Politehnica University Timişoara
Faculty of Automation and Computers
Computers and Information Technology

Developing a smart home solution using Iotivity framework

Bachelor Thesis

Candidate:
Ioan GUŢ

Supervisors:
Assist. Prof. dr. eng. **Alexandru TOPÎRCEANU**

Timişoara
2019

Table of contents

List of abbreviations

HTTP - Hypertext Transfer Protocol
UI - User Interface
JSON - JavaScript Object Notation
IOT - Internet of Things
RFID - Radio-frequency identification
DC - Direct current
AC - Alternating Current
Wifi - Wireless Fidelity
BLE - Bluetooth low energy
Mb - Megabits
IEEE - Institute of Electrical and Electronics Engineers
LiPo - Lithium polymer
UEXT - Universal Extension
USB - Universal Serial Bus
CRI - Color rendering index
OCF - Open connectivity Foundation
IETF - Internet Engineering Task Force
CoRE - Constrained RESTful environments
CoAP - Constrained Application Protocol
CRUDN - Create Read Update Delete Notify
CBOR - Concise Binary Object Representation
JSON - JavaScript Object Notation
IANA - Internet Assigned Numbers Authority
REST - Representational State Transfer
API - Application Programming Interface
URL - Uniform Resource Locator
MVC - Model View Controller
PWM - Pulse Width Modulation
RTOS - Real-Time Operating System

1 Introduction

Nowadays the world is an environment dependent on technology and because of that the technology is reaching new limits every day. Almost all the time when you are thinking to do something there is a big chance you will have to use a technology. Even though we do not realize that we are using this resource as a daily element in our life, we are using it and we will use it more and more in the future.

Technology is involved as a main element in the Internet of Things domain as well. All the products, cars, industrial components, sensors used to offer powerful capabilities are the consequence of using different technologies in order to develop them. The focus of this thesis is the Internet of things and more exactly how we can combine different technologies in order to obtain a reliable IoT infrastructure. All the devices connected to the Internet can be traced and monitored in order to collect, analyze and store data. These devices can also communicate with each other over the Internet in such a way that they can even influence the behavior of the other devices and they can also be monitored and controlled remotely. The concept of IOT technology is almost all the time seen or understood as a "smart home" system. The thing does that this project will also focus on a smart system that will be attached to a house and will cover some functionalities that are part of each home.

IoT is in general about an ecosystem that consists of web-enabled smart devices that are using embedded processors. These devices can contain sensors and they have also the hardware capability to collect, communicate the data collected from their environments and even to act over what was acquired. IoT devices share their data by enabling a connection to an IoT gateway where the data is sent to be analyzed locally or can be also sent in the cloud to be analyzed.

As the name of this thesis suggests, the focus of it is to develop an IoT ecosystem together with an android application that supports features like smart switch with adjustment of color temperature and a smart thermostat that receives data from smartphone sensors. The system will combine technologies such as Iotivity, Open Connectivity Foundation, power switch energy.

Motivation

The motivation for building such a system together with the android application, is to access and control all the devices that have an embedded module supporting power over ethernet attached to

them, through which they can be connected to the internet, they can receive or send data about the state of the devices and how they should change their state, and also to be supplied with electric power. From an engineering point of view in my opinion this will be an improvement of a home environment because there will be used instead of electric wires and a lot of switches just a component which should act as it is programmed and it is handling both the internet connection and the electric power.

By combining technologies this system will provide a compact and secured environment that can be accessed and handled by known users that have permissions to that channel.
There are a lot of such systems implemented already and they are doing the connectivity part over Wi-Fi or other communication channels and even with power over ethernet but with last pointed one, not so many.

This thesis contains a proof of concept that presents how such a system is designed and implemented in order to achieve a “smart home”.

2 State of the art

2.1. Overview

IoT is not anymore a small topic in the world. Its journey is so eventful and so successful since its beginning, that nowadays IoT is one of the main factors in business growth. Since the MIT professor, Kevin Ashton, reinvented RFID, which is a method for automatic identification which has as foundation the capability to store and find data without any contact through radio waves, this became the approach of connecting and monitoring remotely several entities under the concept of IoT.

2.2. Key aspects

IoT is infiltrated in almost all the domains starting from a simple chat communication to a building that is fully functional on an IoT infrastructure. This is happening as a consequence of its evolution over the last years and besides that, because IoT has achieved improvements such as energy efficiency, security, interoperability, maintainability and all of these helped in obtaining trust and perseverance in the world of technologies.

This section contains details about what IoT is offering and information about how they achieved all of these key aspects.

2.2.1. Energy Efficiency

Power over Ethernet or PoE facilitates network and power distribution wherever a power outlet cannot be installed. Taking into account that solar panels are widespread in the world and they are becoming more accessible from a financial point of view, IoT can benefit from them because in this way it can be reduced the energy consumption over a system. Solar panels have the ability to convert sunlight into electricity, producing DC power and after that, stores the energy into batteries that has to use inverters to convert DC power into AC power. Regularly bulbs light for

example are using DC power, but the power is not linked directly to them, a conversion to AC being performed in order to transfer the power to the bulb where is lost 15 - 20% from energy. After the energy is reaching the bulb there is perform one more conversion to DC power which is lost again 15 - 20% from the energy.

To solve this problem of wasting power energy automatization system is the key. Besides the fact that increases the comfort it is reducing the consumption of energy (thermal, electric, etc.). Implementing this system over a PoE equipment is the solution that offers the possibility of using solar panels with no more worries about losing energy. The Power Sourcing Equipment can be sourced directly with DC power that comes from the batteries charged by solar panels without losses and provide power on the ethernet cable. This device can be a switch for network and it also can be a device that takes a non-PoE-cable switch and a PoE device and make the interconnection between them.
In this way most of the devices distributed in the home can be sourced with DC power from the PSE switch saving up to 30% energy that in the other cases will be lost doing the conversions.

On the other hand, the automatization system can be implemented in such a way that, if nobody is at home, it is no longer used the heating system. Collecting data provided by different sensors placed in the entire home and knowing where the people are, it is used or not the heating system or the lighting one just when it is needed. When sufficient data is collected it can be created an occupancy pattern using artificial intelligence tools to optimize the heating and lightening usage automatically.

2.2.2. Maintainability

Using PoE is much simpler and easier to install power and network in buildings or spots where it is hard and expensive to install power lines. In a PoE system the worst-case scenario can appear if there is a problem between the switch and the device and in this case, it is very easy to change that ethernet cable just by unplugging the old one and plugging the new one. In case the switch has no power than there are two possible situations, the power supplier is down or the solar panels are not producing DC power at that moment.

Most of the computer engineers try to implement their own infrastructure with Arduino connecting all the light switches to this microcontroller and control a relay which turns on or off the light or other devices connected to it. It is a good idea but there will always be a huge amount of wires that comes to Arduino and if a problem pops up a lot of work has to be done in order to find out from where it comes from. This is where PoE has no problems being easy to track the issues.

2.2.3. Interoperability

Due to the existence of the PSE switch in a PoE system makes the process of connecting the devices from this network with the other ones that are non-PoE devices easier. The PSE can be used as a bridge between a non-PoE switch and PoE device, in this case the device being called external PoE injector.

2.2.4. Security

Wi-Fi or radio "smart home" systems are easier to implement compared to PoE systems and they are very efficient because there is no need of wiring or embedded devices added if the devices supports already these types of connection, but from a security point of view they are less secure. The Wi-Fi connection can be accessed sometimes even from outside of the building or home area and it can be decrypted by somebody and can be accessed, which is not so easy in PoE systems.

In order to access a PoE connection the device has to be connected to the switch. Also implementing the system based on existing frameworks another layer of security can be introduced. This is done by the library that will be used in the implementation of system which is called Iotivity.

2.2.5. Ethernet over Wi-Fi

Wi-Fi connection it is fragile when it is crowded. If too many devices are connected to the router, collision can occur, the performance and the data transmission is slowing down. Transmission over the cable is always fast and no interference are taking place. For Wi-Fi, when the connection drops the router needs to be restarted to establish the connection such that it is possible again to act over the devices. In PoE system this kind of problems are not present. The only problem that can occur and is applying for Wi-Fi too, is when the internet connection is cut off.

2.2.6. Environmentally clean

Solar panels are the ones that can give power energy without pollution. This is why for our environment, nowadays it will be a good approach to use them in order to avoid destruction of this world and make it uninhabited. There are a lot of pollution sources already and using power energy from sun energy for our needs is a start in getting this world better.

2.3. Similar systems

There are already in the market some solutions implemented for a “smart home”. Alexa, Nest, Google Assistant, Homekit offered by Apple, Home Assistant all of these are solutions for what everyone wants to have in their homes because they are helpful and interesting.

Alexa Amazon offers a “Smart Home Made Easy” package that includes the following functionalities [2]:

- Outlets - turn on your coffee pot from bed
- Thermostats - control your heating system using your voice
- Lighting - set the light depending on your mood using colored smart bulbs
- Locks - in case you forget to lock your doors
- TV streaming - using your voice change to your desired channel)
- Cameras - monitor the home using Echo Show
- Doorbells - know immediately who’s in the front of your using the phone.

Figure 2.1 - Amazon Alexa User Interface [3]

Nest from Google is offering packages like[4]:

- Google Home(streams music and works with the thermostat)
- Nest x Yale Lock(a system to lock your Nest home),
- Smart lights that can shine bright in any color

- Connection with your thermostat and oven in order to control them when nobody is home.

