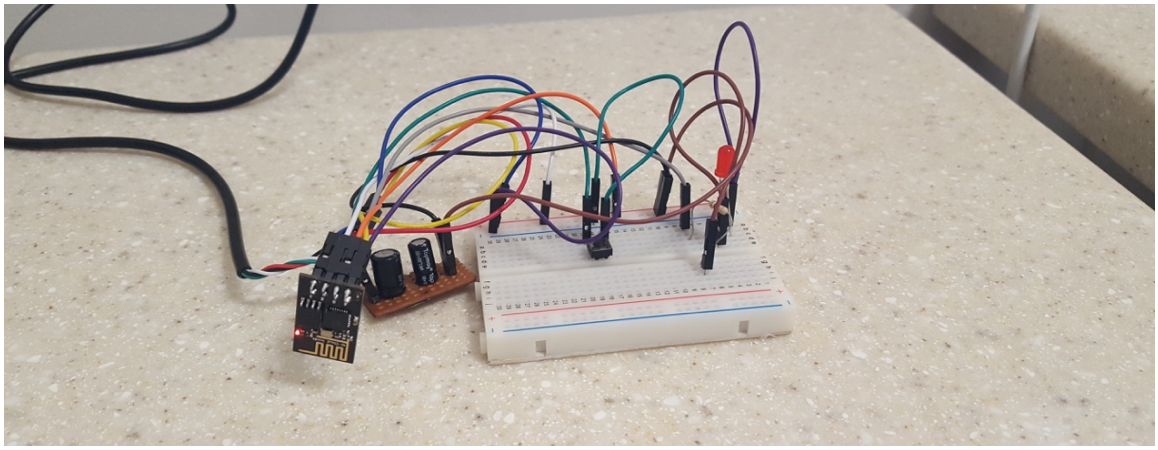

Figure 4.1: Breadboard Connection of ESP8266 to an LED

Figure 4.2: Soldered board with ESP connected to a light bulb

Figure 4.3 shows a connected sensor. The sensor takes both temperature and humidity readings and reports them to the broker using the ESP8266.

