



Semester: 4th

Title: Machine Learning as a Service for
a Personalized Smart Home Environment

Aalborg University Copenhagen
A.C. Meyers Vænge 15
2450 København SV

Semester Coordinator: Henning
Olesen

Secretary: Maiken Keller

Project Period:

Spring 2017

Semester Theme:

Master Thesis

Supervisor(s):

Lazaros Nalpantidis
Tsampikos Kounalakis

Project group no.:

ICTE4 SER 4.4

Members

(do not write CPR.nr.):

Konstantinos Gkentzoglanis

Pages: 94

Finished: 07.06.2017

Abstract:

This project emphasizes on designing and developing a personalized service for Smart Homes and their inhabitants, with the objective of achieving a self-operated house. The main idea is that all “*smart*” devices installed in a house would be able to function by themselves without human intervention, based on the preferences of the house’s residents. To achieve that, several technologies need to be utilized.

Internet of Things (IoT), Cognitive Computing and the Cloud are used in order to control the “*smart*” devices and collect historical data over time. These data contain information about user preferences inside a house, e.g. a room’s temperature, or if a room’s lights are on. A Cloud-based IoT platform was utilized, along with several software-developed IoT devices (simulating real “*smart*” devices in a house), with the purpose of collecting the aforementioned historical dataset. The Smart Home user can interact with the IoT devices directly, or by using a smartphone application. The application was developed to exploit Cloud-based Cognitive Computing capabilities, resulting in a more natural interaction between the user and the IoT devices. Once a dataset has been collected, several Supervised Machine Learning algorithms are used, as well as Time Series Forecasting, in order to train Machine Learning models. These models are trained based on the aforementioned dataset. Following that, the trained models are evaluated in order to detect the one performed best. Then, the model which provided the finest results is selected to be used in a Smart Home environment.

Finally, the smartphone application can contact the previous trained Machine Learning model, through a web service, and request future predictions for the status of all IoT devices inside a house over time. Then, the status of all IoT devices is adjusted according to the received forecasts.

When uploading this document to Digital Exam each group member confirms that all have participated equally in the project work and that they collectively are responsible for the content of the project report. Furthermore, each group member is liable for that there is no plagiarism in the report.

Table of Contents

1.	Introduction.....	1
1.1.	Motivation	2
1.2.	Proposed Solution	2
1.3.	Problem Formulation.....	3
1.4.	Methodology	5
1.5.	Expected Outcome	7
2.	State-of-the-Art.....	8
2.1.	Internet of Things.....	8
2.1.1.	IoT - Devices.....	9
2.1.2.	IoT - Network	10
2.1.3.	IoT – Computing.....	12
2.2.	Machine Learning	12
2.2.1.	Supervised Learning	13
2.2.2.	Time Series Forecasting.....	15
2.2.3.	Evaluation	17
2.3.	Cognitive Computing	18
2.4.	Cloud Platforms.....	19
2.4.1.	Microsoft Azure	20
2.4.2.	IBM Bluemix	22
2.4.3.	Amazon Web Services (AWS)	23
2.4.4.	Differences between Cloud Platforms	24
2.5.	Smart Home Solutions	27
2.5.1.	Commercial Solutions.....	28
2.5.2.	Research Solutions.....	31
2.6.	Remarks.....	32
3.	Analysis.....	33
3.1.	Proposed Solution	33
3.1.1.	Differences with related work.....	35
3.1.2.	Security and Privacy	37
3.2.	Scenarios & Use Cases.....	37
3.2.1.	Scenario 1 – Controlling Home Devices	37
3.2.2.	Scenario 2 – Machine Learning as a Service	38
3.2.3.	Use Cases	38
3.3.	Requirements Specification.....	41
3.4.	Remarks.....	43
4.	Design and Implementation	45
4.1.	Design Choices.....	45
4.1.1.	System Architecture.....	46
4.2.	Implementation.....	50

4.2.1.	Azure IoT Hub	50
4.2.2.	Database	51
4.2.3.	IoT Devices	51
4.2.4.	Smartphone Application	57
4.2.5.	Cognitive Computing Service.....	60
4.2.6.	Machine Learning Service	62
4.3.	Experimental Assessment	63
4.3.1.	Dataset Acquisition.....	63
4.3.2.	Machine Learning Experiments.....	64
4.3.3.	Comparison between Time Series, Regression and Classification Models.....	69
4.4.	System Prototype Deployment & Testing.....	73
4.4.1.	Prototype Testing	74
5.	Conclusion	78
5.1.	Discussion	78
5.2.	Future Recommendations.....	80
6.	References.....	83