Figure 2.2 - Amazon Alexa User Interface [4]

The Edge of Amsterdam building is one of the most intelligent buildings in the world which is using 5 900 square meters of solar panels that are distributed all around the building to provide energy for inside usage. The electricity and network connectivity is covered by 750 PoE switches, eliminating the need of power cabling. It can be considered as the state of the art for systems that are using PoE.[5]

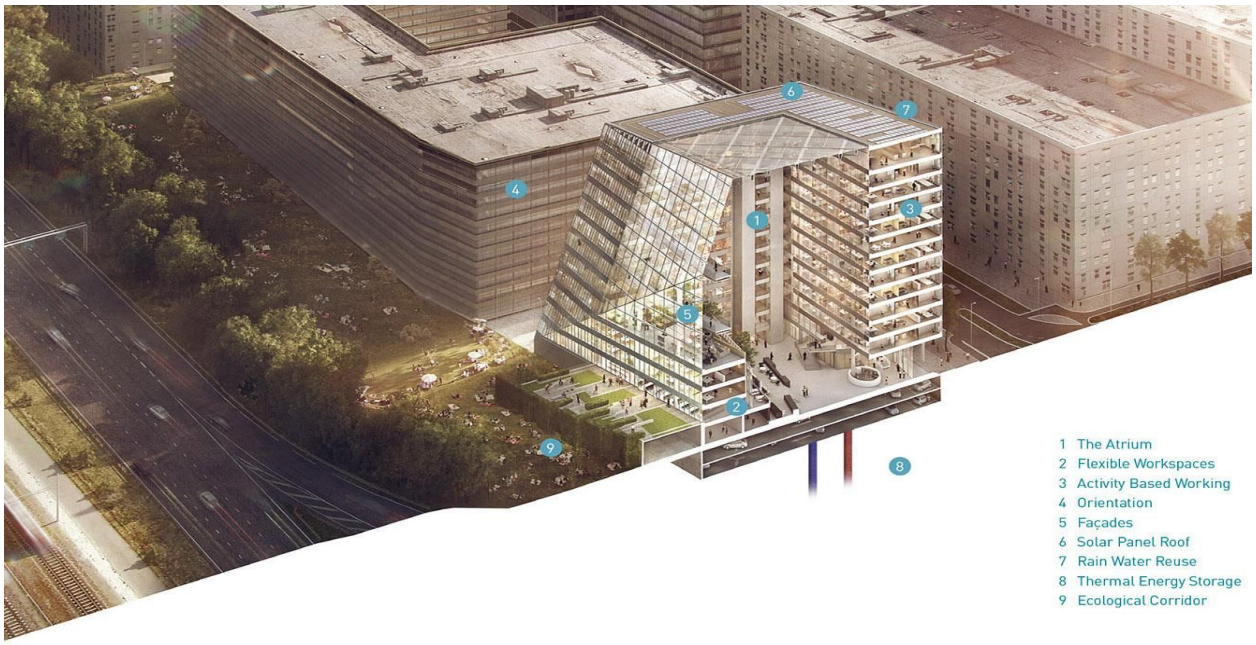

Figure 2.3 - Amazon Alexa User Interface [6]

3 Project specifications

This chapter contains the full list of specifications of the entire system, together with the requirements and functionality description.

3.1. Hardware components

The system developed has to include at least the following modules and pieces:

- A smartphone device that has preferably ambient sensors embedded, such as temperature sensor, pressure sensor and humidity sensor. For now, there are not so many smartphones with these specifications and therefor in this system the smartphone used is a Samsung Galaxy S4. This device has a total number of nine sensors seamlessly combined to deliver a better user experience. [7]
    - Gesture Sensor - recognizes the user's hand movements by using infrared rays
    - Proximity Sensor - recognizes whether or not the mobile phone is near the user using infrared rays
    - Gyro Sensor - detects the smartphone rotation state based on three axes
    - Accelerometer - detects the smartphone movement state based on three axes
    - Geomagnetic Sensor - detects magnetic field intensity based on three axes
    - Temperature/Humidity Sensor - verifies humidity and temperature levels
    - Barometer - identifies the atmospheric pressure
    - Hall Sensor - recognizes whether the cover is open or closed
    - RGB Light Sensor - Measure the intensity of the light source based on the red, green, blue and white colors.

- ESP32-PoE is an IoT development board that supports Power-over-Ethernet feature. As it is described in [8] the board is including pre-standard PoE support, supplied with power from the Ethernet and it can be extended with different sensors and actuators. The board has the following features:
    a. ESP32-WROOM-32 Wi-Fi/BLE module
    b. Ethernet 100Mb interface with IEEE 802.3 PoE support

c. LiPo battery charger
d. LiPo battery connector
e. UEXT connector
f. User button
g. Reset button
h. Micro USB with programmer for ESP32 programming
i. MicroSD card
j. Two extension connectors 0.1” step spaced at 1”

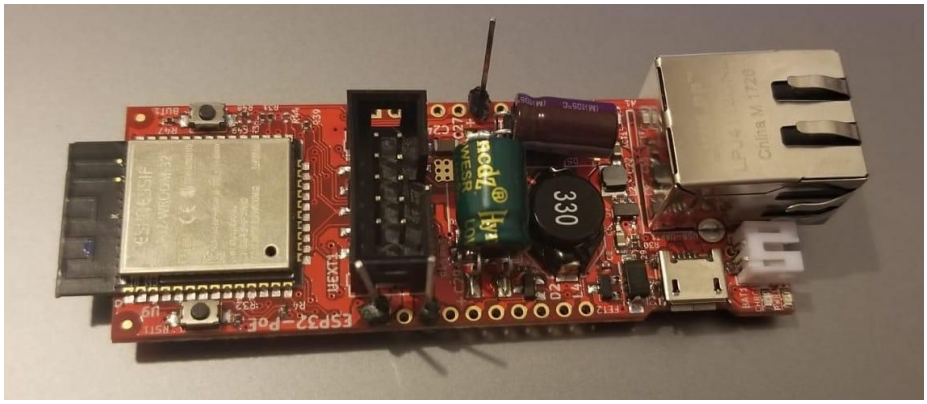

Figure 3.2 - ESP32-POE

- LED Strips 5m High CRI > 90 with double color in order to set the temperature of room color

Figure 3.3 - LED Strip Light White 24V

- LM2596S DC-DC LM2577S Step Up Down Boost Buck Voltage Power Converter (Voltage regulator) with the following specifications:
    a. Input voltage: 4 - 35 V
    b. Output voltage: Continuously adjustable (1.25-25V unload adjust)

c. Output current: 3A Max
d. Output power: 15W natural cooling
e. Indicator: charging is red, charging completed the indicator is blue.

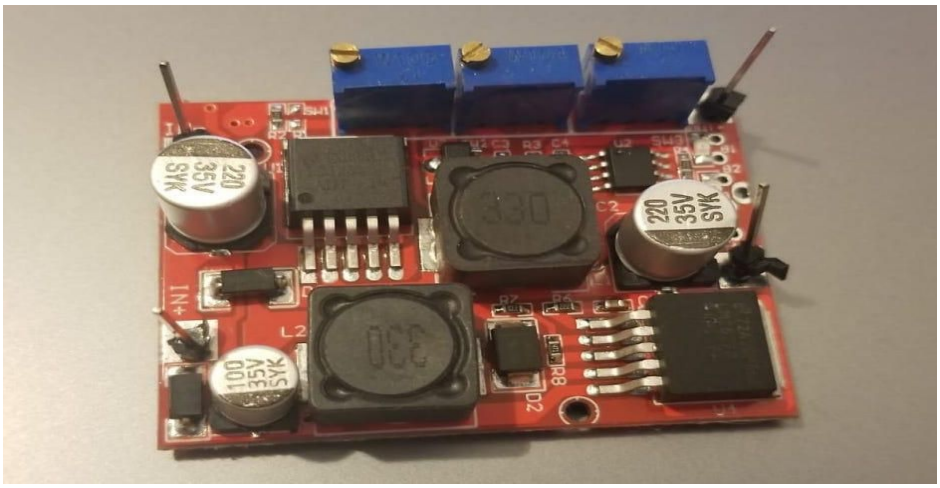

Figure 3.4 - Voltage Regulator DC-DC

The board ESP32-PoE is taking as input from the network 48V and is providing an output of 5V. The LED Strips from above are working with a voltage of 24V, such that there it is needed a convertor step-up to convert from ESP device output of 5V to 24V .

- FR120N LR7843 AOD4184 D4184 Isolated MOSFET MOS Tube FET Module Replacement Relay 100V 9.4A 30V 161A 40V 50A Board with the following features:
    a. Optocoupler isolation - transfer electrical signals using light
    b. Compatibility: MCU and Arduino board
    c. Used to control motor start and stop
    d. High level start, low level stop and PWM speed regulation
    e. Input and output signal can be welded to the pins themselves.

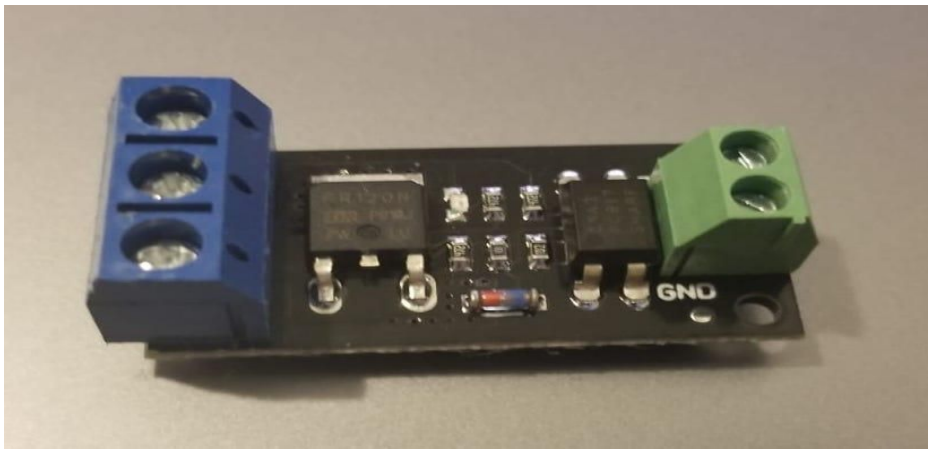

Figure 3.5 - Isolated MOSFET MOS Tube FET Module Replacement Relay

It is a power transistor which receives signal from a pin of ESP32-PoE board and is supplying the LEDs with the necessary voltage when the signals are sent.