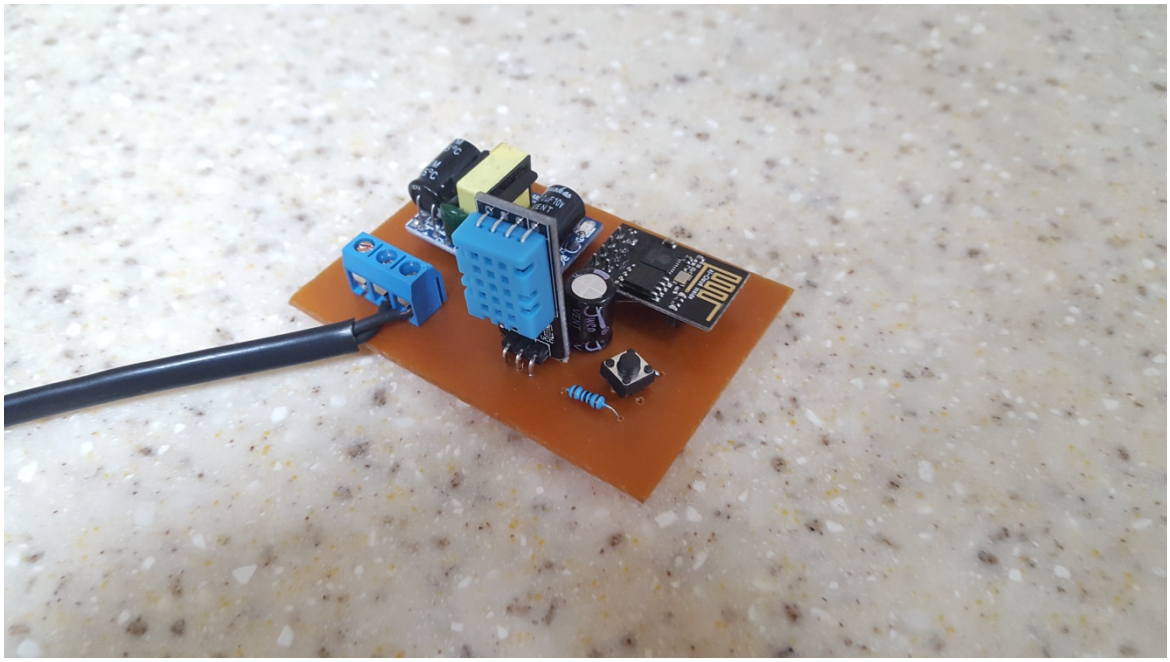

Figure 4.3: Soldered board with Humidity/Temperature Sensor and ESP8266

4.4.2 Software

The following screens show the implemented mobile application. Figure 4.4 shows the screen that shows the list of connected device. On selecting a device, Figure 4.5 shows the controls available for the device.

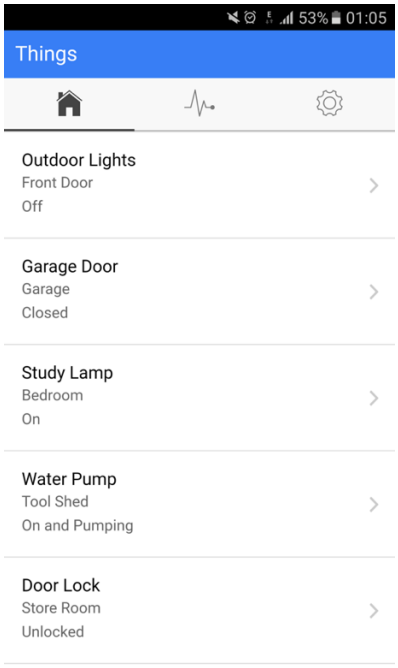

Figure 4.4: Interface showing the list of connected devices

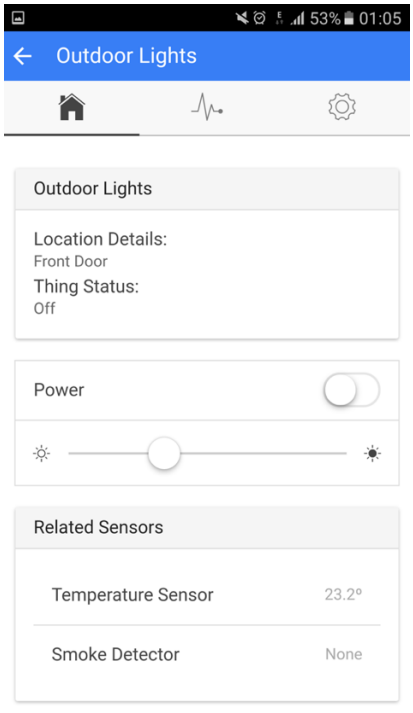

Figure 4.5: Interface showing controls for a selected device

Figure 4.6 shows a list of sensors connected to the system. The list shows their current readings and locations.

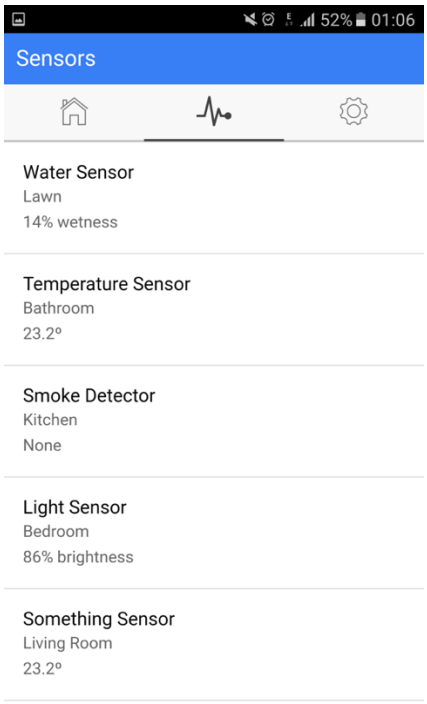

Figure 4.6: Interface showing a list of connected sensors

5. Chapter 5: Testing and Results

5.1 Approach

To ensure that the solution that was developed adequately meets the requirement that were set out at the beginning of the project, appropriate measures were taken to test the components of the system. This chapter discusses the tests that were run and their results.

