



ONLINE HOME AUTOMATION CONTROL SYSTEM



By:

437004875 Reem Ali AlGhamdi
436200058 Abeer Ahmed Ezzat
437004100 Nouf Abdullah AlDajani
437004005 Mona Saud AlKhathlan
436200063 Doha Nidal AlZouhbi
436006939 Sarah Khalid AlHussain

Supervised By:

Dr. Abeer Mahmoud

A Graduation Project Report Submitted to
College of Computer and Information Sciences at PNU
in Partial Fulfillment of the Requirements for the
Degree of
Bachelor of Science
in
Computer Sciences

CCIS, PNU
Riyadh, KSA
1440 - 1441H



Acknowledgment

First and foremost, we thank Allah the Almighty for always blessing us and for helping us complete this project on time. We would like to express our special thanks of gratitude to our project supervisor Dr. Abeer Mahmoud for her continuous support, motivation, and for always commenting on our best qualities and enhancing them. Without her direction and proper guidance, this project would not have succeeded. We thank her for her unwavering support. Not to forget our loving families and supporting friends for helping us sticking to the schedule. We also would like to thank Princess Nourah bint Abdulrahman University. Last but not least we want to thank anyone who has helped us by providing information, insight, comments or facilities that we needed to complete our project. Our heartfelt thanks to everyone we have mentioned.



Contents

Acknowledgment	i
Contents	ii
List of Tables	v
List of Figures	vi
List of Symbols & Abbreviations	ix
Keywords	x
Abstract	xi
1 Introduction	1
1.1 Problem statement & Significance	1
1.2 Proposed Solution	1
1.2.1 Aims	2
1.2.2 Goals	2
1.3 Project Domain & Limitation	2
1.3.1 Domain	2
1.3.2 Limitation	3
1.4 Gantt Chart	3
2 Background Information & Related Work	5
2.1 Background Information	5
2.1.1 IoT	5
2.1.1.1 IoT Architecture	5
2.1.1.2 IoT Applications	5
2.1.2 Hardware	6
2.1.3 Programming Languages & Frameworks	7
2.1.4 SDLC Model	7
2.2 Related Work	8
2.2.1 Insteon - Insteon Hub	8
2.2.2 Wink - Wink Hub 2	10
2.2.3 Samsung Smart Things Hub	12
2.3 Proposed & Similar System Comparison	13
3 System Analysis	17
3.1 Requirement Specification	17
3.1.1 Overview	17
3.1.1.1 Input	19
3.1.1.2 Output	19
3.2 Requirement Analysis	20
3.2.1 Software Requirements	20
3.2.2 Hardware Requirements	22



3.2.3	Functional Requirements	23
3.2.4	Non-Functional Requirements	24
3.2.5	Structured Diagrams	26
3.2.5.1	Use Case Diagram	26
3.2.5.2	Use Case Scenarios & Sequence Diagrams	28
3.2.5.3	Flowchart Diagram	54
3.2.5.4	Entity Relationship Diagram	57
3.2.5.5	Dataflow Diagram	60
3.2.6	Object-Oriented Diagrams	63
3.2.6.1	Class Diagram	63
3.2.6.2	UML representation of the REST API Diagram . .	66
4	System Design	70
4.1	System Architecture	70
4.1.1	The android application	70
4.1.2	The raspberry pi script	70
4.1.3	The web application	70
4.1.4	The system database	71
4.2	User Interface Design	73
4.2.1	Login Activity	73
4.2.2	Raspberry Activity	74
4.2.3	Hardwares Activity	75
4.2.4	Hardware Activity	76
4.2.5	New Command Activity	77
4.2.6	Schedule Activity	78
4.2.7	Progress Fragment	79
4.2.8	Message Fragment	80
5	Implementation	82
5.1	Implementation Requirements	82
5.1.1	Software Requirements	82
5.1.2	Hardware Requirements	82
5.2	Implementation Details	83
5.2.1	Web Application	84
5.2.2	Raspberry Pi	86
5.2.3	Android Application	88
5.3	I/O Screens	93
5.3.1	Login Screen	93
5.3.2	Raspberry Pi Screen	94
5.3.3	Hardware Screen	95
5.3.4	Command & Schedule Screen	96
6	Testing	98
6.1	Testing	98
6.2	Test Plan	98
6.2.1	Unit Test	98
6.2.2	System and Integration Testing	98
6.2.3	The Scope	99
6.2.4	Features to Be Tested	99



6.2.5 Schedule of Test Activities	100
6.2.6 Test Tasks	100
6.3 Test Case	101
6.4 Test Result	103
7 Deployment	107
7.1 Web Application	107
7.2 Android Application	109
8 Conclusion	112
8.1 Conclusion	112
8.2 Evaluation	112
8.2.1 Problems faced:	113
8.3 Future Work	114
References	116
9 Appendices	122
A Figures	123
A.1 Similar systems hub design	123
A.2 Some more Validation Cases	124
B REST API Documentation	125
C Code	136
C.1 Web Application	136
C.2 Raspberry Pi	160
C.3 Android Application	164



List of Tables

1	List of Symbols & Abbreviations	ix
2	Keywords	x
2.1	Proposed & Similar System Comparison	13
3.1	Non-functional requirements	24
6.1	Test Schedule	100
6.2	Test Case	102
6.3	Expert User's Functionalities	103
6.4	Web Application's Functionalities	103
6.5	Raspberry Pi's Functionalities	104
6.6	Android Client's Functionalities	104
6.7	Non-functional requirements	105



List of Figures

1.1 Gantt Chart	3
2.1 Related Work: Insteon	8
2.2 Related Work: Wink	10
2.3 Related Work: Samsung	12
2.4 Relationship between cost, flexibility and usability	15
3.1 Use case diagram	27
3.2 sequence diagram: Expert user adds a new hardware to system . . .	29
3.3 sequence diagram: Expert user adds a new configuration to system	30
3.4 sequence diagram: User login or register	32
3.5 sequence diagram: User adds new raspberry pi	33
3.6 sequence diagram: User removes raspberry pi	34
3.7 sequence diagram: Android app gets hardware list	36
3.8 sequence diagram: Android app gets command list	37
3.9 sequence diagram: Android app deletes a scheduled command . . .	39
3.10 sequence diagram: Android app edits a scheduled command . . .	40
3.11 sequence diagram: Android app adds a new command	41
3.12 sequence diagram: Android app gets responses	42
3.13 sequence diagram: Raspberry pi gets commands from server . . .	44
3.14 sequence diagram: Raspberry pi updates local queue	46
3.15 sequence diagram: Raspberry pi executes commands	48
3.16 sequence diagram: Raspberry pi submits a command response to server	50
3.17 sequence diagram: Web application deletes executed immediate commands	52
3.18 sequence diagram: Web application deletes responses	53
3.19 Flow chart - Android app	55



3.20 Flow chart - Raspberry pi	56
3.21 Entity-relationship diagrams	59
3.22 Dataflow diagram - Level 0	60
3.23 Dataflow diagram - Level 1	61
3.24 Dataflow diagram - Level 2	62
3.25 Class diagram for the web application	64
3.26 Class diagram for the raspberry pi	65
3.27 REST API	67
3.28 REST API models	68
4.1 Component Diagram	71
4.2 Deployment Diagram	71
4.3 Architecture Diagram	72
4.4 UI: Login Activity	73
4.5 UI: Raspberry Activity	74
4.6 UI: Home Activity	75
4.7 UI: Hardware Activity	76
4.8 UI: New Command Activity	77
4.9 UI: Schedule Activity	78
4.10 UI: Progress Fragment	79
4.11 UI: Message Fragment	80
5.1 Connecting Raspberry Pi with the hardware	83
5.2 I/O: Login Screen	93
5.3 I/O: Raspberry Pi Screens	94
5.4 I/O: Hardware Screens	95
5.5 I/O: Command Screens	96
6.1 Login Validation	98
6.2 Postman testing the API	99
7.1 Landing Page	108



7.2 The Application in Google Play	110
A.1 Similar systems: hub design	123
A.2 Validation	124



List of Symbols & Abbreviations

Symbols & Abbreviations	Meaning
API	Application programming Interface
CRUD	Create, Read, Update and Delete
GPIO	General Purpose Input Output
GUI	Graphical User Interface
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LED	Light Emitting Diode
ORM	Object relational mapping
REST	Representational State Transfer
SDK	software development kit
SDLC	Software Development Life Cycle
SQL	Structured Query Language
UI	User Interface

Table 1: List of Symbols & Abbreviations



Keywords

Keyword	Definition
Raspberry Pi	low cost, credit-card sized computer[1].
Linear solenoid	type of electromagnetic actuator that converts an electrical signal into a magnetic field producing a linear motion[2].

Table 2: Keywords



Abstract

With the recent very rapid progress in technology and automation, and towards easier daily life tasks, there has become a need for remote control of almost all possible aspects of living. Some examples are already available. However, they lack flexibility, which restrict the user to a certain set of hardwares with limited variety. The aim for this project is to control light buttons, air conditioners, television or other home appliance regardless of the person's location. The methodology is simple: an android app will send controlling requests to a web server. Raspberry Pi will be getting all the new requests from the server, processing it accordingly and controlling the hardware components connected to it. Such a system will allow someone in the United States to turn the lights in their house in Saudi Arabia on. However, an active connection to the internet must be present all the time.

CHAPTER NO. 1

INTRODUCTION



1 Introduction

1.1 Problem statement & Significance

With the recent very rapid progress in technology and automation, and towards easier daily life tasks, there has become a need for remote control of almost all possible aspects of living. Especially the house appliances that surround us, that allow focusing on main work of each day.

Some examples we have already encountered and used in our daily lives include using apps to control a cleaning robot or adjust the heating in the house or switch the house lights on or off. For the latter, there have been many applications that can do that. However, they all work in a small set of devices and sensors. It is necessary for such applications to exist, as a service like this would be important for many people. An example is working moms who are outside the house and want to switch the lights on at a certain time to wake their children up. Another example is pet owners who need to have UV lights switched on for their pets at certain times of the day but can't do so immediately and so on.

However, the main challenge in creating a device to solve this problem is where the idea of IoT (Internet of Things) comes in; learning how to control this device through the Internet from afar, rather than being controlled by infrared rays locally as is the case with most similar applications.

1.2 Proposed Solution

This project proposes building an intelligent application that is based on IoT techniques. The created app should enable the user, by clicking on the appropriate buttons, to control a physical apparatus that needs to be pushed to work, such as lights buttons. This will be done by designing and creating an Android application, then using a small laptop, called Raspberry Pi, to control a small piece that will



be pushed forward (on command) to switch the light on or off, the API is a web application hosted on a server.

1.2.1 Aims

At the end of this project, we intend to achieve the following aims:

- Learn how to design a mobile application using previously learned and new knowledge
- Learn how to invoke a web API and use it in our application
- Learn Python programming language to control Raspberry Pi
- Learn Flask web micro-framework

1.2.2 Goals

At the end of this project, we expect to deliver:

- An Android application with a user friendly, simple, clear interface with buttons to control a LED and linear solenoid. However, theoretically, the system can accept any other electronic device as long as it can be installed in the electronic circuit.
- A physical apparatus composed of the Raspberry Pi connected to and controlling the piece.
- A web application following REST architecture, managing user requests and Raspberry Pi's responses.

1.3 Project Domain & Limitation

1.3.1 Domain

This project falls under the Internet of Things (IoT) scope, which is a branch of cloud computing.



1.3.2 Limitation

This project aims to control any hardware connected to the electronic circuit. Thus, the flexibility and freedom are extremely high. However, this advantage comes at a non-financial cost: the learning curve. The users must be knowledgeable about the basics of electrical engineering and computer science. A manual could be made to make the process easier.

Although the application will be available for all kinds of users to use, we expect that the ones who would make the most use of it would be people who have some background in electrical engineering and computer science.

1.4 Gantt Chart

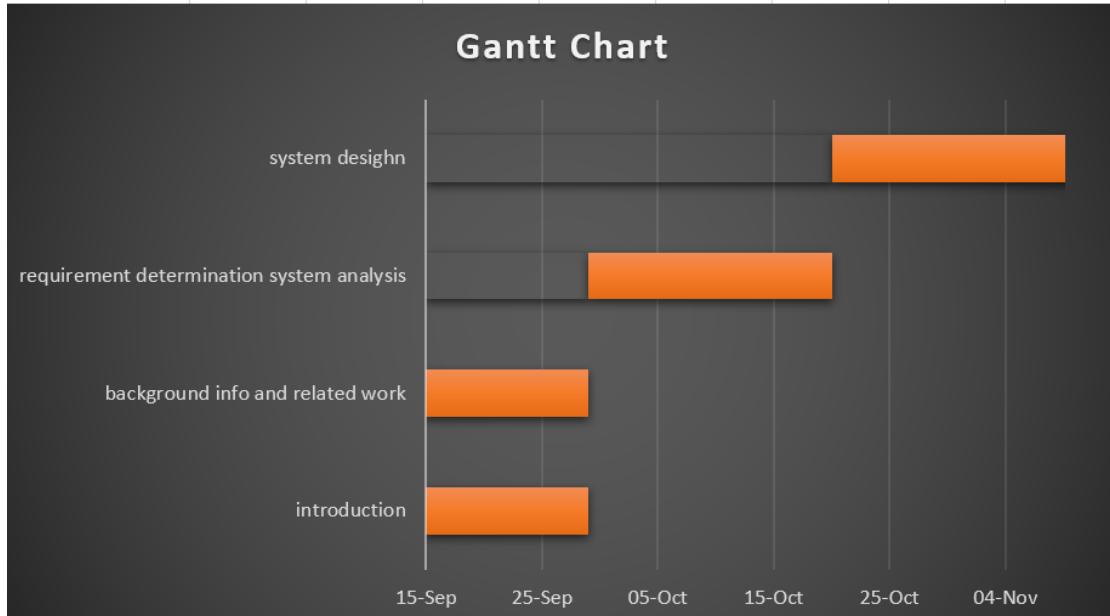


Figure 1.1: Gantt Chart

CHAPTER NO. 2

BACKGROUND INFORMATION &

RELATED WORK



2 Background Information & Related Work

2.1 Background Information

2.1.1 IoT

The internet of things (IoT) is one of the most important technological developments in the last decade. Do you wonder where the term IoT came from? It was coined by Kevin in 1999 [3]. This means we can describe IoT as the ability to connect as many things by the internet without humans help by using technologies such as cloud computing, Radio Frequency Identification (RFID), wireless communication, sensors, Internet protocol, ultra-low-power processors and others [4].

2.1.1.1 IoT Architecture

It is basically what IoT is made of. It contains three layers; Perception layer, Network layer, and Application layer. So, to know these layers better we need to know the definition of them.

- **Perception layer:** is responsible for perceiving and identifying objects or things in the environment [5].
- **Network layer:** is responsible for receiving and transmitting data between layers [6].
- **Application layer:** is the interface for all previous layers used to process and transport data to provide services to the users [7].

2.1.1.2 IoT Applications

There are different kinds of applications that make life easier and more secure. These are some examples of them:

- **Automobiles:** by IoT technology, it makes cars smarter and works to improve safety on the road by helping cars detect obstacles and assist braking



or adapt their speed to the flow of traffic. Also it helps protect the environment by reducing fuel consumption, etc [8].

- **Smart Home:** You can control your entire home with a touch screen or your smartphone whether you are inside or outside a home, for example, you can lock or open your doors, turn on/off the light and air conditioner, etc [9].
- **Healthcare:** IoT has an important role in healthcare applications by providing the ability to easily monitor and manage patient health without having to manually visit each patient the doctor can give a remote diagnosis to provide quality care more quickly and manage the health care environment more efficiently [6].

2.1.2 Hardware

- **Raspberry Pi:** a small general purpose computer. All hardware components will be connected to it. An active connection to the internet is needed for it to fetch data from the server[1].
- **Ubuntu Web Server:** hosts the web application. Digital Ocean servers[10] were chosen for this project.
- **LED:** since the hardware components controlled depends heavily on the user needs, this project main aim will be controlling a small LED. LED stands for light-emitting diode[11]. Basically a small light source.
- **Linear Solenoid:** once the LED works, linear solenoid will be installed for demonstrating the idea[2]. It is a small component that generates a linear motion. It will be used to press in anything, such as lights, TV remote, and coffee machine.



2.1.3 Programming Languages & Frameworks

- **Python:** raspberry pi can be controlled by either c++ or python. Python was chosen because a REST API can be made using it fast.
 - **GPIO:** a library for controlling any hardware component connected to the GPIO pins[[12](#)].
 - **Flask:** a lightweight framework to build web applications[[13](#)].
- **Java:** mobile application are made in a native way with either swift or java.
 - **Android:** a framework for making android apps.
 - **Retrofit:** type-safe HTTP client for Android and Java[[14](#)]. It will be used to send and receive commands and status from the web server.
- **PostgreSQL:** an open-source RDBMS[[15](#)]. It will be installed on the server.

2.1.4 SDLC Model

Incremental model will be used in this project. This model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing / verification, maintenance[[16](#)]. The reason this model was chosen is the pieces will be installed, tested and connected to the system gradually. First a LED, then a linear solenoid and so on.



2.2 Related Work

2.2.1 Insteon - Insteon Hub

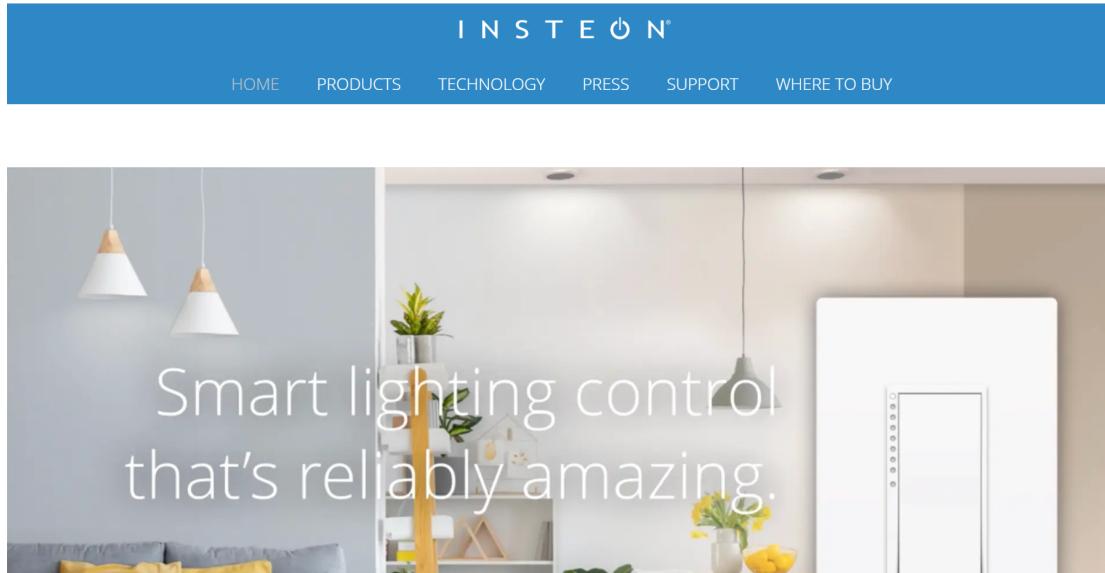


Figure 2.1: Related Work: Insteon

Insteon Hub is a simple and straightforward system that connects you to your home from any smartphone or tablet, anywhere in the world. Control Insteon light bulbs, wall switches, outlets, and thermostats at home or remotely and receive instant email or push notification alerts from motion, door and window, water leak, and smoke sensors while you're away[17]. *Figure 2.1* shows the home page for Insteon.

• **Advantage:**

1. Control Multiple Devices Simultaneously with a Basic Scene.
2. Create Schedules to Turn Your Lights On and Off at Specific Times.
3. Automatically Turn Lights On and Off with Sensors.
4. Monitor Your Home with Email or Push Notification Alerts.

• **Disadvantage:**



1. Hub setup takes a couple of minutes and a few moments per light switch, sensor.
2. Its need to connect it to power and your home's internet router so if the internet die all devices need to start over again.
3. fixed the hub take more cost than its original price.
4. There is no database save/restore. You have to recreate all the devices, scenes, schedules if its replaced.

2.2.2 Wink - Wink Hub 2



Figure 2.2: Related Work: Wink

Wink Hub 2 is the world's first smart home hub created for the mainstream consumer. With industry-leading smart home protocol support, enhanced connectivity features, and a sleek design, Wink Hub 2 brings hundreds of products from best-in-class brands together for a simple, intuitive experience[18]. *Figure 2.2 shows the home page for Wink.*

- **Advantage:**

1. Support Different platforms such as iOS or Android.
2. Once you've created an account, Wink has the ability to recognize the products within Wink Bright, guide you through a few simple steps, and then you're ready to go.
3. Wink works with Cortana Microsoft's voice assistant and Amazon Alexa.
4. One Important features in Wink, its can see what you're spending even before the bill arrives.

- **Disadvantage:**



1. One major problem with the Wink 2 hub is that the device sometimes loses connectivity and must be reset in order for it to connect again.
2. Wink app doesn't always let you access other devices' full features.
3. High price.
4. Takes 14 days to arrive.



2.2.3 Samsung Smart Things Hub

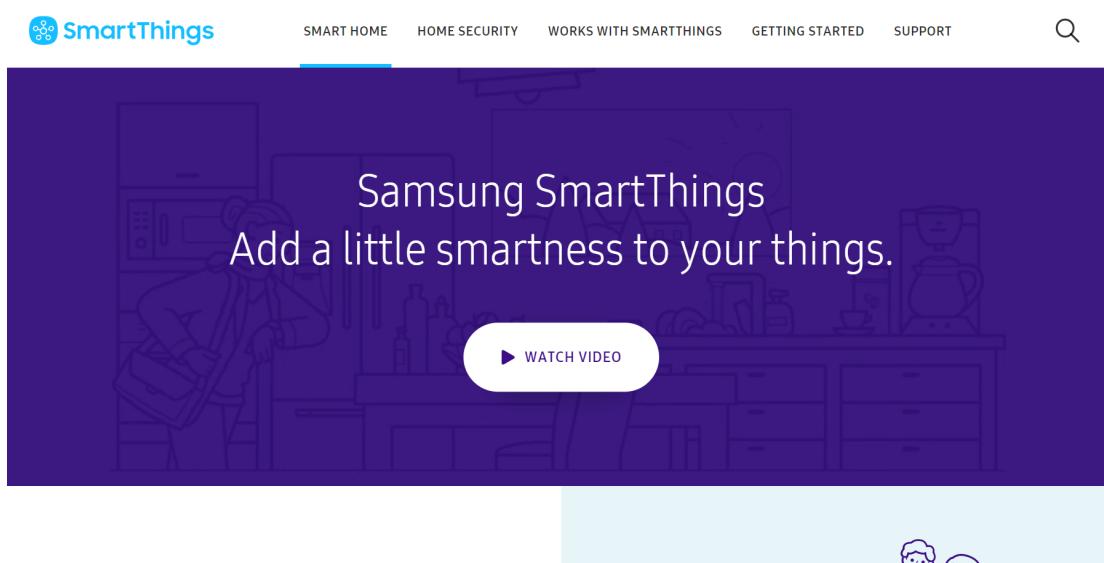


Figure 2.3: Related Work: Samsung

Smart things hub Connect wirelessly with a wide range of smart devices and make them work together[19]. *Figure 2.3* shows the home page for Samsung.

- **Advantage:**

1. Monitor and control connected devices in your home using a single SmartThings app for iPhone or Android.
2. Manage connected devices in your home with SmartThings Routines for Good Morning, Goodbye, Good Night, and more.
3. Receive alerts from connected devices when theres unexpected activity in your home.

- **Disadvantage:**

1. Some compatible components may not work as efficiently or smoothly as you want them to, which may be inconvenient.
2. Some users report it stops working at times.
3. Difficult to upgrade from older hub.
4. In US Only.



2.3 Proposed & Similar System Comparison

	Raspberry Pi	Insteon	Wink hub 2	Samsung (smart things)
design				
Hub Price	25\$	80\$	99\$	70\$
Hardware Price	parts are very cheap	expensive parts	very expensive parts	expensive parts
Flexibility	any hardware that can be physically connected	42 hardwares of basic variety [20]	102 hardwares of different variety [21]	only 6 hardwares support [22]
Installation & Configuration Difficulty	require one expert user. Easy control for all other users via mobile app	easy to install and control hardwares with mobile app	easy to install and control hardwares with mobile app	easy to install and control hardwares with mobile app

Table 2.1: Proposed & Similar System Comparison

Better quality can be found at Appendix A.1



Although similar systems already exist, our system has its own special advantages. The biggest being **hardware freedom**. In other systems, there exists a main hub receiving user command from the mobile app. So far, the ideas and implementation is identical. The previous systems require the consumer to buy additional parts for it to work, such as special LED lights that needs installation or a small component controlling air conditioners. Those parts are usually limited in numbers, usage and can get very expensive fast. On the other hand, our system works with any hardware component as long as connecting it to the electrical circuit is possible.

However, this great flexibility comes with a great sacrifice: the steep learning curve. To keep the hardware prices as low as possible while maintaining high flexibility requires the buyer household to have one expert user. For example, in a family of four, all can control the AC. However, one of them must be an expert who can configure the hardware controlling the AC and adding it to raspberry pi manually. Of course, making it easier is possible, but it will result in a higher component cost and less flexibility as hardware components would need to be bought from our company (future implementation) in comparison to buying it from any cheap electronic store (current proposed solution).

figure 2.4 shows this relationship between flexibility, cost and usability for the three systems, our proposed solution and the merchant edition(future work).

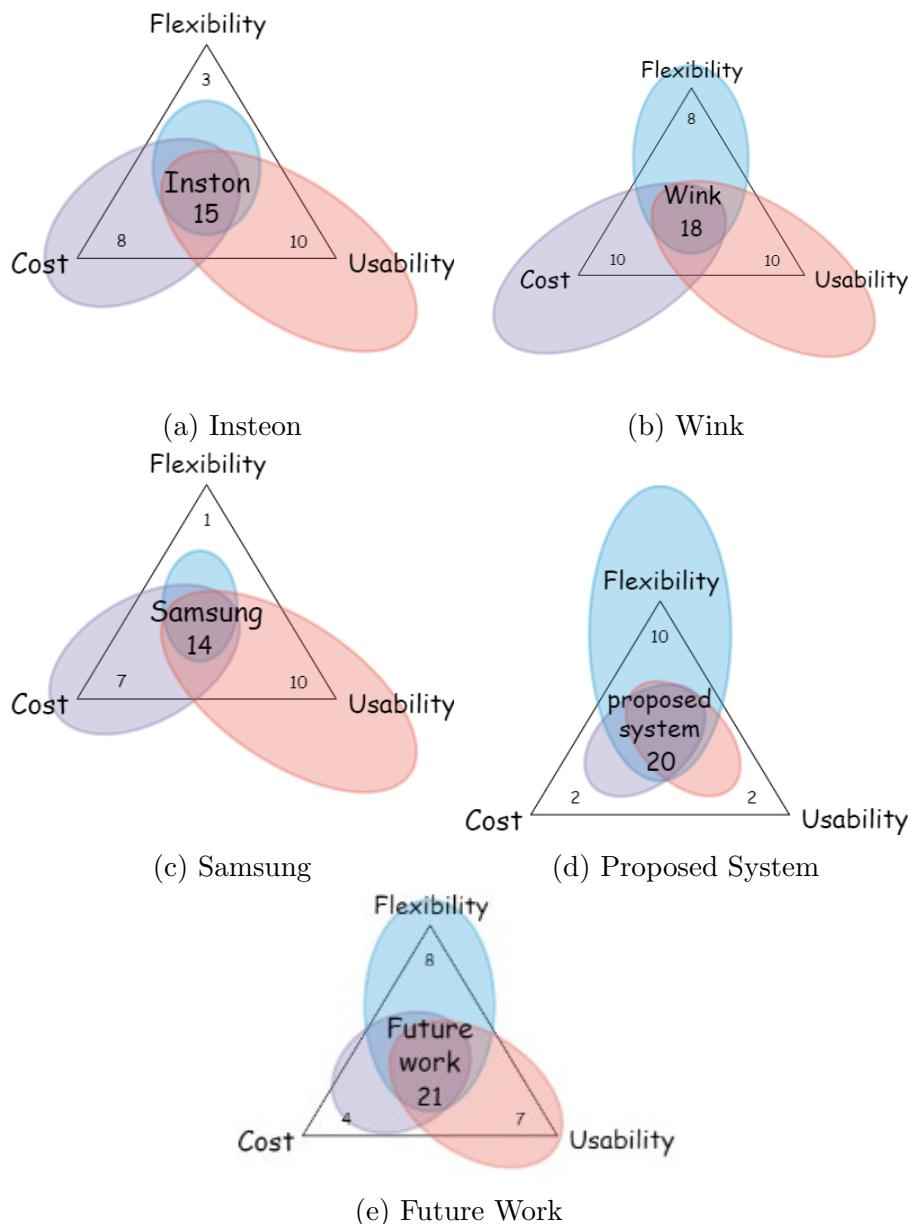


Figure 2.4: Relationship between cost, flexibility and usability

CHAPTER NO. 3

SYSTEM ANALYSIS



3 System Analysis

3.1 Requirement Specification

These requirements were selected after discussing the proposed idea thoroughly. Top-down approach was used, in which we discussed the general goal of this project, then broke it down even further to smaller manageable units.

3.1.1 Overview

The proposed project consists of three main systems: the android application which will be the user interface, the Raspberry Pi, which is the small computer where all the hardware pieces will be connected to, and the web server which will host the REST web application and be the connection between the android application and the Raspberry Pi.

First, the user is required to install the android app on his mobile phone. When the app is opened, the user either register or login. Once logged in, a list of raspberry pis is shown with the possibility of adding more. To add a new raspberry pi, the user must scan the QR code that belongs to the raspberry pi. Any user or raspberry added is immediately uploaded to the web server.

Once a raspberry is clicked, an activity displaying the hardwares that are connected to the Raspberry Pi is shown. It gets this list of hardwares from the webserver by using the GET method. When the user clicks on any hardware that is there, a new activity opens. In it is mentioned the name of the hardware, the status (e.g. whether it is on or off), the commands and the scheduling configuration. All of this information is obtained from the server.

Next to the commands title, there will be a small button that when clicked on will open a third activity, which gives the option of adding a new command. The command can be instantaneous (for example, switching an LED light immediately)



or it can be scheduled for a later date or time. For the instantaneous command, the POST method will be used, and for scheduling the commands, the user will have the option to choose the date and time he wishes the commands to be undertaken in. Whatever the outcome of this process is, a popup message will appear to the user either confirming the success of the command the user issued, or denying it while explaining the reason for that failure.

Under the commands tab, there will be a configuration section where all the scheduled commands will appear, along with their dates and times and options to edit them or delete them. The edit option will be done by the PUT method and the deleted option by the DELETE method. All the methods work on the data in the servers database i.e. they either add a new command (POST), edit a scheduled command (PUT), or delete an existing command (DELETE).