- G3MB-202P 5V DC Solid Relay Module:
    a. Input power: 5V DC
    b. Trigger voltage for high relay ON: 3.3 - 5V
    c. Trigger voltage for low relay OFF: 0 - 2.5V

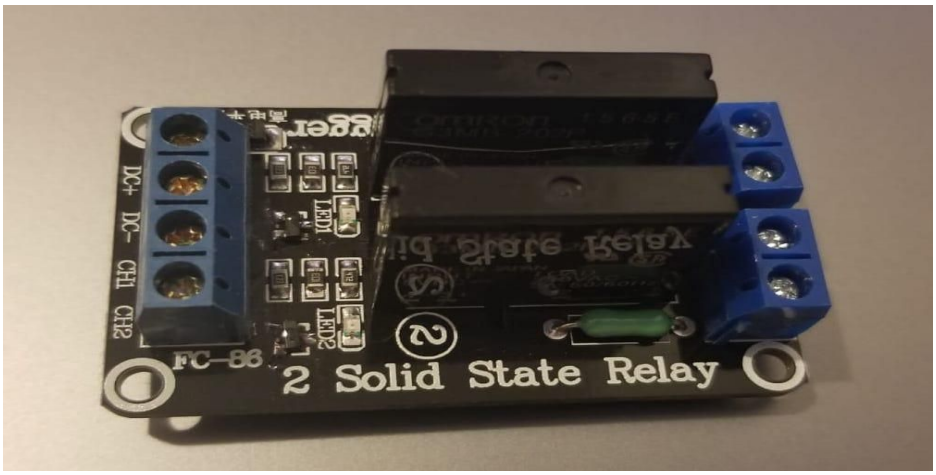

Figure 3.6 - 5V DC Solid Relay Module

This is a module with 2 solid state relays that will be used to control the heating system inside the home.

- RTL8152 Micro USB Ethernet Adapter Micro USB to RJ45 LAN Card

Characteristics:

a. Switch 10/100 Mbps network automatically
b. Compliant to USB interface version 1.0/1.1/2.0
c. Compatible with the following systems: Windows, Linux, Mac, Android

Figure 3.7 Micro USB Ethernet Adapter

It will be used to charge and connect the smartphone to the PoE switch

- TP-Link TL-SF1008P 8-Port 10/100Mbps Desktop Switch with 4-Port PoE
  a) Provides seamless network connection
  b) Integrates 100Mbps Fast Ethernet and 10Mbps Ethernet network capabilities.
  c) PoE ports can be automatically detected and supplied with power with those IEEE 802.3af compliant Powered Devices (PDs).
  d) Electrical power together with data are transmitted over one single cable

Figure 3.8 TP-LINK POE Switch

With such a system that is using PoE, the electrical power along with the data provided over the internet connection it can be transmitted in places where no outlets are present by extending the network infrastructure.

3.2. System specifications

The android application should at least fulfill the following requirements:

- The android application should run continuously on a smartphone device plugged in the PSE switch such that whenever the battery goes down it will be supplied with power.
- The application will run in kiosk mode which means that only the application developed for this system can be accessed.
- The power consumption of the smartphone has to be reduced
- On the display will be always provided information about: temperature, pressure, humidity and time.
- The user should have the ability to interact with the application
- The heating system should be accessible through the application such that the user can set the selected temperature inside the building.
- The lighting should be accessible through the application such that the end user can turn on/off the light and more than that, he can set the temperature of the light.
- The application should give the possibility of dimming the light shining.
- The application should find all the devices that are connected to the PSE switch and should be able to control them.
- The application should provide a list with all the devices and should offer the capability to choose one of the devices.
- After the device is selected the application should provide information about the device and possibility to act over the state of the device.
- The application should provide the ability to control the system vocally.

3.3. System functionality

This section contains details about how the system is functioning in order to provide the specified requirements together with information about the communication layer between the devices(servers) and the client(user).

The android application is implemented in Android Studio IDE using the provided standard libraries to retrieve the needed information from the smartphone, together with external libraries such as Iotivity. The standard libraries are used to design the UI, to get the data from nearby environment such as temperature, humidity and pressure, and to develop the application in such a way to support multiple API versions.

The system has to be supplied with power all the time in order to have the entire functionality. The Samsung S4 Galaxy has installed the application implemented for this system and it is working in kiosk mode.

To run an application in kiosk mode some features of the Android device are disabled. There will be a single application running and the user is not able to exit the application or access other settings of the device if it is rebooted. The interaction with the application it is not interrupted, not even with a phone call. They are turned off. Kiosk mode is keeping the application awake all the time in full screen mode as long as the device is powered.

The smartphone is connected with the PSE Switch that supports sending through ethernet cable both power energy and internet connection. In this way the device is accessing the network system with no Wi-Fi connection and it is more secure.

When the user decided that the temperature displayed by the application it is not how he would like to be, he can solve the problem just by sliding the seek bar up or down, depending if the temperature inside the home has to be increased or not.

On the other hand, if the client decides that he wants to turn off the light in a specific room, he can access the server of that room just by selecting the device founded by the application that is there and act over it. This is the situation with all the devices that are connected to the network through the cables to the switch.

What can be found interesting is that if the user wants a warm color in a room, he can assign to a seek bar the light system from a room and use it to dime the shine light in that room, and even to change a bit the temperature of the light.

Kids almost all the time are more sensitive and because of that sometimes, the parents have to support even in their room a higher temperature than they would like to. Even this problem can be solved with this system. Every room can have its actuator in order to save

energy. This means that, if in the kids' room it is needed a higher temperature, select that actuator from his room and set whatever temperature it fits. In the other situation when not the entire home it is used, it will be very efficient to warm up just the rooms which are occupied in order to save energy. It can be done just by turning on the heating system in that room acting over the server that controls that room heating actuator.

3.4. Market and business analysis

The market and business analysis it done by every company or person that wants to present and introduce their product or service on the market. It is done in order to identify the customer demand for that product type. The analysis helps them to determine rather or not they will have success on the market or if they can successfully enter a market and bring profits too.
As mentioned in the second chapter in 2.3. Similar applications, there are already a lot of such applications develop by the giant companies as Apple, Google, Amazon. They invested money in these applications and devices because the future will come with more and more demand, the industry analysis predicting a grow from year to year, reaching in 2023 a total value of $137.91B.

However, the things weren't as expected for these companies because there wasn't a market explosion, it just grown steadily. The customers were not very excited about this future tech and because of that just for some of them it seems that these new "smart home" are an improvement of a house.

3.5. Design

This section contains information about how the system is designed in order to provide the presented requirements. It describes the communication level between the devices involved in the system, the architecture of the system together with high level overview of the project.

3.5.1. Open Connectivity Foundation

This foundation has born with a precise goal, that being IoT.[11] Nowadays a lot of devices can communicate with each other, can interact and share information, but there is a problem that even if it seems that it does not exist, it still cause difficulties in IoT world, and that being the absence of a universal language for Internet of Things.
The devices are made on different disparate frameworks such as Amazon, Alexa or Google and this is limiting their market, but there is also the possibility to choose to develop these devices

across multiple ecosystems, but in this case the costs will be increased. The problem is that all the work that has to be done is on the end user which has to determine if the product they want it is compatible with the ecosystem or they have to find a way to integrate it with their devices, meaning that they have to solve the interoperability issues.
These are the topics that OCF is trying to address in order to make all of this easier.
Their mission as they are describing in the following words is [12]:

1. "Provide specifications, code and a certification program to enable manufacturers to bring OCF Certified products to the market that can interoperate with current IoT devices and legacy systems.
2. Make the end user's experience better by seamlessly bridging to other ecosystems within a user's smart home and ensure interoperability with OCF compliant devices."

This foundation provides developers and manufacturers a framework for different platforms, operating systems, modes of communication and use cases, OCF Bridging Specification such that the devices are discoverable and can be connected with other ecosystems, OCF Security Framework and the opportunity for innovation.

The difference between OCF certified products and the other products that are already on the market is the following: the users can get a product that is not dependent on a particular brand and that can work all together.

OCF wants to "connect the next 25 billion devices for the IoT" [13] and also to provide a secure and reliable structure such that the devices are discoverable and they can be connected across multiple operating systems/platforms.

The Core Framework specifications for OCF listed in [13]:

- Discovery - uses IETF CoRE for device discovery
- Messaging - supports as default IETF CoAP but it also supports protocol translation via bridges
- Common Resource Model - resources are real world entities defined as fata models
- CRUDN - it is a RESTful API that manipulates the data(resources)
- ID & Addressing - represents the OCF IDs that help to address to the entities (servers, clients, devices, resources)
- Security is fundamental for the entire ecosystem and it is applied to all the elements.

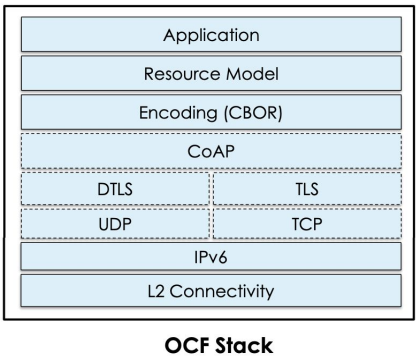

Figure 3.9 - Protocol Stack [13]

Because everything in OCF is a Resource, servers may contain one or more Resources that are describing the real existing entity. Each Resource has its own properties that are defined in order to describe an aspect that is exposed via a Resource together with meta-information about that Resource. An interface of the Resource is provided, such that a first view into the resources is done and then it can be defined the requests and responses that are permissible on that Resource view.

The encoding that is adopted at this moment by OCF is called CBOR and it comes from Concise binary object representation which is based on JSON data model (it is a lightweight format used in order to store data or to transport it).

The Resource discovery it is realized using a customized CoAP Discovery that uses IANA. The CoAP get method it is set to multicast retrieve resources from URI "*/oic/res"*, and the response it is an array with all the links found over that internet connection.

The concept of Endpoint in OCF is seen as a source or destination of a request/response message that can have different Transport Protocol suites and based on these protocols the endpoint can alter. An OCF device can have more than one endpoint that is connected to it, this means that the device may have multiple functions.