5.2 Unit Testing

The testing process began with the smallest units of the system. Unit Tests were run to ensure that all the implemented functions worked correctly. For the mobile application and the backend, most of the functionality was implemented in JavaScript so a JavaScript Testing Framework was needed. Jasmine, a "Behavior Driven Development (BDD) testing framework for JavaScript" was used to run unit tests (Jasmine, 2016). Figure 5.1 shows the results of a unit test on the functions that make up the Remote Control Engine and Table 5.1 summarizes the results.

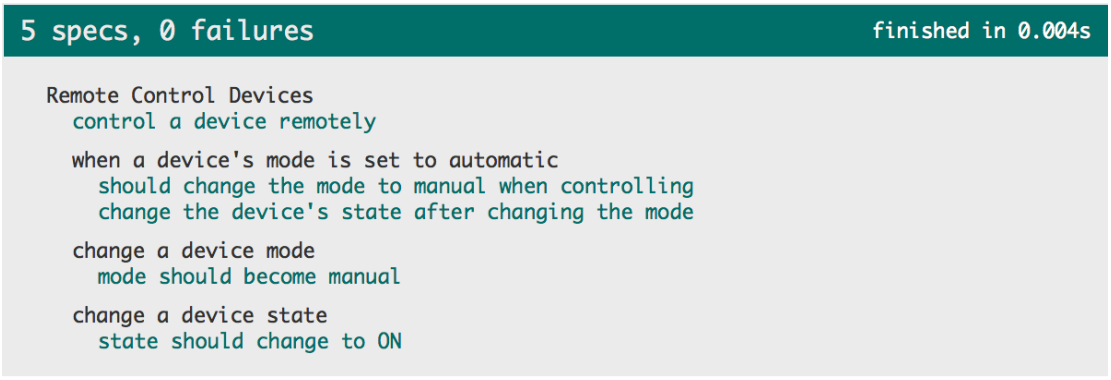

Figure 5.1: Screenshot of Jasmine Unit Test Results

Table 5.1: Summary of unit test results for Remote Control

Functionality	**Expected Result**	**Actual Result**
Remote Control (ON)	Device state changes to ON	Device state changes to ON

	Device mode changes to MANUAL	Device mode changes to MANUAL
Change Mode (MANUAL)	Device mode changes to MANUAL	Device mode changes to MANUAL
Change State	Device state changes to ON	Device state changes to ON

As seen from the snippet of automated testing code in Listing 5.1*: Unit test code for remote control engine* below, functions were tested under different conditions, and the expected outputs were compared to the actual output. Whenever the tests failed, the functions were corrected using the stack trace the testing framework provided and the unit tests were carried out again.

5.2.1 Snippet of unit test code for the remote control engine

Listing 5.1: Unit test code for remote control engine

```
describe("Remote Control Devices", function() {
 var dev;
 beforeEach(function() {
  dev = new Device();
  dev.Name = "Test LED";
  dev.State = "OFF";
  dev.DeviceLocation = "Breadboard";
  dev.Controls = ["ON","OFF"];
  dev.Mode = "Manual";

  // rule = new AutomationRule();
  // msgevent = new IncomingMessage();
 });

 it("control a device remotely", function() {
  var ctrl = "ON";
  RemoteControl(dev, ctrl);
  expect(dev.State).toEqual(ctrl);
 });
});
```

5.3 Component Testing

Once the individual units were tested and working correctly, the components were tested as well. The different components of the were tested to ensure that at a component level, the system functioned correctly.