For the Raspberry Pi, the sequence it works according to is timed. Every 5 minutes, it puts the hardware status to the server so that it can show on the users application hardware list. Also, every 30 seconds, it checks the server for any new commands posted by the user from the android application. If there are any new ones that have to be, it updates its own local database (a local queue) according to the priorities and scheduled dates and times of the commands. This local database is organized according to the time the command was issued (i.e. the instantaneous commands are put at the front of this queue because of their precedence and the scheduled ones are put in the command order) and contains the command ID, which hardware this command was issued for, when this command was issued, and whether the command was successfully done or not, all gotten from the server by the GET method except the successfully done column, which the Raspberry edits according to the hardware.

Whatever the result of the command was, the Raspberry posts the response of the command to the server. The android application gets this response from



the server every 5 or 10 minutes, depending on the user's choice. The response is displayed as a push notification in the user's mobile phone. Depending on this response, the status and configuration information in the app will be updated to reflect the success or failure of the command response. Finally, it is important to note that any new hardware or configuration added to or connected to the Raspberry Pi will have its information posted to the server by the Raspberry Pi computer, where the user can view it then as soon as he opens his application to the first activity. When the response is successfully done and read by the user, the webserver deletes it from the database to save space. The webserver also deletes instantaneous commands once the user is notified the execution result.

3.1.1.1 Input

The user command issued using the Android client is the main input. Each command consists of the following:

- chosen hardware.
- configuration wanted.
- optional scheduling information.

The **hardware** is the physical component connected to raspberry pi. Each hardware has a set of possible states that it can be in. Those states are called **configuration**. The **schedule** indicates the time of day and days of week the user might want the command to run at.

3.1.1.2 Output

- a response

The raspberry pi issues a **response** indicating whether the command has been successfully done with an optional message. This response is saved in the webserver, which in turn is read by the android client periodically.



3.2 Requirement Analysis

3.2.1 Software Requirements

- **Android Application**

- **Java programming language:** it is a general-purpose, concurrent, class-based, object-oriented programming language[23]. It is going to be used to develop the android mobile application.
- **Android SDK:** a software development kit for developing android mobile apps. It includes a comprehensive set of development tools and libraries to aid the development process[24].
- **Retrofit library:** a type-safe HTTP client for Android and Java[14]. It will be used to send and receive commands and status from the web server.
- **Android studio:** Android Studio is the official integrated development environment for Google's Android operating system[24]. This IDE is going to be used to write the android application code.

- **Web Application**

- **Python programming language:** it is a high-level interpreted general purpose language[25]. It will be used in making the web application.
- **Flask framework:** a lightweight micro-framework to build web applications[13]. It will be used to build the REST API and interact with the database.
- **SQLalchemy library:** flask does not offer a database abstraction layer[13], so a supporting library must be installed to interact with the database. SQLAlchemy is an Object relational mapper that does not restrict its user to an RDBMS[26]. However, this project will use postgres.



- **SQL programming language:** it is a database manipulating programming language. It is used to store and manage information in relational databases that enables users to create view, stored procedure and functions in a database[27]. It will be used to save commands, users, raspberries, hardwares, and responses.
- **Postgresql RDBMS:** an open-source RDBMS[15]. It will be installed on the server.
- **pycharm IDE:** an IDE to develop mobile application by JetBrains[28]. This IDE will be used to make the web application.
- **Nginx web server:** an open-source web server[29]. This web server is going to be installed to serve the web application and static files, such as the REST API docs.

• **Raspberry pi Application**

- **Python programming language:** it will be used to program the raspberry pi.
- **SQL programming language:** it will be used in entering the data to the local queue.
- **GPIO library:** a library for controlling any hardware component connected to the GPIO pins[12].
- **SQLite RDBMS:** it is a file-based lightweight RDBMS[30]. It will be used to store data in the local queue.



3.2.2 Hardware Requirements

- **Raspberry Pi**

- Raspberry Pi 3 B+.
- a minimum of 2 GB of RAM.
- a minimum of 10 GB space in SD card.
- a monitor, a keyboard and a mouse, alternatively SSH connection could be established.
- internet connection, either via Wi-Fi or Ethernet cable.
- breadboard, cables, and resistors for circuit.
- RGB LED, solenoid, or any other hardware components satisfying user needs.

- **Server**

- Ubuntu 16.04+ server, we chose digital ocean's. The web server and database server shall be installed here.
- Minimum of 1GB of RAM.
- Minimum of 10GB of available space.

- **Android mobile phone**



3.2.3 Functional Requirements

3.2.3.1 Expert User's Functionalities:

- 3.2.3.1.1. Expert user shall be able to add new hardware to the system.
- 3.2.3.1.2. Expert user shall be able to add new configuration to the system.

3.2.3.2 Android Client's Functionalities:

- 3.2.3.2.1. Android client shall allow user to login or register
- 3.2.3.2.2. Android client shall allow user to add a new raspberry pi to the system
- 3.2.3.2.3. Android client shall allow user to remove a raspberry pi from the system
- 3.2.3.2.4. Android client should get hardware list from webserver.
- 3.2.3.2.5. Android client should get scheduled commands from webserver.
- 3.2.3.2.6. Android client shall be able to submit a new command, might be scheduled, to webserver.
- 3.2.3.2.7. Android client shall be able to delete a scheduled command from webserver.
- 3.2.3.2.8. Android client shall be able to edit a scheduled command from webserver.
- 3.2.3.2.9. Android client shall get responses from webserver automatically.

3.2.3.3 Raspberry pi's Functionalities:

- 3.2.3.3.1. Raspberry pi should get command list each 30 seconds.
- 3.2.3.3.2. Raspberry pi shall be able to update local queue.
- 3.2.3.3.3. Raspberry pi shall execute commands saved in queue.
- 3.2.3.3.4. Raspberry pi shall be able to submit a command response to server.

3.2.3.4 Web application's Functionalities:

- 3.2.3.4.1. Web application should delete executed immediate commands.
- 3.2.3.4.2. Web application should delete read responses.



3.2.4 Non-Functional Requirements

Requirement	Description
Availability	The system shall not be shut down for maintenance for more than 1 minute.
Usability	Except the expert user, all other users of the system shall be able to use the system immediately without much training.
Verifiability	The system shall check the user identity.
Performance and Delay	The raspberry pi shall read user commands each 30 seconds to avoid delay or overhead in the system.
Flexibility	Any hardware component, such as linear solenoid or an infra-red controller, can be added to the system as long as it can be connected to the electric circuit.
Security	Each user can only communicate with assigned raspberry pi. In a similar manner, raspberry pi only reads commands meant for it and not for other ones.
Efficiency	The web server cleans the database and deletes any unneeded rows routinely.

Table 3.1: Non-functional requirements



Table 3.1 shows the non-functional requirements for the system and their description. The most important requirement is flexibility, as it is the core feature differentiating the proposed system from other systems. Next comes security and verifiability. They guarantee that no intruders are going to interrupt the system even if they communicated with the REST API directly without the android app because they won't be authorized -or authenticated- to enter the system. The other requirements are mainly for performance and ease of use.



3.2.5 Structured Diagrams

3.2.5.1 Use Case Diagram

Figure 3.1 shows the use-case diagram of the system. Although the system learning curve is steep, each raspberry pi can be connected to multiple users and vice versa -i.e. many to many relationship-. In other words, in a household of four people, all of them can connect to the same raspberry and control the AC. However, only one of them must be knowledgeable about the basics of electrical engineering and computer science. This person is referred to as the **Expert User**. All other users including the expert user are referred to as **Users**. In this system, there are four main actors:

- **User**: any person using the system. They don't have to have any kind of background information at all. All they need to do is install the android app and clicks buttons on it.
- **Expert user**: a type of user than configure the raspberry pi. They must manually connect the hardware to the GPIO pins, add it to the database and configure it.
- **Android client**: the android application that the user uses. The android app makes requests and takes responses from the web server either by user order, such as deleting a certain command, or by itself such as getting the raspberry pi responses from the webserver periodically.
- **Raspberry Pi**: the computer controlling the hardwares. It communicates with the web server to get the commands periodically then executes it.

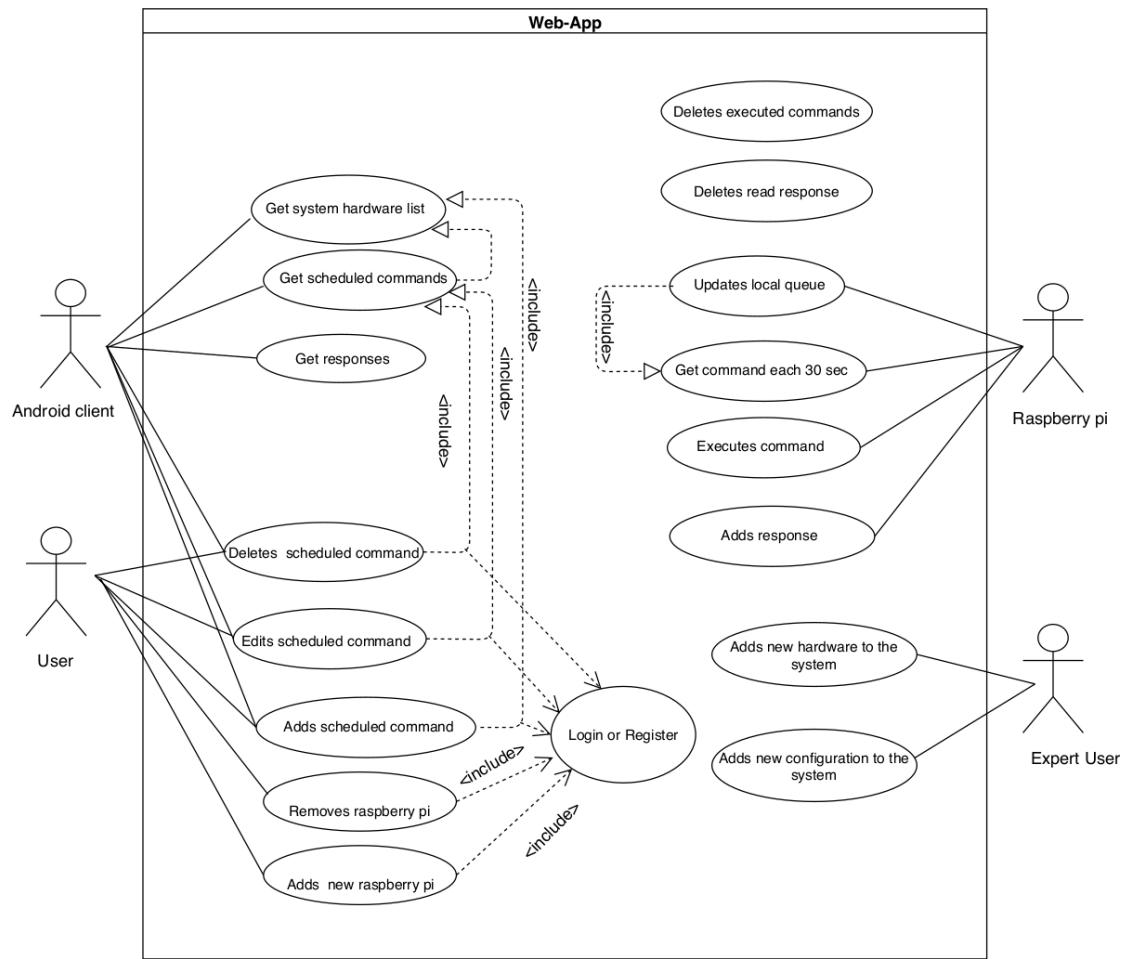


Figure 3.1: Use case diagram



3.2.5.2 Use Case Scenarios & Sequence Diagrams

Expert user adds a new hardware to system

Sequence diagram: *Figure 3.2*

Goal: to add a new hardware device that the user can control to the Raspberry Pi.

Actors: Expert user, Raspberry Pi, Web server.

Precondition: None.

Primary Scenario:

1. Expert user physically connects the new hardware to the Raspberry Pi
2. Expert user inserts new hardware attributes into hardware table in the raspberry local database.
3. Raspberry pi uploads information to the webserver using *POST /hardware*

Variant:

- 1.A. A physical defect might appear in the new hardware, the wires connecting it to the Raspberry Pi, or its ports, all resulting in the new hardware not working correctly.

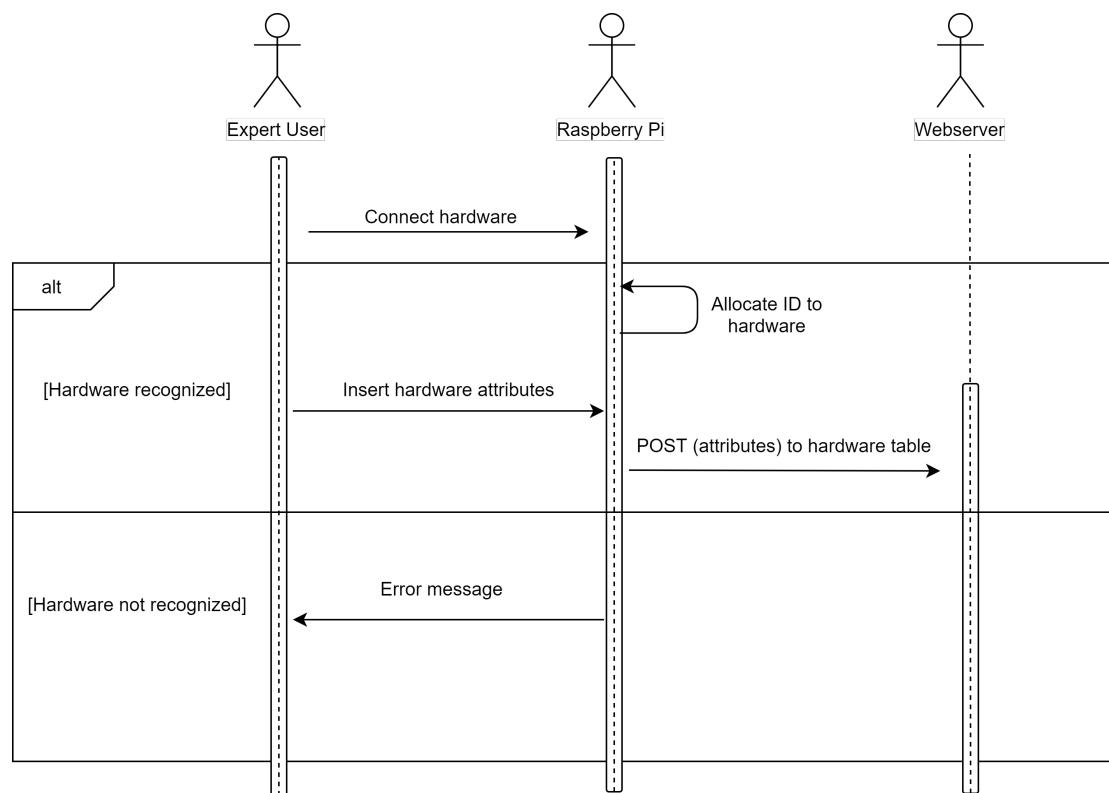


Figure 3.2: sequence diagram: Expert user adds a new hardware to system



Expert user adds a new configuration to system

Sequence diagram: *Figure 3.3*

Goal: to add a new configuration to a new hardware.

Actors: Expert user, Raspberry Pi, Web server.

Precondition: Hardware has been added to the system successfully.

Primary Scenario:

1. Expert user inserts new configuration attributes into configuration table in the raspberry local database.
2. Raspberry pi uploads information to the webserver using *POST /configuration*

Variant:

- 1.A. A physical defect might appear in the new hardware, the wires connecting it to the Raspberry Pi, or its ports, all resulting in the new hardware not working correctly.

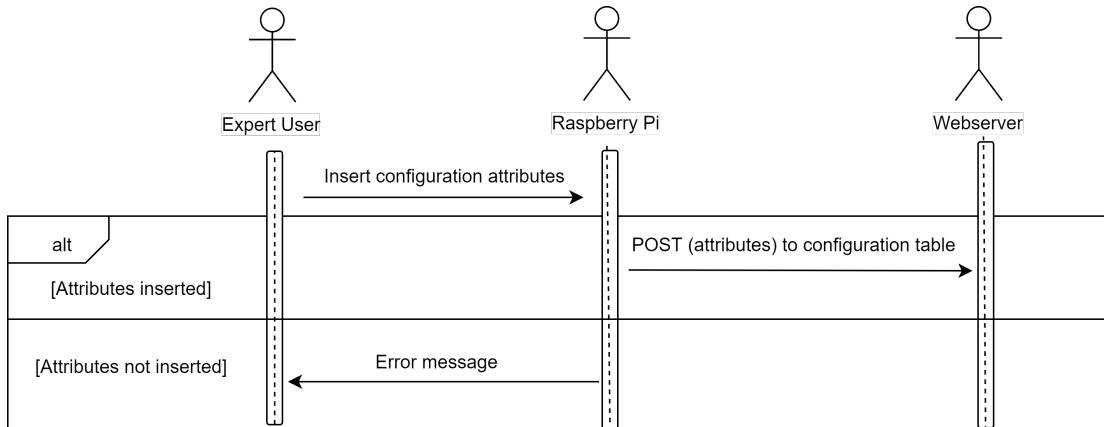


Figure 3.3: sequence diagram: Expert user adds a new configuration to system



User login or register

Sequence diagram: *Figure 3.4*

Goal: to login a user if exists, or make a new user otherwise.

Actors: Android app, Web server.

Precondition: user opens android app.

Primary Scenario:

1. android app prompt user to enter credentials.
2. android POST data to webserver's endpoint `/login`
3. web app checks email and password. If the user already exists and credentials are correct, return token. If the user doesn't exist create user and return token.
4. Android calls web server's endpoint `/raspberry` using the token in the authorization header to get the list of raspberry pi's related to the user
5. Android display this array to user in UI.

Variant:

- *. user might exit android app.
- 2.A. android app might fail to connect to the internet
- 3.A. email might not match password, return error message

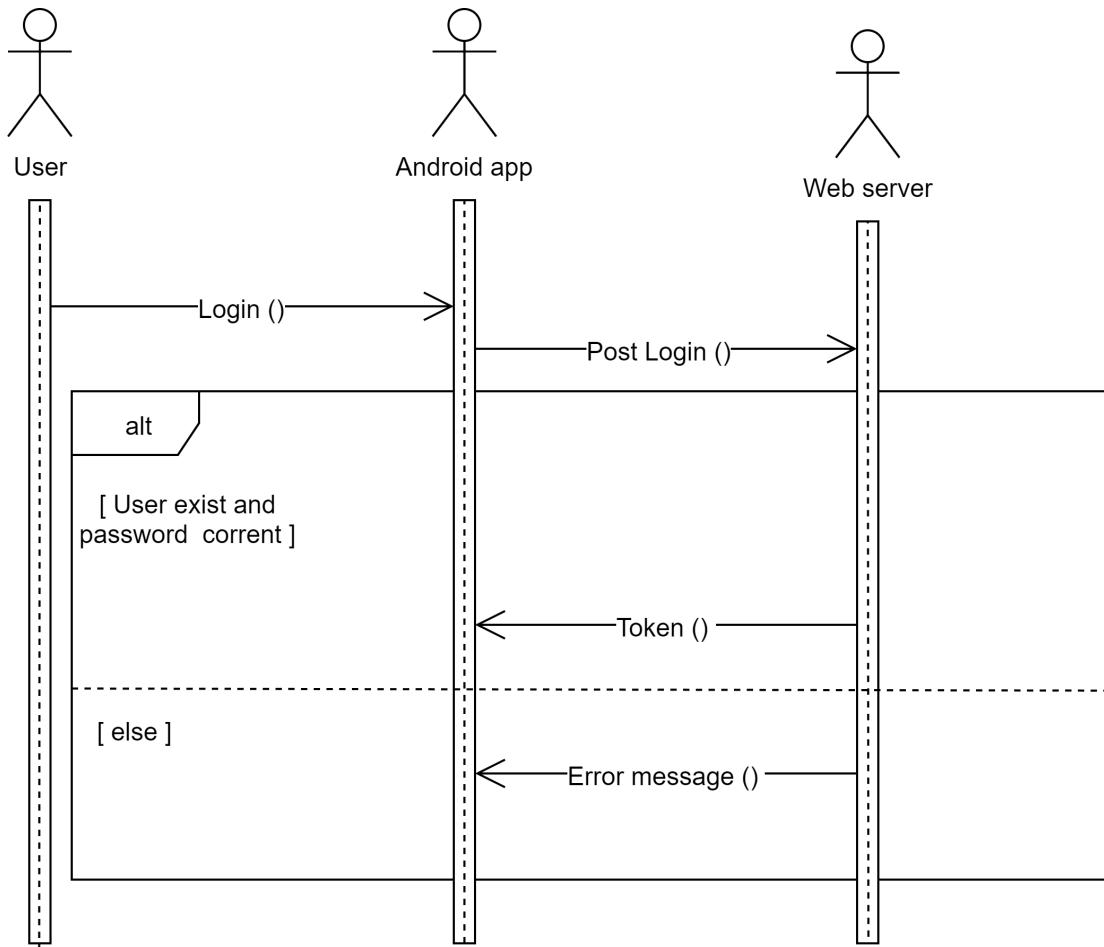


Figure 3.4: sequence diagram: User login or register



User adds new raspberry pi

Sequence diagram: *Figure 3.5*

Goal: to add a raspberry pi to the system

Actors: Android app, Web server.

Precondition: user logged in.

Primary Scenario:

1. user clicks on add new raspberry pi button.
2. android app opens camera for the user to scan the QR code on the raspberry pi.
3. once the code is scanned, the android app POST the raspberry pi id to the server's endpoint **/raspberry**.

Variant:

- *. user might exit android app.
- 2.A. user might close camera or scan incorrect item.
- 3.A. android app might fail to connect to the internet

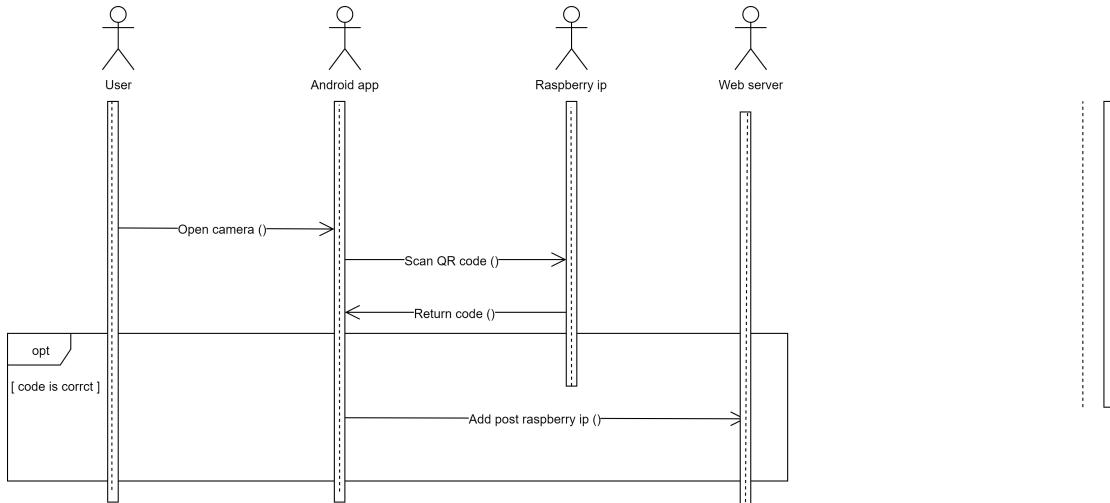


Figure 3.5: sequence diagram: User adds new raspberry pi



User removes raspberry pi

Sequence diagram: *Figure 3.6*

Goal: to remove a raspberry pi from the system.

Actors: Android app, Web server.

Precondition: user logged in.

Primary Scenario:

1. user clicks on delete raspberry pi.
2. Android client requests URL `/raspberry/{id}` with HTTP method *DELETE* where `{id}` is the raspberry pi selected id.
3. webserver receives requests and deletes the raspberry pi from user.

Variant:

- *. user might exit android app.
- 2.A. android app might fail to connect to the internet

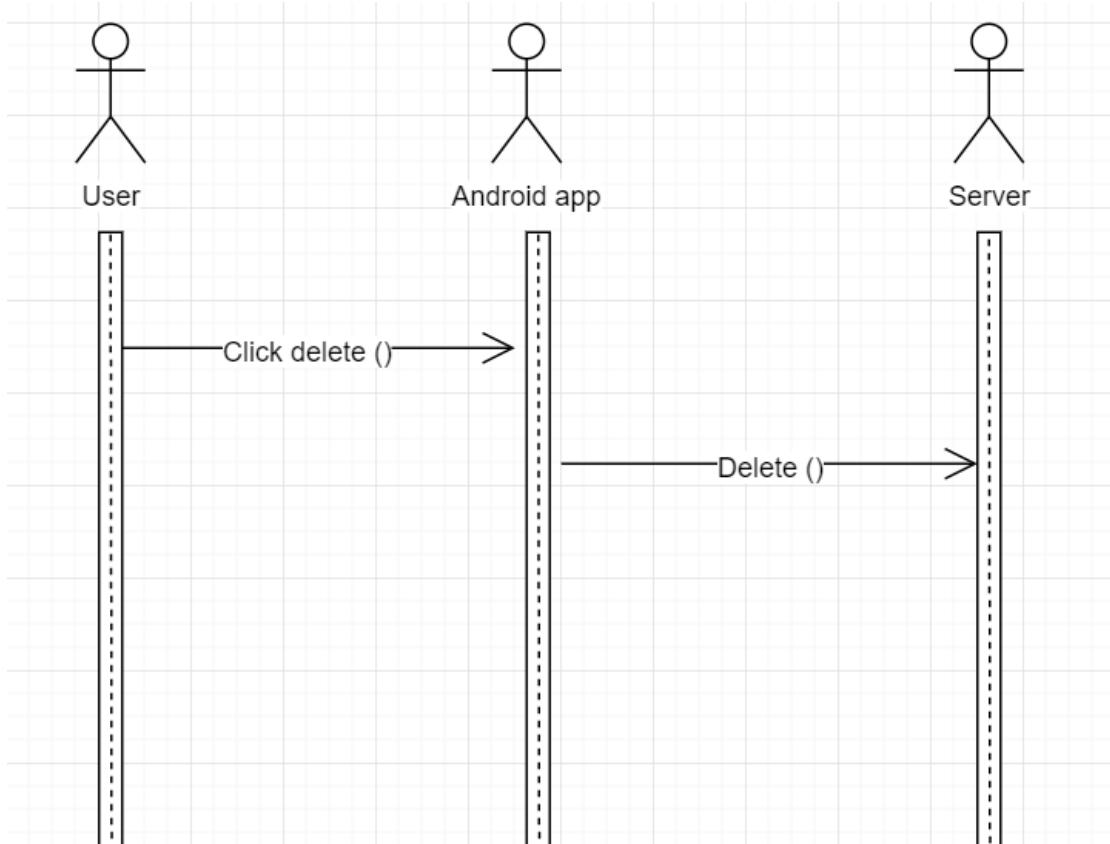


Figure 3.6: sequence diagram: User removes raspberry pi



Android app gets hardware list

Sequence diagram: *Figure 3.7*

Goal: to get all hardware in the system from webserver.

Actors: Android app, Web server.

Precondition: user logged in successfully

Primary Scenario:

1. user clicks on a raspberry pi.
2. android app calls the webserver endpoint `/hardware`
3. webserver fetches data from database using `hardware.index()`
4. webserver responses to request with an array of hardwares in the response's body
5. Android display this array to user in UI.

Variant:

- *. user might exit android app.
- 1.A. android app might fail to connect to the internet

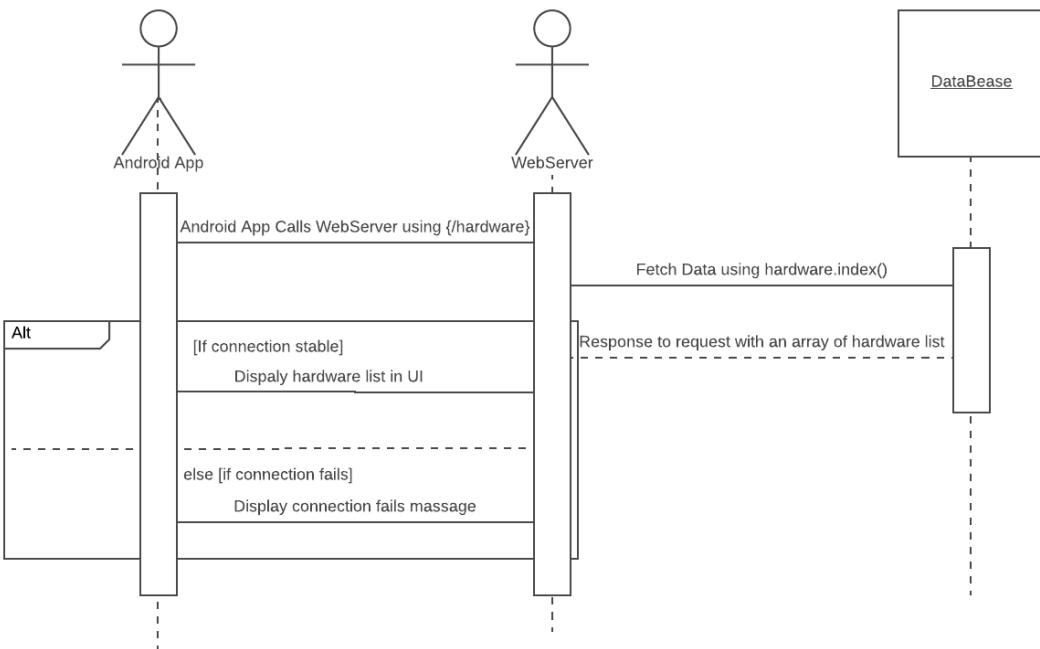


Figure 3.7: sequence diagram: Android app gets hardware list



Android app gets command list

Sequence diagram: *Figure 3.8*

Goal: to get all commands in the system related to selected hardware.

Actors: Android app, Web server.