List of Figures

Figure 1: Proposed Solution.....	3
Figure 2: Methodology.	6
Figure 3: Cloud Platform.	20
Figure 4: Samsung SmartThings.....	29
Figure 5: Nest Thermostat.	30
Figure 6: Google Home.	31
Figure 7: MLaaS – Context Diagram.....	34
Figure 8: MLaaS - Context Diagram 2.	34
Figure 9: MLaaS – Use Case Diagram.	39
Figure 10: MLaaS Complete Architecture.....	47
Figure 11: MLaaS Implemented Architecture.	49
Figure 12: User House – Rooms and IoT Devices.	52
Figure 13: IoT Device – Type 1.....	53
Figure 14: IoT Device – Type 2.....	54
Figure 15: IoT Device – Registration.	55
Figure 16: IoT Device – Cloud-to-Device Communication.	55
Figure 17: IoT Device – Register to Local Database.....	56
Figure 18: IoT Device – Store Device Status.	56
Figure 19: Smartphone Application.....	57
Figure 20: Smartphone Application – Connect to Cognitive Service.....	58
Figure 21: Smartphone Application – Retrieve Devices.	59
Figure 22: Smartphone Application – Bind Devices to UI.....	59
Figure 23: Smartphone Application – Retrieve Recent Status.	60
Figure 24: Smartphone Application – Connect to Machine Learning Web Service.	60
Figure 25: LUIS – Training and Testing.....	61
Figure 26: Methodology - Step 5.....	62
Figure 27: User Preferences.....	64
Figure 28: Azure ML Studio – Dataset every second 1.....	65
Figure 29: Azure ML Studio – Dataset every second 2.....	65
Figure 30: Azure ML Studio – th01 Line Plot.....	66
Figure 31: Azure ML Studio – lb01 Line Plot.....	67
Figure 32: Azure ML Studio – th01 ETS Training Model.	68
Figure 33: Azure ML Studio – ARIMA Prediction for th02 and lc01.....	72
Figure 34: Azure ML Studio – Web Service.	74
Figure 35: IoT Devices – Change lb03 Status 1.	75
Figure 36: IoT Devices – Change lb03 Status 2.	75
Figure 37: Smartphone Application – Change lb03 Status.....	76
Figure 38: Smartphone Application – Change Devices' Status.	77
Figure 39: IoT Devices – Change Devices' Status.....	77

List of Tables

Table 1: Communication Technologies Comparison.	10
Table 2: Communication Protocols Comparison.	11
Table 3: Cloud Platforms – IoT.	24
Table 4: Cloud Platforms – Machine Learning.	26
Table 5: Cloud Platforms – Cognitive Computing.	27
Table 6: Functional Requirements.	42
Table 7: Non-Functional Requirements.	43
Table 8: Azure ML Studio – Regression & Classification Algorithms.	69
Table 9: Mean Values of Thermostats – MAPE.	70
Table 10: Mean Values of Light Bulbs and Lock – MASE.	71

List of Abbreviations

ACF	AutoCorrelation Function	ML	Machine Learning
AI	Artificial Intelligence	MLaaS	Machine Learning-as-a-Service
AMQP	Advanced Message Queuing Protocol	MPE	Mean Percentage Error
APIs	Application Programming Interfaces	MQTP	Message Queue Telemetry Transport
AR	AutoRegressive	OMG	Object Management Group
ARIMA	AutoRegressive Integrated Moving Average	OS	Operating System
AWS	Amazon Web Services	PaaS	Platform-as-a-Service
BES	Batch Execution Service	PACF	Partial AutoCorrelation Function
BLE	Bluetooth Low Energy	REST	REpresentational State Transfer
BPNN	Back Propagation Neural Network	RMSE	Root Mean Squared Error
CoAP	Constrained Application Protocol	RRS	Request-Response Service
DDS	Data Distribution Service	SaaS	Software-as-a-Service
DTLS	Datagram Transport Layer Security	SSL	Secure Sockets Layer
ETS	Error Trend Seasonality	STL	Seasonal and Trend using Loess
ETSI	European Telecommunications Standards Institute	SVM	Support Vector Machine
HTTP	Hypertext Transfer Protocol	TCP	Transmission Control Protocol
IaaS	Infrastructure-as-a-Service	TLS	Transport Layer Security
IEEE	Institute of Electrical and Electronics Engineers	UDP	User Datagram Protocol
IETF	Internet Engineering Task Force	UI	User Interface
IHMM	Improved Hidden Markov Model	URIs	Uniform Resource Identifiers
IoT	Internet of Things	UWP	Universal Windows Platform
IPA	Intelligent Personal Assistant	UX	User Experience
LTE-A	Long Term Evolution-Advanced	WLAN	Wireless Local Area Network
LUIS	Language Understanding Intelligent Service	WPANs	Wireless Personal Area Networks
M2M	Machine-to-Machine		
MA	Moving Average		
MAE	Mean Absolute Error		
MAPE	Mean Absolute Percentage Error		
MASE	Mean Absolute Scaled Error		
MDP	Markov Decision Process		