5.3.1 Broker Testing

The first component to be tested was the Broker. This was tested by developing a test MQTT client to publish messages and subscribe to topics as seen in Figure 5.2. These messaging transactions were monitored on the Mosquitto broker command line interface. This application was used to understand how to name topics for clients. The application was also used to test authentication on the broker. First the client was used to attempt to connect to the broker with details not in Mosquitto's password file. Finally, to test the encryption configuration of the password file, it was opened and viewed to see if its contents were intelligible. As expected, a six-character password was hashed to a 107-character string in the file. Having completely tested the Broker component of the system, it was certain that it had been set-up correctly and functioned as designed.

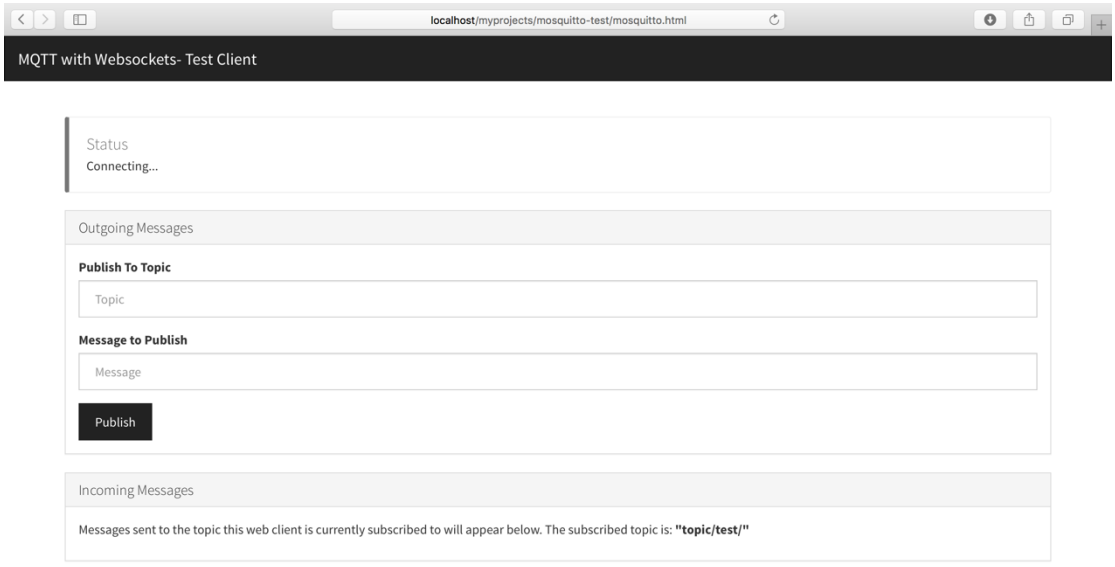

Figure 5.2: Screenshot of Broker Test Application

5.3.2 Mongo Document Database

Once the Mongo database was created, the system's RESTful Application Programming Interface (API) written in Node JS was used to test it. The GET, PUT, POST and DELETE functions in the API tested at the Unit Testing phase were now tested at the component level. Postman, a professional API testing tool was used to test the Database. As seen in Figure 5.3 a GET request for 'things' in the database with its respective URL returned a JSON array of MongoDB documents representing connected objects stored in the database. Testing the different HTTP methods in the API demonstrated that database was properly configured and its contents were accessible.