Precondition: android app gets hardware list

Primary Scenario:

1. user selects hardware from UI.
2. android app calls the webserver endpoint `/hardware/{hardwareId}/command`, where *hardwareId* is the id of the hardware the user selected.
3. webserver fetches data from database using `command.indexByHardwareId({hardwareId})`
4. webserver responds to request with an array of commands in the response's body

Variant:

- *. user might exit android app.
- 2.A. android app might fail to connect to the internet

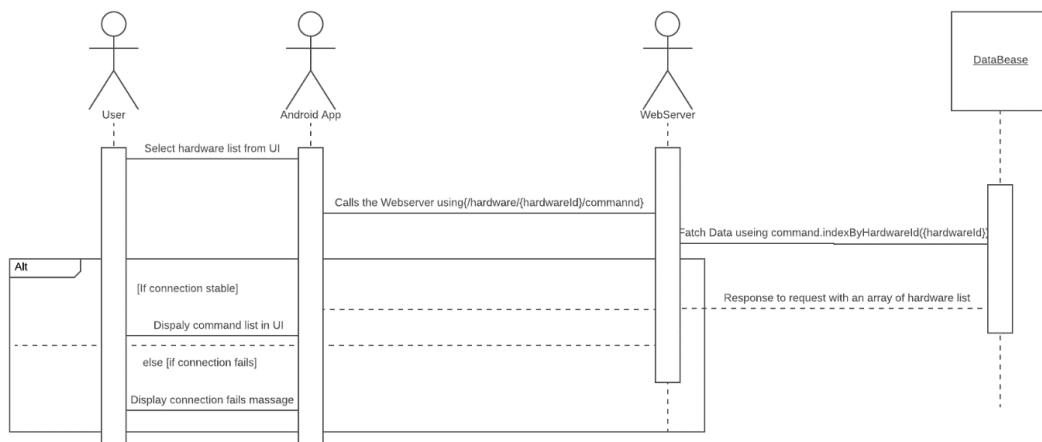


Figure 3.8: sequence diagram: Android app gets command list



Android app deletes a scheduled command

Sequence diagram: *Figure 3.9*

Goal: to delete a scheduled command from webserver related to selected hardware.

Actors: User, Android app, Web server

Precondition: android app gets all scheduled command.

Primary Scenario:

1. User clicks on delete a scheduled command.
2. Android client requests URL `/command/{id}` with HTTP method *DELETE*
where `{id}` is the command selected id.
3. webserver receives requests and deletes the command.

Variant:

- 1.A. user might not have a schedule command.
- 2.A. android app might fail to connect to the internet

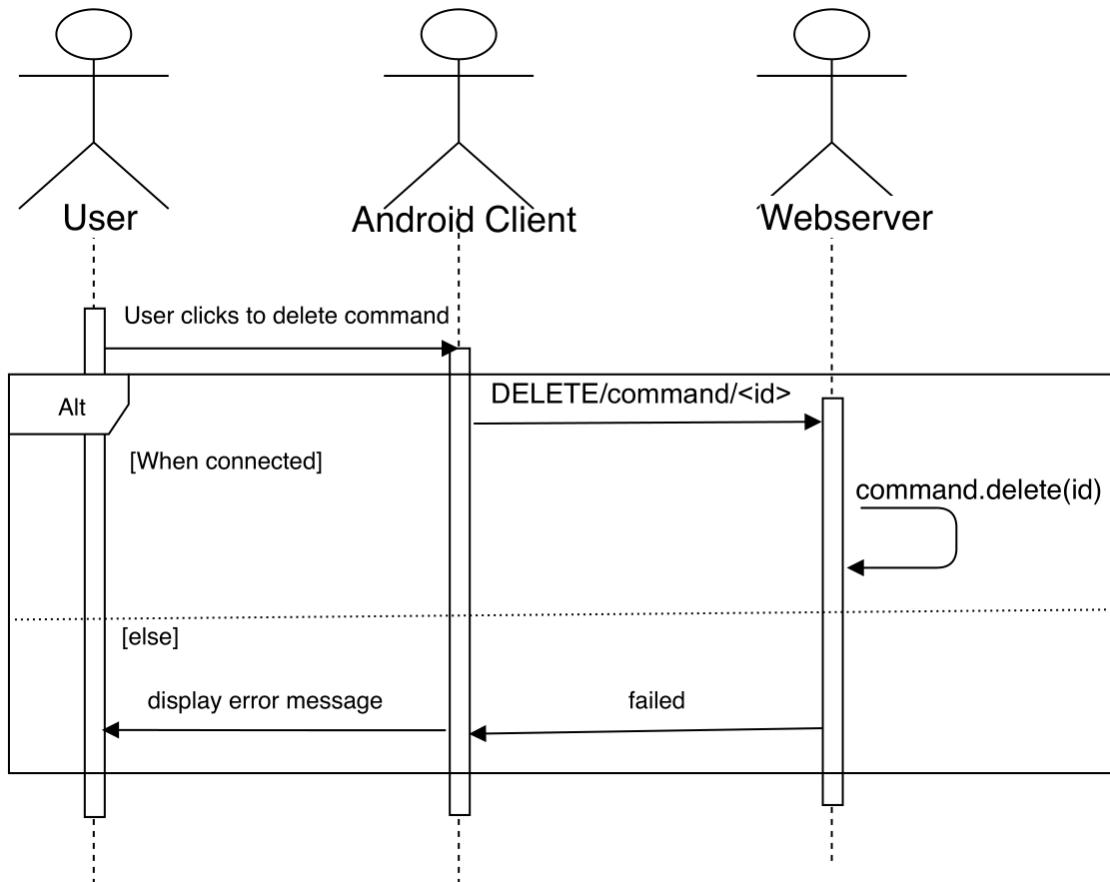


Figure 3.9: sequence diagram: Android app deletes a scheduled command



Android app edits a scheduled command

Sequence diagram: *Figure 3.10*

Goal: to edit a scheduled command from webserver related to selected hardware.

Actors: User, Android app, Web server

Precondition: android app gets all scheduled command.

Primary Scenario:

1. User clicks on editing a scheduled command.
2. Android client requests URL `/command/{id}` with HTTP method `PUT` where `{id}` is the command selected id.
3. webserver receives requests and edits the command.

Variant:

- 1.A. user might not have a schedule command.
- 2.A. android app might fail to connect to the internet

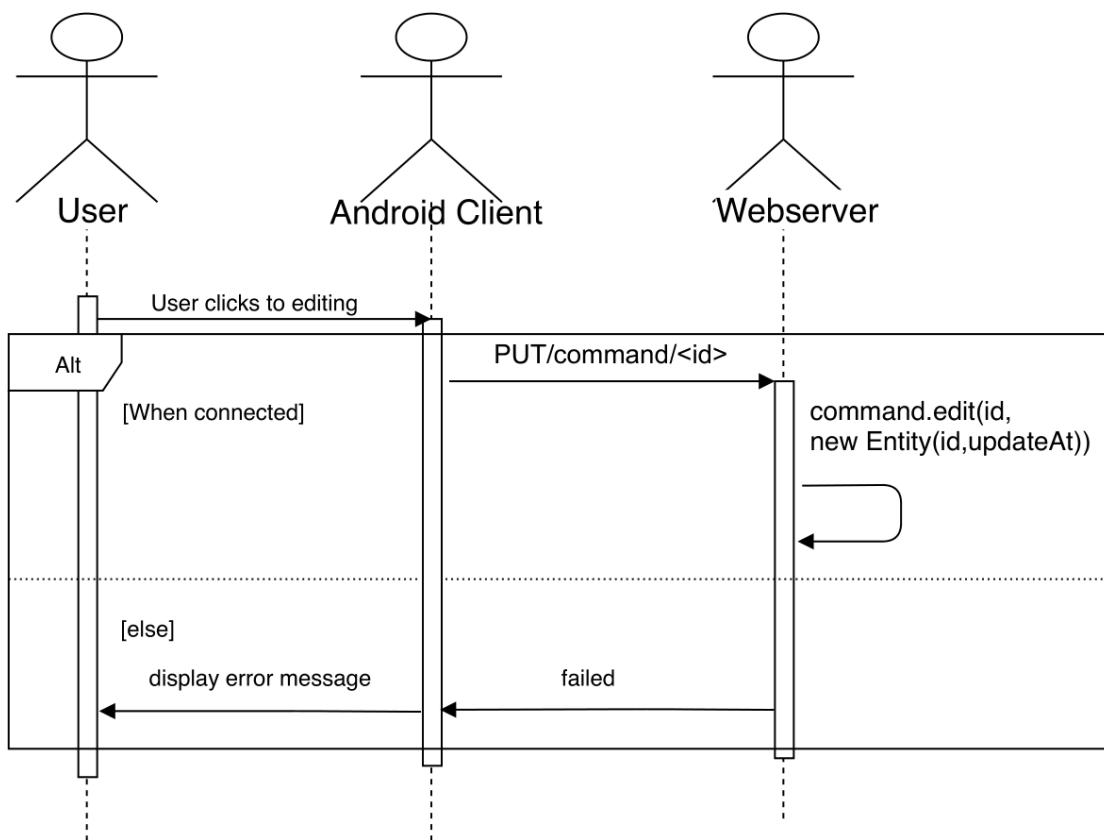


Figure 3.10: sequence diagram: Android app edits a scheduled command



Android app adds a new command

Sequence diagram: *Figure 3.11*

Goal: to add a new command to webserver.

Actors: User, Android app, Web server

Precondition: android app gets all hardwares and user chooses the hardware to add command to.

Primary Scenario:

1. User clicks on adding a new command.
2. activity appears taking the new command attributes, such as schedule or configuration.
3. Android client requests URL `/command` with HTTP method *POST*.
4. webserver receives requests and adds the command.

Variant:

- *. internet might disconnect

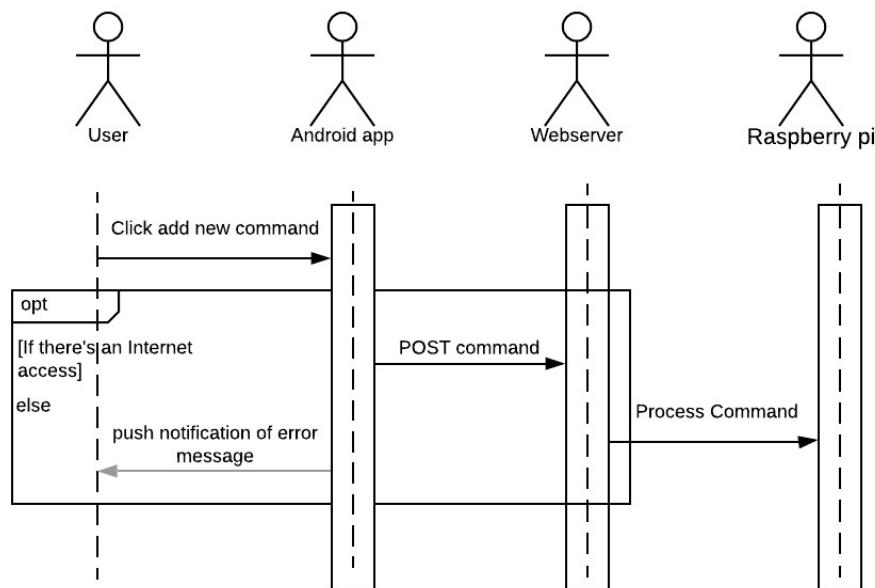


Figure 3.11: sequence diagram: Android app adds a new command



Android app gets responses

Sequence diagram: *Figure 3.12*

Goal: to get the result of the command execution to the user as a notification

Actors: Android app, Web server

Precondition: None

Primary Scenario:

1. Android client requests URL `/response` every 5 or 10 minutes.
2. if responses exist, android app shows it to user as a push notification.

Variant:

- *. internet might disconnect

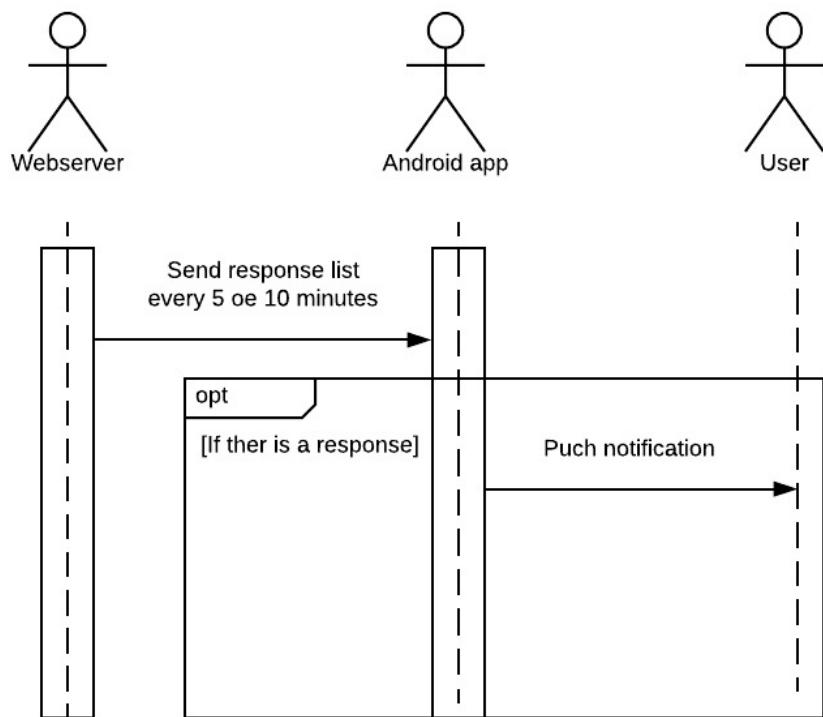


Figure 3.12: sequence diagram: Android app gets responses



Raspberry pi gets command list

Sequence diagram: *Figure 3.13*

Goal: to get all commands in the system.

Actors: Raspberry pi, Web server.

Precondition: None

Primary Scenario:

1. Raspberry pi calls the webserver endpoint `/command` to get all the commands
2. webserver fetches data from database using `command.index()`
3. webserver responds to request with an array of commands in the response's body

Variant:

- *. raspberry pi might fail to connect to the internet

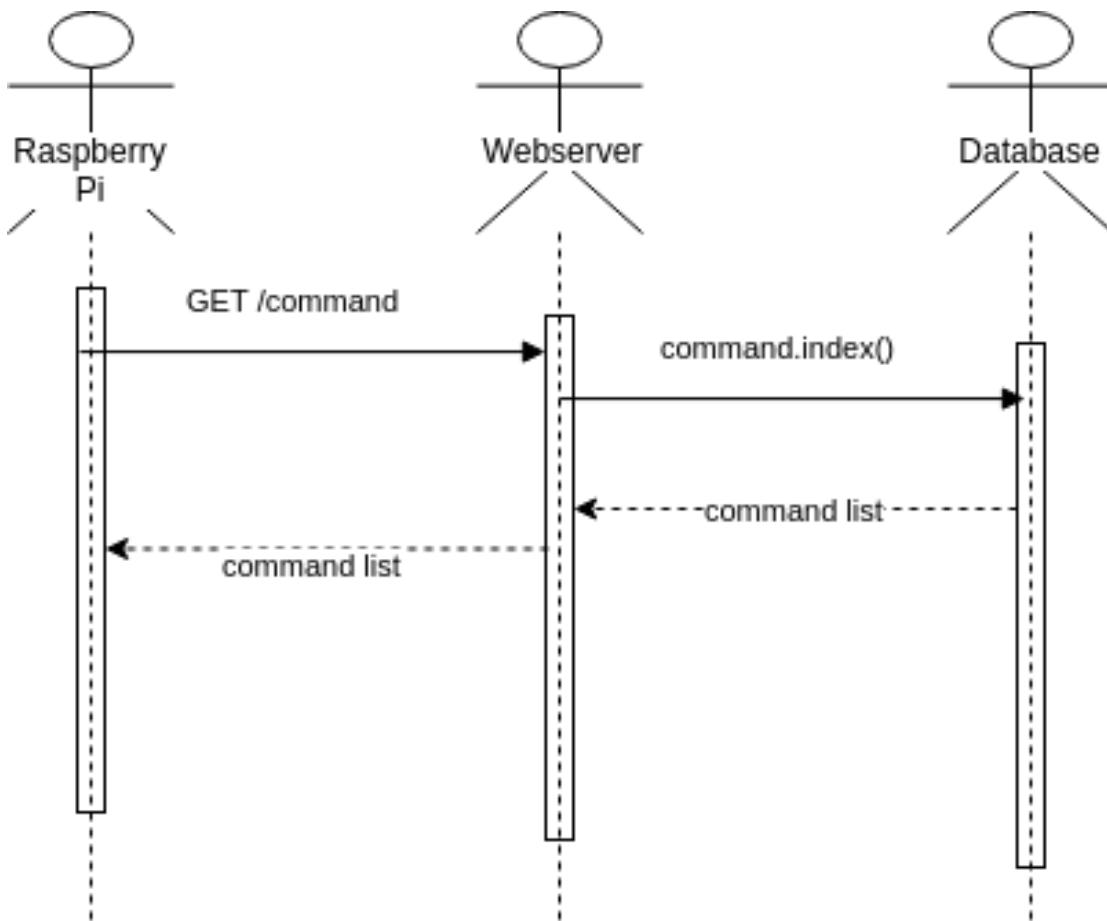


Figure 3.13: sequence diagram: Raspberry pi gets commands from server



Raspberry pi updates local queue

Sequence diagram: *Figure 3.14*

Goal: to update local queue.

Actors: Raspberry pi.

Precondition: raspberry pi got command List from web server.

Primary Scenario:

1. Raspberry pi gets a command from list.
2. If command is immediate,i.e. has no schedule, the execution time for the processed command will be the time the command was created - accessed using `command.getUpdateAt()` -.
3. else -meaning command has a schedule- and today is part of the schedule days, add to queue with execution Time set to today at the schedule's time.
4. order queue by execution Time

Variant:

- 3.A. if today is not part of the days, discard command

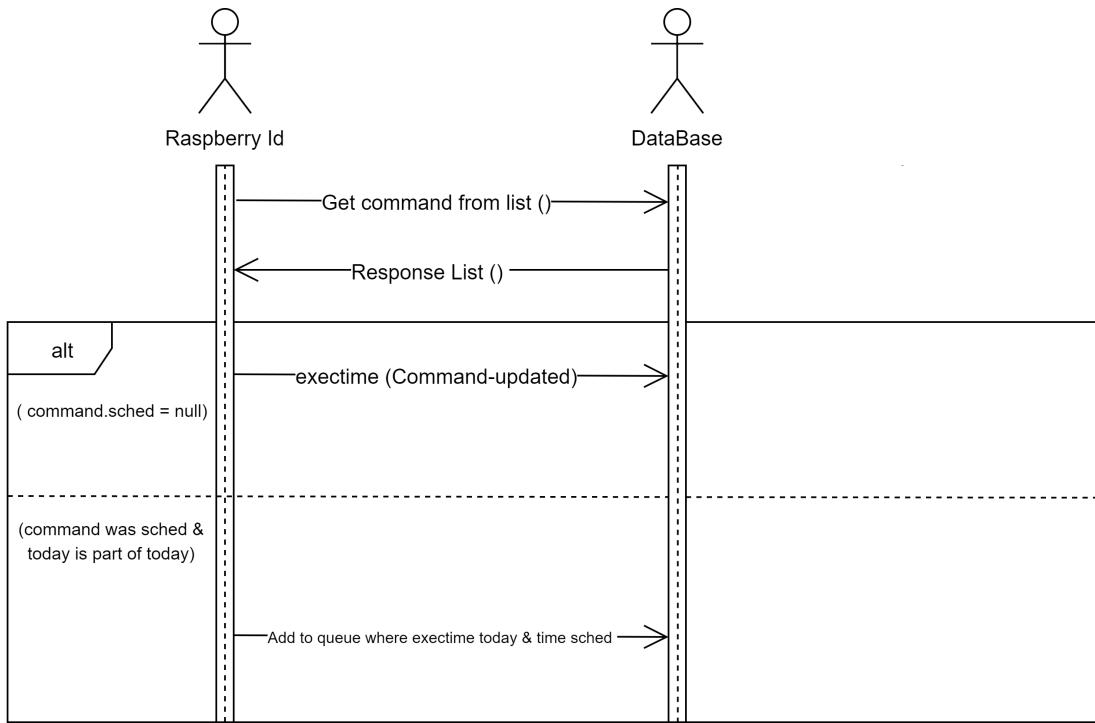


Figure 3.14: sequence diagram: Raspberry pi updates local queue



Raspberry pi executes commands

Sequence diagram: *Figure 3.15*

Goal: to execute commands saved in the local queue on a timely fashion.

Actors: Raspberry Pi.

Precondition: raspberry pi got the commands from webserver, updated local database.

Primary Scenario:

1. raspberry pi fetches first processed command saved in local queue.
2. raspberry pi calls the function `EexecuteCommand()` and pass it the command as a parameter.
3. raspberry pi look for the pin connecting the hardware and apply the desired configuration to it.

Variant:

- 1.A. local queue might be empty, no action is taken.
- 2.A. a hardware error might happen, such as disconnected cable.

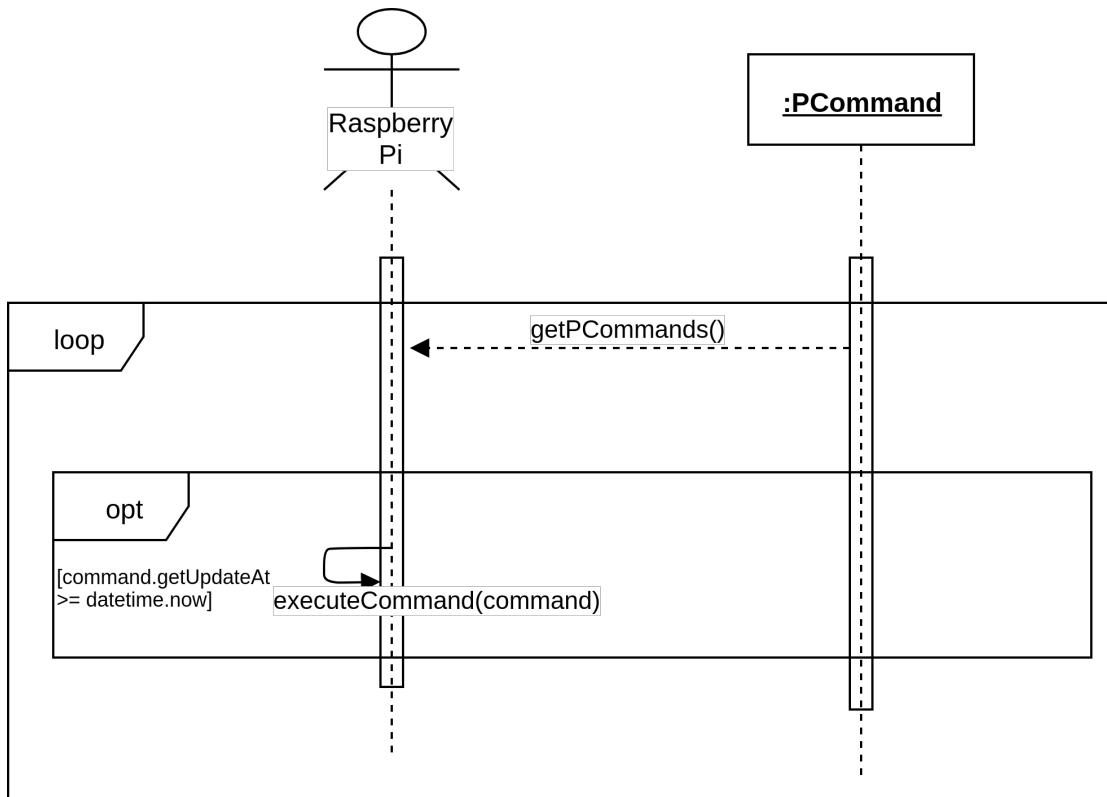


Figure 3.15: sequence diagram: Raspberry pi executes commands



Raspberry pi submits a command response to server

Sequence diagram: *Figure 3.16*

Goal: to submit the result of the command execution to the server by saving it as a response row.

Actors: Raspberry Pi, webserver.

Precondition: raspberry pi executed commands.

Primary Scenario:

1. raspberry pi determines whether the execution was successful.
2. raspberry pi includes a message if any.
3. raspberry pi requests the webserver endpoint **/response** with the method *POST*, needed parameter are added to the request body.
4. webserver receives request, a new row to the database response table is added and ready for the android client to read.

Variant:

- *. internet might disconnect.

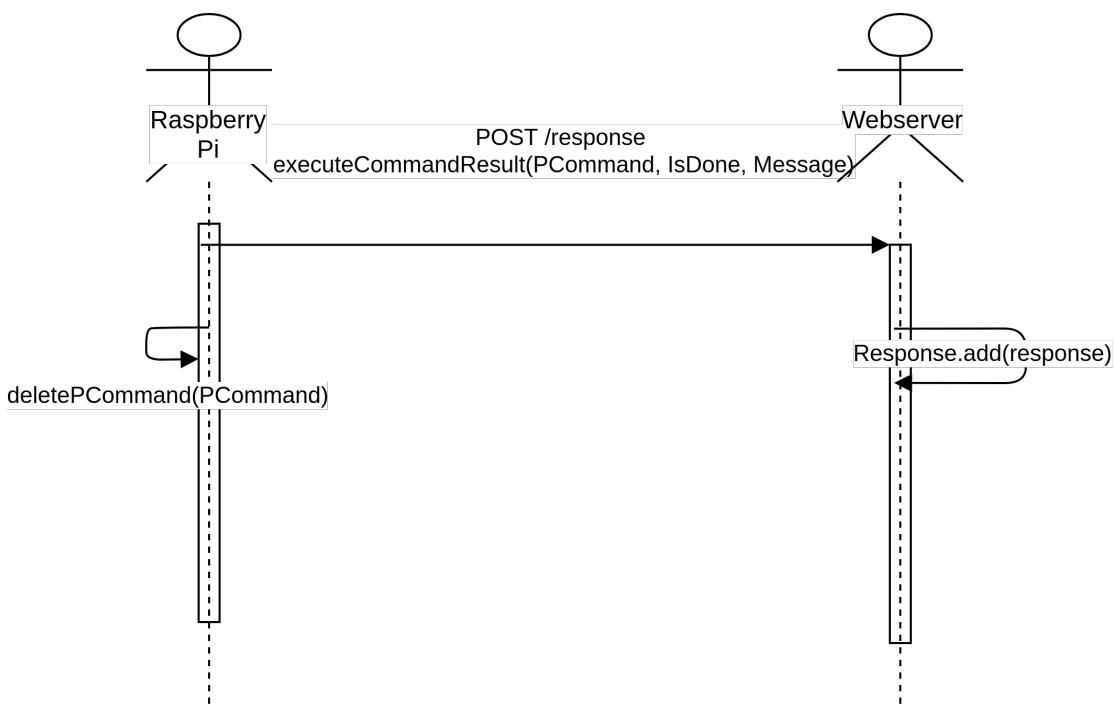


Figure 3.16: sequence diagram: Raspberry pi submits a command response to server



Web application deletes executed immediate commands

Sequence diagram: *Figure 3.17*

Goal: to delete all executed immediate commands.

Actors: Web application, database.

Precondition: None

Primary Scenario:

1. Web application joins between a command table and Response table from database.
2. Web application will select the immediate command from database using `Command.scheduled=null`.
3. Web application checks if response has been read by the user. using `response.isRead()` then, deletes command.

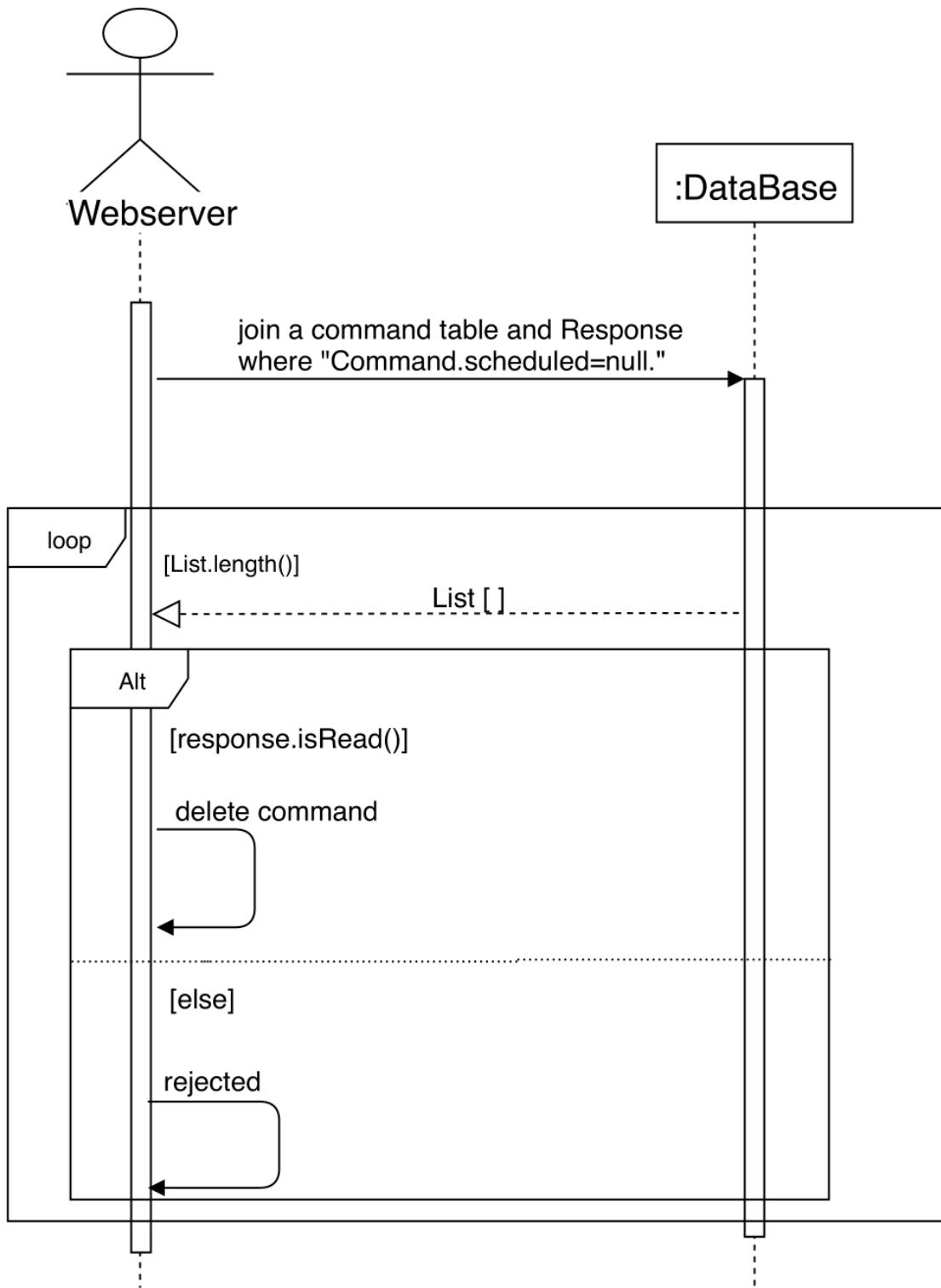


Figure 3.17: sequence diagram: Web application deletes executed immediate commands



Web application deletes read responses

Sequence diagram: *Figure 3.18*

Goal: to delete all read responses.

Actors: Web application, database.