The communication between the server, which is the android application, and the clients, which are the ESP32-POE modules, it is realized under the known architecture, RESTful. The client sends a request to the server and the server will change its state and responds back with information about its state. Servers may also notify the client if any changes occur over time and the client may or may not act over the server depending on the settings or on the user wishes. As mentioned, the messages sent from client to server and vice versa, are JSON format.
OTGC Android control application it is an open source code and is offered by OCF community for their members such that the system is spread and in continuous evolution. With all of these

there is the possibility to build their own application that discovers and controls OCF devices.[11]

3.5.2. Iotivity

IoTivity, as the web page is describing, is an open source software project enabling seamless device-to-device connectivity where billions of wired and wireless Internet of Things (IoT) devices can securely connect to each other and to the internet [15]. All the specification presented about Open Connectivity Foundation regarding interoperability guidelines between devices, specifications and also certification program are implemented by Iotivity.
APIs exposed by Iotivity framework are available in several programming languages and they are supported by multiple operating systems. It is working like a middleware between all operating systems and connectivity platforms. The main methods that are essential for building such an ecosystem and which are implemented by Iotivity are: discovery, data transmission, data management and device management.

- Discovery - in order to discover the devices in proximity or remotely they are implemented multiple mechanisms.
- Data transmission - supports exchange of information and control based on messaging and streaming model
- Data management - all the data provided by the resources can be collected, stored and analyzed in order to make the system responsible
- Device management - resources that are seen as devices can be configured and provisioned with user input data.

I chose to use this open source project because it is still in development and I can even contribute to the development of this framework at some point. They are providing documentation about initialization and setup and also documentation in different languages about the APIs exposed.

3.5.3. REST

“Representational State Transfer is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other”. Nowadays almost all the applications are developed on the Client-Server architecture. The UI or the frontend part which is exposed and represents the client, communicates with the server or the backend to get or save the data provided as a consequence of user actions. CoAP protocol is a transfer protocol that handle constrained nodes and networks used in IoT such that the communication between Server - Client is done by exposing on the server side some services that are accessible via this protocol.

To understand how REST is working we will take an example from the android application where we are trying to get the devices that are available in the network system. The client application can manage the servers which are basically the devices and can discover them. To discover these devices the client has to search for an endpoint that is exposed as a service on the server side.
The client can send CoAP request to this endpoint to communicate with the server, but it can also extend this communication to other protocols such as http, https or coap, coaps. It depends in most cases on the application requirements which means that if we want to exchange data on a secure channel we would use "https" or "coaps". The URL is composed by the protocol part together with the resource (endpoint locator). Each device will have a is represented by at least one endpoint.

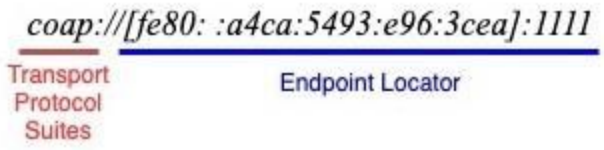

Figure 3.10 - REST endpoint structure with CoAP protocol

The REST standard methods are:

- GET - fetch the data from an endpoint
- POST - create new data
- PUT - update already existing data
- DELETE - delete existing data

The GET method it is used every time when we want to fetch some data or resources from an external service (that expose an endpoint). For example, in order to retrieve all the devices currently connected over the network, a get request is executed with a URL that is exposed by each server: GET "/oic/res" . This endpoint is returning an CBOR data model with information about the device founded that contains even the endpoint needed in order to communicate with that specific device. The endpoint looks like the one provided in Figure 3.10.

3.6. Android Application Architecture

In 2017 the Android Framework team had introduced a set of new Architecture Components that deals with the known problem of screen rotation. "Android Architecture components are a collection of libraries that help you design robust, testable, and maintainable apps. Start with

classes for managing your UI component lifecycle and handling data persistence" [9]. Improvements regarding the functionality of an application were designed such that, with ease, it can be managed the lifecycle of your app, survive to different configuration changes, no memory leaks present and easily it can be loaded data into UI.

The components that were introduced are :

- LiveData - this component is used to store data objects that notify, usually the UI, when LiveData objects are changing.
- ViewModel - as the documentation of Android components is describing the Android framework manages the UI controllers lifecycle (activities and fragments). In this case, if the user controls the device, the framework is deciding if it has to re-create or destroy the UI controller. In such cases any transient UI-related data stored in is lost. ViewModel is a helper class that is responsible for any data that has to be exposed to UI. These objects "are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance. For example, if you need to display a list of users in your app, make sure to assign responsibility to acquire and keep the list of users to a ViewModel, instead of an activity or fragment" [10].
- Room - it is used to save data in a local database. It provides an abstraction layer over SQLite and it is used when you want to save a significant amount of structured data locally such that if the internet connection is lost, the changes made by the user are saved and then when the connection is reopened the server is synced and the changes are made.

The application is designed over a known and used architecture, MVC architecture. MVC stands for Model, View, Controller and it separates an application in those three layers called in the same way.

Model - this component represents the shape of the data together with business logic. The object model state is stored in the database and it can be retrieved such that the application is acting based on the information stored in the model.

View - it represents the application interface. Here it is displayed the data to the user and also the user has the possibility to modify the data.

Controller - this component is managing all the user requests. Basically, every time the user is interacting with the application view an appropriate URL request is raised and is handled by this component. After the request is sent, an appropriate view based on the model it is rendered and the user can see the response of his action.

The main view of the application is designed such that the user can interact with some of the components and it can also see live data about the time, temperature and humidity, data that is retrieved from that environment with Samsung s4 Galaxy sensors. On the right side of the screen there is a fragment view that can be loaded with data about devices that are connected to that internet connection that the smartphone is. All this information is an example of how the

application is using this architecture. The Model is presented as a device, view is the graphical interface that the user is able to observe and the controller part is covered by acting over devices/model.

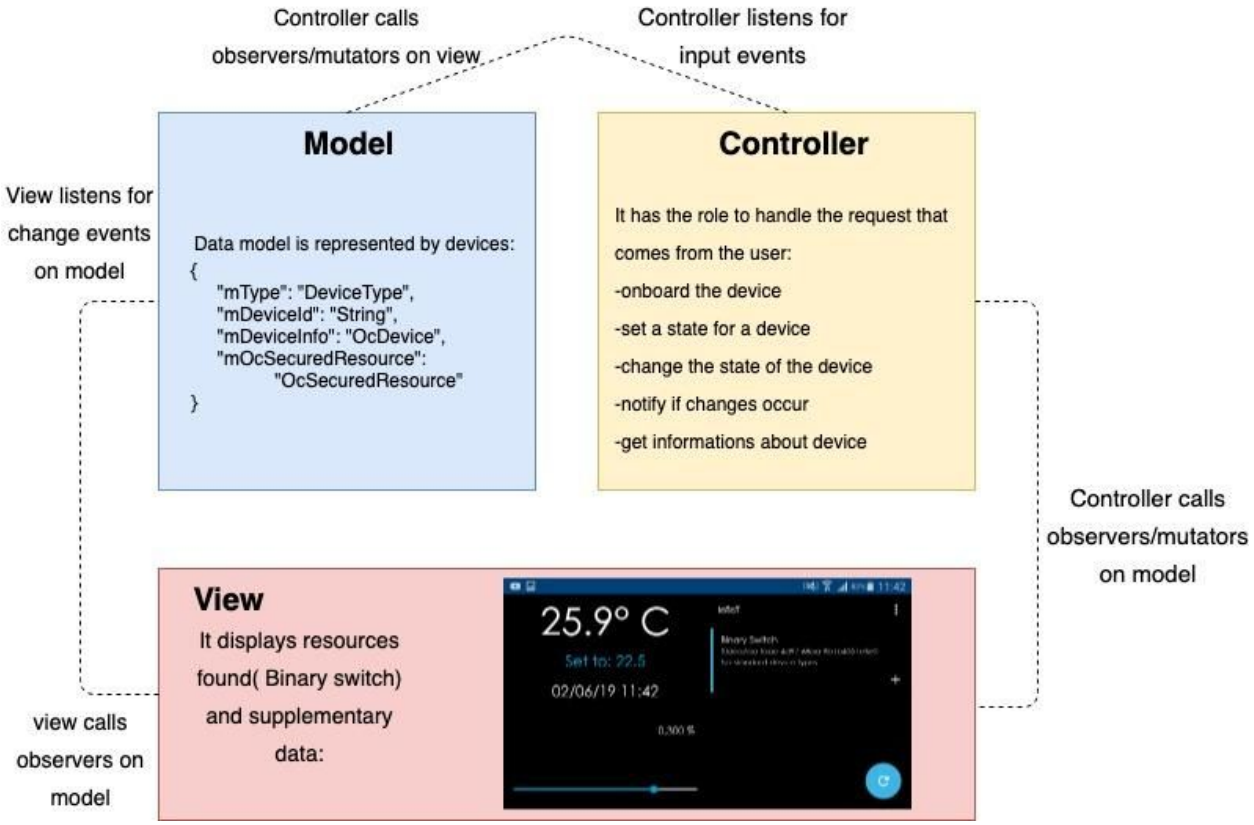

Figure 3.11 - Model, View, Controller Architecture

4 Implementation

This section contains information and technical details about implementation and development of the entire system together with components binding and connection between each other. It also contains details about building the component Iotivity that is used in the Android Application and on the server side, details about how to develop all the characteristics that are needed such that your device supports diverse actions.

4.1. Hardware overview

In this section it is presented the hardware components overview and all the connections between these modules.

The main hardware component of the entire system is the **ESP32-PoE** module. To this device it is connected sensor or actuator of the system that has to handle the request asked by the client. The devices are linked to this module using GPIO ports and also the port that provides continuous voltage, in this case the 5V. The ESP32-PoE responsibility is to understand the user request and to act based on what the user wants, such that if he asked to turn on a light it has to provide the bulb or the led with the needed power. The power supplying of this module is realized through the network cable.

The **PoE Switch** is used to provide for devices power and also internet connection. It is a network switch that simply connect network devices to it as normally a switch does, and it has the ability to detect whether or not these devices are PoE compatible such that it is enabling or not the power. This switch it is connected to the internet and all the devices that are linked to it and are using ESP32-PoE as control device are accessible and powered based on the request send.