1. Introduction

The world as we know is changing rapidly, because of the accelerated evolution of existing technologies and the invention of new ones. Internet of Things (IoT) is one example of the fast-paced evolution of technology. The term IoT refers to physical-electronic devices which among other functionalities (software, sensors, actuators, etc.) are also equipped with capabilities of internet connectivity [1]. By using an internet connection, these devices can communicate with other devices, i.e. Machine-to-Machine (M2M) communication. According to [2, p. 1] the exchange of collected data is very important for achieving the IoT paradigm.

Apart from IoT, other technologies and services are constantly developed be used along with IoT devices. They assist in the understanding of collected data and enhance their decision making. These technologies can be found under the field of Artificial Intelligence (AI). Machine Learning (ML) is one of them and even though is not new, it has undergone tremendous development in recent years. By combining IoT and ML, different use cases can be implemented in different environments, industrial and residential alike. For instance, in [3] a framework was proposed in healthcare where an eHealth service is capable of detecting in real-time face disorders. In that scenario, images of people's faces can be identified and collected through IoT devices (in the specific scenario only cameras were used) [3, p. 1]. Then, using the K-Means, i.e. an unsupervised machine learning algorithm, faces can be further analyzed and abnormalities can be detected [3, p. 3]. Another example of IoT cooperating with ML can be seen in [4], where the behavior of people with disabilities could be predicted by using an Improved Hidden Markov Model (IHMM) in a Smart Home environment. In that paper, the goal was to predict the next behavior of a person by studying his/her habits. By doing so, the necessary appliances inside the house can be automatically activated. The interaction between the user and the Smart Home devices is based on voice commands [4, p. 2], which makes it easier for people with disabilities.

Moreover, voice commands become more and more popular in many different use case scenarios and many solutions are focusing on that. Companies like Google and Amazon have already on the market products like Google Home [5] and Amazon Echo [6] which utilize speech recognition. These devices, among other functionalities, can be also used in a house to manage all the smart appliances installed in it [6], [7]. For instance, the user would be able to adjust the heat of the house, turn on or off the lights, request to turn on the TV or play a specific song. Investigating further Google Home and Amazon Echo, it can be seen that they do not only understand voice commands but they use the power of Cognitive Computing to understand their context. To do so, each one uses its own Intelligent Personal Assistant (IPA), Google Assistant [8] and Amazon Alexa [6]. Because of that, the user does not have to syntax the command exactly the way the computer is anticipating it, which also means that the user does not have to remember specific commands. However, voice commands are only one of the few capabilities of Cognitive Computing. Machine Learning, Speech/Vision Recognition, Natural Language Processing and many more are also included [9, p. 8].

Although implementations of the technologies described above exist in a home environment, the focus of this report is the further investigation of the aforementioned technologies (IoT, ML and

Cognitive Computing). Then, a solution is proposed, on how they can be better combined and used to deliver a unified personalized service to Smart Home users.

The rest of the chapter is structured as follows. First, the motive of why the field of Smart Homes is of interest is examined. Then, the solution to be proposed, designed and implemented during this project is briefly described. Furthermore, the problem that the proposed solution is going to solve is discussed, along with the necessary research questions, objectives and limitations. Last but not least, the methodology which was followed during the course of the project is documented and the outcome that is expected by the end of this project is stated.

1.1. Motivation

As it was briefly discussed previously, IoT, ML and Cognitive Computing play already a big part in all kind of “*smart*” scenarios, like Smart Homes and eHealth. Particularly, Smart Homes is a popular research field that attracts many researchers and companies. Both are interested in the development of different services and products which can make homes even smarter.