Precondition: None

Primary Scenario:

1. Web application fetches responses from database
2. if response has been read by the user, delete it.

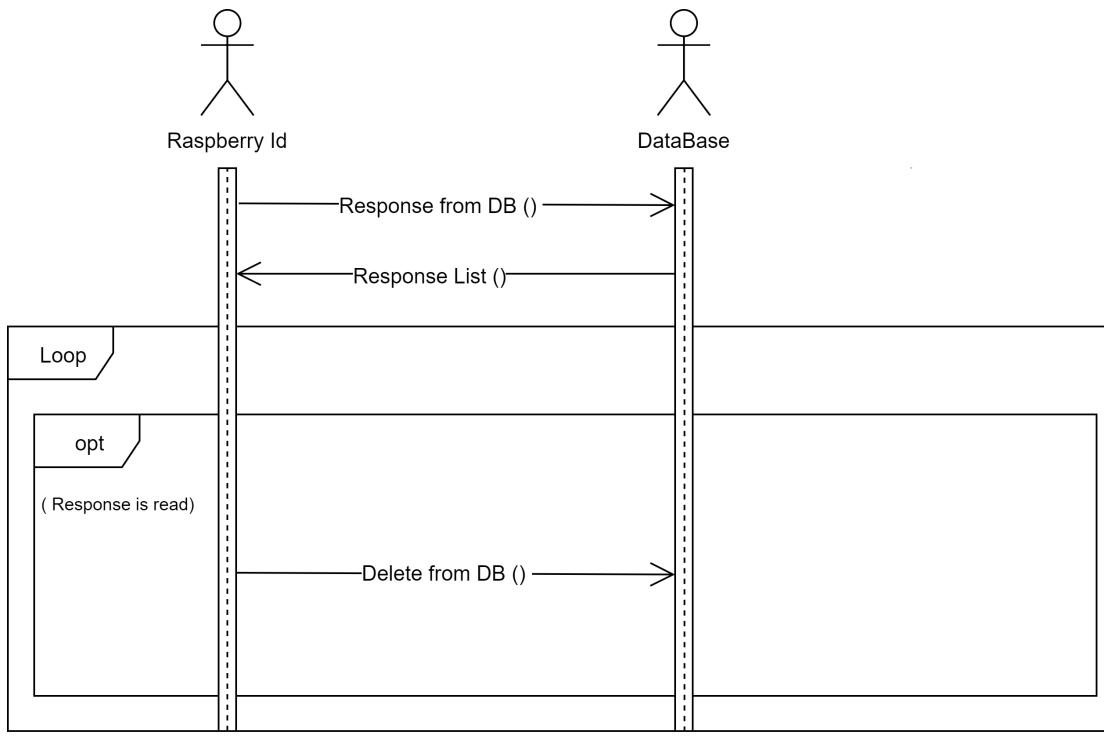


Figure 3.18: sequence diagram: Web application deletes responses



3.2.5.3 Flowchart Diagram

Figure 3.19 shows the flow chart for the android app, and figure 3.20 shows the raspberry pi's flowchart.

- **Raspberry pi:** Every 30 seconds, raspberry first requests command list from the server, the web application then fetches the list from the database and returns it. Raspberry pi updates the local queue with them. It then checks the first command in the queue, if it was due it gets executed and post the response to the server. After execution, it checks the first in queue again.
- **Android app:** when android app is first opened, it requests hardware list from the webserver. After the web app fetches the hardware list from the database, it returns it to the android app, which displays the hardwares to the user. If the user clicks in a hardware, a new activity is opened containing the hardware information and the command list for that hardware. The user then can do 3 things:
 1. Edit a certain command: android request PUT to the command chosen from the server.
 2. Delete a certain command: android request DELETE to the command chosen from the server.
 3. Add a new command: android shows a new activity prompting the user for the command information, then it requests POST command to the server.

Regardless of the choice, the web app then process the request.

- **Android app back work:** each 5 or 10 minutes, android requests the responses from the webserver. When there is a response to be read, it shows it to the user as a push notification.

- **Web app:** when a command has no schedule and it was executed, the web app immediately deletes it from the database. Web app also deletes any read response.

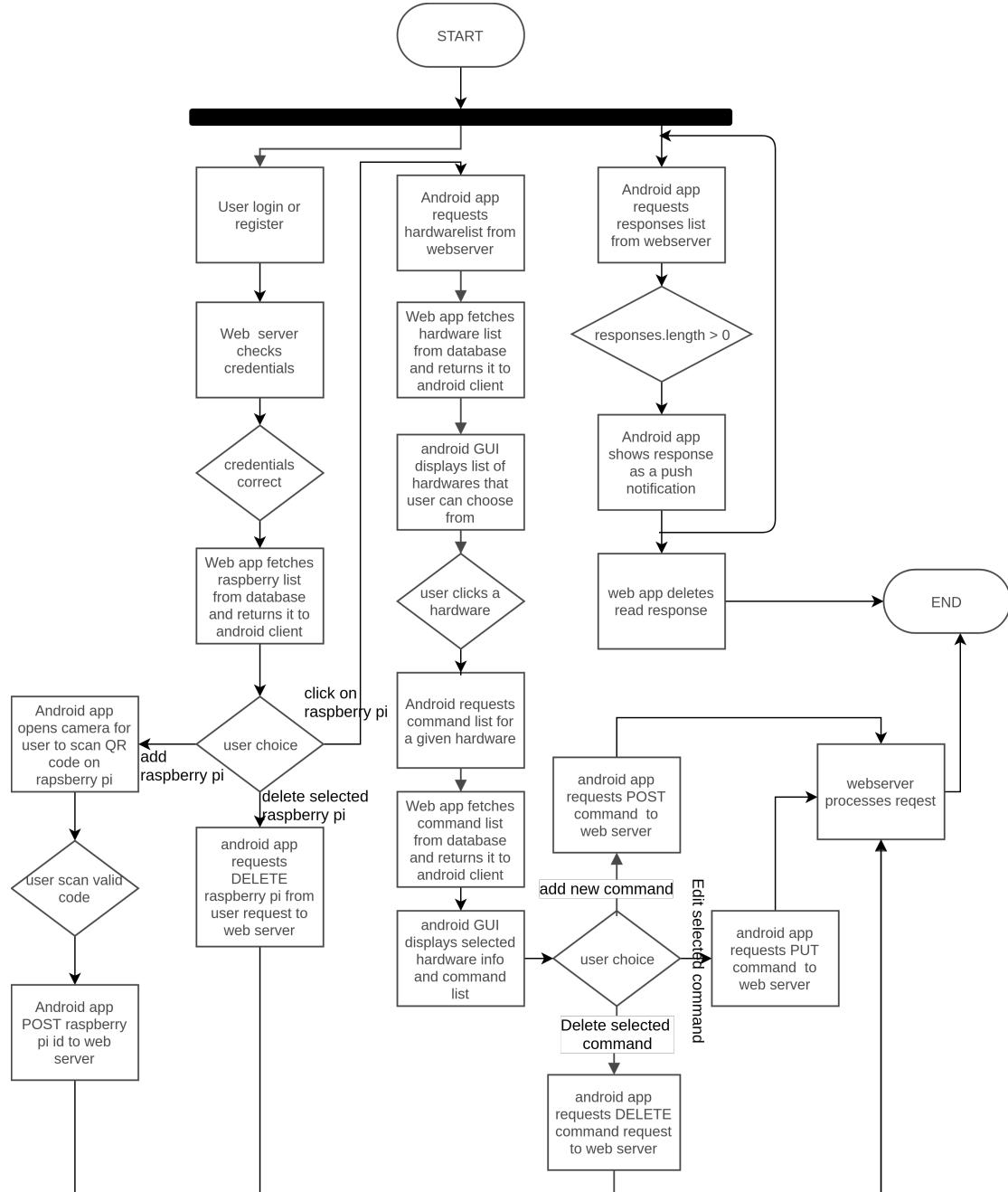


Figure 3.19: Flow chart - Android app

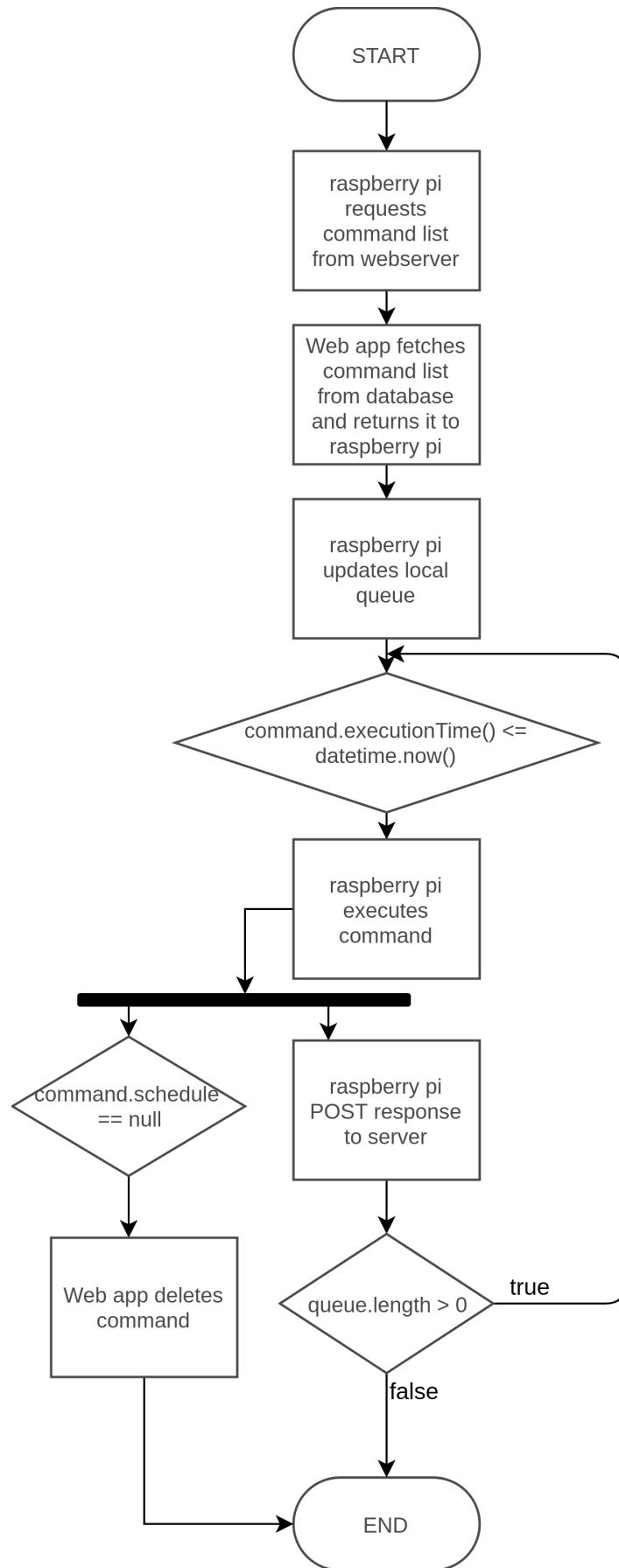


Figure 3.20: Flow chart - Raspberry pi



3.2.5.4 Entity Relationship Diagram

In our system, there are two databases: the *system database* stored in the web server, and *the queue*, a local database for the raspberry pi. To make understanding the databases' easier, we represented it using entity relationship diagram. It is a graphical representation that demonstrates the relationship between concept, people, places, objects or events inside a system. The main components are the entities, relationship and cardinality. Entities are concepts or object that need their data stored. Cardinality defines that relationship in terms of numbers[31].

- **System database:** This database is the main database. It will be stored in the web server and gets accessed by the android client and the raspberry pi. There are 7 main entities:

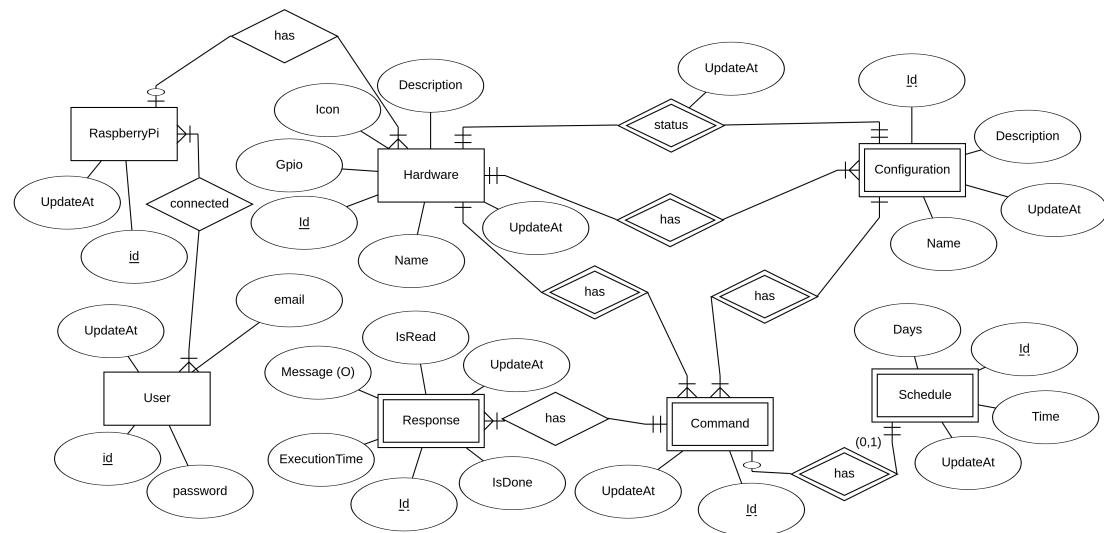
1. **User:** to manage access, a user entity is essential. IT will store the user credentials.
2. **RaspberryPi:** Here, the data for each physical raspberry pi is saved. Each user can have many raspberry pis and each raspberry can belong to multiple people.
3. **Hardware:** it will store information related to the physical components connected to a raspberry pi. Such as LED lights, linear solenoid or an infra-red controller.
4. **Configuration:** hardwares can have different states. For example, a LED light could be on or off. An RGB LED can be ON on a certain color, or off. Configurations save the possible states a hardware can be in.
5. **Command:** users can change raspberry pi's hardwares to be in a certain configuration. These commands are stored here. The users requests could either be instant, or scheduled.
6. **Schedule:** since commands can be scheduled, those data should be saved here. The user can choose the days and time a command fires.



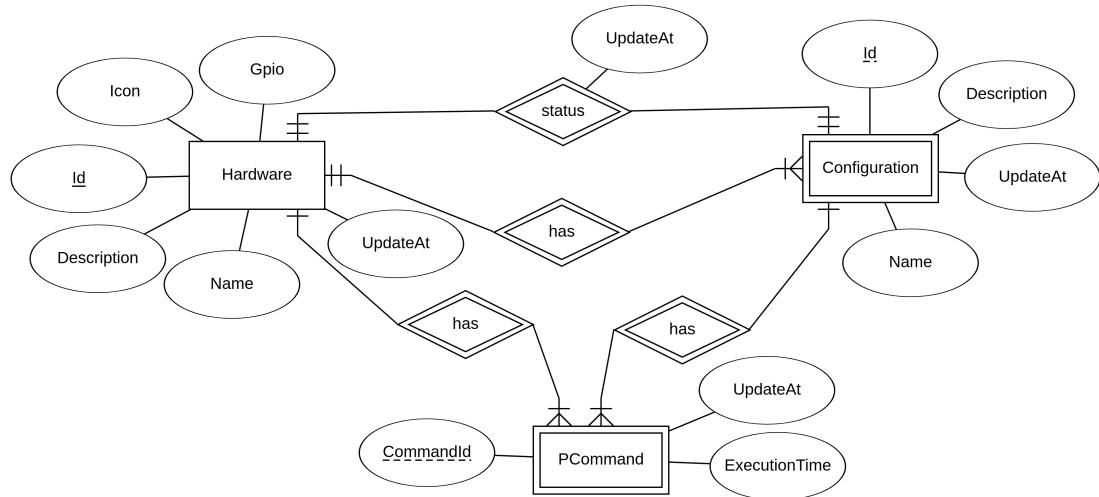
7. Response: when raspberry pi finish executing a command, it issues a response.

- **Local Queue:** This database is stored locally in raspberry pi. After raspberry gets user commands from server, it processes them and orders them based on their execution time.
 1. **Hardware:** similar to the server's hardware list, this entity stores the hardware connected. The system database gets data related to hardware from this entity
 2. **Configuration:** this table stores the possible configuration. When the table is edited, the server database is updated.
 3. **PCommand:** only the commands to be executed are stored here, once a command is executed, it gets deleted from the queue. This entity contains processed commands. i.e each queue stores the exact date and time a command will be executed instead of the schedule information.

Figure ?? shows the Entity-relationship diagram.



(a) System database

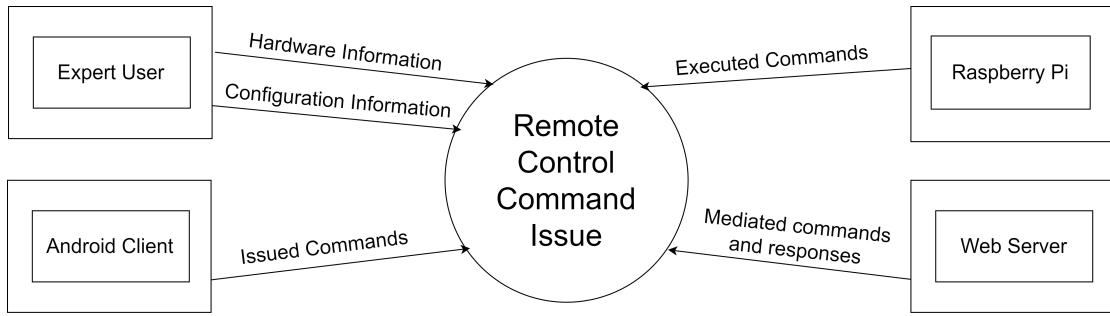


(b) Local queue

Figure 3.21: Entity-relationship diagrams



3.2.5.5 Dataflow Diagram



Level 0 DFD

Figure 3.22: Dataflow diagram - Level 0

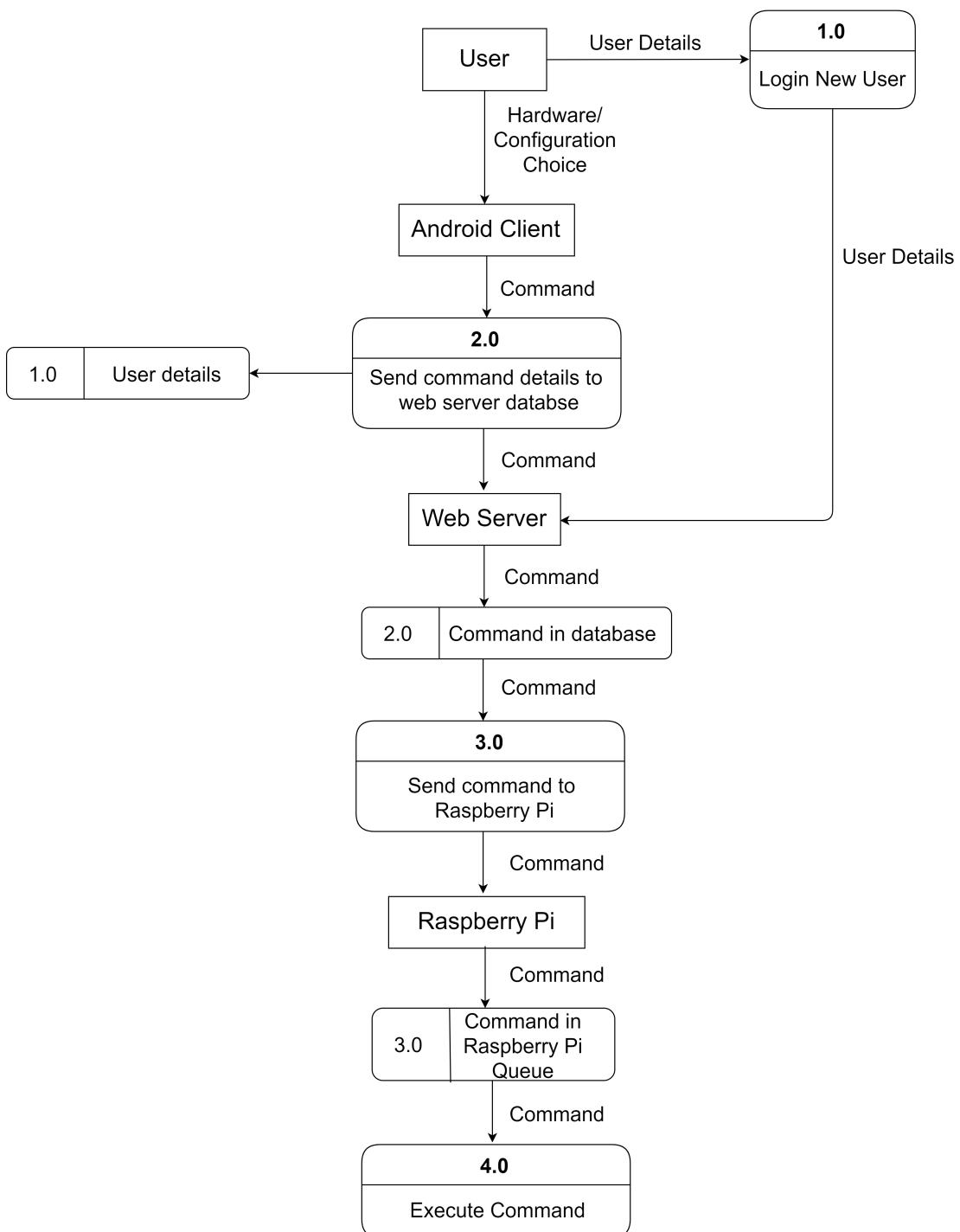


Figure 3.23: Dataflow diagram - Level 1

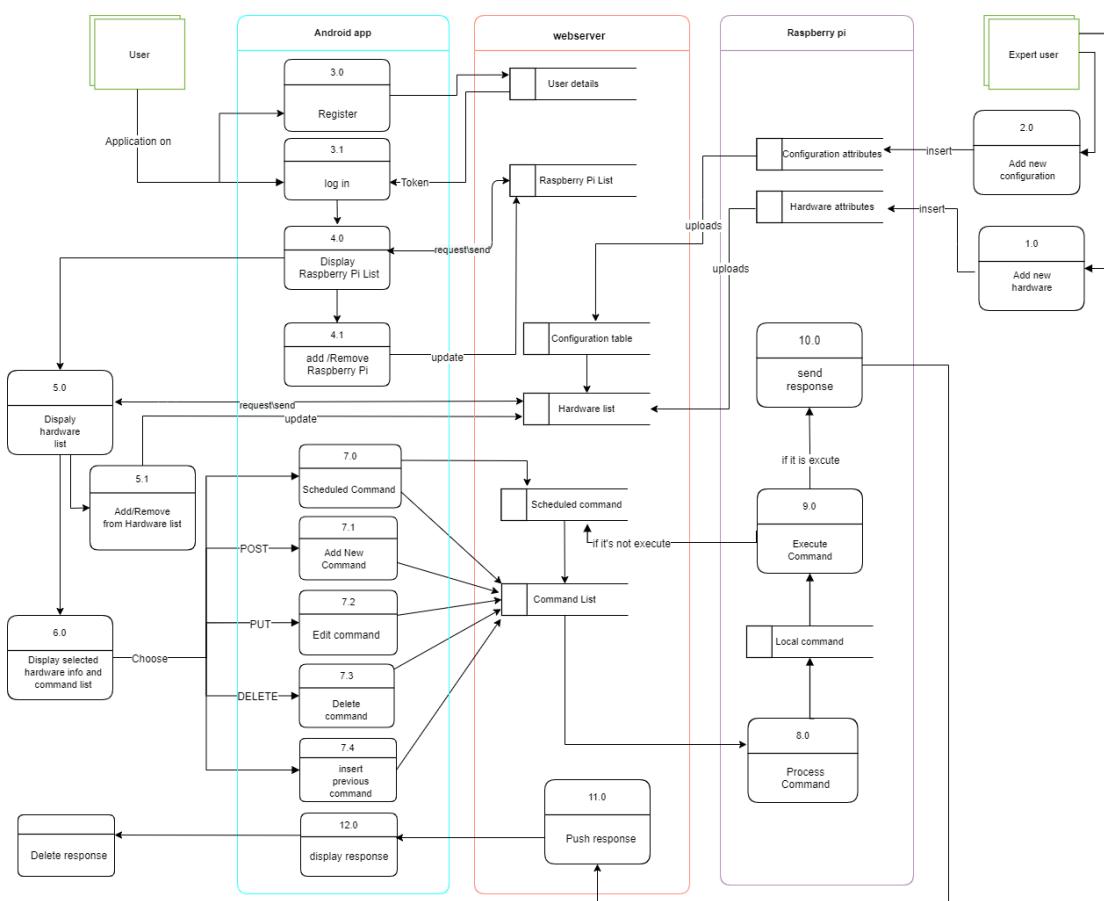


Figure 3.24: Dataflow diagram - Level 2



3.2.6 Object-Oriented Diagrams

3.2.6.1 Class Diagram

Figure 3.25 shows the class diagram for the web application, and *figure 3.26* shows the class diagram for raspberry pi. The classes are essentially wrappers for the databases tables, except for the class **RaspberryPi**, where the functions for controlling the hardwares are.

The classes which represent database's tables inherit from a **BaseEntity** generic class. This generic class has the entity's id and **updateAt** property, which determines when the entity was created/last updated. This generic class is important as it has the methods needed to communicate with the web server. An example is `hardware.edit(1)` , which corresponds to the REST API **PUT /hardware/1** .

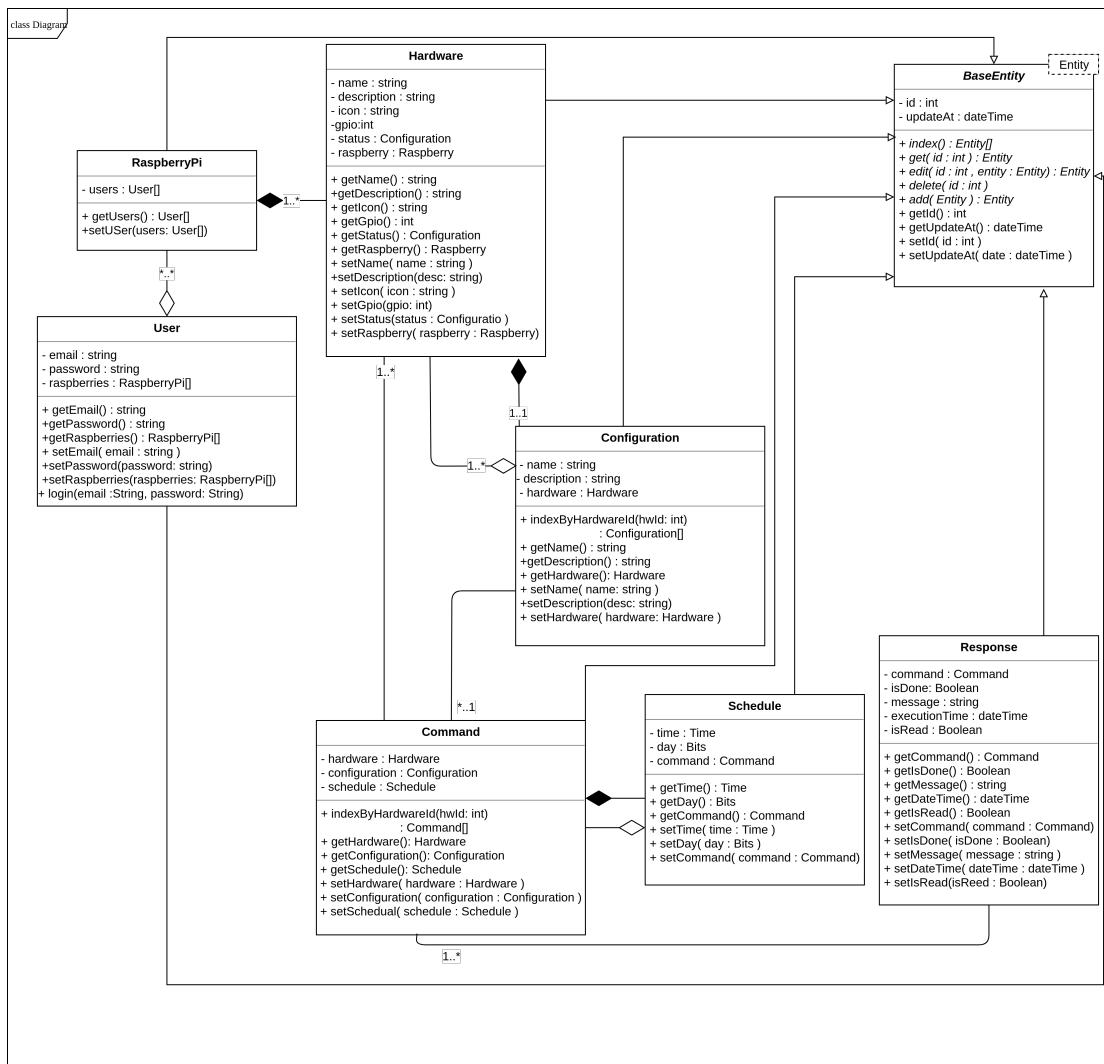


Figure 3.25: Class diagram for the web application

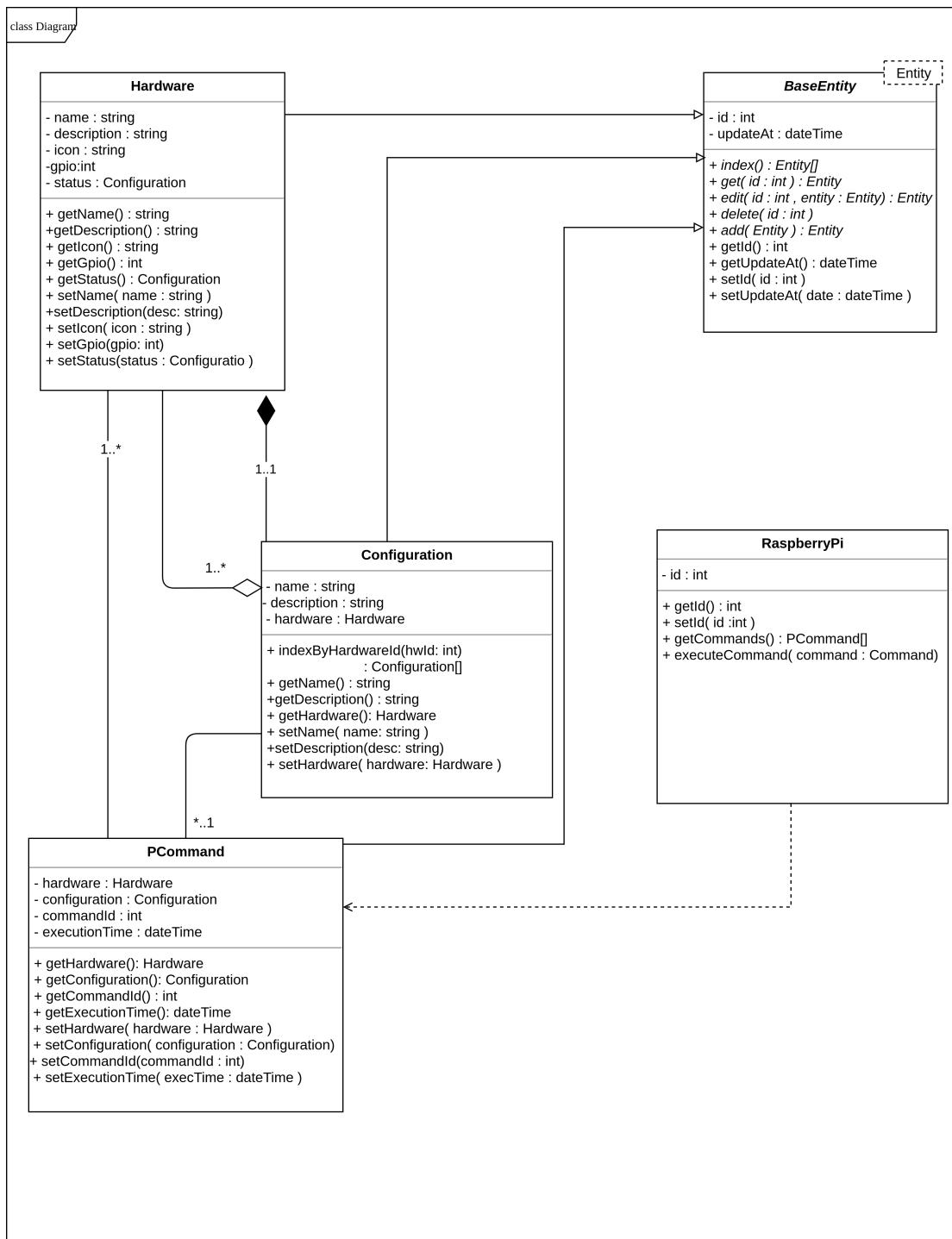


Figure 3.26: Class diagram for the raspberry pi



3.2.6.2 UML representation of the REST API Diagram

Correct calls to the web application via the REST API is the building block of our project. Thus, talking about the structure is essential. The API has 5 entities representing the database's tables. For each entity, client can use 2 methods with the entity url -e.g. `/hardware` -:

- **GET**: this method corresponds to the SQL's `SELECT * from ...;`.
- **POST**: a body of type JSON is required. It holds the object attributes.