The **voltage power convertor DC-DC** it is needed because the LED Strip needs to be supplied with 24V. This component takes an input voltage of 5V from the ESP32-PoE device and it gives at the output an adjustable power voltage from 1.25 to 25V. Since the LED Strips supports dimming it can be loaded with variable power such that the light can have different temperature. In order to realize that, we are using an PWM(GPIO) pin from the ESP32-PoE module to deliver an average power to a transistor hardware component.

The **power transistor** it is used in order to provide an average voltage output for the LEDs. It supports PWM speed regulation from the ESP32-PoE and together with the power received from the voltage power converter of 24V it gives a load power based on the PWM input frequency.

The **LED Strip** component as already specified it has two types of bulb light. One is for warm color of light and the other one for cold color. In order to access both led colors we will use 2 power transistor that will be connected to different PWM pins of the ESP32-PoE module.

On the client side the hardware used it is a **Samsung S4 Galaxy** because it has embedded more sensors than other smartphones, such as pressure, humidity and temperature for external environment. These sensors are used, and the client application acts based on some information that is provided by them.

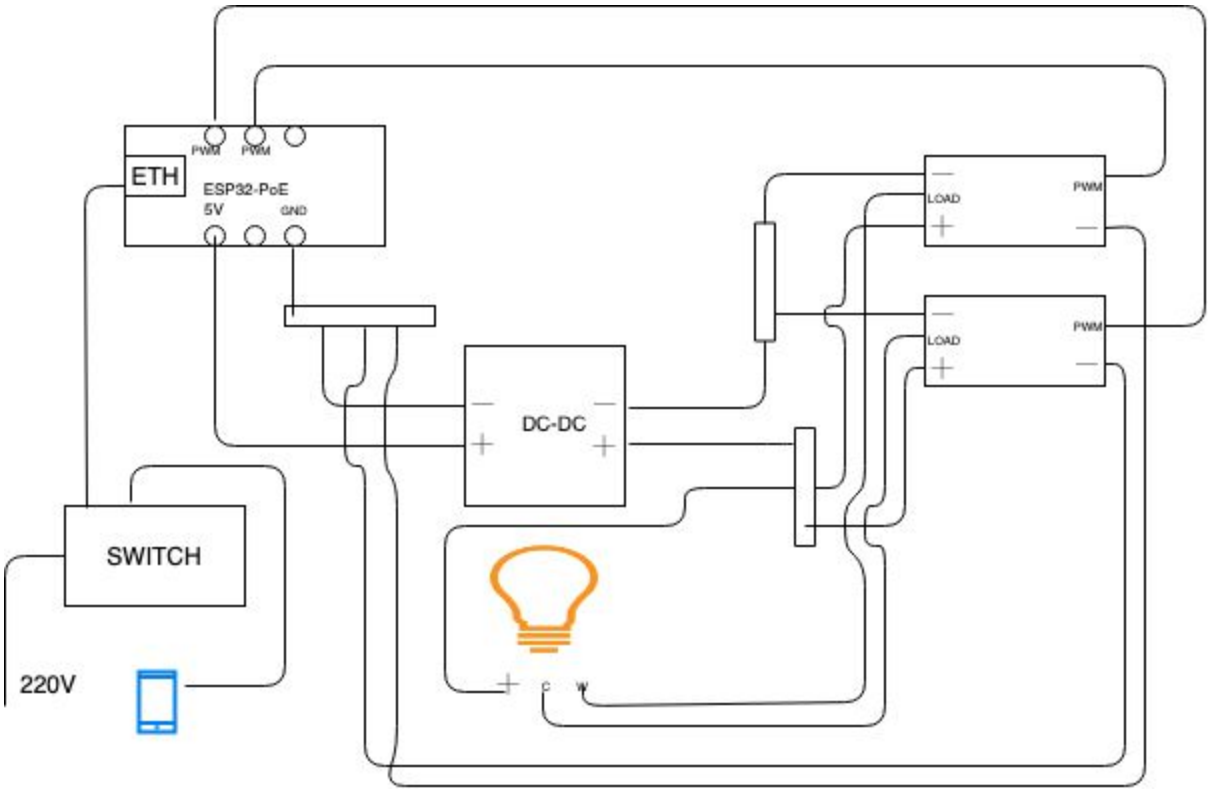

Figure 4.1 - Light System Hardware Architecture

4.2. Software implementation

In order to develop an embedded system, it is usually slower and it takes more time than developing a classic application that is running on cloud or on a computer. The testing of the

system it also takes as much as the development itself. To do things correctly, I started with a minimal android application implemented in such a way to discover all the devices that are connected to the network and respects a specific pattern. After each improvement the system had to be tested. In order to test the functionality of the application, a server device was simulated on an ubuntu system that provides the behavior of a light.

4.2.1. Prerequisites

This system is designed based on the open source framework implemented by OCF community in order to provide services that connect securely devices in the world of IoT.

In this section it is presented all the necessary components and software needed such that this system will work properly:

OTGC Android Application provided by OCF under the project Iotivity is it used as a base code in order to retrieve devices and connect to them such that we are able to act over them. In order to make this application work on S4 Galaxy the Iotivity project has to be built for this smartphone architecture.

For **Iotivity** I have cloned the repository provided by the Iotivity project "git clone -b 1.3-rel --depth 1 https://github.com/iotivity/iotivity.git" on a Linux operating system and I did the following steps:

- Make sure the OpenJDK version is 1.7 or later it is used by running the following command in the terminal "java -version" and if it is not run the following commands: "sudo apt-get update && sudo apt-get install openjdk-7-jdk"
- Into the root directory where the Iotivity is cloned, run the following command in order to build the Iotivity for your device: "scons TARGET_OS=android TARGET_ARCH=armeabi-v7a RELEASE=0"
- Scons utility is a tool for building software by determining based on file named sconstruct, Sconstruct or SConstruct which component pieces has to be rebuilt and is executing the necessary commands to rebuild them. In this case the "sconstruct" file was provided by Iotivity. The necessary changes were done for the Linux operating system in order to export some system variables that the script does not managed to do it.
- The output resulted after build has to be imported in the OTGC application as module dependencies such that it works properly for that specific architecture.

At this step there were encountered some problems related to OTGC application at run time when the device met in the network was found. I spent some time debugging where the problem is and finally, I realized that the version of this application it wasn't a stable one.

All the information provided above is related to the client. On the server side there are required the following prerequisites:

Toolchain setup to build the application for ESP32. This is available for download from the Espressif website and it has to be extracted in a “esp” directory. Once this step is done successfully the PATH variable has to be updated such that xtensa-esp32-elf is available from terminal.

The second step is to get **ESP-IDF**(Espressif IoT Development Framework) by running the following command in a new directory: “git clone --recursive https://github.com/espressif/ esp-idf.git". After the repository is cloned in *esp-idf* directory run “git submodule update --init”.

4.2.2. Android Application - Front End

The main functionality of the android application in this system is to provide a UI for the entire system functionality such that the user is able to navigate through the devices that are distributed over the house using PoE cables.

The application is build over architectural pattern ViewModel. The ViewModel class can store and manage data that are related to UI in a lifecycle conscious way. As mentioned above its data does survive to configuration changes such as screen rotations. These types of objects used to implement the application are retained whenever configuration changes occur such that they hold and share the data immediately to the next activity or fragment.

Before going further and present the implementation of how the devices are scanned and hot the request is sent from UI to the servers, the application is asking for some data provided by the smartphone itself. In order to have the fully details for UI it is recommended a Samsung Galaxy S4 because we are using some sensors that not all the smartphones have. Information about the ambient temperature and also humidity percentage is displayed in real time based on the sensor’s information provided. When changes in temperature are read by these sensors, they are updating the UI with the new temperature that was read, and it is the same in case of humidity sensor.

Code Listing 4.1. Retrieve information from humidity sensor

```
private void getHumiditySensorData() {
    humidity= mSensorManager.getDefaultSensor(Sensor.TYPE_RELATIVE_HUMIDITY);
    if (humidity == null) {
        mHumidity.setTextSize(20);
        mHumidity.setText(NOT_SUPPORTED_MESSAGE);
    } else {
        mHumidity.setText(String.format("%.3f %%", humidity.getPower()));
    }
}
```

Code Listing 4.2. Callback on sensor changes

```
@Override
    public void onSensorChanged(SensorEvent event) {
        String tempStr;
        if (event.sensor.getType() == Sensor.TYPE_AMBIENT_TEMPERATURE) {
            ambient_temperature = event.values[0];
            if (Math.abs(ambient_temperature) < 10)
                 tempStr = String.valueOf(ambient_temperature).substring(0, 3) +
getResources().getString(R.string.celsius);
            else
                 tempStr = String.valueOf(ambient_temperature).substring(0, 4) +
getResources().getString(R.string.celsius);

            tempToSet(mTemperature, "", tempStr);
        } else if (event.sensor.getType() == Sensor.TYPE_RELATIVE_HUMIDITY) {
            float ambient_humidity = event.values[0];
            mHumidity.setText(String.format("%.3f %%", ambient_humidity));
        }
    }
```

The output of this code sequence can be observed in the left side of the screen application. On the right side there is a refresh button that sends the request for retrieving the devices when it is pressed and the same request it is send even when on right side of the screen you swipe down with your finger.
The main screen that pops up after the smartphone is connected to the Wi-Fi it is wrapped into a LinearLayout that is split in two: one LinearLayout and one CoordinatorLayout. On the left side of the screen a LinearLayout again, is wrapping the components that display information provided by sensors in TextViews. In TextView is also printed the DateTime in real time retrieved from the smartphone. Two more components are used here: a SeekBar and another TextView. The SeekBar acts over the new TextView that is displaying the temperature wanted inside home and it also acts over temperature actuators. In case the difference is bigger than 0.5°C between the actual temperature and what the user would like to be the actuators will start the heating system.