However, as seen in the State-of-the-Art chapter, chapter 2, solutions coming from researches are quite different from the actual solutions available on the market. For instance, most of the services and products currently provided on the market are focused on how individuals could access and control different IoT devices installed in their house. Yet, they neglect solutions which could automate the entire process based on user needs and habits. On the other hand, researches mostly focus on the automation of the house without considering how their solution could be deployed in a larger scale. The gap between the two “*worlds*”, i.e. the scientific and the industrial, will become more transparent during chapter 2: State-of-the-Art. This phenomenon constitutes the motivation factor, which initiated ideas on “How a unified service could be built which is intelligent and, at the same time, feasible for a real-world application?”

1.2. Proposed Solution

The main purpose of this project is to design and develop a solution that involves many different technologies. This solution is focused on Machine Learning (ML) and its use as a service for identifying and then emulating user interactions with IoT devices inside a house over time. The objective is to make a house autonomous, through the use of those technologies. The Smart Home IoT devices can be accessed and controlled through the Cloud using a smartphone application. In the Cloud, an ML web service will firstly observe user interactions with home devices. Then, an ML model will be trained based on these interactions, thus being capable of predicting future user-home devices interactions and control the devices instead of the user. Cognitive Computing comes into play when the user needs to interact with the IoT devices. Specifically, Natural Language Processing will enable users to give commands in a more natural way without the need of remembering specific ones, for specific types of IoT devices. This proposed solution is illustrated below in Figure 1.

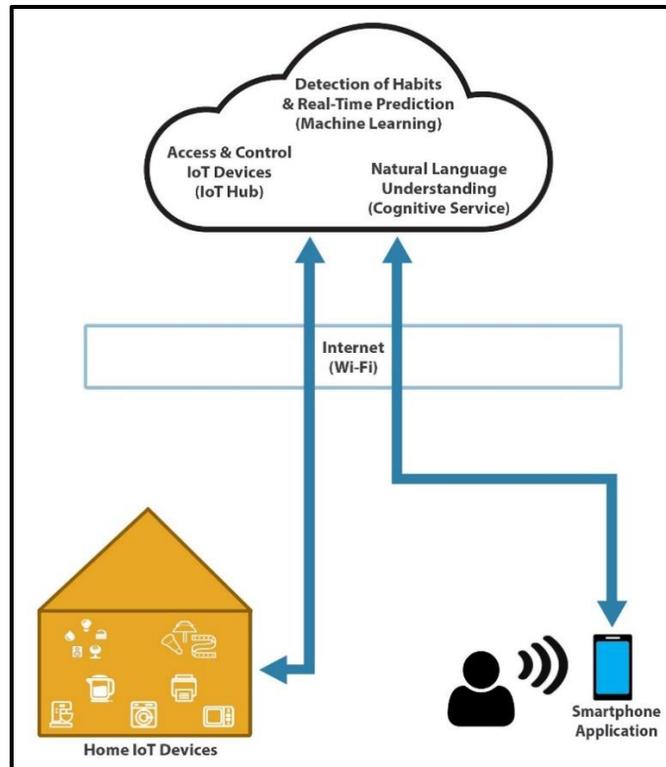


Figure 1: Proposed Solution.

By taking a closer look at Figure 1, one can observe that there can be many IoT devices in a house with a connection to the Cloud through the Internet. These devices can be accessed and controlled from a smartphone application, which the user interacts with by text or voice. The smartphone application can understand voice and text commands by utilizing the power of the Cloud to understand natural language. Once the user starts interacting with the smartphone application to control the IoT devices, the ML component of the Cloud records user preferences over time. Then, the trained ML algorithm is capable of predicting user preferences for specific time frames and control all the IoT devices in the house, without the need of user intervention. At any time, the user can see the status of all IoT devices on his/her smartphone and change it.

The first challenge of the proposed solution is the creation of a common service where many heterogeneous devices (e.g. door locks, light bulbs, thermostats, etc.) can be connected to in order to be accessed and controlled by the user. The next and greater challenge is to build a Machine Learning service which would be able to train itself based on historical data of user habits, thus predicting the future status of all installed IoT devices in real time.

1.3. Problem Formulation

In order to solve the challenges presented in the proposed solution, the research question of this project is shaped as:

How the Internet of Things, Machine Learning and Cognitive Computing technologies can be used to design and develop a unified personalized service in a Smart Home environment for a purely autonomous house?