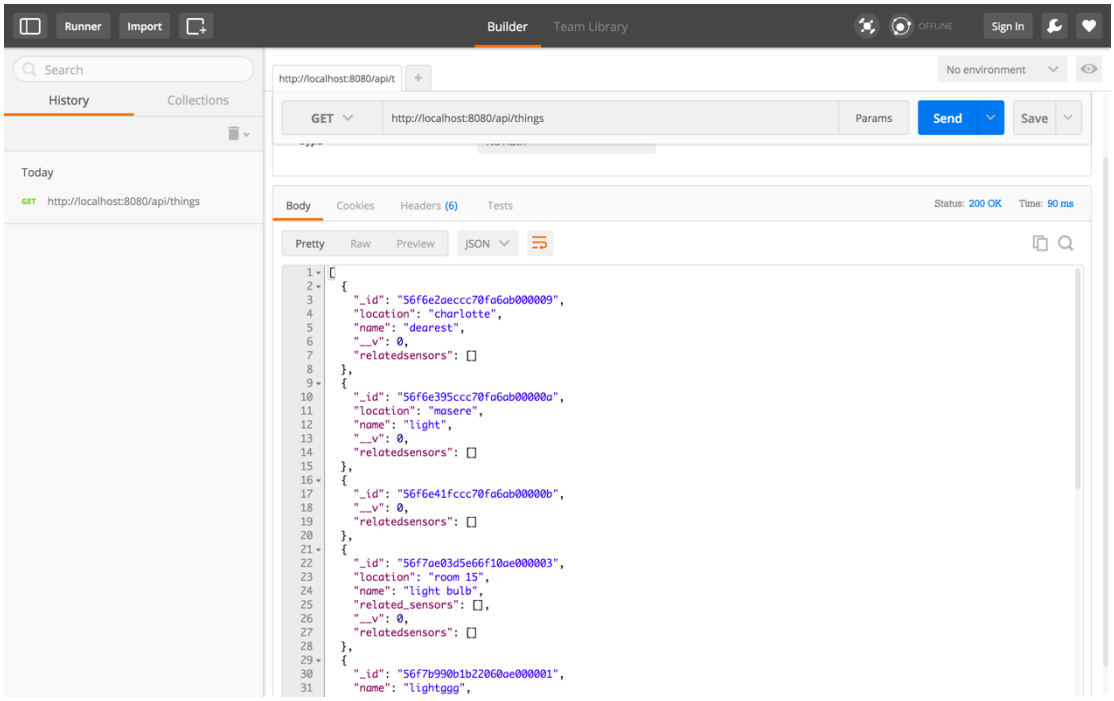

Figure 5.3: Screenshot of Postman application for testing database

5.3.3 Mobile Application

The mobile application component built with the Ionic Framework was tested using a browser at the component level. The Ionic command line interface has an inbuilt webserver that allows a developer to view changes in real time. With this the mobile application was tested, using the browser console to log and monitor different occurrences

within the application. For example, when an MQTT message was published from the application, logging to the console helped to record the timestamp and actual contents of the message. These logs were compared with the changes in the on/off state of a test LED.

5.4 System Testing

At the system level, more technical aspects of the system were tested to be able to determine if the set requirements had been met adequately. As part of the system tests response time, scalability and conflict resolution were the main concerns.

5.4.1 Response Time

For the remote control requirement to be meaningful to a user, the system would have to minimize response time. Hence, once the system was implemented the time between a button press within the application and a change in the state of a device was to be measured. However, by observation, the control seemed instantaneous. To accurately measure, this time required additional hardware components including a real-time clock. Unable to get a real-time clock, the response time could not be calculated. However, because of the seemingly instantaneous nature of the response, it can be concluded that with an estimated response time of less than 1 second, the requirement has been met.

5.4.2 Conflict Resolution

From tests on the system's ability to handle conflicts, it was found that conflicts between Remote Control and Device Automation were resolved by the system. However, when two or more remote controlled events conflict, the system is unable to meaningfully handle them. Similarly, when there are two or more conflicting Device Automation events the system fails to resolve them. In order to intelligently resolve these conflicts, machine

learning and data mining techniques would be needed to determine the most desirable outcome. However, that was out of the scope of this project.