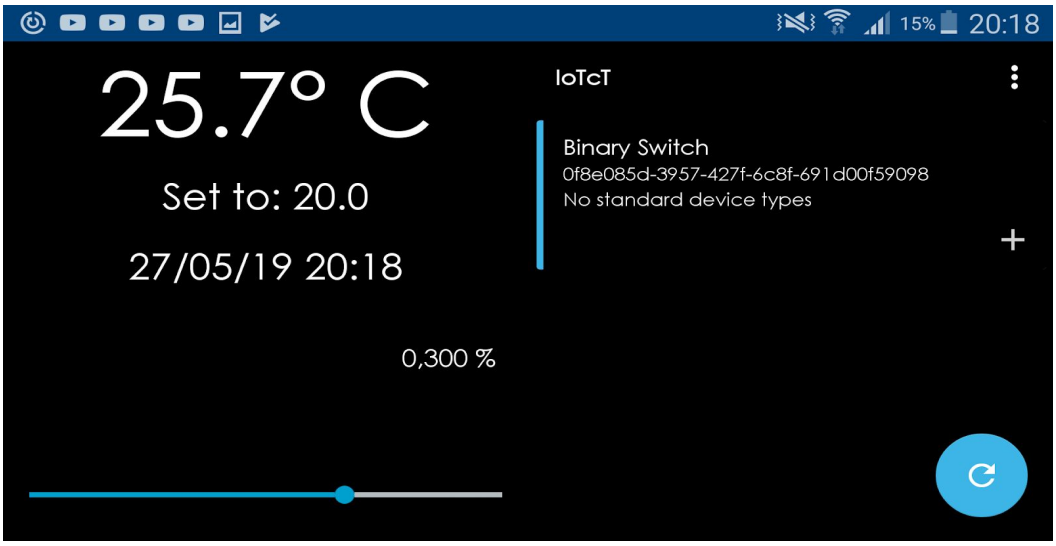

Figure 4.2 - Main View of Android Application

On the right side of the screen there is a list with all the devices found over the network. Once the refresh button is pressed the scanning of devices starts. As long as the backend is searching for resources, a progress bar pops up on the top of the screen. The resources found are wrapped each of them in a different device fragment.

Code Listing 4.3. Callback on sensor changes

```
<fragment
    android:id="@+id/devices_fragment"
            android:name="org.openconnectivity.otgc.devicelist
                                    .presentation.view.DoxsFragment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

At this point we managed to retrieve the resources found over the network and if we want to act over them, we have to do the transfer ownership. By pressing “+” button the transfer ownership starts.

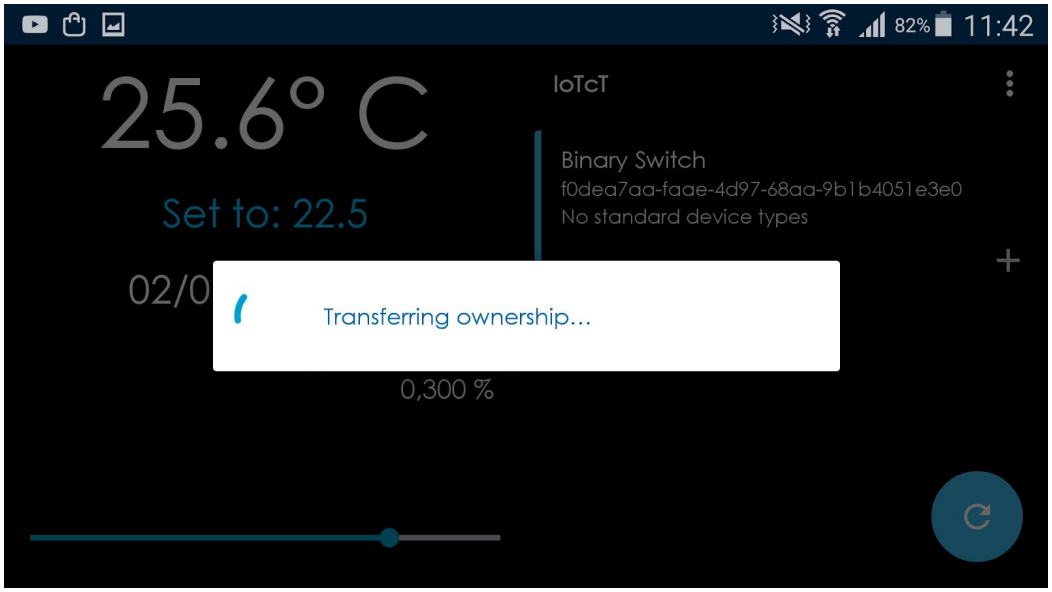

Figure 4.3 - Loading pop up for transfer ownership

After the transfer ownership is realized successfully a window shows up asking to name the device as you wish. Once the confirm button it is pressed the name of the device will be the one that was inserted, and the selected devices will have now new options.

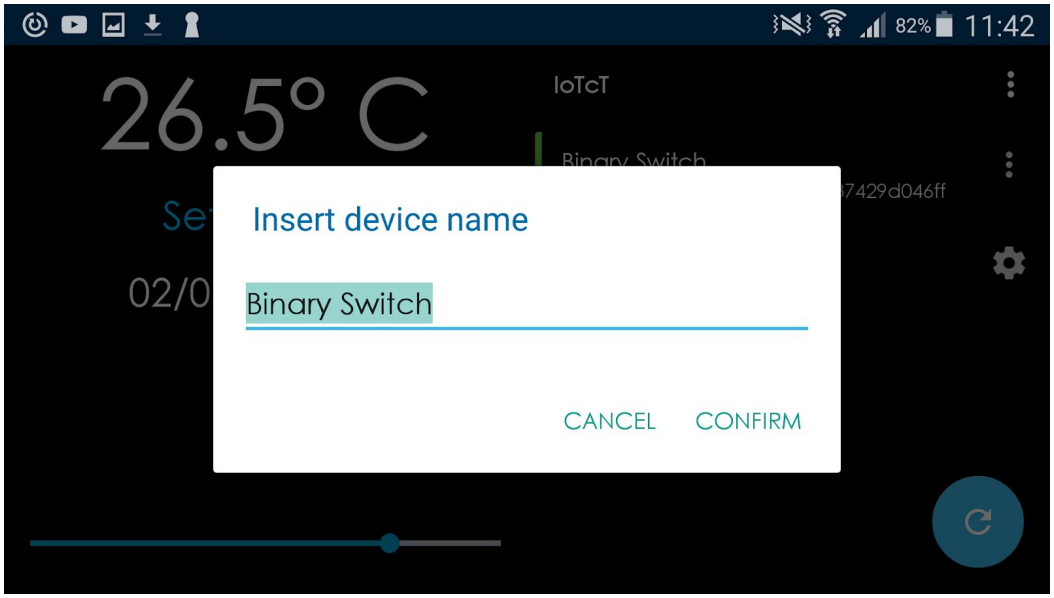

Figure 4.4 - Insert device name after transfer ownership tab

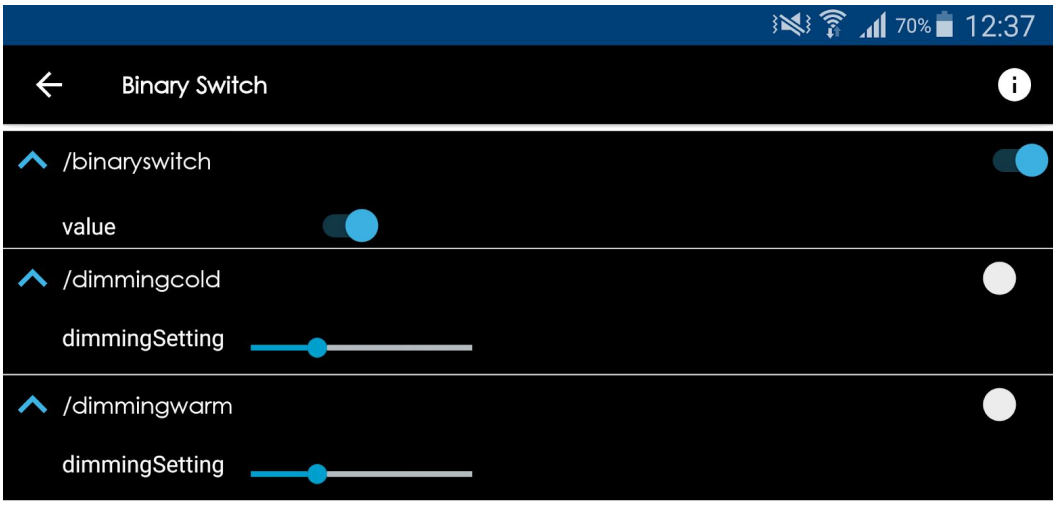

Figure 4.5 - Device supported operations which implies GET/POST request

This device from *Figure 4.5* that was opened has 3 possible function implemented. For any new function that the device supports there will be extended dynamically the list of function. There are available 2 types of function, both of them being captured above.

The first one is a binary switch because on the server side the actuator can act as a switch, a Boolean value. Every time when a Boolean it has to be handled on the server side, a switch will be draw on UI.

The second type of variable is a double. In this case in order to change the value of this variable a Seek Bar element is drawn on UI and the name of the actual function is displayed revealing exactly what the function does. This name it is also the last part of the endpoint that is called when the value is changed.

For more details about the device that is selected, swipe from right to left and a drawer will show all the information about it.

Figure 4.6 - Device details coming from the right drawer

4.2.3. Iotivity - Backend

In order to realize the interconnection between client and servers there were implemented different services. These are provided by Iotivity library built for this application and smartphone architecture as explained in section 4.2.1. The communication is done via OCF protocol.
It uses a REST architectural style, meaning that all the devices, things are modeled as resources. Servers expose resources to Clients and the Clients discover and access resources that are hosted by servers. All the modifications are possible as was already specified just after the onboarding or the transfer ownership was realized.

The first API that is used in this application does the scanning of devices. The call is made once the client refreshes the list of devices. If there are already devices these will be erased and if they are still available will be found again.

Code Listing 4.3. Method called for resource scanning

```
@Override
public void onSwipeRefresh() {
    mAdapter.clearItems();
    mViewModel.onScanRequested();
}
```

Inside the method "**onScanRequested**()" it is checked if the client device it is still connected to Wi-Fi. If it does the scanning process begins. All the resources(devices) can have one of the three possible states:

- UNOWNED - the resource found does not belong to any Client, it has no owner.
- OWNED_BY_SELF - the resource found does belong to a Client, which in this case is the actual client that just found the resource
- OWNED_BY_OTHER - the resource belongs to another Client, it is owned by other application

Code Listing 4.4. DeviceType enumeration