To answer that question, it is needed to break it down to smaller sub questions and relevant objectives.

- **Research sub-questions**

Following the more generic research question, sub-questions that are needed to be answered are:

1. What is the current State-of-the-Art in Smart Homes considering the IoT, ML and Cognitive Computing?
2. How a new solution can be designed for Smart Homes to achieve a self-operated house?
3. How a proof of concept can be developed based on this new solution for Smart Homes?

- **Objectives**

In order for the main research question along with the three sub questions to be answered, certain objectives need to be defined. These are described below.

1. Knowledge about IoT, Machine Learning and Cognitive Computing must be gained.
2. Research is required about other technologies such as the Cloud and how it can be used in relation to IoT, ML and Cognitive Computing.
3. An investigation is necessary to be conducted on how IoT, ML and Cognitive Computing technologies are currently used in a Smart Home environment.
4. Specification requirements are essential for the design of the proposed solution and the implementation of the proof of concept.

To efficiently fulfill all the aforementioned objectives and answer the research main question, some limitations need to be defined.

- **Limitations**

Due to time restrictions, several delimitations need to be set. These are:

1. It is presumed that internet connectivity is always available and no research is made on how the proposed system would function without one.
2. Communication standards between devices are restricted to Wi-Fi only. Further research is needed about other connectivity protocols that could be used, e.g. Z-Wave, ZigBee, Bluetooth etc.
3. The “*full-scale*” solution is not implemented due to time constraints. Instead, a big part of it will be presented as a proof of concept.
4. Although concerns about security and privacy are raised, they are not taken under consideration during the design of the solution and the implementation of the

prototype, except the standard security measures already implemented and required by the communication protocols in use.

5. No physical IoT devices are used in the prototype. Instead, software applications simulating real IoT devices are developed and used, as the focus of the solution is not the devices themselves but the interaction between them and the rest of the system.
6. It is assumed that all IoT devices are already installed in a house and no initialization steps are included in the proposed solution.
7. IoT devices included in the solution are limited in number and type (only thermostats, light bulbs and door locks were used). Further research is needed on how the system would behave with more or less IoT devices and different types.
8. The produced data by the simulated IoT devices are going to be gathered under a controlled environment, i.e., for only one user and for a restricted time frame. Further research is needed for collecting and analyzing data in different environments and users with diverse needs and habits for a longer period of time.
9. As the project is limited to only one user, it is also limited to only one user type with full authorization access to all IoT devices. Additional research is needed to design a system with different types of users and authorization access.
10. The data to be gathered from the IoT devices are going to be stored in a SQL database and further investigation is needed to conclude whether it is the ideal one or a NoSQL or a Graph database is the better choice.
11. The smartphone application is developed only for one platform, i.e. for Windows 10 Mobile, and further research is needed to decide which platform is the ideal target.
12. The smartphone application User Interface (UI) and User Experience (UX) are designed and developed solely on technical aspects based on the developer's opinion. Thereby collaboration and feedback from users is not considered, which could improve both UI and UX.
13. Even though Machine Learning algorithms are used, only their fundamentals are studied and an extended research is needed. As a result, the selection of one algorithm over another is based only on the results each one produces during the evaluation process.
14. Only a few Cloud Platforms are reviewed and the selection of one is based on that. Further research is needed to investigate all the available platforms and which one is the better choice.

1.4. Methodology

The methodology described in this section is the overall methodology followed through the course of this project. As it can be seen in the following figure, Figure 2, six distinct stages are presented.