5.4.3 Scalability Testing

The Mosquitto broker does not place a limit on the number of connections to the broker. However, according to benchmark tests, with high load on the CPU and high transmission latency, Mosquitto succeeds in connecting 60,000 clients. To explore how this statistic compares to the implemented system, a test client was developed to publish 1,000 control messages to the web browser to repeatedly switch a connected object on and off. This timed experiment was run 4 times and for all 4 tests. Table 5.2 below shows a summary of the results. The variations in duration may be attributed to the load on the network.

Table 5.2: A table showing the results for a scalability test

Round of Experiment	Controls	Duration (s)	Rate (controls/sec)
1	1000	117	8.547
2	1000	166	6.024
3	1000	233	4.292
4	1000	216	6.630
Average rate			23.49 ≈ 24controls/sec

As seen from Table 5.2, at high load the broker can handle about 24 control requests per second. Although this seems like a rather high estimate, the rate is not good enough because the Internet of Things is expected to scale at a remarkable rate.

5.5 User Testing

A hallway user test was run with a novice user who found the basic features of the application easy to use. The user was able to navigate to the view devices in the system and change the state of a given device.

6. Chapter 6: Conclusions and Recomendations

6.1 Summary

Overall the system successfully meets the set requirements. The system is able to successfully automate the control of physical devices and allow a user to remotely control devices in the home. The system does this with a minimum viable set of open-source protocols. Compared to related systems Eclipse Kura Framework and Windows IoT Core, the project is comparatively lightweight.

6.2 Limitations

Despite the success of the project in meeting the requirements, there are some limitations that could not be addressed with the resources allotted for the project. These include:

- The project is limited to Wi-Fi connections
- Reduce latency on the broker to scale more efficiently

6.3 Future Work

To extend the project additional features to improve the system as an IoT solution include:

- Use geofencing technology to make the system smarter
- Apply machine learning and data mining to improve the system's engines
- Use NFC, RFID or Barcode technologies extend interactions with physical system
- Use natural language voice recognition to control devices in a more natural way

7. References

Castellani, A. P., Dissegna, M., Bui, N., & Zorzi, M. (2012). WebIoT: A web application framework for the internet of things. In *Wireless communications and networking conference workshops (WCNCW), 2012 IEEE* (pp. 202-207). Retrieved from Google Scholar.

CoAP,. (2016). *CoAP — Constrained Application Protocol | Overview*. *Coap.technology*. Retrieved 17 April 2016, from http://coap.technology

Dorsemaine, B., Gaulier, J., Wary, J., Kheir, N., & Urien, P. (2015). Internet of Things: A Definition & Taxonomy. *2015 9Th International Conference On Next Generation Mobile Applications, Services And Technologies*. http://dx.doi.org/10.1109/ngmast.2015.71

Eclipse,. (2016). *iot.eclipse.org — Standards*. *Iot.eclipse.org*. Retrieved 17 April 2016, from http://iot.eclipse.org/standards#etsi-smartm2m

Hamann, H. (2015). From Smart Sensors to Smarter Solutions with Physical Analytics. *Proceedings Of The 13Th ACM Conference On Embedded Networked Sensor Systems - Sensys '15*. http://dx.doi.org/10.1145/2809695.2823463

Logvinov, O. (2014). *Open Standards Will Enable the IoT's Growth*. *Electronicdesign.com*. Retrieved 17 April 2016, from http://electronicdesign.com/iot/open-standards-will-enable-iot-s-growth

Microsoft,. (2016). *Windows 10 IoT for your buisness*. *Microsoft.com*. Retrieved 17 April 2016, from https://www.microsoft.com/en-us/WindowsForBusiness/windows-iot

Mosquitto,. (2016). *mqtt*. *Mosquitto.org*. Retrieved 17 April 2016, from http://mosquitto.org/man/mqtt-7.html

MQTT,. (2016). *FAQ - Frequently Asked Questions | MQTT*. *Mqtt.org*. Retrieved 17 April 2016, from http://mqtt.org/faq