This method corresponds to the SQL's `INSERT INTO`.

Also, client can use 3 methods with the entity url + id -e.g. `/hardware/6` -:

- **GET**: this method corresponds to the SQL's `SELECT * from ... WHERE id = ...;`.
- **PUT**: a body of type JSON is required. It holds the object attributes. This method corresponds to the SQL's `UPDATE ... WHERE id = ..;`.
- **DELETE**: this method corresponds to the SQL's `DELETE FROM ... WHERE id = ..;`.

Figure 3.27 shows the API's hierarchy and figure 3.28 shows the models. For full documentation see Appendix B

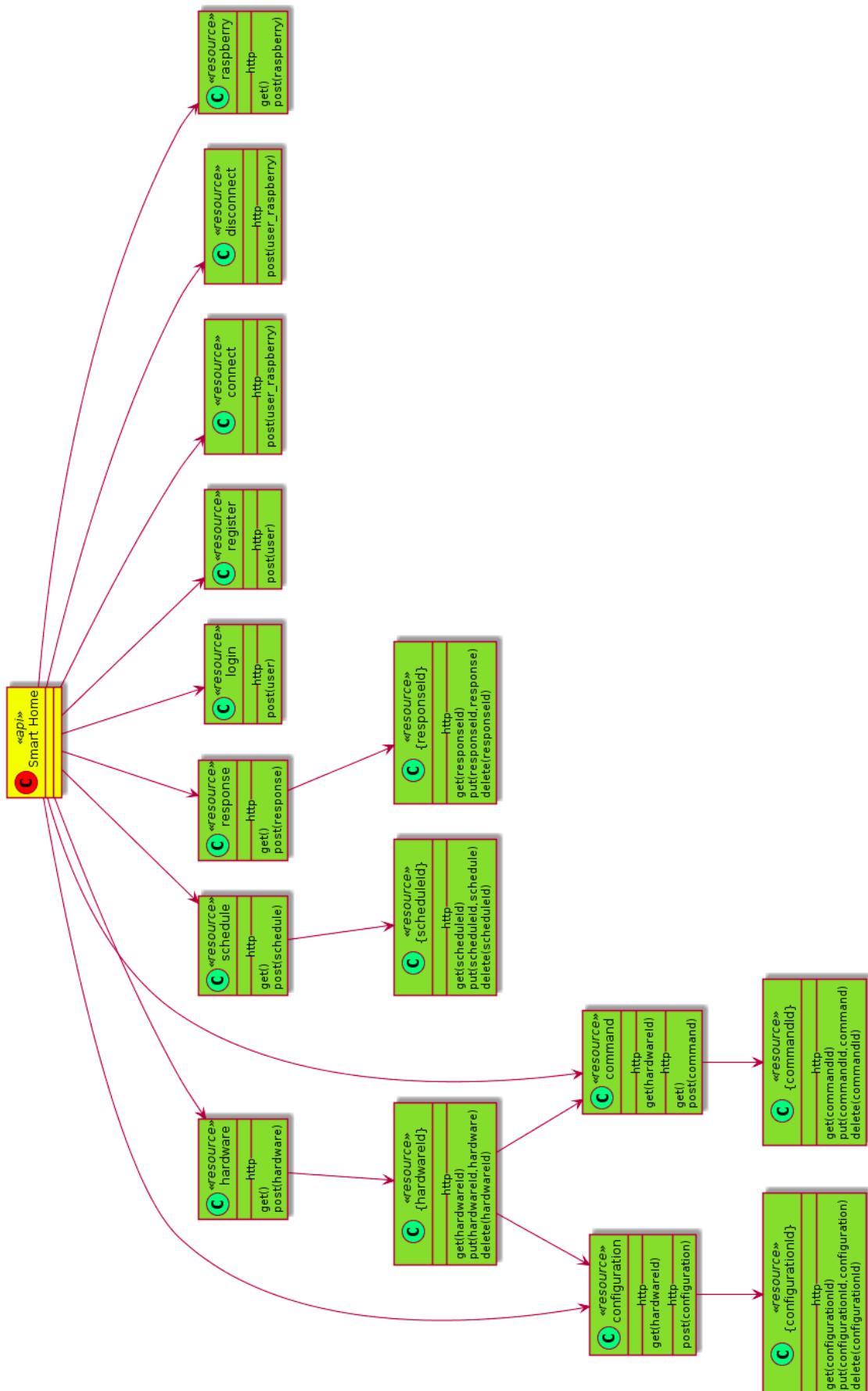


Figure 3.27: REST API



Representations/Messages



Figure 3.28: REST API models

CHAPTER NO. 4

SYSTEM DESIGN



4 System Design

4.1 System Architecture

The proposed project has 4 tiers. The Android App, the web app and finally the raspberry pi script and finally the system database. The android app works as the user interface. The raspberry pi app is the one responsible for controlling the actual hardwares. The web app works as a medium for managing user requests and raspberry pi's response. Finally, the database that stores these data. *Figure 4.1 and figure 4.2 show the architecture for the system using component and deployment diagrams.*

4.1.1 The android application

Also known as the client tier. It enables the user to interact with the system through a simple graphical user interface. Android application can send requests to the web server and receive responses. This app is designed using Java programming language, Android framework, Retrofit library for managing networking and finally Gson for parsing JSON responses.

4.1.2 The raspberry pi script

Also known as the aggregation tier. It executes the commands and send responses to the web application. The script is a simple python program that controls different hardwares using GPIO library.

4.1.3 The web application

Also known as the delivery tier. It works as an immediate medium between the user and raspberry pi. The advantage for adding this layer is guaranteed execution regardless of user location and distance from raspberry pi. The web app will be deployed on an ubuntu server. It is made using python programming language and flask framework. Also, sqlalchemy, which is an ORM for managing database



interaction, will be used to ease the development.

4.1.4 The system database

Also known as the Services tier. It manages and stores all system data.

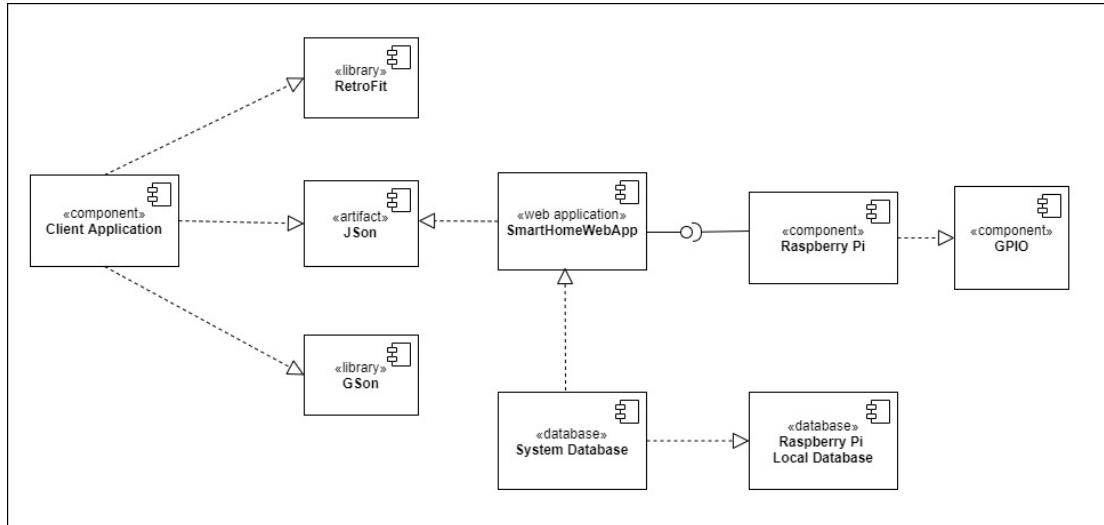


Figure 4.1: Component Diagram

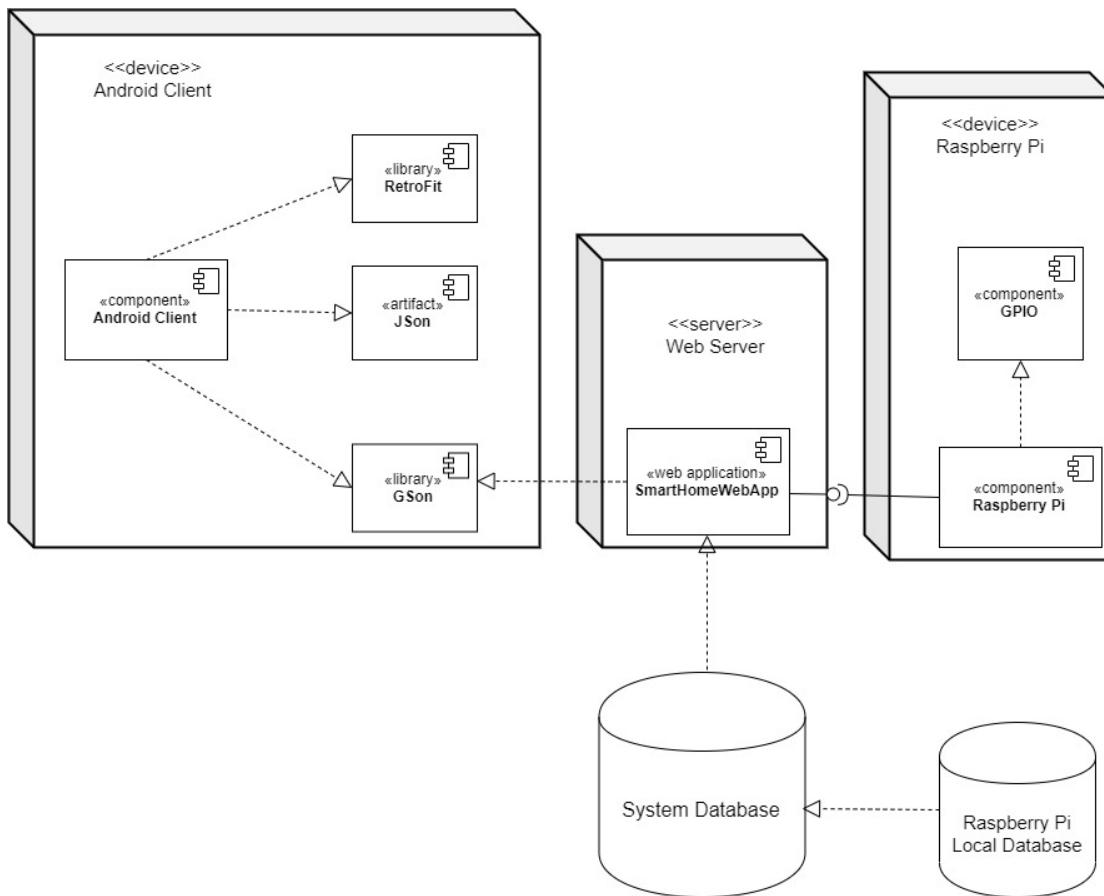


Figure 4.2: Deployment Diagram



Figure 4.3 shows the four-tier architecture explained above. In our system, the best architecture to use is four-tier architecture because of its flexibility and scalability to add multiple services. Also, it adopts user experiences that are fast, responsive, and tailored unique needs. Furthermore, multi-tier architecture also reduces traffic on the network[32].

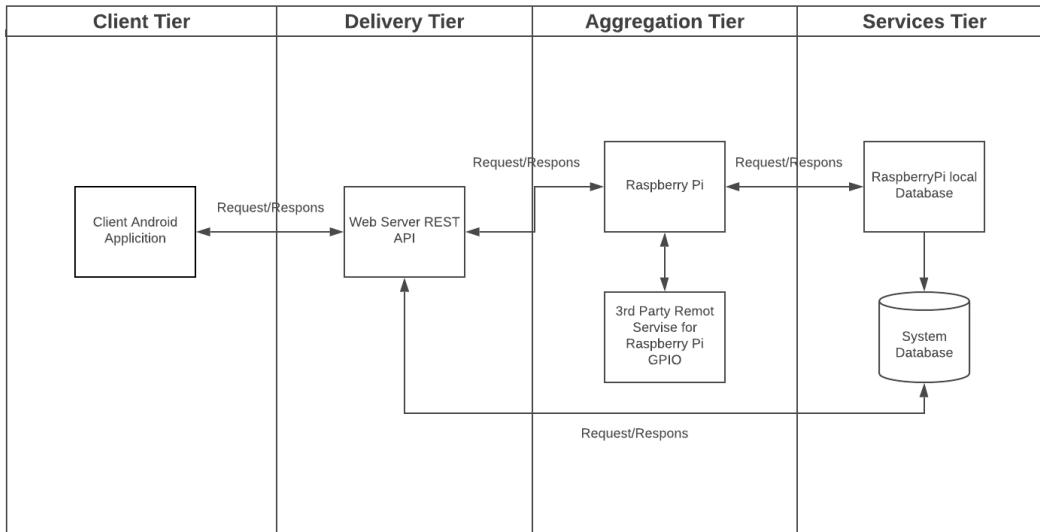


Figure 4.3: Architecture Diagram



4.2 User Interface Design

4.2.1 Login Activity

Figure 4.4 shows the login activity. It is the activity that the user will see when the app is first launched to prompt the user to enter their credentials.

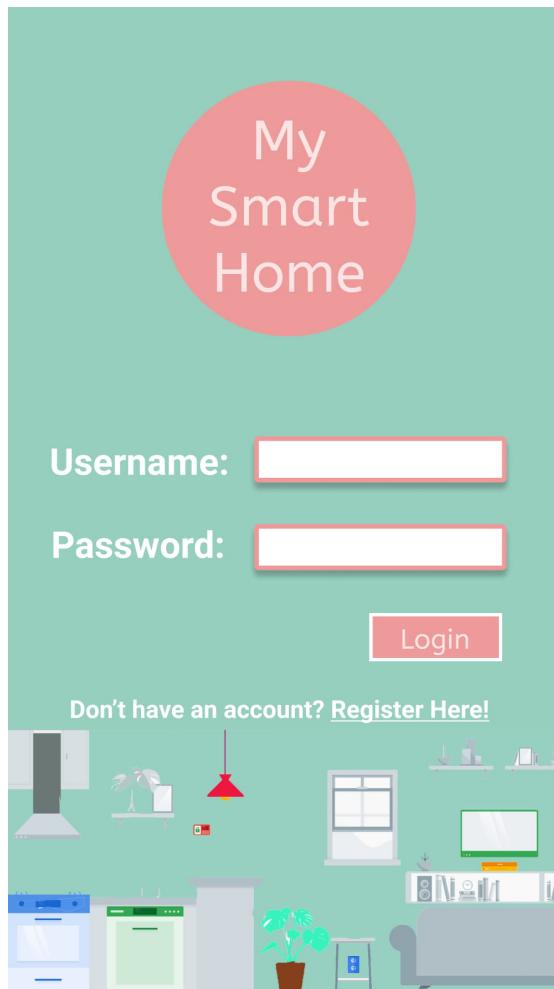


Figure 4.4: UI: Login Activity



4.2.2 Raspberry Activity

Figure 4.5 shows the raspberry activity. All the raspberry connected to the user shall be viewed here, with the option of deleting them or adding new ones.



Added Raspberry Pi's +

- Raspberry_1
- Raspberry_2
- Raspberry_3

Figure 4.5: UI: Raspberry Activity



4.2.3 Hardwares Activity

Figure 4.6 shows the hardwares activity. All the hardwares in the system shall be displayed in a grid.

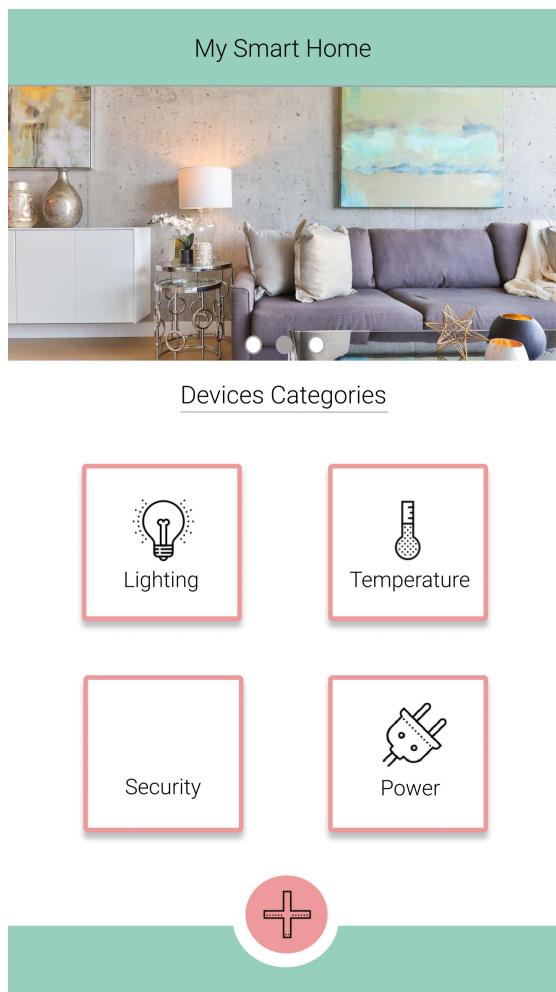


Figure 4.6: UI: Home Activity



4.2.4 Hardware Activity

Figure 4.7 shows the hardware activity. The user first is shown the current status for the hardware. Then, a list of active command is presented with the ability to edit/delete each command. Also, the user can click on the + icon to add a new command.

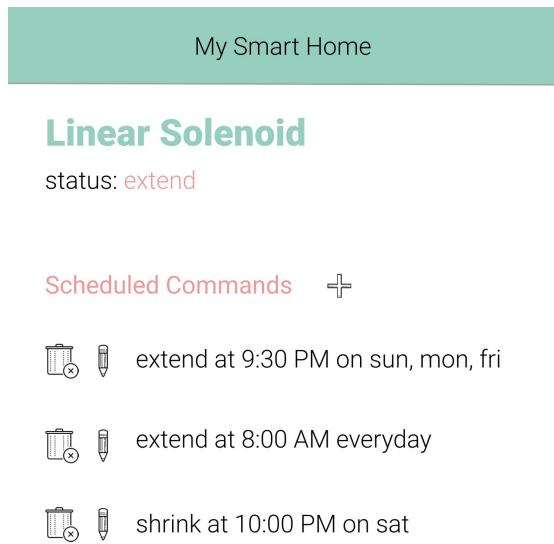


Figure 4.7: UI: Hardware Activity



4.2.5 New Command Activity

Figure 4.8 shows the new command activity, where the user can add a new command to be executed. First, the user choose the configuration wanted. Then either execute the command now, or click on the schedule, which will open a new activity to choose the scheduling information desired.

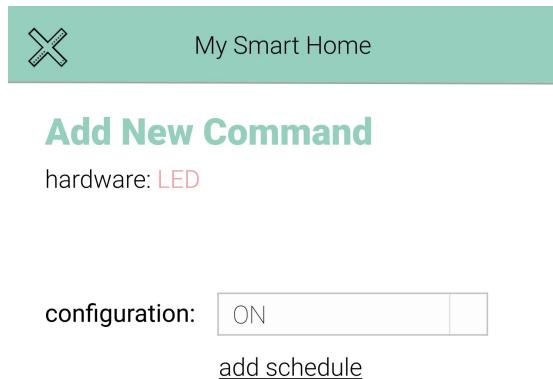


Figure 4.8: UI: New Command Activity



4.2.6 Schedule Activity

Figure 4.9 shows the schedule activity. This activity appears once the user clicks on add schedule in the previous activity. The user can choose the time of execution and the days to execute the action here.

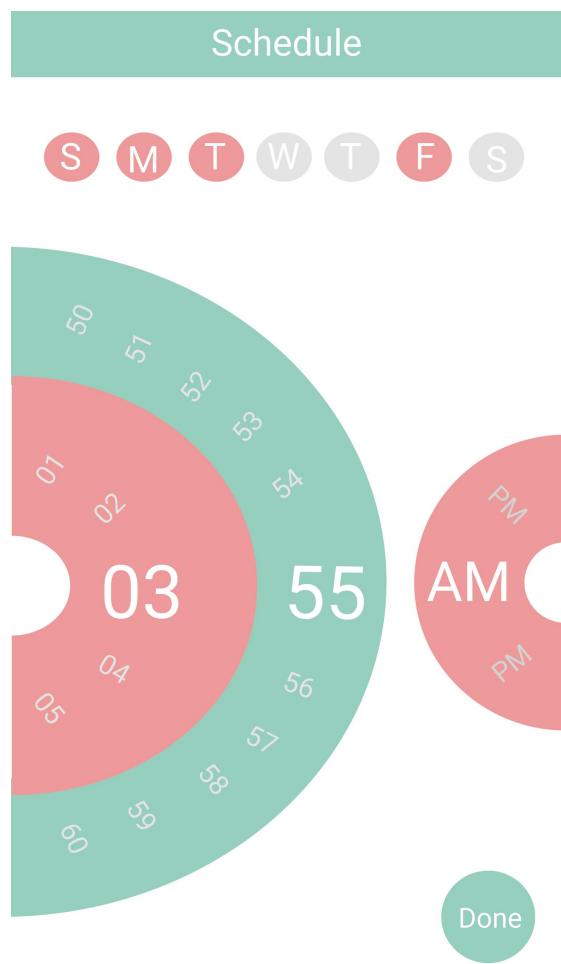


Figure 4.9: UI: Schedule Activity



4.2.7 Progress Fragment

Figure 4.10 shows the waiting progress fragment that will appear until the web server respond with the code 201, which denotes a successful creation of a command.

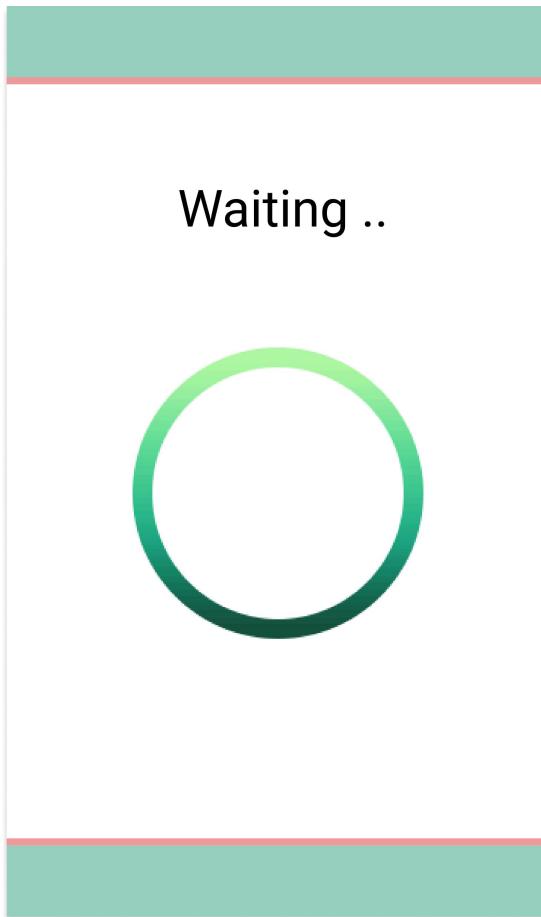


Figure 4.10: UI: Progress Fragment



4.2.8 Message Fragment

Once the server responds, the message embedded with the response shall be displayed to the user. Either **Done!** with **201** or the error message when the code is different. *Figure 4.11* shows the message when the command is successfully created.



Done!



Figure 4.11: UI: Message Fragment

CHAPTER NO. 5

IMPLEMENTATION



5 Implementation

5.1 Implementation Requirements

As mentioned before, the system has three major components: **Web application**, **mobile application** and **raspberry pi script**. The software requirement was detailed in section 3.2.1 while the hardware requirements was in section 3.2.2. However, after writing the code for each component, an installation has to be made.

5.1.1 Software Requirements

Jet brain's IDE were used to write the code, specifically **PyCharm**. For the android application, the official **Android Studio** was used. **Postman** was used to test the API.

5.1.2 Hardware Requirements

For the web application, a server was brought. The OS is *ubunut 16.04*. The official digitalocean [33] guide was followed. After that, a domain name was assigned to the web application and it is now available over gp.reem-codes.com.

For the android application, an android mobile is required. Our application is available in google play. https://play.google.com/store/apps/details?id=com.reem_codes.gp_android

Finally for the raspberry script, raspberry pi was bought along with its accessories mentioned before. *Figure 5.1* shows raspberry pi connected to two LEDs.



Figure 5.1: Connecting Raspberry Pi with the hardware

5.2 Implementation Details

The project was divided into three parts. Each part is a stand-alone application. However, the communication is done using the web api application. Since the system is broken down into 3 components, a functionality as straight-forward as *login* for instance may be written in two or more of these applications.

Below are the main functionality of the system, along with their major code and the way they work



5.2.1 Web Application

Function Name	Login
Input	email and password, or raspberry id
Output	token or an error message
Explanation	a straightforward method. When an email and password is passed to it, a token is returned if the authentication was correct. Otherwise an error message appears. This method also is used for raspberry login with the same procedure.
<pre> 12 @app.route('/login', methods=['POST']) 13 def login(): 14 if not request.is_json: 15 return jsonify({"message": "Missing JSON in request"}), 400 16 17 email = request.json.get('email', None) 18 password = request.json.get('password', None) 19 raspberry_id = request.json.get('raspberry_id', None) 20 21 if raspberry_id: 22 raspberry = Raspberry.query.filter_by(id=raspberry_id).first() 23 if not raspberry: 24 return jsonify({"message": "Bad raspberry pi id"}), 401 25 ret = { 26 'access_token': create_access_token(identity=raspberry.id), 27 } 28 return jsonify(ret), 200 29 else: 30 if not email: 31 return jsonify({"message": "Missing email parameter"}), 400 32 if not password: 33 return jsonify({"message": "Missing password parameter"}), 400 34 35 user = User.query.filter_by(email=email).first() 36 if not user or not bcrypt.check_password_hash(user.password, password): 37 return jsonify({"message": "Bad email or password"}), 401 38 39 # Identity can be any data that is json serializable 40 ret = { 41 'access_token': create_access_token(identity=user.id), 42 'user': user 43 } 44 return jsonify(ret), 200 </pre>	



Besides the login function, the web application offers methods to access the database's tables.

Function Name	API functions
Input	the token, id and body if needed
Output	message and the object if needed
Explanation	For each model, 5 functions are written: index, edit, delete, get and insert.
	<pre> @app.route("/hardware", methods=["GET"]) @jwt_required def hardware_index(): raspberry = request.args.get('raspberry_id') ids = {"raspberry_id": raspberry} if raspberry else {} return jsonify(Hardware.index(ids)) @app.route("/hardware", methods=["POST"]) @jwt_required def hardware_post(): body = request.get_json() obj = Hardware.post(body) return {"message": Config.POST_MESSAGE, "object": obj}, 201 @app.route("/hardware/<_id>", methods=["GET"]) @jwt_required def hardware_get(_id): raspberry = request.args.get('raspberry_id') ids = {"id": _id} if raspberry: ids["raspberry_id"] = raspberry obj = Hardware.get(ids) return jsonify(obj) @app.route("/hardware/<_id>", methods=["DELETE"]) @jwt_required def hardware_delete(_id): Hardware.delete({"id": _id}) return {"message": Config.DELETE_MESSAGE}, 203 @app.route("/hardware/<_id>", methods=["PUT"]) @jwt_required def hardware_put(_id): body = request.get_json() obj = Hardware.put({"id": _id}, body) return {"message": Config.PUT_MESSAGE, "object": obj} </pre>



5.2.2 Raspberry Pi

Function Name	Main
Input	All the commands gotten from the server
Output	Execution of the method execute which carries out the commands and switches on the LED
Explanation	<p>The function, when invoked, uses the GET method to get all the commands saved in the server. It iterates through them and checks if the commands schedule_id attribute is set to some value. In that case, the command is scheduled. The time string of the schedule is taken from schedules time attribute and is saved then turned into a format that includes the days date. The day of the week is identified and turned into a binary digit using the binary_to_position function. Then, if the day and execution time are less than the present, the execute function is called, and the command is executed. Otherwise, if the schedule_id is not set to some value, this means that the command is immediate, and the execute function is called right away.</p> <pre> def main(): now = datetime.datetime.now() # today's datetime # GET all commands from server and save it in data variable data = api.get_commands() for command in data: # looping through commands print(command) hw_id = command["hardware_id"] gpio = api.get.hardware(command["hardware_id"])["gpio"] conf = command["configuration"] command_id = command["id"] if command["schedule_id"] is not None: # if the command is scheduled (i.e. schedule_id != null) # GET schedule_id from the server and save it in schedule variable schedule = api.get_schedule(str(command["schedule_id"])) # the schedule has a time string (e.g. "04:45 AM") and an integer from 0-126 representing the binary days schedule_time = datetime.datetime.strptime(schedule["time"], "%I:%M %p") # time string into datetime object # turn the time string into today's date with the time given execution_time = datetime.datetime(now.year, now.month, now.day, schedule_time.hour, schedule_time.minute) # find today's weekday (e.g. sun, mon, tues) weekday = execution_time.today().weekday() # return the day of execution from the schedule days = binary_to_position("0:07b").format(schedule["days"]) # if the weekday is part of the execution days and the time is before no if weekday in days and execution_time < now: execute_gpio(gpio, _on_=conf, command_id=command_id, hw_id=hw_id) # execute else: # immediate command # turn update from string into datetime datetime_object = datetime.datetime.strptime(command["updateAt"], "%a, %d %b %Y %H:%M:%S %Z") if datetime_object < now: # if the execution time before now execute_gpio(gpio, _on_=conf, command_id=command_id, hw_id=hw_id) </pre>



Function Name	Execute
Input	<code>_gpio: The GPIO pin connected. _on: Whether to make it high or low. Command_id: The commands that will be executed. hw_id: The physical components connected to a raspberry pi. Such as LED lights, linear solenoids.</code>
Output	Post response to the server that indicates whether the commands has been successfully done with date and time.
Explanation	The function will turn the hardware ON and OFF depending on the GPIO pin. The POST method will be sending a response and the PUT method will change the status of hardware from the OFF status to ON or conversely.
	<pre>def execute(_gpio, _on, command_id, hw_id): print("I'm executing...") print(_gpio, _on) GPIO.setmode(GPIO.BCM) GPIO.setup(_gpio, GPIO.OUT) GPIO.output(_gpio, GPIO.LOW if _on else GPIO.HIGH) res = {"isDone": True, "message": "command executed successfully", "executionTime": datetime.datetime.now().strftime('%a, %d %b %Y %H:%M:%S %Z'), "command_id": command_id } api.post_response(res) api.put_hardware(hw_id, _on)</pre>



5.2.3 Android Application

Even though the majority of functions are in android studio, a few will be mentioned. The rest are filler functions and boiler plates. The most important functions are:

- create, get, delete and edit command, hardware and raspberry: basically, providing UI interfaces for the users to manipulate the models of the system. These are done using **Adapters** and **onActivityResult**.
- get response each 5 minutes from the server: to see the execution result of a command. This is done using **onStartCommand** function.
- communicating with the server: and it is done using different methods all following the same **methodModelApiformat**. In which the method is GET, POST, PUT or DELETE. The model is either command, schedule, hardware, response or raspberry.