```
public enum DeviceType {
 UNOWNED,
 OWNED_BY_SELF,
 OWNED_BY_OTHER
}
```

The scanning is provided by IotivityRepository class where are implemented three methods that are handling this request. One of them is searching for devices that are not owned, one for devices owned by the client itself and the third one is searching for resources that are owned by others. The devices found are collected into an Observable object, which means that these devices are continuously under observation.

Code Listing 4.5. Observable Unowned Devices

```
Observable<Device> unownedObservable =
        mIotivityRepository.scanUnownedDevices()
                           .map(ocSecureResource ->
                                 new Device(DeviceType.UNOWNED,
                                 ocSecureResource.getDeviceID(),
                                 new OcDevice(),
                                 ocSecureResource)
                           );
```

The discovery itself it is accomplished once again with the help provided by the library Iotivity. OcProvisioning services are looking into the network connection and tries to discover resources.

The devices discovered are set as Disposable instances and the emitter implementation will dispose/cancel each instance of this kind when the flow is canceled or when the methods Emitter.onError(Throwable), Emitter.onComplete() succeeds.

Code Listing 4.5. Scan method for unowned devices

```
public Observable<OcSecureResource> scanUnownedDevices() {
        return Observable.create(emitter -> {
            try {
                mUnownedDevices=OcProvisioning
                    .discoverUnownedDevices(getDiscoveryTimeout());

                for (OcSecureResource ocSecureResource : mUnownedDevices) {
                    emitter.onNext(ocSecureResource);
                }
            } catch (OcException e) {
                Timber.e(e);
                emitter.onError(e);
            }
            emitter.onComplete();
        });
    }
```

After this point, if there were found devices/resources over the network the android application will list them, and the user will have the possibility to ask for ownership transfer.

The ownership transfer or the onboarding process requires connectivity check and at the same time bringing additional methods that can be used after onboarding.

- First check is done on the connectivity side, verifying if the device it is still connected to Wi-Fi

Code Listing 4.6. Wi-Fi connectivity check to run the app

```
    return wlanRepository.isConnectedToWiFi()
         .map(isConnected -> {
              if (!isConnected) {
                   throw new NetworkDisconnectedException();
              }

         return true;
    }).toCompletable();
```

- The next step is to obtain the methods supported after onboarding and set them on the selected device.

Code Listing 4.7. Set Ownership Transfer methods

```
.andThen(mGetOTMethodsUseCase.execute(ocSecureResource)
      .map(oxms -> {
           if (oxms.size() > 1) {
               return mOxmListener.onGetOxM(oxms);
           } else {
               return oxms.get(0);
           }
      }).filter(oxm -> oxm != null)
      .flatMapCompletable(oxm -> mSetOTMethodUseCase
           .execute(ocSecureResource, oxm))  ...
```

- At this point the onboarding starts and the device is updated with all the new methods supported and also with the information about what exactly this device can do, it's capabilities. Also, here the device receives its name.

Once the onboarding process has finished the Client has all the right over the device. It can get its actual state, it can change its state and it can update its state.
The resource has to have some properties such that the client can find it. Those are called common properties and the other ones that are called specific properties, and those are the ones that differ from device to device.

Code Listing 4.8. Binary Resource

```
    /the/light/1
{
          "rt": ["oic.r.switch.binary"],
          "if": ["oic.if.a",
          "oic.if.baseline"], "value": false
}
```

The method that is handling all the changes and are also sending the request to the server is called **processResource().** Inside this method there is an algorithm that checks what type of resource is processed and what type of event does occur on the client side. If there is a boolean resource, or also called a switch, on the UI will be drawn a switch component and if the resource supports also settings for intensity or power that takes as input an Integer on the UI will be developed a SeekBar component. At this level of development are implemented also the listeners for these components such that if any changes occur on the resource the listener acts and sends the message further to the server.

Code Listing 4.9. Handle Integer resource from server

```
if (entry.getValue() instanceof Integer) {
    if (isViewEnabled(response.getResourceInterfaces())) {
       SeekBar sb = new SeekBar(getContext());
       sb.setMax(100);
       sb.setProgress(((Integer) entry.getValue()).intValue());
       sb.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener()
        {
              …

              @Override
              public void onStopTrackingTouch(SeekBar seekBar) {
                  Integer number;

                  try {
                      number = seekBar.getProgress();
                  } catch (NumberFormatException e) {
                      return;
                  }

                  OcRepresentation rep = new OcRepresentation();
                  try {
                      rep.setValue(entry.getKey(), number);
                  } catch (OcException e) {
                       e.printStackTrace();
                      TextView textView = new TextView(getContext());
                      textView.setText("The resource could not be set");
                  }
                   mViewModel.postRequest(mDeviceId,mResource.getUri(),
                       mResource.getResourceTypes(),
                       mResource.getResourceInterfaces(), rep);
              }
          });
       view= sb;
  }
```

In the **postRequest()** method of "**mViewModel"** instance is realized the communication with the server. Before the post is sent, from the Iotivity repository it is extracted the host CoAP of the device based on provided "**mDeviceID"** and just after this data is obtained the connection is made and the request is sent to server.

Code Listing 4.8. Post changes using Iotivity provided methods

```
return mIotivityRepository.getDeviceCoapsIpv6Host(deviceId)
             .flatMap(host ->
                 mIotivityRepository.constructResource(
                         host,
                         uri,
                         resourceTypes,
                         interfacesList))
                .flatMap(ocResource -> mIotivityRepository.post(ocResource,
              true, rep));
```

This is the way the communication between server and client it is realized. If there are sensors that has to be read, Iotivity framework provides GET method too. Everything from this point is about imagination and needs.

4.2.4. Server Implementation

In order to implement the server on ESP32-PoE device, I had to set the software development environment based on Espressif ESP32. This is realized with the following prerequisites from **4.2.1. Prerequisites** section:

- Toolchain to build the app for ESP32
- ESP-IDF (Espressif IoT Development Framework) - has APIs for ESP32 and scripts to operate over the Toolchain

Once the environment for building the server is set, next steps are: implementing the server and integrating iotivity-lite into the ESP-IDF project.
A large part of the code for this server is generated using a procedure provided by the following environment [14]:

Install the environment to create the OCF server device by running the:

- "*curl https://openconnectivity.github.io/IOTivity-Lite-setup/install.sh | bash*"

In the environment that was just created using the above command, move into "iot-lite" folder and run the following commands:

- edit example.json - adding all the desired resources with their properties (dimming value)
- gen.sh - this script generates the code that represent the device(server)
- build.sh - this script builds the code that was generated
- run.sh - this script runs the compiled code that was generated.

After all the steps from above were followed and some changes at the internet connection layer were made such that the connection is realized through ethernet cable, I managed to have a server for ESP32-PoE. The changes that were made in order to realize this connection through ethernet cable were realized by integrating parts from different projects that already used this type of devices, with some adjustments that were necessary such that the communication between the server and the client is possible.
The needed functionality for my "smart home" system was developed over this generated server..
A basic lightbulb server was generated and using this model I implemented my servers for the system.
One of the servers has to support the following functions:

- Turn on/off the light
- Set brightness of cold color temperature LEDs
- Set brightness of warm color temperature LEDs

In the "simpleserver.c" file there were developed APIs that are able to support the functionality from above. For each property was implemented a **POST** and **GET** function. When a **POST** request is realized by the client, the post function from the server is triggered and the data that is sent through the json model is modifying the resource (the state of the device). The **GET** request called by the client over a resource triggers the "get" resource function from server and this is returning the values/information about that resource.

Code Listing 4.9. Get request response for client

```
   PRINT("   Adding Baseline info\n" );
   oc_process_baseline_interface(request->resource);
    /* property "value" */
   oc_rep_set_boolean(root, value, g_binaryswitch_value);
PRINT("%s:%d\n",g_binaryswitch_RESOURCE_PROPERTY_NAME_value,
g_binaryswitch_value );  /* not handled value */
  oc_rep_end_root_object();
  oc_send_response(request, OC_STATUS_OK);
```

The functions *oc_rep_set_boolean, oc_rep_end_root_object* and *oc_send_response* are used to create the resource json model from the server side such that the client can retrieve and process the data as a device resource.

When a **POST** request is triggered by the user, the payload is sent to the server and that resource is updated with the new data that was provided. This is done in a function that is set in the server as being an “oc_resource_handler” for that post request that was called. For each of the requests it is set a handler on the server side that is calling the specific function for that request any time a request is triggered. The data that is sent through the **POST** request it has to modify the state of the server in order to give the wanted output. This can be seen in the following code sequence:

Code Listing 4.10. Get request response for client

```
  /* set the response */
    PRINT("Set response %d\n",g_dimmingcold_dimmingSetting);
    oc_rep_start_root_object();
    /*oc_process_baseline_interface(request->resource); */
    oc_rep_set_int(root, dimmingSetting, g_dimmingcold_dimmingSetting );
    oc_rep_end_root_object();

    set_dimming_cold(g_dimmingcold_dimmingSetting); // the resource state it
                           // is changed here
    oc_send_response(request, OC_STATUS_CHANGED);

//response for the client if the resource is valid
```

The algorithm to maintain the state of the light was implemented in the following way. If the light is turned on/off using just the switch both of the light colors will give 75% from the led power or 0%. If the user wants to modify one of the led intensities, he can use the other two function for dimming the warm color led and for cold color led. Once one of values are modified the led will respond to that request based on the percentage provided by the user. The seekbar sends values from 0 to 100 which are being translated as percentage for the light power. The PWM pin is sending power to the leds based on the frequency which is calculated based on the percentage.

Code Listing 4.12. Algorithm handling the light resource

```
  void lightbulb_set_on(void *p)
{
    bool value = *(bool *)p;

    //PRINT("lightbulb_set_on : %s\n", value == true ? "true" : "false");

    if (value == true) { // if light on

        if(w_on == true || c_on == true){  //check if dimming used
            set_dimming_warm(s_cw_val.w); //set the value used from dimming
            set_dimming_cold(s_cw_val.c); //set the value used from dimming
        } else {
```