Function Name	methodModelApi
Input	the id and body if needed. The access token.
Output	Show message to user.
Explanation	These methods family are used to communicate with the server using the okhttp library. A request is build then a response is received. The response is validated and turned into an object for further processing
	<pre>public void getResponseApi() throws IOException { url = getString(R.string.api_url) + "/response"; Request request = new Request.Builder() .url(url) .get() .addHeader("Authorization", "Bearer " + currentLoggedUser.getAccess_token()) .build(); client.newCall(request).enqueue(new Callback() { @Override public void onFailure(Call call, IOException e) { Toast.makeText(getApplicationContext(), "failed", Toast.LENGTH_SHORT).show(); } @Override public void onResponse(Call call, Response response) throws IOException { String result = response.body().string(); System.out.println("DEBUG results are " + result); Gson gson = new Gson(); TypeToken<ArrayList<Response>> typeToken = new TypeToken<ArrayList<Response>>() {}; responses = gson.fromJson(result, typeToken.getType()); for(Response res : responses) { addNotification(res); } } }); }</pre>



Function Name	onStartCommand
Input	the time interval of executing
Output	show response to users as a push notification.
Explanation	At first, this method check if the user is logged in. If they are the getResponseApi -part of the method-ModelApi family- is called. This methods get all responses belonging to this user and show them as push notification. This process is repeated each 5 minutes.
	<pre> @Override public int onStartCommand(Intent intent, int flags, int startId) { checkUser(this); if(currentLoggedUser != null) { final Handler handler = new Handler(); Runnable runnable = new Runnable() { @Override public void run() { try{ getResponseApi(); } catch (Exception e){ System.out.println("GPDEBUG "+ e.getMessage()); } handler.postDelayed(this, DELAYED_TIME); } }; handler.post(runnable); } return START_STICKY; } </pre>



Function Name	user event listeners in adapters
Input	user clicks on delete icon or edit icon
Output	Show the user a form for editing or creating.
Explanation	In android, adapters are classes managing list of data. For each item, we attached a delete and edit methods. Also, create method is attached to the whole list so an item can be added. All of these functions work on the user interface but call one of the methodModelApi family in the background to enforce these changes in the server.
edit event listener and delete event listener	// if the user clicked on the edit button, open the NewCommandActivity <pre>edit.setOnClickListener(new View.OnClickListener() {</pre> <pre>// if the user clicked on delete button, call the API to delete it delete.setOnClickListener(new View.OnClickListener() {</pre>



Function Name	onActivityResult
Input	the action: edit or create
Output	return user to the list view page
Explanation	When a user clicks on create button or the edit button, a form is shown to take the input. Once the input is taken, a function of the methodModelApi family is called.
<pre> @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) { super.onActivityResult(requestCode, resultCode, data); if (requestCode == LAUNCH_ADD_HARDWARE requestCode == HardwareAdapter.LAUNCH_EDIT_HARDWARE) { if (resultCode == Activity.RESULT_OK) { int gpio = data.getIntExtra("gpio", -1); String name = data.getStringExtra("name"); String desc = data.getStringExtra("desc"); String icon = data.getStringExtra("icon"); boolean isEdit = data.getBooleanExtra("isEdit", false); if(isEdit){ TypeToken<Hardware> token = new TypeToken<Hardware>(){}; hardware = (Hardware) Gson.fromJson(data.getStringExtra("hardware"), token.getType()); System.out.println("GODEBUG IS EDIT"); } String text = String.format("(\\\"name\\\":\\\"%s\\\", \\\"gpio\\\":%d,\\\"icon\\\": \\\"%s\\\", \\\"desc\\\": \\\"%s\\\", \\\"raspberry_id\\\": %d)", name, gpio, icon, desc, raspberry.getId()); try { postPutHardwareApi(text, isEdit); } catch (IOException e) { Toast.makeText(this, "please check network and try again", Toast.LENGTH_LONG).show(); } if (resultCode == Activity.RESULT_CANCELED) { Toast.makeText(this, "no hardware created", Toast.LENGTH_LONG).show(); } } } } //onActivityResult </pre>	



5.3 I/O Screens

The following are the main input and output screens

5.3.1 Login Screen

First, the user is required to login or register before using the system. *Figure 5.2* shows the login interface.

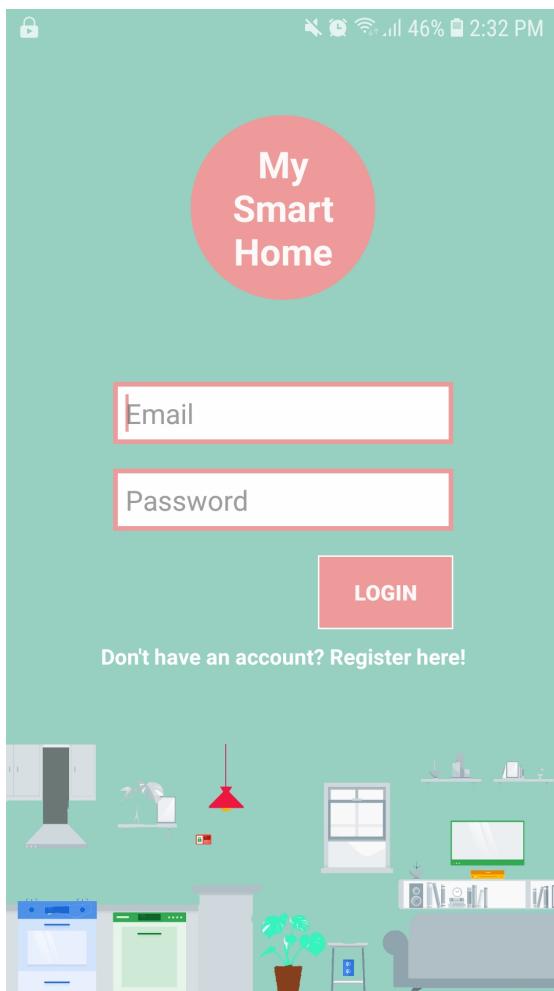
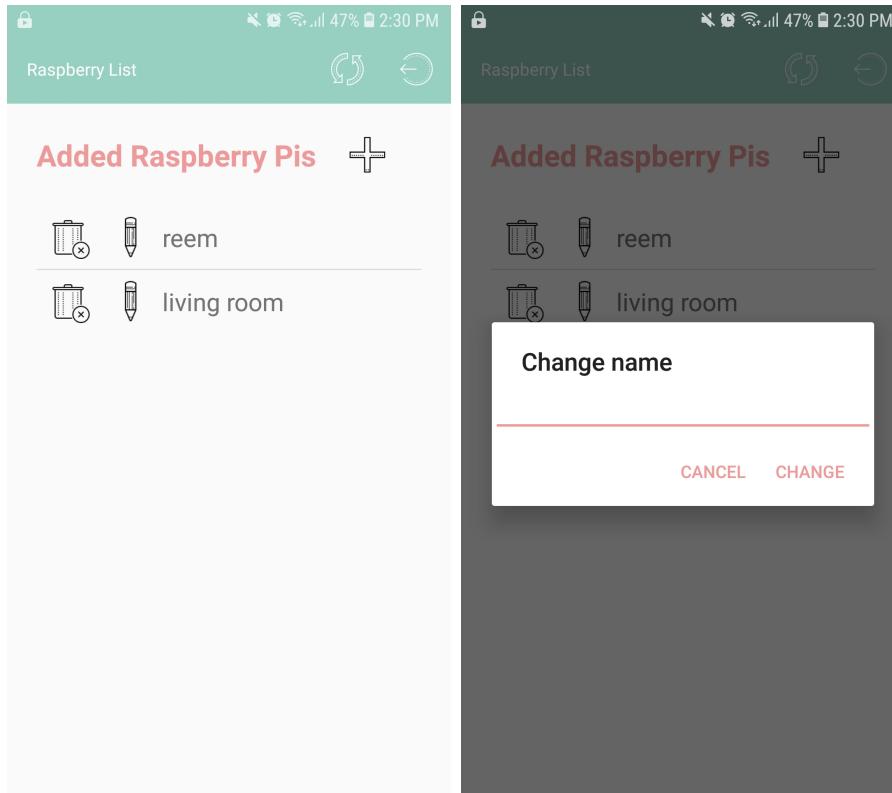


Figure 5.2: I/O: Login Screen



5.3.2 Raspberry Pi Screen

After successful login, the user is redirected to the raspberry pi's list of connected devices. *Figure 5.3* shows the Raspberry Pi interfaces.



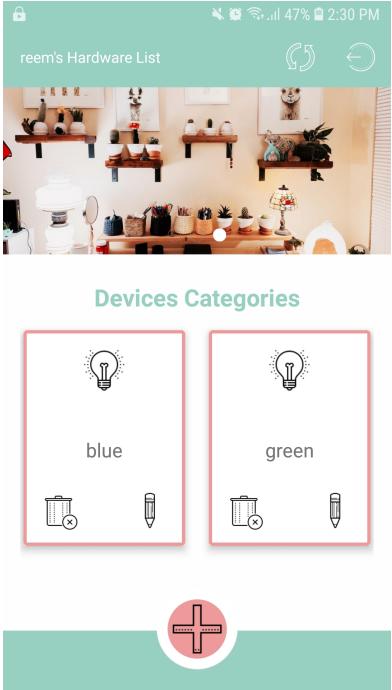
(a) I/O: Raspberry Pi List (b) I/O: Raspberry Pi Edit

Figure 5.3: I/O: Raspberry Pi Screens

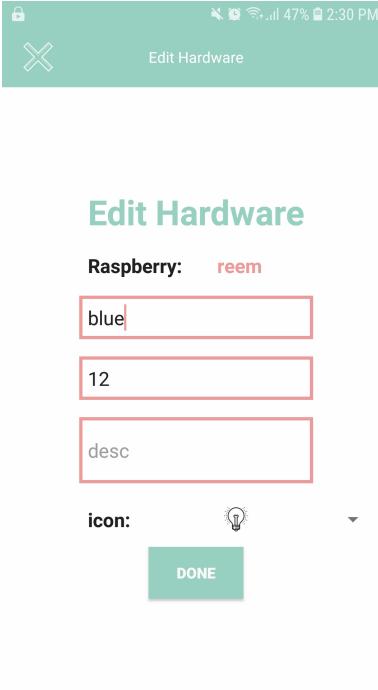


5.3.3 Hardware Screen

After clicking on one of the raspberry pi's, the hardware list appears. *Figure 5.4* shows the Hardware interfaces.



(a) I/O: Hardware List



(b) I/O: Hardware Edit

Add New Hardware

Raspberry: reem

name
gpio
desc

icon: 

DONE

(c) I/O: Hardware Create

Figure 5.4: I/O: Hardware Screens



5.3.4 Command & Schedule Screen

For each hardware, a user can create a command and see the command list.

Figure 5.5 shows the Command and Schedule interfaces.



blue

Status: **OFF**

Schedule Command +

	ON at 9:30 PM on Mon, Thurs, Sat
	OFF at 12:30 AM on Tue, Fri
	ON at 6:45 PM on everyday

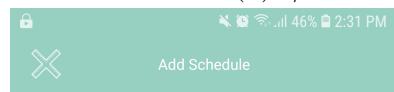
Add New Command

Hardware: **blue**

Configuration: **ON** ▼

DONE ADD SCHEDULE

(a) I/O: Command List (b) I/O: Command Create



9 : 30 AM PM

Mon	Tues	Wed	Thurs
Fri	Sat	Sun	

DONE

(c) I/O: Schedule Create

Figure 5.5: I/O: Command Screens

CHAPTER NO. 6

TESTING



6 Testing

6.1 Testing

In the last chapter, implementation was finished. This chapter looks into the methods used to test and validate the different parts and scopes of the system mentioned in previous chapters.

6.2 Test Plan

In this stage, the concern is in eliminating the majority of bugs that could be easily found with a bit of usage and focused code reading. Basically, addressing those bugs before launching the website and publishing the application. *Unit Testing* and *System and Integration Testing* were used.

6.2.1 Unit Test

The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as expected. As shown in *Figure 6.1* shows multiple cases of validation offered to users. *Appendix A.2* demonstrates more cases.

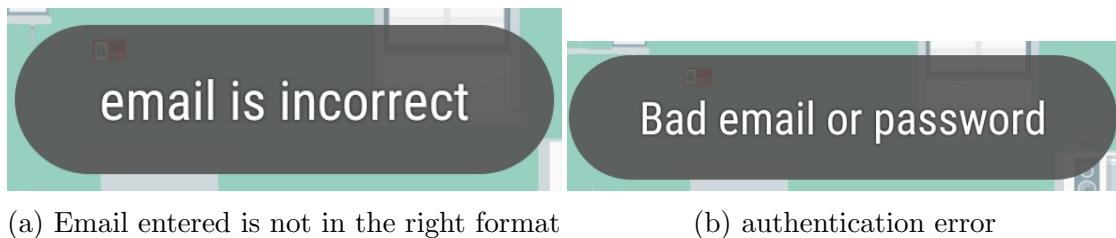


Figure 6.1: Login Validation

6.2.2 System and Integration Testing

Here, the concern is on the harmony between the 3 subsystems and their communication. After testing each component under each sub-system individually, a complete system test must be performed to check for any communication error.



Postman was used to ensure correct REST API behavior as shown in the *figure*

6.2

The screenshot shows the Postman application interface. At the top, there's a toolbar with 'File', 'Edit', 'View', 'Help', and various buttons for 'New', 'Import', 'Runner', and 'Invite'. Below the toolbar, a navigation bar includes 'My Workspace' and 'Sign In'. The main area is titled 'Untitled Request' and shows a 'GET' request to 'https://gp.reem-codes.com/command'. The 'Params' tab is selected, showing a single entry 'Key' with 'Value'. The 'Body' tab is also selected, displaying a JSON response with two objects:

```

1  [
2   {
3     "configuration": false,
4     "hardware_id": 1,
5     "id": 1,
6     "schedule": null,
7     "schedule_id": null,
8     "updateAt": "Sat, 22 Feb 2020 08:16:17 GMT"
9   },
10  {
11    "configuration": false,
12    "hardware_id": 1,
13    "id": 2,
14    "schedule": {
15      "days": 109,
16      "id": 1,
17      "time": "09:40 AM",
18    }
19  }
20 ]

```

Below the body, the status bar indicates 'Status: 200 OK Time: 882 ms Size: 4.94 KB'. The bottom of the window has icons for 'Bootcamp', 'Comments 0', and a help icon.

Figure 6.2: Postman testing the API

6.2.3 The Scope

The scope of testing includes the **testable coded parts**; which means testing the hardware and its power is outside of the testing scope. To be specific, no checking will be made to make sure the server is constantly up. Secondly, no testing for the mobile device's memory usage or battery -a network check, however, will be performed-. Finally, no checking to see if the raspberry pi's components are in good shape.

6.2.4 Features to Be Tested

On the other hand, any user input or manageable units shall be tested. The user's explicit CRUD operations shall be tested and managed accordingly. In



addition, The network connection shall be checked and a suitable error shall be thrown if needed.

6.2.5 Schedule of Test Activities

Some test activities are done each time before the operation execution, others are done periodically or once per installation time. *Table 6.1 represents the schedule of test activities.*

No.	Task activity	Deliverable	Start Date	End Date
1	Plan testing	Test Plan	20 February	24 February
2	Develop test cases	Test Cases	24 February	28 February
3	Execute test		28 February	4 March
4	Evaluate test		4 March	8 March
5	Documentation	Test Results	8 March	10 March

Table 6.1: Test Schedule

6.2.6 Test Tasks

The test is conducted as followed:

1. Write test plan.
2. Build test cases.
3. Execute the tests.
4. Evaluate the test results.



6.3 Test Case

a Test case can be defined as a set of test inputs, execution conditions, and expected results developed for a particular goal to verify compliance with a requirement [34]. The following test cases were chosen for being challenging. *Table 6.2 shows the test cases for the system.*



No.	Function	Input	Expected Output	Resulting Output
1	Login	email and password	If exists in the database and credentials correct, go to raspberry pi's activity.	Display raspberry pi's activity.
2	Adding a resource	resource attributes	Add a new resource if data is in the right format and required fields are filled.	Return successfully added message
3	Editing a resource	resource id and its attributes	Edit resource if id is correct and data is in the right format and required fields are filled.	Return successfully edited message
4	Deleting a resource	resource id	Delete resource if id is correct.	Return successfully deleted message

Table 6.2: Test Case



6.4 Test Result

Testing is done for functional and non-functional requirements. If those results are satisfactory, then it is a success. The following tables demonstrate the results.

Requirement	Success	Failure	Description
Manage hardware	✓		the expert user can add manage hardware via the application. however, they must physically connect the components respectfully.
Manage configuration		✓	the only available configurations are booleans: Either high or low voltage.

Table 6.3: Expert User's Functionalities

Requirement	Success	Failure	Description
Delete executed immediate commands	✓		once a response is read by the android client, they are immediately deleted from the server
Delete read responses	✓		once a response is read by the android client, it is immediately deleted from the server

Table 6.4: Web Application's Functionalities



Requirement	Success	Failure	Description
Get command list	✓		raspberry pi read commands each 30 seconds.
Update local queue	✓		The local queue was replaced by a list for faster implementation
Execute commands saved in queue	✓		Raspberry pi execute commands accordingly
Submit a command response to server	✓		Once done, a response is POST to the server

Table 6.5: Raspberry Pi's Functionalities

Requirement	Success	Failure	Description
User login or register	✓		validation and authentication are implemented too.
Manage resources	✓		a user can get, delete, create and edit each of the following: raspberry pi, hardware, command and schedule.
Receive Responses	✓		Response service pulls responses from the server each 5 minutes and show them in the notifications.

Table 6.6: Android Client's Functionalities



Requirement	Success	Failure	Description
Availability	✓		The server bought guarantee a 99.99% up-time.
Usability	✓		The system is easy to use, especially that the hardware can be added via the mobile application instead of raspberry pi's console.
Verifiability	✓		the system require authentication in each step via a token
Performance and Delay	✓		The raspberry pi read user commands within 30 seconds of adding the command
Flexibility		✓	although the application accepts any hardware in theory, only LEDs have been tested.
Security	✓		tokens are used as means of communication.
Efficiency	✓		the web application cleans up executed command periodically.

Table 6.7: Non-functional requirements

CHAPTER NO. 7

DEPLOYMENT



7 Deployment

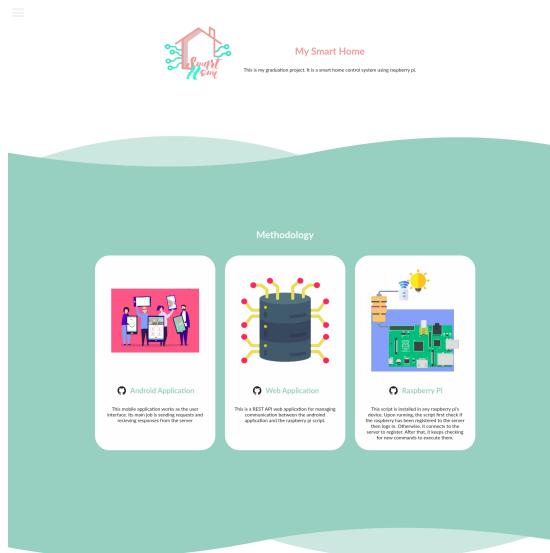
After implementing the system and testing it, the system was deployed. Again, for each sub-system, the deployment method is different. As mentioned briefly in *Section 5.1.2*.

7.1 Web Application

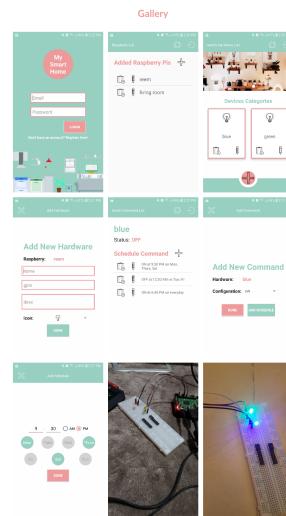
First, a domain name and a web server were bought. The official tutorials and help-guides were followed for each service used. Although the web application is a REST API, we made an interface for people who visit it. *Figure 7.1* shows the interface. The interface can be accessed here gp.reem-codes.com. For security purposes, SSL certificate was installed from **Let's Encrypt**. The domain was brought from **namescheap** and the server from **DigitalOcean**.

7.1.1 Raspberry Pi Script

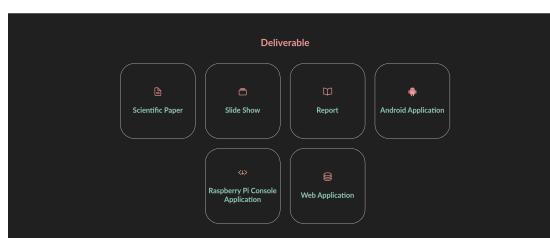
The other use of this page is downloading the raspberry pi script. User can access the website and find the application to be installed in their raspberry pi.



(a) Landing Page - Methodology



(b) Landing Page - Gallery



This Project Is Made By

Name	Email
Penn Ali	www.aldibeg@gmail.com
Ahmed Ezzat	alemnazzat@gmail.com
Nour Alzahrani	nouralzahrani@gmail.com
Maha Alzahrani	maha.alzahrani@nu.edu.sa
Dalia Alzahrani	dalia.alzahrani@gmail.com
Sarah Alzahrani	sarah.alzahrani@nu.edu.sa

©2020 [mycodee.com](#). All rights reserved.

(c) Landing Page - Deliverable

Figure 7.1: Landing Page



7.2 Android Application

Android studio offers a way to build the application into **apk** files then sharing them freely. However, the majority of phones disable downloading from them by default. Unless the user has a good knowledge of how to change the option, they won't be able to use the application. Thus, deploying it in **google play** was essential.

Again, official tutorials were followed. The problems all came from waiting for the approval. Once it was approved, it was published in google play here https://play.google.com/store/apps/details?id=com.reem_codes.gp_android. Figure 7.2 shows the application in google play.



My Smart Home

reem-codes House & Home

3+

This app is compatible with all of your devices.

★★★★★ 8 2

Installed

This is my graduation project. It is a system that aims to control any hardware online as long as it is connected to a registered raspberry pi.

ABSTRACT:

With the recent very rapid progress in technology and automation, and towards easier daily life tasks, there has become a need for remote control of almost all possible aspects of living. Some examples

[READ MORE](#)

Figure 7.2: The Application in Google Play

CHAPTER NO. 8

CONCLUSION



8 Conclusion

8.1 Conclusion

In summary, this project aimed to create a system that controls electronic household items from afar. In short, an android application was used as an interface for the user to issue control commands, which are in turn sent to and stored in the server. They are gotten from the server by a small computer (Raspberry Pi) that controls an electric circuit, to which the desired household items are connected and manipulated as per the commands. The project is a direct application of the concept of Internet of things and automated house systems.

8.2 Evaluation

Users all over the world can benefit greatly from an increase in systems and applications offered to enable control of devices over the Internet. As the trend in recent years has been for faster and more inclusive control of aspects of human life, more and more of these systems have popped up to help users control their cars, home appliances, doors etc. non manually using Internet of things, without having to actually be there.

For that, we have created a downloadable android application. It allows the user to add different Raspberry Pis, each connected to its own hardware. He can then choose a certain item of hardware that he wishes to control, like the LED and issue a command to turn the light on. He can then choose to schedule the command by choosing the days of the week he wants it to be turned on and the timings, or he can just choose for it to be executed immediately. After execution, he receives a push notification to confirm the command being executed successfully.

The application we have created successfully ticks all the boxes for an acceptable, feasible smart control system. It doesn't cost much; \$25-\$35 for a Raspberry



Pi and very low costs for the wires and relays that connect the house appliances to the laptop. It is secure, as it uses login tokens to ensure that, at every step in the command issuing process, the same correct user is logged in. It is also relatively fast, as the commands are gotten from the server every 30 seconds so this allows for very fast execution of the command for a system this cheap.

8.2.1 Problems faced:

During the course of working on this project, we faced several problems in every phase, especially in the final phases of the project. They can be summarized as follows:

1. In the system analysis phase, we faced some problems knowing exactly the types of software and hardware that we would need, since our project is the first of its kind to be done. This included not knowing the types of files and programs that would be needed.
2. A second problem we faced was finding a time we were all free in to be able to work simultaneously on the project because of our conflicting schedules. Several times, we couldn't do that so we had to repeat to some peers what they had missed out and re-explain it to them. This also resulted in us having online meetings during weekends to be able to catch up and not miss any work.
3. Many problems occurred while deploying the system. Google took 7 days to verify us to allow us to post in google play which took another week. Also, deploying the API took a lot of time to configure and fix the problems. The difficulty comes from the fact that no error messages are displayed. So we had to manage it ourselves.
4. Another problem we faced was being able to bring the hardware pieces, like the Raspberry Pi laptop and the wires and the LEDs which were necessary for the project into the university building. We had to go to several places



to be able to get permission slips and that wasted a lot of time of the two hours a week we got to be able to work together.

5. Usually before any task, we would all sit together and agree exactly how that task is to be done and divide the work accordingly. Because of the quarantine during the last weeks, we couldnt meet together and work face to face on the project. This resulted in some of the work being submitted needing some corrections as it wasnt done and agreed upon in cooperation with the whole group.
6. The final problem we faced was because of the project being implemented in Android. Since only two girls in the group had Android OS devices, the rest had to design their application pages and test them out on these two phones. So, there was no possibility of downloading the finished application and testing it out on everyones phone. That made the testing phase a bit difficult to be done by everyone, especially because, as mentioned above, we could not meet up together to be able to test our pages out and screenshot the results.

8.3 Future Work

For our project, we had several more things in mind that we couldnt implement because of the time crunch towards the end. First, we had wanted to try out the linear solenoid that was supposed to push forward and press the button of whatever device the user wanted to control, like the coffee machines or an air conditioners remote control. Unfortunately, we only had time to test out the LED bulbs.

Secondly, we had intended to include a Configuration table in the database. We wanted it to contain all the different types of configurations possible for all the different types of connected hardware. So, for example, the table would store the fans settings which would be on/off or any of its three speeds, the air conditioners



different settings which would be the temperatures, swing options and sleep, etc. We also did not have time to try out enough hardware so we only had the LEDs configuration as on/off, which did not require a whole table to be made for it.

As with every system, there are definitely improvements and enhancements to be made. Our first idea is definitely making the application available for iOS users too. That will require knowing how to code in Swift but it will make the application available to a bigger portion of the community as most people here use iPhones instead of Android run devices.



References

- [1] “What’s raspberry pi?.” [https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.](https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/)
- [2] “Linear solenoid actuator.” [https://www.electronics-tutorials.ws/io/io_6.html.](https://www.electronics-tutorials.ws/io/io_6.html)
- [3] M. G. Samaila, M. Neto, D. A. Fernandes, M. M. Freire, and P. R. Inácio, “Challenges of securing internet of things devices: A survey,” *Security and Privacy*, vol. 1, no. 2, p. e20, 2018.
- [4] F. Samie, L. Bauer, and J. Henkel, “Iot technologies for embedded computing: A survey,” in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, p. 8, ACM, 2016.
- [5] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, “Architecting the internet of things: state of the art,” in *Robots and Sensor Clouds*, pp. 55–75, Springer, 2016.
- [6] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, “Future internet: the internet of things architecture, possible applications and key challenges,” in *2012 10th international conference on frontiers of information technology*, pp. 257–260, IEEE, 2012.
- [7] A. Khalid, “Smart applications for smart live,” *International Journal of Computer Science and Mobile Computing*, vol. 5, pp. 97–103, 2016.
- [8] I.-I. Pătru, M. Carabaş, M. Bărbulescu, and L. Gheorghe, “Smart home iot system,” in *2016 15th RoEduNet Conference: Networking in Education and Research*, pp. 1–6, IEEE, 2016.



- [9] M. S. Hossain and G. Muhammad, “Cloud-assisted industrial internet of things (iiot)-enabled framework for health monitoring,” *Computer Networks*, vol. 101, pp. 192–202, 2016.
- [10] “Droplets on digitalocean - more than just virtual machines.” <https://www.digitalocean.com/products/droplets/>.
- [11] “How light emitting diodes work — howstuffworks.” <https://electronics.howstuffworks.com/led.htm>.
- [12] “Gpio - raspberry pi documentation.” <https://www.raspberrypi.org/documentation/usage/gpio/>.
- [13] “Foreword.” <https://flask.palletsprojects.com/en/1.1.x/foreword/>.
- [14] “Retrofit.” <https://square.github.io/retrofit/>.
- [15] “Postgresql: The world’s most advanced open source database.” <https://www.postgresql.org/>.
- [16] “Incremental model in sdlc: Use, advantage & disadvantage.” <https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html>.
- [17] “Insteon.” <https://www.insteon.com/>.
- [18] “Wink — about us.” <https://www.wink.com/about/>.
- [19] “Smartthings. add a little smartness to your things..” <https://www.smartthings.com/>.
- [20] “All products - insteon.” <https://www.insteon.com/all-products>.
- [21] “Wink — buy and view smart home products.” <https://www.wink.com/products/>.