```
            s_cw_val.w = 75;  // if no dimming used than the light is used at
            s_cw_val.c = 75;  // 75% power
            s_bulb_state.set_on = true;
        }

    } else { // if light off
        s_cw_val.c = 0;  // set the led power to 0, and reset the values also
        s_cw_val.w = 0;  // for dimming settings
        s_bulb_state.set_on = false;
        w_on = false;
        c_on = false;
    }

    lightbulb_update();
    return;
}
```

The maths used to calculate the intensity of the led is using a frequency of 2048. The formula can be observed in the next call which is realized in order to set the intensity of the led.

Code Listing 4.12. Algorithm handling the light resource

```
 static void lightbulb_update()
{
     lightbulb_set_aim(s_cw_val.c * PWM_TARGET_DUTY / 100, s_cw_val.w *
                          PWM_TARGET_DUTY / 100,
                         (100 - 50) * 5000 / 100, 20 * 2000 / 100, 1000);
}
```

For the second server that is handling the “cooling system” the implementation is almost the same as for the light, the only difference is that here we have just one function that is setting the state of the fan as the client wants.

5 Testing

In this section it will be presented how the testing of the application was tested and which were the main problems encountered during the entire development and design.

The Android Application is built over an existing base application. First of all, I downloaded a working application that was loaded over the internet and try to discover my device. In this case the device was simulated on a Linux machine as a virtual device and the connection was realized successfully.

To build the code of the application in Android Studio there were necessary to follow the steps of building the Iotivity library for android. A lot of problems were encountered here because the architecture and the code of the app had to be compatible one to each other. After fixing the version of the code I managed to use some of the functionality. The implemented functionality added to this code was checked after each step. The sensors used have to provide the user with the real information. The time should be displayed, and the list of the devices found were checked to be accessible. To test the communication layer between the client and server an action was triggered such that I could see that the actual change is triggering the server function.

The server implementation on the ESP32-PoE device was the next step in developing the system. The code had to be adapted such that the application could find the device over the network. After managing to realize the connection between the server and client, the remaining step was to use the system.

The actual test phase was realized in the following manner:

Test the android application by using a virtual server that was simulated on my Linux machine. The application discovery phase should give as output only one server that is connected to that network. After the server was found I checked that the ID of the server it is the same with the one that application found.

Code Output Listing 5.1. Server connection to network and the resource registration

```
 ETHERNET: Ethernet Got IPv6 Addr
=== got IPv4 addr:192.168.1.100=== got IPv6 addr:FE80::827D:3AFF:FEE6:BF4FUsed
input file : "../device_output/out_codegeneration_merged.swagger.json"
OCF Server name : "Binary Switch"
Intialize Secure Resources
Register Resource with local path "/binaryswitch"
     number of Resource Types: 1
     Resource Type: "oic.r.switch.binary"
     Default OCF Interface: "oic.if.baseline"
Register Resource with local path "/dimmingcold"
     number of Resource Types: 1
     Resource Type: "oic.r.light.dimming"
     Default OCF Interface: "oic.if.baseline"
Register Resource with local path "/dimmingwarm"
     number of Resource Types: 1
     Resource Type: "oic.r.light.dimming"
     Default OCF Interface: "oic.if.baseline"
OCF server "Binary Switch" running, waiting on incomming connections.
```

The onboarding process of a device should give the client permissions to act over it and this can be seen in the android application when the color of the device becomes green. This is how I tested that the device was successfully onboarded and is now accessible from my app.

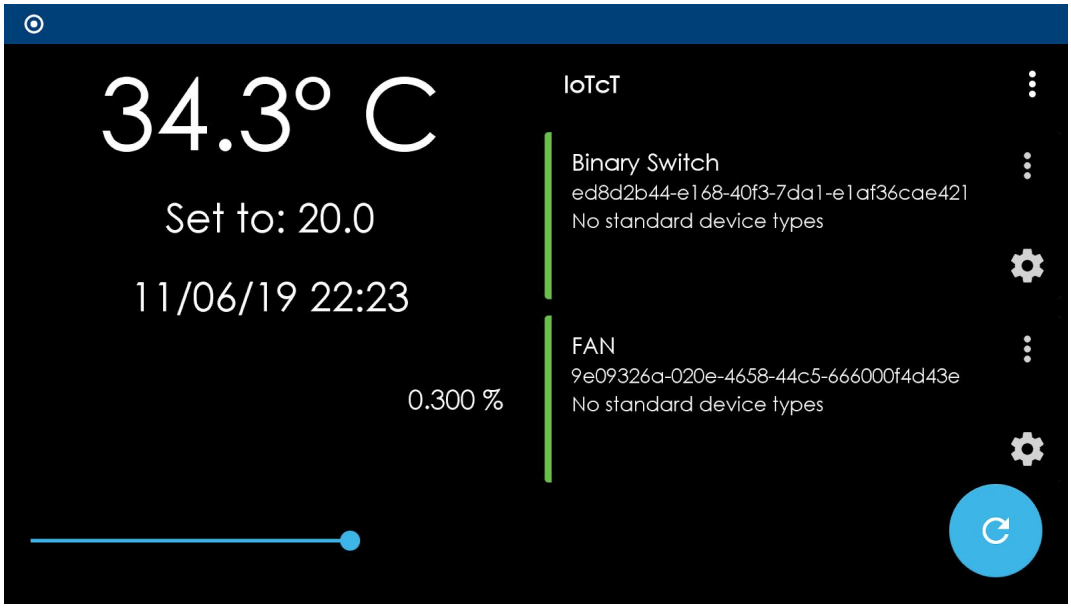

Figure 5.1 - Device details

It tested also that the request that were sent to the server are correct and no changes or errors occurred during the transmission process.

Code Output Listing 5.2. Get request executed when device route "/**binaryswitch**" is called

```
get_binaryswitch: interface 2
Adding Baseline info
value : 0
post_binaryswitch:
key: (check) value
key: (assign) value
Set response
```

When the value on the "/**binarywitch**" route is triggered the resource is modified and a POST request is executed. The server changes the state of the device and is sending the response to the server.

Code Output Listing 5.3. Post request executed when device route "/**binaryswitch**" is called

```
  post_binaryswitch:
key: (check) value
key: (assign) value
```

```
Set response
```

6 Conclusions

In this thesis I have presented you how I implemented my own “smart home” system using Power over Ethernet and how the system can be extended using new type of devices that can seamlessly connect to each other. All these devices being accessible through Internet and their state being changeable based on the user preferences using the application presented above. I also presented you a proof of concept that explains how I see such a system and which are the necessary components in developing your own “smart home” system as a developer.

Every time when you are trying to develop systems that implies integration between different software there will pop up problems. The difficult part from my point of view in developing the android application was finding out what was the problem with the built library from Iotivity. I tried different approaches such as trying to build the framework for debug mode application, with different architecture than my phone has and did not work. I tried to use existing methods that were doing the same discovery process but with other parameters because there were problems at compile time saying that the library does not provide those methods. After using the existing methods there was still one more problem. The application did not find the virtual device (server) that was running on my laptop. Finally, I found and solve the problem by using a previous version of base android application. Problems were encountered even when I tried to switch from the simulated server to the real device server. This was happening because the operating system differs, it is a RTOS on the ESP32-PoE device, and some changes in the implementation had to be made.

The resulted system after this implementation it is a demonstration of a “smart home” system where the lights and a cooler it is controlled using Power Over Ethernet. The system can be extended to use more devices and can be used for personal purposes.

For future development I am thinking to extend this solution such that the system will be accessible from any browser using a web page. The data that is stored now just internally in the app will be pushed in cloud and some analyzing algorithms will be implemented such that the system will be able to act by itself when no movements or nobody is home. Voice control will be another improvement for the android application that will be implemented.

As a conclusion, this system can be a good solution that can be incorporated into any building with the additional devices that are useful. There is plenty of space to improve this system and the application to,o and I am planning to work on it in the near future.

References

[2]- "Jefferson Graham, USA TODAY, Published 6:30 AM EST Jan 1, 2019, Amazon says putting Alexa to work in your smart home's easy. Here's exactly what you need "

[3]-https://cdn.macrumors.com/article-new/2018/10/alexa-app-update-800x395.jpg

[4]- "Julie Jacobson · May 1, 2018, Nest's New Video Doorbell and Smart Lock are Shipping"

[5]- "11 January 2017 Netherlands, The Edge: Amsterdam office building with highest BREEAM score to date"

[6]- "January 2017 Netherlands, The Edge: Amsterdam office building with highest BREEAM score to date, Key innovative feature @ PLP Architecture"

[7]- "April 10, 2013, What You May Not Know About GALAXY S4 Innovative Technology"

[8]- https://www.olimex.com/Products/IoT/ESP32/ESP32-POE/open-source-hardware

[9]- "Android Architecture Components, Part of Android Jetpack: https://developer.android.com/topic/libraries/architecture/index.html"

[10]- "ViewModel Overview, Part of Android Jetpack: https://developer.android.com/topic/libraries/architecture/viewmodel"

[11]- "OCF Core Specification VERSION 1.3.1 | February 2018 Part 1"

[12]-https://openconnectivity.org/foundation

[13]- "February 2019, OCF Specification Overview Core Technology Specification OCF 2.0.1 Release"

[14]- https://openconnectivity.org/developer/reference-implementation/iotivity

[15]-https://github.com/openconnectivity/IoTivity-Lite-setup

DECLARAȚIE DE AUTENTICITATE A

LUCRĂRII DE FINALIZARE A STUDIILOR

Subsemnatul

_____

_____ ,

legitimat cu _____seria _____nr. _____,

CNP_____

autorul lucrării _____

_____

_____

elaborată în vederea susținerii examenului de finalizare a studiilor de _____ _____organizat de către Facultatea _____ _____din cadrul Universității "Politehnica" din Timişoara, sesiunea _____ a anului universitar _____, luând în considerare conținutul art. 39 din RODPI – UPT, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române şi a convențiilor internaționale privind drepturile de autor.

Timişoara,

Data Semnătura

_____ _____