[22] “Samsung smartthings - for your connected smart home — samsung uk.”

<https://www.samsung.com/uk/smartthings/#products>.

[23] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, *The Java Language Specification - Java SE 8 Edition*. Oracle, 8 ed., 2015.

[24] “Download android studio and sdk tools : Android developers.” <https://developer.android.com/studio/index.html>.

[25] Z. Shaw, *Learn more Python 3 the hard way: the next step for new Python programmers*. Addison-Wesley, 2018.

[26] “Sqlalchemy — the python sql toolkit and object relational mapper.” <https://www.sqlalchemy.org/>.

[27] L. Beighley, *Head first SQL*. W. Ross MacDonald School Resource Services Library, 2011.

[28] “Pycharm: the python ide for professional developers by jetbrains.” <https://www.jetbrains.com/pycharm/>.

[29] “High performance load balancer, web server, & reverse proxy.” <https://www.nginx.com/>.

[30] “Sqlite home page — what is sqlite?” <https://www.sqlite.org/index.html>.

[31] T. M. Connolly, I. Jacobson, and C. E. Beg, *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson, 2005.

[32] P. Nommensen, “Its time to move to a fourtier application architecture.” <https://www.nginx.com/blog/time-to-move-to-a-four-tier-application-architecture/>, February 2015.



[33] DigitalOcean, “How to serve flask applications with uwsgi and nginx on ubuntu,” Sep 2019.

[34] C. Kaner, “What is a good test case?,” 2003.



CHAPTER NO. 9
APPENDICES



9 Appendices

A Figures

A.1 Similar systems hub design



Figure A.1: Similar systems: hub design



A.2 Some more Validation Cases

Schedule added

(a) Telling the user an item was added successfully

You must enter a password

(b) a required field cannot be left empty

resource deleted successfully

resource updated successfully

(c) Telling the user an item was deleted successfully

(d) Telling the user an item was updated successfully

Figure A.2: Validation



B REST API Documentation



Smart Home

Overview

This is the documentation for smart home web rest api by [reem alghamdi](#)

Version information

Version : 1.0

Contact information

Contact Email : reem.brain@gmail.com

License information

License : MIT

License URL : <https://opensource.org/licenses/MIT>

Terms of service : null

URI scheme

Host : gp.reem-codes.com

BasePath : /api

Schemes : HTTPS, HTTP

Tags

- Command : What user wants raspberry to do. It is a mapping between the time, the hardware and the configuration
- Configuration : All the possible configuration and status for a given hardware. eg: on, off, red, extended ...
- Hardware : Everything about the sensors and actuators connected to raspberry pi
- RaspberryPi : raspberry pi connected to users
- Response : What the raspberry wants the user to know. It is a mapping between the time of execution, the command triggered and whether the action was done or not
- Schedule : scheduling information for command if any. Specifys the days and time of day the user would like a command to be triggered
- User : saving user information to correctly connect them with their respectful raspberry pi



Security

bearerAuth

Type : apiKey
Name : Authorization
In : HEADER

Paths

Add a new command

POST /command

Parameters

Type	Name	Description	Schema
Body	command <i>optional</i>	The command to create.	command

command

Name	Description	Schema
configurationId <i>required</i>	the id of the configuration to apply to the hardware Example : 3	integer
hardwareId <i>required</i>	the id of the hardware this command targets Example : 1	integer
scheduleId <i>optional</i>	the id of the schedule if any Example : 453	integer

Responses

HTTP Code	Description	Schema
201	Created	Command
405	Invalid input	No Content



Consumes

- application/json

Produces

- application/json

Tags

- Command

get all commands

GET /command

Responses

HTTP Code	Description	Schema
200	return an array of command objects	< Command > array

Produces

- application/json

Tags

- Command

get a command by id

GET /command/{commandId}

Parameters

Type	Name	Schema
Path	commandId <i>required</i>	integer



Responses

HTTP Code	Description	Schema
200	get the command	Command
404	Not found	No Content

Produces

- application/json

Tags

- Command

edit an existing command given its id

```
PUT /command/{commandId}
```

Parameters

Type	Name	Description	Schema
Path	commandId <i>required</i>		integer
Body	command <i>optional</i>	The command to edit.	command

command

Name	Description	Schema
configurationId <i>required</i>	the id of the configuration to apply to the hardware Example : 3	integer
hardwareId <i>required</i>	the id of the hardware this command targets Example : 1	integer
scheduleId <i>optional</i>	the id of the schedule if any Example : 453	integer



Responses

HTTP Code	Description	Schema
200	Edited	Command
404	Not found	No Content
405	Invalid input	No Content

Consumes

- application/json

Produces

- application/json

Tags

- Command

delete a command by id

```
DELETE /command/{commandId}
```

Parameters

Type	Name	Schema
Path	commandId <i>required</i>	integer

Responses

HTTP Code	Description	Schema
204	deleted	No Content
404	Not found	No Content

Tags

- Command



Add a new configuration to a hardware

POST /configuration

Parameters

Type	Name	Description	Schema
Body	configuration <i>optional</i>	The configuration to create.	configuration

configuration

Name	Description	Schema
description <i>optional</i>	information about the configuration Example : "EXTEND means that it will become longer by 3cm"	string
hardwareId <i>required</i>	the hardware this configuration belongs to Example : "ON belongs to the hardwareID 1, which is RED LED"	integer
name <i>required</i>	the configuration name Example : "ON"	string

Responses

HTTP Code	Description	Schema
201	Created	Configuration
405	Invalid input	No Content

Consumes

- [application/json](#)

Produces

- [application/json](#)

Tags

- Configuration



get a configuration by id

GET /configuration/{configurationId}

Parameters

Type	Name	Schema
Path	configurationId <i>required</i>	integer

Responses

HTTP Code	Description	Schema
200	get the configuration	Configuration
404	Not found	No Content

Produces

- [application/json](#)

Tags

- Configuration

edit an existing configuration given its id

PUT /configuration/{configurationId}

Parameters

Type	Name	Description	Schema
Path	configurationId <i>required</i>		integer
Body	configuration <i>optional</i>	The configuration to edit.	configuration

configuration



Name	Description	Schema
description <i>optional</i>	information about the configuration Example : "EXTEND means that it will become longer by 3cm"	string
hardwareId <i>optional</i>	the hardware this configuration belongs to Example : "ON belongs to the hardwareID 1, which is RED LED"	integer
name <i>optional</i>	the configuration name Example : "ON"	string

Responses

HTTP Code	Description	Schema
200	Edited	Configuration
404	Not found	No Content
405	Invalid input	No Content

Consumes

- application/json

Produces

- application/json

Tags

- Configuration

delete a configuration by id

```
DELETE /configuration/{configurationId}
```

Parameters

Type	Name	Schema
Path	configurationId <i>required</i>	integer



Responses

HTTP Code	Description	Schema
204	deleted	No Content
404	Not found	No Content

Tags

- Configuration

connect a user to raspberry pi

POST /connect

Parameters

Type	Name	Description	Schema
Body	user_raspberry	The user and raspberry data <i>optional</i>	user_raspberry

user_raspberry

Name	Description	Schema
raspberry_id <i>required</i>	the raspberry id that needs to be connected to the logged in user Example : 3	integer

Responses

HTTP Code	Description	Schema
200	return an array of raspberry pi objects connected	< RaspberryPi > array
405	Invalid input	No Content



Consumes

- application/json

Produces

- application/json

Tags

- RaspberryPi
- User

disconnect a user from a raspberry pi

POST /disconnect

Parameters

Type	Name	Description	Schema
Body	user_raspberry	The user and raspberry data <i>optional</i>	user_raspberry

user_raspberry

Name	Description	Schema
raspberry_id	the raspberry id that needs to be disconnected from the logged in user Example : 4	integer

Responses

HTTP Code	Description	Schema
200	return an array of raspberry pi objects connected	< RaspberryPi > array
405	Invalid input	No Content

Consumes

- application/json



C Code

C.1 Web Application

Listing C.1: __init__.py

```
from flask import Flask
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_bcrypt import Bcrypt
from flask_jwt_extended import JWTManager
```

"""

In this file, the flask app and its plugins are made and configured.

"""

```
app = Flask(__name__)
app.config.from_object(Config)
```

```
db = SQLAlchemy(app)
migrate = Migrate(app, db)
```

```
bcrypt = Bcrypt(app)
jwt = JWTManager(app)
```

```
from web import view, model, auth
```

Listing C.2: view.py

```
from sqlalchemy.exc import IntegrityError, InvalidRequestError
```

```
from web import app, db
```



```
from flask import jsonify, request, render_template
from web.model import Hardware, Command, Schedule, Response, User, Raspb
from config import Config
from flask_jwt_extended import jwt_required, get_jwt_identity
from sqlalchemy import or_
"""
error handling
"""
@app.errorhandler(400)
def error_400(error):
    return jsonify({'message': '{}'.format(error)}), 400
@app.errorhandler(401)
def error_401(error):
    return jsonify({'message': '{}'.format(error)}), 401
@app.errorhandler(403)
def error_403(error):
    return jsonify({'message': '{}'.format(error)}), 403
@app.errorhandler(405)
def error_405(error):
    return jsonify({'message': '{}'.format(error)}), 405
```



```
@app.errorhandler(404)  
  
def error_404(error):  
  
    return jsonify({'message': '{}'.format(error)}), 404
```

```
@app.errorhandler(409)  
  
def error_409(error):  
  
    return jsonify({'message': '{}'.format(error)}), 409
```

```
@app.errorhandler(422)  
  
def error_422(error):  
  
    return jsonify({'message': '{}'.format(error)}), 422
```

```
@app.errorhandler(500)  
  
def error_500(error):  
  
    return jsonify({'message': '{}'.format(error)}), 500
```

```
@app.route("/")  
  
def index():  
  
    # return render_template("actual_main.html")  
  
    return jsonify({"message": "hello :D"})
```

```
@app.route("/privacy_policy")
```



```
def privacy():
    return render_template("privacy-policy.html")
    # return jsonify({"message": "hello :D"})
```

"""

Each model has 5 routes accessing it:

- index: to GET all the rows of the table
- post: to add a new row in the database
- put: to edit an existing row
- delete: to delete an existing row
- get: to get one row of the database

"""

"""

HARDWARE: REEM

"""

```
@app.route("/hardware", methods=["GET"])
@jwt_required
def hardware_index():
    raspberry = request.args.get('raspberry_id')
    ids = {"raspberry_id": raspberry} if raspberry else {}
    return jsonify(Hardware.index(ids))
```

"""

```
@app.route("/hardware", methods=["POST"])
@jwt_required
def hardware_post():
    pass
```



```
try:  
  
    body = request.get_json()  
  
    obj = Hardware.post(body)  
  
    return {"message": Config.POST_MESSAGE, "object": obj}, 201  
  
  
except InvalidRequestError:  
  
    db.session().rollback()  
  
    return jsonify({'message': 'an_invalid_request'}), 409  
  
except IntegrityError:  
  
    return jsonify({'message': 'an_integrity_error_occurred'}), 409  
  
  
  
  
@app.route("/hardware/<_id>" , methods=["GET"] )  
@jwt_required  
  
def hardware_get(_id):  
  
    raspberry = request.args.get('raspberry_id')  
  
    ids = {"id": _id}  
  
    if raspberry:  
  
        ids["raspberry_id"] = raspberry  
  
    obj = Hardware.get(ids)  
  
  
    return jsonify(obj)  
  
  
  
  
@app.route("/hardware/<_id>" , methods=["DELETE"] )  
@jwt_required  
  
def hardware_delete(_id):  
  
    Hardware.delete({"id": _id})  
  
    return {"message": Config.DELETEMESSAGE}, 203
```



```
@app.route("/hardware/<_id>", methods=["PUT"])
@jwt_required

def hardware_put(_id):
    body = request.get_json()
    obj = Hardware.put({"id": _id}, body)
    return {"message": Config.PUT_MESSAGE, "object": obj}

#
# """
# CONFIGURATION: ABEER
# """
# @app.route("/configuration", methods=["GET"])
# def configuration_index():
#     return jsonify(Configuration.index())
#
#
# @app.route("/configuration", methods=["POST"])
# def configuration_post():
#     body = request.get_json()
#     obj = Configuration.post(body)
#     return {"message": Config.POST_MESSAGE, "object": obj}, 201
#
#
# @app.route("/configuration/<_id>", methods=["GET"])
# def configuration_get(_id):
#     obj = Configuration.get({"id": _id})
#     return jsonify(obj)
```



#

#

```
# @app.route("/configuration/<_id>", methods=["DELETE"])
def configuration_delete(_id):
    Configuration.delete({"id": _id})
    return {"message": Config.DELETEMESSAGE}, 203
#
#
# @app.route("/configuration/<_id>", methods=["PUT"])
def configuration_put(_id):
    body = request.get_json()
    obj = Configuration.put({"id": _id}, body)
    return {"message": Config.PUTMESSAGE, "object": obj}
```

"""

COMMAND: SARAH

"""

```
@app.route("/command", methods=["GET"])
```

```
@jwt_required
```

```
def command_index():
```

```
    ids = {}
```

```
    raspberry = request.args.get('raspberry_id')
```

```
    hardware = request.args.get('hardware_id')
```

```
    schedule_id = request.args.get('schedule_id')
```

```
    if raspberry:
```

```
        obj = Command.query.join(Hardware).\\
```

```
            outerjoin(Response, Response.command_id==Com-
```

```
            filter(Hardware.raspberry_id==raspberry # fi
```



```

        , or_ ( Response . isDone==None , Response
        ). all()

    return jsonify ( obj )

if hardware:

    ids [ " hardware_id" ] = hardware

if schedule_id and schedule_id == " not_null ":

    print ( " no_schedule" )

return jsonify ( Command . index ( ids , True ) )

return jsonify ( Command . index ( ids ) )



@app . route ( "/command" , methods=[ "POST" ] )

@jwt_required

def command_post ( ):

    try:

        body = request . get_json ( )

        obj = Command . post ( body )

        return { " message": Config . POST_MESSAGE , " object": obj } , 201

    except InvalidRequestError:

        db . session ( ) . rollback ( )

        return jsonify ( { ' message': ' an_invalid_request' } ) , 409

    except IntegrityError:

        return jsonify ( { ' message': ' an_integrity_error_occurred' } ) , 409


@app . route ( "/command/<_id>" , methods=[ "GET" ] )

@jwt_required

def command_get ( _id ):

    ids = { " id": _id }

```



```
raspberry = request.args.get('raspberry_id')
hardware = request.args.get('hardware_id')

if raspberry:
    ids["raspberry_id"] = raspberry

if hardware:
    ids["hardware_id"] = hardware
obj = Command.get(ids)

return jsonify(obj)

@app.route('/command/<_id>', methods=['DELETE'])
@jwt_required
def command_delete(_id):
    Command.delete({'id': _id})
    return {"message": Config.DELETE_MESSAGE}, 203

@app.route('/command/<_id>', methods=['PUT'])
@jwt_required
def command_put(_id):
    body = request.get_json()
    obj = Command.put({'id': _id}, body)
    return {"message": Config.PUT_MESSAGE, "object": obj}

"""

SCHEDULE: NOUF

"""

@app.route('/schedule', methods=['GET'])
```



```
@jwt_required

def schedule_index():
    return jsonify(Schedule.index())

@app.route("/schedule", methods=["POST"])
@jwt_required

def schedule_post():
    try:
        body = request.get_json()
        obj = Schedule.post(body)
        return {"message": Config.POST_MESSAGE, "object": obj}, 201
    except InvalidRequestError:
        db.session().rollback()
        return jsonify({'message': 'an_invalid_request'}), 409
    except IntegrityError:
        return jsonify({'message': 'an_integrity_error_occurred'}), 409

@app.route("/schedule/<_id>", methods=["GET"])
@jwt_required

def schedule_get(_id):
    obj = Schedule.get({"id": _id})
    return jsonify(obj)

@app.route("/schedule/<_id>", methods=["DELETE"])
@jwt_required

def schedule_delete(_id):
```



```
Schedule.delete({“id”: _id})  
  
return {“message”: Config.DELETE_MESSAGE}, 203  
  
  
  
@app.route(”/schedule/<_id>”, methods=[”PUT”])  
@jwt_required  
  
def schedule_put(_id):  
  
    body = request.get_json()  
  
    obj = Schedule.put({“id”: _id}, body)  
  
    return {“message”: Config.PUTMESSAGE, “object”: obj}  
  
”””  
  
RESPONSE: MONA  
”””  
  
@app.route(”/response”, methods=[”GET”])  
@jwt_required  
  
def response_index():  
  
    obj = Response.query.join(Command)\br/>        .join(Hardware).join(Raspberry) \br/>        .join(User, Raspberry.users)\br/>        .filter(User.id == get_jwt_identity())  
  
    r_obj = obj.all()  
  
    for response in obj:  
  
        command = Command.query.filter_by(id=response.command_id, schedule_id=_id)  
  
        if command is not None:  
  
            db.session.delete(command)  
  
            db.session.delete(response)  
  
    db.session.commit()
```



```
return jsonify( r_obj )
```

```
@app.route("/response" , methods=["POST"])
@jwt_required

def response_post():
    try:
        body = request.get_json()
        obj = Response.post(body)
        return {"message": Config.POST_MESSAGE, "object": obj} , 201
    except InvalidRequestError:
        db.session().rollback()
        return jsonify({ 'message': 'an_invalid_request' }) , 409
    except IntegrityError:
        return jsonify({ 'message': 'an_integrity_error_occurred' }) , 409

@app.route("/response/<_id>" , methods=["GET"])
@jwt_required

def response_get(_id):
    obj = Response.get({"id": _id})
    return jsonify(obj)

@app.route("/response/<_id>" , methods=["DELETE"])
@jwt_required

def response_delete(_id):
    Response.delete({"id": _id})
    return {"message": Config.DELETEMESSAGE} , 203
```



```
@app.route("/response/<_id>" , methods=["PUT"] )  
@jwt_required  
def response_put(_id):  
    body = request.get_json()  
    obj = Response.put({"id": _id} , body)  
    return {"message": Config.PUTMESSAGE, "object": obj}  
"""  
USER: REEM  
"""  
  
@app.route("/user" , methods=["GET"] )  
@jwt_required  
def user_index():  
    raspberry_id = request.args.get('raspberry_id')  
    if raspberry_id:  
        obj = User.query.join(Raspberry , User.raspberries).filter(Raspberry.id == raspberry_id)  
        return jsonify(obj)  
    else:  
        return jsonify(User.index())  
  
@app.route("/user" , methods=["POST"] )  
def user_post():  
    try:  
        body = request.get_json()
```



```

obj = User.post(body)

return {"message": Config.POST_MESSAGE, "object": obj}, 201

except InvalidRequestError:
    db.session().rollback()
    return jsonify({'message': 'an_invalid_request'}), 409

except IntegrityError:
    return jsonify({'message': 'an_integrity_error_occurred'}), 409

@app.route("/user/<_id>", methods=["GET"])
@jwt_required

def user_get(_id):
    obj = User.get({"id": _id})
    return jsonify(obj)

@app.route("/user/<_id>", methods=["DELETE"])
@jwt_required

def user_delete(_id):
    User.delete({"id": _id})
    return {"message": Config.DELETE_MESSAGE}, 203

@app.route("/user/<_id>", methods=["PUT"])
@jwt_required

def user_put(_id):
    body = request.get_json()
    obj = User.put({"id": _id}, body)
    return {"message": Config.PUT_MESSAGE, "object": obj}

```



” ” ”

RASPBERRY: REEM

” ” ”

```
@app.route("/raspberry", methods=["GET"])
@jwt_required

def raspberry_index():
    user_id = request.args.get('user_id')
    if user_id:
        obj = Raspberry.query.join(User, Raspberry.users).filter(User.id == user_id)
        return jsonify(obj)
    else:
        return jsonify(Raspberry.index())

@app.route("/raspberry", methods=["POST"])
def raspberry_post():
    try:
        body = request.get_json()
        obj = Raspberry.post(body)
        return {"message": Config.POST_MESSAGE, "object": obj}, 201
    except InvalidRequestError:
        db.session().rollback()
        return jsonify({'message': 'an invalid request'}), 409
    except IntegrityError:
        return jsonify({'message': 'an integrity error occurred'}), 409
```



```
@app.route("/raspberry/<_id>", methods=["GET"])
@jwt_required

def raspberry_get(_id):
    obj = Raspberry.get({"id": _id})
    return jsonify(obj)

@app.route("/raspberry/<_id>", methods=["DELETE"])
@jwt_required

def raspberry_delete(_id):
    Raspberry.delete({"id": _id})
    return {"message": Config.DELETE_MESSAGE}, 203

@app.route("/raspberry/<_id>", methods=["PUT"])
@jwt_required

def raspberry_put(_id):
    body = request.get_json()
    obj = Raspberry.put({"id": _id}, body)
    return {"message": Config.PUT_MESSAGE, "object": obj}

@app.route("/raspberry-user", methods=["PUT"])
@jwt_required

def raspberry_user_put():
    body = request.get_json()
    obj = Raspberry.get({"id": body["raspberry_id"]})
    user = User.get({"id": get_jwt_identity()})
    if user:
        if user.id == obj.id:
            user.raspberry_id = obj.id
            db.session.commit()
            return {"message": "User assigned to raspberry successfully"}, 200
        else:
            return {"message": "User already assigned to another raspberry"}, 400
    else:
        return {"message": "User not found"}, 404
```



```
        user.raspberries.append(obj)
        db.session.commit()
        return {"message": Config.POSTMESSAGE, "object": obj}, 201
```

```
@app.route("/raspberry-user", methods=["DELETE"])
@jwt_required

def raspberry_user_delete():
    body = request.get_json()
    obj = Raspberry.get({"id": body["raspberry_id"]})
    user = User.get({"id": get_jwt_identity()})
    user.raspberries.remove(obj)
    db.session.commit()
    return {"message": Config.DELETEMESSAGE}, 203
```

Listing C.3: model.py

```
"""ORM=> Object Relational Mapping"""
from web import db, bcrypt
from dataclasses import dataclass
from datetime import datetime
"""


```

Each class represent a model i.e. a table in the database all of the models inherits from the Base class , which has the CRUD methods RevokedToken class is there for the logout token strings , part of the auth module

```
class Base:
```

```
    id: int
```



updateAt: datetime

```
id = db.Column(db.Integer, primary_key=True, autoincrement=True)
updateAt = db.Column(db.DateTime, default=datetime.utcnow)
```

@classmethod

```
def get(cls, ids):
    obj = cls.query.filter_by(**ids).first_or_404()
    return obj
```

@classmethod

```
def index(cls, ids=None):
    obj = cls.query
    if ids:
        obj = obj.filter_by(**ids)
    obj = obj.all()
    return obj
```

@classmethod

```
def delete(cls, ids):
    obj = cls.query.filter_by(**ids).first_or_404()
    db.session.delete(obj)
    db.session.commit()
    return
```

@classmethod

```
def post(cls, kw):
    obj = cls(**kw)
    obj.updateAt = datetime.utcnow()
```



```
db.session.add(obj)
db.session.commit()
return obj
```

```
@classmethod
def put(cls, ids, kw):
    obj = cls.query.filter_by(**ids).first_or_404()
    for k in kw:
        obj.__setattr__(k, kw[k])
    obj.updateAt = datetime.utcnow()
    db.session.commit()
    return obj
```

```
#
# @dataclass
# class Configuration(db.Model, Base):
#     id: int
#     updateAt: datetime
#     name: str
#     desc: str
#     hardware_id: int
#
#     name = db.Column(db.String)
#     desc = db.Column(db.Text)
#
#     hardware_id = db.Column(db.Integer, db.ForeignKey('hardware.id'))
#     commands = db.relationship("Command", backref="configuration", lazy=True)
#     # hardware = db.relationship("Hardware", useList=False, foreign_key=hardware_id)
# 
```



@dataclass

class Hardware(db.Model, Base):

id: **int**

 updateAt: datetime

 name: **str**

 icon: **str**

 desc: **str**

 gpio: **int**

 raspberry_id: **int**

 status: **bool**

 name = db.Column(db.String)

 icon = db.Column(db.String)

 desc = db.Column(db.Text)

 gpio = db.Column(db.Integer)

 raspberry_id = db.Column(db.Integer, db.ForeignKey('raspberry.id'), o

 commands = db.relationship("Command", backref="hardware", lazy='dyna

 status = db.Column(db.Boolean)

def __hash__(self):

return hash(self.id)

@dataclass

class Schedule(db.Model, Base):

id: **int**

 updateAt: datetime



days: int

time: str

days = db.Column(db.Integer)

time = db.Column(db.String)

def __hash__(self):

return hash(self.id)

@dataclass

class Command(db.Model, Base):

 id: int

 updateAt: datetime

 hardware_id: int

 configuration: bool

 schedule_id: int

 schedule: Schedule

 hardware_id = db.Column(db.Integer, db.ForeignKey('hardware.id'), on

 configuration = db.Column(db.Boolean)

 schedule_id = db.Column(db.Integer, db.ForeignKey('schedule.id'), nu

 schedule = db.relationship("Schedule", foreign_keys=[schedule_id], p

 responses = db.relationship("Response", backref="command", lazy='dyn

@classmethod



```

def index( cls , ids=None , schedule_not_null=False ):

    obj = cls . query

    if ids :

        obj = obj . filter_by (**ids)

    if schedule_not_null :

        print( "inside_not_null" )

        obj = obj . filter( cls . schedule_id . isnot( None ) )

    obj = obj . all()

    return obj


def __hash__( self ):

    return hash( self . id )



@dataclass

class Response( db . Model , Base ):

    id: int

    updateAt: datetime

    isDone: bool

    message: str

    executionTime: str

    isRead: bool

    command_id: int

    isDone = db . Column( db . Boolean )

    message = db . Column( db . String )

    executionTime = db . Column( db . DateTime )

    isRead = db . Column( db . Boolean )
  
```



```
command_id = db.Column(db.Integer, db.ForeignKey('command.id'), onde
```

```
def __hash__(self):  
    return hash(self.id)
```

```
RaspberryUser = db.Table(  
    'raspberry_user',  
    db.Column('raspberry_id', db.Integer, db.ForeignKey('raspberry.id')),  
    db.Column('user_id', db.Integer, db.ForeignKey('user.id')),  
    db.PrimaryKeyConstraint('raspberry_id', 'user_id'))  
)
```

```
@dataclass  
class Raspberry(db.Model, Base):  
    id: int  
    updateAt: datetime  
    name: str  
  
    name = db.Column(db.String, nullable=True)  
    hardwares = db.relationship("Hardware", backref="raspberry", lazy='dynamic')  
    users = db.relationship('User', secondary=RaspberryUser, back_populates='raspberry')  
  
    def __hash__(self):  
        return hash(self.id)  
  
class RevokedToken(db.Model):
```



```
id: int
jti: str
id = db.Column(db.Integer, primary_key=True, autoincrement=True)
jti = db.Column(db.String(120))

def add(self):
    db.session.add(self)
    db.session.commit()

@classmethod
def is_jti_blacklisted(cls, jti):
    query = cls.query.filter_by(jti=jti).first()
    return bool(query)

@dataclass
class User(db.Model, Base):
    id: int
    updateAt: datetime
    email: str
    password: str

    email = db.Column(db.String, unique=True)
    password = db.Column(db.String)

raspberries = db.relationship('Raspberry', secondary=RaspberryUser,
@classmethod
def post(cls, kw):
    db.session.rollback()
```



```
kw[ 'password' ] = bcrypt.generate_password_hash(kw[ 'password' ]).d  
obj = cls(**kw)  
obj.updateAt = datetime.utcnow()  
db.session.add(obj)  
db.session.commit()  
return obj
```

```
@classmethod  
  
def put(cls, ids, kw):  
    obj = cls.query.filter_by(**ids).first_or_404()  
    for k in kw:  
        if k is 'password':  
            kw[ 'password' ] = bcrypt.generate_password_hash(kw[ 'password' ])  
            obj.__setattr__(k, kw[k])  
        else:  
            obj.__setattr__(k, kw[k])  
    obj.updateAt = datetime.utcnow()  
    db.session.commit()  
    return obj  
  
def __hash__(self):  
    return hash(self.id)
```

C.2 Raspberry Pi

Listing C.4: main.py

```
import datetime  
  
import time  
  
from config import Config, config_extract  
import RPi.GPIO as GPIO
```



```
import api_calls as api
```

```
def execute(_gpio, _on, command_id, hw_id):
```

```
    """
```

```
:param _gpio: the GPIO pin connected
```

```
:param _on: Whether to make it high or low
```

```
    """
```

```
    print("I'm executing..")
```

```
    print(_gpio, _on)
```

```
    GPIO.setmode(GPIO.BEAN)
```

```
    GPIO.setup(_gpio, GPIO.OUT)
```

```
    GPIO.output(_gpio, GPIO.LOW if _on else GPIO.HIGH)
```

```
    # time.sleep(0.5)
```

```
    # GPIO.output(_gpio, GPIO.HIGH)
```

```
    # time.sleep(0.5)
```

```
    res = {"isDone": True,
```

```
            "message": "command_executed_successfully",
```

```
            "executionTime": datetime.datetime.now().strftime('%a, %d %b %Y %H:%M:%S'),
```

```
            "command_id": command_id}
```

```
}
```

```
    api.post_response(res)
```

```
    api.put.hardware(hw_id, _on)
```

```
def main():
```

```
    now = datetime.datetime.now() # today's datetime
```



```
# GET all commands from server and save it in data variable
data = api.get_commands()

for command in data: # looping through commands
    print(command)

    hw_id = command["hardware_id"]
    _gpio = api.get.hardware(command["hardware_id"])[ "gpio" ]
    _conf = command[ "configuration" ]
    command_id = command[ "id" ]

    if command[ "schedule_id" ] is not None: # if the command is scheduled
        # GET schedule/<id> from the server and save it in schedule
        schedule = api.get_schedule(str(command[ "schedule_id" ]))

        # the schedule has a time string (e.x. "9:45 AM") and an int
        schedule_time = datetime.datetime.strptime(schedule[ "time" ] , "%H:%M %p")

        # time string into datetime object
        # turn the time string into today's date with the time given
        execution_time = datetime.datetime(now.year, now.month, now.day, schedule_time.hour, schedule_time.minute, schedule_time.second)

        # find today's weekday (e.x. sun, mon, tues)
        weekday = execution_time.today().weekday()

        # return the days of execution from the schedule
        days = binary_to_position('{0:07b}'.format(schedule[ "days" ]))

        # if the weekday is part of the execution days and the time
        # is valid
        if weekday in days and execution_time <= now:
            execute( _gpio=_gpio, _on=_conf, command_id=command_id, hardware_id=hw_id, update_at=schedule_time)

    # execute

else: # immediate command
    # turn updateAt from string into datetime
    datetime_object = datetime.datetime.strptime(command[ "update_at" ], "%H:%M %p")
```



```
if datetime_object <= now:    # if the execution time before now

    # EXECUTE METHOD

    execute( _gpio=_gpio , _on=_conf , command_id=command_id , h

def binary_to_position(binary):
    """
    :param binary:
    :return: array of positions on
    This method takes the binary string that has 7 bits representing the
    if the bit is equal to 1, it means that day is an execution day and
    """
    positions = []
    for index , digit in enumerate(binary):
        if int(digit) == 1:
            positions.append(index)
    return positions

if __name__ == '__main__':
    config_extract()
    if Config.ID == -1:
        api.register()
    if Config.TOKEN == "":
        api.login()
    while True:
        main()
        time.sleep(Config.SECONDS)
    # while True:
    # main(Config.BASE)
```



```
# time.sleep(Config.SECONDS) # Time loop
```

C.3 Android Application

Listing C.5: Command.java

```
package com.reem_codes.gp_android.model;

public class Command extends Base{

    private int id;
    private String updateAt;

    private Schedule schedule;
    private int hardware_id;
    private boolean configuration;

    public Command() {

    }

    public Command(int id, String updateAt, Schedule schedule, int hardw
        this.id = id;
        this.updateAt = updateAt;
        this.schedule = schedule;
        this.hardware_id = hardware_id;
        this.configuration = configuration;
    }

    public int getHardware_id() {
        return hardware_id;
    }
}
```



}

```
public void setHardware_id(int hardware_id) {
```

```
    this.hardware_id = hardware_id;
```

```
}
```

```
public String getUpdateAt() {
```

```
    return updateAt;
```

```
}
```

```
public void setUpdateAt(String updateAt) {
```

```
    updateAt = updateAt;
```

```
}
```

```
public int getId() {
```

```
    return id;
```

```
}
```

```
public void setId(int id) {
```

```
    this.id = id;
```

```
}
```

```
public Schedule getSchedule() {
```

```
    return schedule;
```

```
}
```

```
public void setSchedule(Schedule schedule) {
```

```
    this.schedule = schedule;
```

```
}
```



```
public boolean isConfiguration() {  
    return configuration;  
}  
  
public void setConfiguration(boolean configuration) {  
    this.configuration = configuration;  
}  
}
```

Listing C.6: CommandAdapter.java

```
package com.reem_codes.gp_android.adapter;  
  
import android.app.Activity;  
import android.content.Context;  
import android.content.Intent;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.ImageButton;  
import android.widget.ListView;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
  
import com.google.gson.Gson;
```



```

import com.google.gson.reflect.TypeToken;
import com.reem_codes.gp_android.R;
import com.reem_codes.gp_android.activity.BaseActivity;
import com.reem_codes.gp_android.activity.HardwareListActivity;
import com.reem_codes.gp_android.activity.NewCommandActivity;
import com.reem_codes.gp_android.model.Base;
import com.reem_codes.gp_android.model.Command;
import com.reem_codes.gp_android.model.Hardware;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import okhttp3.Call;
import okhttp3.Callback;
import okhttp3.Request;
import okhttp3.Response;

public class CommandAdapter extends ArrayAdapter<Command> {
  public final static int LAUNCH_EDIT_COMMAND = 4;

  Context context;
  static Command command;
  ListView listView;
  Hardware hardware;

  public CommandAdapter(Context context, List<Command> commands, ListView
    super(context, R.layout.command_item_adapter, commands);
  this.context = context;
}

```



```
this.listView = listView;  
}  
  
public CommandAdapter(Context context, List<Command> commands, ListView listView) {  
    this(context, commands, listView);  
    this.hardware = hardware;  
}  
  
@NotNull  
@Override  
public View getView(@Final int position, @Nullable View convertView,  
    LayoutInflater inflater = LayoutInflater.from(getContext());  
  
    // specify each item's layout  
    View item = inflater.inflate(R.layout.command_item_adapter, parent);  
  
    // get the current command from the list passed to the adapter  
    command = getItem(position);  
  
    // take the views from the layout  
    final TextView config = (TextView) item.findViewById(R.id.config);  
    ImageButton delete = (ImageButton) item.findViewById(R.id.delete);  
    ImageButton edit = (ImageButton) item.findViewById(R.id.edit);  
  
    /* This is the code to turn the days integer value into binary,  
     * then into an array of on positions representing the days in which  
     */  
    ArrayList<Integer> positions = intToBinaryPositions(command.getDays());  
    String days = "";  
    if (positions.size() == 7) {  
        days = days + "1";  
        for (int i = 0; i < positions.size(); i++) {  
            days = days + ((positions.get(i)) ? "1" : "0");  
        }  
    } else if (positions.size() == 8) {  
        days = days + "0";  
        for (int i = 0; i < positions.size(); i++) {  
            days = days + ((positions.get(i)) ? "1" : "0");  
        }  
    }  
    config.setText(days);  
    delete.setOnClickListener(v -> deleteCommand(command));  
    edit.setOnClickListener(v -> editCommand(command));  
}
```



```

days = "everyday";

} else {

for(int i = 0; i < positions.size(); i++) {

    int day = positions.get(i);

    switch (day) {

        case 0: days += "Mon"; break;
        case 1: days += "Tue"; break;
        case 2: days += "Wed"; break;
        case 3: days += "Thurs"; break;
        case 4: days += "Fri"; break;
        case 5: days += "Sat"; break;
        case 6: days += "Sun"; break;

    }

    if(i != positions.size() -1 ) {

        days += ", ";

    }

}

// make the config string and set the view text to the string

String configString = String.format("%s at %s on %s", command.is
config.setText(configString);

// if the user clicked on delete button, call the API to delete

delete.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View view) {

        command = getItem(position);

        try {


```



```
String url = context.getString(R.string.api_url) + "  
  
// then, we build the request by providing the url,  
Request request = new Request.Builder()  
    .url(url)  
    .delete()  
    .addHeader("Authorization", "Bearer " + ((BaseActivity)context).client.token)  
    .build();  
  
((BaseActivity)context).client.newCall(request).enqueue(new Callback() {  
    @Override  
    public void onFailure(Call call, IOException e)  
        ((Activity)context).runOnUiThread(new Runnable() {  
            @Override  
            public void run() {  
                Toast.makeText(context, "command not found", Toast.LENGTH_SHORT).show();  
            }  
        });  
    }  
  
    @Override  
    public void onResponse(Call call, Response response) {  
        String result = response.body().string();  
        System.out.println("GPDEBUG_results_are_" + result);  
  
        // use Gson to parse from a string to object  
    }  
});
```



```
// first create Gson object
Gson gson = new Gson();
// specify the object type: is the string a
TypeToken<Base> typeToken = new TypeToken<Base>(){
    // create the login object using the response
    final Base res = gson.fromJson(result, typeToken);
    ((Activity)context).runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(context, res.getMessage(),
                    (( BaseActivity )context).loadActivity()
            );
        }
    });
};

} catch (Exception e){
    Toast.makeText(context, "error while deleting, please
try again");
}

});

// if the user clicked on the edit button, open the NewCommandActivity
edit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        command = getItem(position);
        Intent intent = new Intent(context, NewCommandActivity.class);
        intent.putExtra("isEdit", true);
        startActivity(intent);
    }
});
```



```
        intent . putExtra ( " hardware " , hardware . getName ( ) ) ;  
        intent . putExtra ( " status " , hardware . isStatus ( ) ) ;  
        Gson gson = new Gson ( ) ;  
        intent . putExtra ( " command " , gson . toJson ( command ) ) ;  
        System . out . println ( " GPDEBUG_ " + gson . toJson ( command ) ) ;  
        System . out . println ( " GPDEBUG_ " + command . isConfiguration ( ) ) ;  
        (( Activity ) context ) . startActivityForResult ( intent , LAUNCH_CODE ) ;  
    }  
}  
  
public static ArrayList<Integer> intToBinaryPositions (int x) {  
    /**  
     * This method takes an integer number,  
     * converts it to binary  
     * and then returns an array of on positions */  
    String binary = toBinary (x, 7);  
    ArrayList<Integer> positions = new ArrayList<>();  
    for (int i = 0; i < binary . length ( ) ; i ++){  
        if (binary . charAt (i) == '1') {  
            positions . add (i);  
        }  
    }  
  
    return positions;  
}  
  
public static String toBinary (int x, int len) {
```



```
StringBuilder result = new StringBuilder();  
  
for (int i = len - 1; i >= 0 ; i--) {  
    int mask = 1 << i;  
    result.append((x & mask) != 0 ? 1 : 0);  
}  
  
return result.toString();  
}  
  
public static boolean[] isSelected(int x){  
    /**  
     * This method takes an integer number,  
     * converts it to binary  
     * and then returns an array of on positions */  
    String binary = toBinary(x, 7);  
    boolean[] positions = new boolean[7];  
    for(int i = 0; i < binary.length(); i++){  
        if(binary.charAt(i) == '1'){  
            positions[i] = true;  
        }else {  
            positions[i] = false;  
        }  
    }  
  
    return positions;  
}
```



}

Listing C.7: CommandListActivity.java

```
package com.reem_codes.gp_android.activity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import com.reem_codes.gp_android.R;
import com.reem_codes.gp_android.adapter.CommandAdapter;
import com.reem_codes.gp_android.model.Command;
import com.reem_codes.gp_android.model.Created;
import com.reem_codes.gp_android.model.Hardware;
import com.reem_codes.gp_android.model.Schedule;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import okhttp3.Call;
```



```
import okhttp3.Callback;  
import okhttp3.Request;  
import okhttp3.RequestBody;  
import okhttp3.Response;  
  
public class CommandListActivity extends BaseActivity {  
  
    public static List<Command> commands;  
    final int LAUNCHADD_COMMAND = 1;  
    Hardware hardware;  
    ListView listView;  
    Command eCommand;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_command_list);  
        // check if user is properly logged in, or redirect to login page  
        checkUser(this);  
  
        // get the parent hardware that was clicked in the previous activity  
        Intent intent = getIntent();  
        int hardware_index = intent.getIntExtra("hardware_index", -1);  
        if(hardware_index != -1) {  
            hardware = HardwareListActivity.hardwares.get(hardware_index);  
            super.setToolbar(hardware.getName() + "'s_Command_List");  
            ((TextView) findViewById(R.id.hw_name)).setText(hardware.getName());  
            ((TextView) findViewById(R.id.hw_config)).setText(hardware.getConfig());  
        }  
    }  
}
```



```
commands = new ArrayList<>();  
  
listView = (ListView) findViewById(R.id.commands);  
  
/** new command  
 * when the add command button is clicked , the NewCommand activity  
 * **/  
  
ImageButton addCommand = (ImageButton) findViewById(R.id.add_command);  
addCommand.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View view) {  
  
        Intent i = new Intent(getApplicationContext(), NewCommandActivity.class);  
        i.putExtra("hardware", hardware.getName());  
        i.putExtra("status", hardware.isStatus());  
        startActivityForResult(i, LAUNCHADDCOMMAND);  
    }  
});  
  
loadActivity();  
  
}  
  
@Override  
public void loadActivity() {  
    try {  
        getCommandApi();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



```
        } catch (IOException e) {  
            Toast.makeText(this, "please_check_network_and_try_again", To-  
            }  
  
public void getCommandApi() throws IOException{  
    /**  
     * This method gets the command list from the api and updates the  
     */  
    url = getString(R.string.api_url) + "/command?schedule_id=not_nu-  
    Request request = new Request.Builder()  
        .url(url)  
        .get()  
        .addHeader("Authorization", "Bearer " + currentLoggedUser  
        .build();  
  
    client.newCall(request).enqueue(new Callback() {  
  
        @Override  
        public void onFailure(Call call, IOException e) {  
            runOnUiThread(new Runnable() {  
                @Override  
                public void run() {  
                    Toast.makeText(CommandListActivity.this, "please  
                }  
            } );  
        }  
    };
```

@Override

```
public void onResponse(Call call, Response response) throws
{
    String result = response.body().string();
    System.out.println("GPDEBUG_results_are_" + result);

    // parse result from string into object
    Gson gson = new Gson();
    TypeToken<ArrayList<Command>> typeToken = new TypeToken<ArrayList<Command>>() {
        commands = gson.fromJson(result, typeToken.getType());
    };
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            // update array adapter
            ArrayAdapter arrayAdapter = new CommandAdapter(CommandListActivity.this, R.layout.list_item, commands);
            listView.setAdapter(arrayAdapter);
        }
    });
}

}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    /**
     * once the results are returned from the new command activity,
     * check if the user cancelled the operation. If not upload the
     * if it was scheduled the schedule must be posted/put first. Then
     */
}
```



*/

```
super.onActivityResult(requestCode, resultCode, data);  
  
if (requestCode == LAUNCHADDCOMMAND || requestCode == Command)  
    if (resultCode == Activity.RESULT_OK) {  
        boolean config = data.getBooleanExtra("config", true);  
        boolean isEdit = data.getBooleanExtra("isEdit", false);  
        boolean isScheduled = data.getBooleanExtra("isScheduled", false);  
        if (isEdit){  
            TypeToken<Command> token = new TypeToken<Command>(){  
                eCommand = (new Gson()).fromJson(data.getStringExtra("command"));  
                System.out.println("GPDEBUG_IS_EDIT");  
            }  
  
        if (isScheduled) {  
            boolean isAM = data.getBooleanExtra("isAM", true);  
            int hour = data.getIntExtra("hour", -1);  
            int minute = data.getIntExtra("minute", -1);  
            int daysInt = data.getIntExtra("days", -1);  
  
            String time = String.format("%d:%d.%s", hour, minute, seconds);  
            String sJson = String.format("{\"%s\": \"%s\", \"%s\": \"%s\"}",  
                "try {  
                    postPutScheduleApi(sJson, config, isEdit);  
                } catch (IOException e) {  
                    Toast.makeText(this, "schedule was not added.",  
                        duration).show();  
                }  
            }  
        }  
    }  
    else {  
        try {  
            if (isEdit){  
                eCommand = (new Gson()).fromJson(data.getStringExtra("command"));  
                System.out.println("GPDEBUG_IS_EDIT");  
            }  
            if (isScheduled) {  
                boolean isAM = data.getBooleanExtra("isAM", true);  
                int hour = data.getIntExtra("hour", -1);  
                int minute = data.getIntExtra("minute", -1);  
                int daysInt = data.getIntExtra("days", -1);  
  
                String time = String.format("%d:%d.%s", hour, minute, seconds);  
                String sJson = String.format("{\"%s\": \"%s\", \"%s\": \"%s\"}",  
                    "try {  
                        postPutScheduleApi(sJson, config, isEdit);  
                    } catch (IOException e) {  
                        Toast.makeText(this, "schedule was not added.",  
                            duration).show();  
                    }  
                }  
            }  
        }  
    }  
}
```



```
postPutCommandApi(config, -1, isEdit);  
} catch (IOException e) {  
    Toast.makeText(this, "schedule was not added.",  
        Toast.LENGTH_SHORT).show();  
}  
}  
  
if (resultCode == Activity.RESULT_CANCELED) {  
    Toast.makeText(this, "no command created",  
        Toast.LENGTH_SHORT).show();  
}  
}  
} // onActivityResult  
  
  
public void postPutScheduleApi(String json, final Boolean config,  
    url = getString(R.string.api_url) + "/schedule";  
    RequestBody body = RequestBody.create(JSON, json);  
    System.out.println("GPDEBUG json is " + json);  
  
    Request request;  
  
    if (isEdit) {  
        url += "/" + eCommand.getSchedule().getId();  
        request = new Request.Builder()  
            .url(url)  
            .put(body)  
            .addHeader("Authorization", "Bearer " + currentLogger.token)  
            .build();  
    } else {  
        request = new Request.Builder()  
            .url(url)
```



```
. post ( body )  
. addHeader ( " Authorization " , " Bearer " + currentLogge  
. build ();  
}  
  
client . newCall ( request ) . enqueue ( new Callback () {  
  
    @Override  
    public void onFailure ( Call call , IOException e ) {  
        runOnUiThread ( new Runnable () {  
            @Override  
            public void run () {  
                Toast . makeText ( CommandListActivity . this , " please  
            }  
        } );  
    }  
  
    @Override  
    public void onResponse ( Call call , Response response ) throws  
        String result = response . body () . string ();  
        System . out . println ( " GPDEBUG_results_are_ " + result );  
        Gson gson = new Gson ();  
        TypeToken < Created < Schedule >> typeToken = new TypeToken < C  
        Created < Schedule > schedule = gson . fromJson ( result , typeT  
        postPutCommandApi ( config , schedule . getObject () . getId () ,  
    }  
});  
}
```



```
public void postPutCommandApi(Boolean config, int schedule_id, final String url = getString(R.string.api_url) + "/command";  
String json = String.format("{\"%s\":%d,\"%s\":%b}", "hardware", schedule_id != -1 ? String.format(",\"%s\":%d", "schedule_id", schedule_id) : "", "json");  
  
RequestBody body = RequestBody.create(JSON, json);  
System.out.println("GPDEBUG_json_is_" + json);  
  
Request request;  
if (isEdit){  
    url += "/" + eCommand.getId();  
    request = new Request.Builder()  
        .url(url)  
        .put(body)  
        .addHeader("Authorization", "Bearer_" + currentLogger.token)  
        .build();  
} else {  
    request = new Request.Builder()  
        .url(url)  
        .post(body)  
        .addHeader("Authorization", "Bearer_" + currentLogger.token)  
        .build();  
}  
System.out.println(url);  
  
client.newCall(request).enqueue(new Callback() {  
    @Override
```



```
public void onFailure(Call call, IOException e) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(CommandListActivity.this, "command failed", Toast.LENGTH_SHORT).show();
        }
    });
}

@Override
public void onResponse(Call call, Response response) throws IOException {
    String result = response.body().string();
    System.out.println("GPDEBUG_results_are_" + result);

    Gson gson = new Gson();
    TypeToken<Created<Command>> typeToken = new TypeToken<Created<Command>>() {};
    final Created<Command> command = gson.fromJson(result, typeToken);
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (!isEdit) {
                if (command.getObject().getSchedule() != null) {
                    commands.add(command.getObject());
                }
            }
            ArrayAdapter arrayAdapter = new CommandAdapter(commands);
            listView.setAdapter(arrayAdapter);
        } else {
            Toast.makeText(CommandListActivity.this, "command updated", Toast.LENGTH_SHORT).show();
            loadActivity();
        }
    });
}
```



```
        }  
    }  
});  
}  
});  
}  
}  
  
Listing C.8: NewCommandActivity.java  
  
package com.reem_codes.gp_android.activity;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.app.Activity;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.ImageButton;  
import android.widget.Spinner;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import com.google.gson.Gson;  
import com.google.gson.reflect.TypeToken;  
import com.reem_codes.gp_android.R;  
import com.reem_codes.gp_android.model.Command;  
  
import java.util.ArrayList;
```



```
import java.util.Arrays;  
  
import java.util.List;  
  
  
public class NewCommandActivity extends Activity {  
  
    final int LAUNCH_ADD_SCHEDULE = 2;  
  
  
    boolean isOn = true;  
    boolean isAM, isScheduled;  
    int hour, minute, days;  
    boolean isEdit;  
    Command command;  
    Gson gson;  
    @Override  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_new_command);  
        gson = new Gson();  
  
  
        TextView toolbarText = (TextView) findViewById(R.id.toolbar_title);  
        toolbarText.setText("Add_Command");  
  
  
        Intent intent = getIntent();  
        String name = intent.getStringExtra("hardware");  
        boolean status = intent.getBooleanExtra("status", false);  
        isEdit = intent.getBooleanExtra("isEdit", false);  
        if(name != null) {  
            ((TextView) findViewById(R.id.hardware_name)).setText(name);  
        }  
    }  
}
```



```
if (isEdit) {  
    ((TextView) findViewById(R.id.textView)).setText("Edit_Command")  
    toolbarText.setText("Edit_Command");  
    TypeToken<Command> token = new TypeToken<Command>(){};  
    command = gson.fromJson(intent.getStringExtra("command"), to  
}  
// Get reference of widgets from XML layout  
final Spinner spinner = (Spinner) findViewById(R.id.spinner);  
  
// Initializing a String Array  
String[] configs = new String[] {  
    "ON",  
    "OFF"  
};  
  
final List<String> plantsList = new ArrayList<>(Arrays.asList(  
  
// Initializing an ArrayAdapter  
final ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<  
    spinnerArrayAdapter.setDropDownViewResource(R.layout.command_spinner);  
    spinner.setAdapter(spinnerArrayAdapter);  
  
// set click listener to options  
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener {  
    @Override  
    public void onItemSelected(AdapterView<?> parent, View view,  
        int position, long id) {  
        if (position == 0) {  
            // Set the selected item to ON  
        } else {  
            // Set the selected item to OFF  
        }  
    }  
});
```



@Override

```
public void onNothingSelected(AdapterView<?> parent) {
```

```
}
```

```
});
```

```
if (isEdit) {
```

```
    System.out.println("GPDEBUG_ " + command.isConfiguration());
```

```
    spinner.setSelection(command.isConfiguration() ? spinnerArray.
```

```
}
```

```
/* user may cancel the addition operation */
```

```
ImageButton close = (ImageButton) findViewById(R.id.close);
```

```
close.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

```
    public void onClick(View view) {
```

```
        Intent returnIntent = new Intent();
```

```
        setResult(Activity.RESULT_CANCELED, returnIntent);
```

```
        finish();
```

```
}
```

```
});
```

```
/* validation and execution once the user is done */
```

```
Button done = (Button) findViewById(R.id.done);
```

```
done.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

```
    public void onClick(View view) {
```

```
        Intent returnIntent = new Intent();
```

```
        returnIntent.putExtra("config", spinner.getSelectedItemP
```

```
        if (isEdit) {
```



```
        returnIntent.putExtra("command", gson.toJson(command));
        returnIntent.putExtra("isEdit", true);

    }

    if(isScheduled){
        returnIntent.putExtra("isScheduled", isScheduled);
        returnIntent.putExtra("isAM", isAM);
        returnIntent.putExtra("hour", hour);
        returnIntent.putExtra("minute", minute);
        returnIntent.putExtra("days", days);
    }

    setResult(Activity.RESULT_OK, returnIntent);
    finish();
}

});

Button schedule = (Button) findViewById(R.id.schedule);
if(isEdit) {
    schedule.setText("Edit_Schedule");
}
schedule.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), ScheduleActivity.class);
        if(isEdit) {
            intent.putExtra("isEdit", true);
            intent.putExtra("days", command.getSchedule().getDays());
            intent.putExtra("time", command.getSchedule().getTime());
        }
    }
});
```



```
        }  
        startActivityForResult(intent , LAUNCH_ADD_SCHEDULE);  
    }  
});  
  
}  
  
@Override  
  
protected void onActivityResult(int requestCode , int resultCode , Intent data)  
{  
    super.onActivityResult(requestCode , resultCode , data);  
    if (requestCode == LAUNCH_ADD_SCHEDULE) {  
        if (resultCode == Activity.RESULT_OK) {  
            isScheduled = true;  
            isAM = data.getBooleanExtra("isAM" , true);  
            hour = data.getIntExtra("hour" , -1);  
            minute = data.getIntExtra("minute" , -1);  
            days = data.getIntExtra("days" , -1);  
  
            Toast.makeText(this , isEdit ? "Schedule_edited" : "Sched-  
            }  
            if (resultCode == Activity.RESULT_CANCELED) {  
                Toast.makeText(this , "no_schedule_added" , Toast.LENGTH_  
            }  
        }  
    }  
}  
}  
}
```

Listing C.9: ScheduleActivity.java

```
package com.reem_codes.gp_android.activity;
```



```
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.Activity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.ImageButton;  
import android.widget.RadioButton;  
import androidx.recyclerview.widget.RecyclerView;  
  
import android.widget.TextView;  
import android.widget.Toast;  
import com.google.android.flexbox.AlignItems;  
import com.google.android.flexbox.FlexDirection;  
import com.google.android.flexbox.FlexWrap;  
import com.google.android.flexbox.FlexboxLayoutManager;  
import com.google.android.flexbox.JustifyContent;  
import com.reem_codes(gp_android.R;  
import com.reem_codes(gp_android.adapter.CommandAdapter;  
import com.reem_codes(gp_android.adapter.DayAdapter;  
  
public class ScheduleActivity extends AppCompatActivity {  
  
    EditText hourEdit, minuteEdit;  
    RadioButton am, pm;  
    int hour, minute;
```



```
boolean isAM, isEdit;
```

```
public static boolean[] isSelected = {false, false, false, false, false, false};  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_schedule);  
    TextView toolbarText = (TextView) findViewById(R.id.toolbar_title);  
    toolbarText.setText("Add_Schedule");  
    hourEdit = (EditText) findViewById(R.id.hour);  
    minuteEdit = (EditText) findViewById(R.id.min);  
    am = (RadioButton) findViewById(R.id.am);  
    pm = (RadioButton) findViewById(R.id.pm);  
  
    Intent intent = getIntent();  
    isEdit = intent.getBooleanExtra("isEdit", false);  
    if (isEdit) {  
        toolbarText.setText("Edit_Schedule");  
        String time = intent.getStringExtra("time");  
        int days = intent.getIntExtra("days", 0);  
        String[] split = time.split(":");  
        hour = Integer.valueOf(split[0]);  
        String[] secondSplit = split[1].split(" ");  
        minute = Integer.valueOf(secondSplit[0]);  
        isAM = secondSplit[1].equals("AM") ? true : false;  
  
        hourEdit.setText(" "+hour);  
        minuteEdit.setText(" "+minute);  
        am.setChecked(isAM);
```



```
pm.setChecked (!isAM);  
  
    isSelected = CommandAdapter.isSelected (days);  
}  
/*  
app:flexWrap="wrap"  
app:justifyContent="space-around"  
app:alignItems="center"  
app:alignContent="center"  
*/  
FlexboxLayoutManager manager = new FlexboxLayoutManager (this , Fl  
manager . setJustifyContent ( JustifyContent .SPACE_AROUND) ;  
manager . setFlexWrap (FlexWrap .WRAP) ;  
manager . setAlignItems ( AlignItems .CENTER) ;  
  
RecyclerView recycler = ( RecyclerView ) findViewById (R. id . days ) ;  
recycler . setLayoutManager ( manager ) ;  
recycler . setAdapter (new DayAdapter ( this ) );  
  
ImageButton close = ( ImageButton ) findViewById (R. id . close ) ;  
close . setOnClickListener (new View . OnClickListener () {  
    @Override  
    public void onClick (View view) {  
        Intent returnIntent = new Intent ();  
        setResult ( Activity .RESULT_CANCELED , returnIntent );  
        finish ();  
    }  
});
```



```
Button done = (Button) findViewById(R.id.done);  
done.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
  
        String h = hourEdit.getText().toString();  
        String m = minuteEdit.getText().toString();  
  
        if(h.length() == 0 || m.length() == 0) {  
            Toast.makeText(getApplicationContext(), "Hours and Minutes must be entered");  
            return;  
        }  
        hour = Integer.valueOf(h);  
        minute = Integer.valueOf(m);  
  
        if(hour > 12 || hour < 1) {  
            Toast.makeText(getApplicationContext(), "Hours must be between 1 and 12");  
            return;  
        }  
        if(minute > 59 || minute < 0) {  
            Toast.makeText(getApplicationContext(), "Minutes must be between 0 and 59");  
            return;  
        }  
        int countOn = 0;  
        for(boolean day : isSelected) {  
            if(day){  
                countOn++;  
            }  
        }  
        if(countOn > 1){  
            Toast.makeText(getApplicationContext(), "You can't select more than one day");  
            return;  
        }  
        Intent intent = new Intent(getApplicationContext(), ShowTime.class);  
        intent.putExtra("hour", hour);  
        intent.putExtra("minute", minute);  
        intent.putExtra("selectedDay", day);  
        startActivity(intent);  
    }  
});
```



```
        countOn++;  
    }  
}  
if( countOn == 0){  
    Toast.makeText(getApplicationContext() , "You must select  
    return;  
}  
int daysInt = 0;  
for(int i = 0; i < isSelected.length; i++) {  
    if( isSelected [ i ]){  
        daysInt += Math.pow(2, isSelected.length - i - 1);  
    }  
}  
  
isAM = am.isChecked();  
Intent returnIntent = new Intent();  
returnIntent.putExtra("hour" , hour);  
returnIntent.putExtra("minute" , minute);  
returnIntent.putExtra("isAM" , isAM);  
returnIntent.putExtra("days" , daysInt);  
setResult(Activity.RESULT_OK, returnIntent);  
finish();  
}  
});  
}  
}
```

Listing C.10: activity_command_list.xml



```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:background="@color/white"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.CommandListActivity">
<include
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    layout="@layout/toolbar_plain" />
<LinearLayout
    android:layout_marginRight="24dp"
    android:layout_marginLeft="24dp"
    android:layout_marginBottom="0dp"
    android:layout_marginTop="24dp"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<TextView
    style="@style/h1"
    android:id="@+id/hw_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Linear_Solenoid"/>
<LinearLayout

```



```
        android:layout_marginTop="8dp"  
        android:orientation="horizontal"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content">>  
        <TextView  
            android:layout_marginRight="8dp"  
            android:textSize="20sp"  
            android:textColor="@color/black"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="Status :"/>  
        <TextView  
            android:id="@+id/hw_config"  
            android:textSize="20sp"  
            android:textStyle="bold"  
            android:textColor="@color/pink"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="extend"/>  
    </LinearLayout>  
  </LinearLayout>  
  
<LinearLayout  
    android:layout_marginRight="24dp"  
    android:layout_marginLeft="24dp"  
    android:layout_marginBottom="0dp"  
    android:layout_marginTop="24dp"  
  
    android:orientation="horizontal"
```



```
        android:layout_width="match_parent"  
        android:layout_height="wrap_content">>  
  
<TextView  
        style="@style/h2"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center"  
        android:text="Schedule_Command" />  
  
<ImageButton  
        android:layout_marginLeft="24dp"  
        android:id="@+id/add_command"  
        style="@style/icons"  
        android:background="@drawable/add" />  
  
</LinearLayout>  
  
<ListView  
        android:layout_marginRight="24dp"  
        android:layout_marginLeft="24dp"  
        android:layout_marginBottom="24dp"  
        android:layout_marginTop="8dp"  
        android:id="@+id/commands"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content" />  
  
</LinearLayout>
```

Listing C.11: command_item_adapter.xml



```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:padding="8dp"
    android:layout_marginBottom="8dp"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageButton
        android:id="@+id/delete"
        style="@style/icons"
        android:layout_marginRight="8dp"
        android:background="@drawable/delete"/>
    <ImageButton
        android:id="@+id/edit"
        style="@style/icons"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:background="@drawable/edit"/>
    <TextView
        android:layout_marginLeft="8dp"
        android:layout_gravity="center"
        android:text="extend_at_9:30_AM_configuration_Sun,_Mon"
        android:id="@+id/config"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
  
```