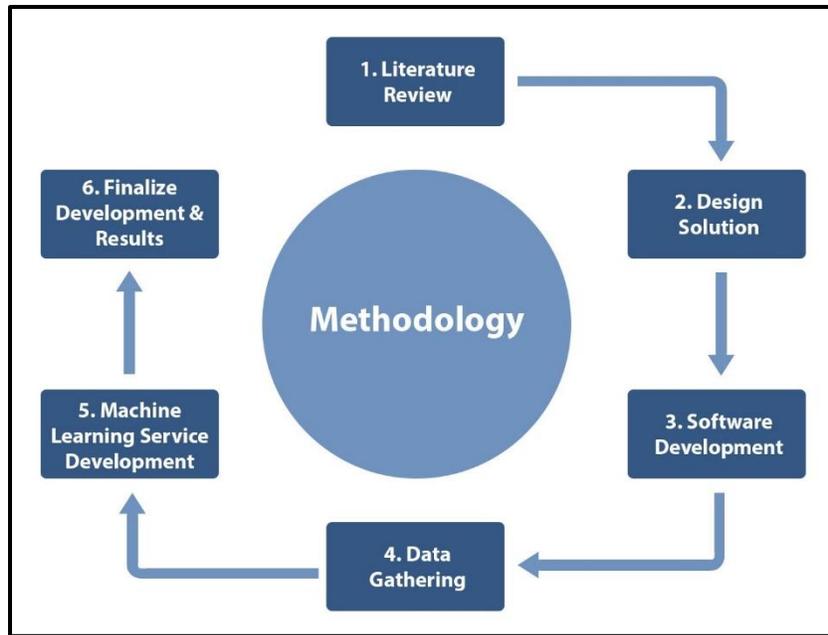


Figure 2: Methodology.

1. **Literature Review:** During this step, secondary research is conducted including academic papers, educational books, articles and journals. This research will provide knowledge useful for fulfilling segments of all the objectives and research sub-questions described in the previous section. As a result, the foundation for answering the main research question will be laid. Literature review is also required for accomplishing the next stages of the methodology.
2. **Design Solution:** Following the secondary research, the solution is designed along with its architecture. This step is dedicated on analyzing all the findings discovered so far, what technologies are going to be used and how.
3. **Software Development:** At this stage of the report, the required software is developed according to the designed solution. This is the first out of three parts of the development process as additional data are required for its continuance.
4. **Data Gathering:** As soon as the software development is completed, it is possible to gather data which are important for the next phase of the development process. These data were gathered by using primary research and more specifically quantitative approach. Data were collected by structured observations where the behavior of people can be documented and analyzed.
5. **Machine Learning Service Development:** Having a dataset, the second phase of the development process can start. There, several Machine Learning algorithms are tested and evaluated. Based on this evaluation, the best performing algorithm is chosen as the one to be deployed as a web service and utilized by the proposed solution.

- 6. Finalize Development & Results:** Finally, the software from step 3 can be combined with the Machine Learning service from step 5 and provide a unified service to smart home users. The next step is to test the proposed solution, document the findings and reflect on them.

In addition to the methodology, a Gantt chart illustrating the project schedule is available in Appendix 01.

1.5. Expected Outcome

By the end of this report, it is expected that a smartphone application will be developed. The user will be able to give a command by text or voice to the application in natural language. Then, the application will be able to understand the context of the speech or text and respond back to the user the same way. Through this interaction, the user will be able to manage the IoT devices installed in the house, e.g. turn on the lights, adjust the heat, lock the doors etc. A single interaction between the user and a IoT device will be accompanied by a timestamp and saved in a database. The application will also display the current status of the house, for instance the temperature in the bedroom.

Once the interaction between the user and the application generates enough data, the Machine Learning algorithm will gain access to those in order to start training itself. The moment the training is completed, the application will be able to ask the ML algorithm for predictions based on time, through a web service. As soon as the predictions are available to the application, it will send these data directly to the IoT devices in order to change their status, e.g. turning the kitchen light on from off. By doing so, the automation of the house is achieved.

It is important to state that the outcome described above is the entire proposed solution and it is further analyzed on chapter 3: Analysis. However, the prototype to be developed will have less functionalities due to time constraints as it was stated in section 1.3 – limitations. It is also worth mentioning that during the design and implementation of the proof of concept, several ML algorithms are going to be tested and the one generating the best results will be selected for deploying it as a web service.

2. State-of-the-Art

This chapter focuses on the latest and greatest technologies currently available for building a Smart Home solution. Moreover, it includes current solutions available on the market, as well as, solutions developed from researchers. The purpose here is to answer the objectives:

1. *Knowledge about IoT, Machine Learning and Cognitive Computing must be gained.*
2. *Research is required about other technologies such as the Cloud and how it can be used in relation to IoT, ML and Cognitive Computing.*
3. *An investigation is necessary to be conducted on how IoT, ML and Cognitive Computing technologies are currently used in a Smart Home environment.*

The purpose of this chapter is to provide an answer to the research sub-question 1:

What is the current State-of-the-Art in Smart Homes considering the IoT, ML and Cognitive Computing?

and set the basis for answering the research sub-question 2:

How a new solution can be designed for Smart Homes to achieve a self-operated house?.

Findings acquired from this chapter are later elaborated in chapter 3: Analysis, for answering the main research question.

The rest of the chapter is structured as follows. First the Internet of Things is researched. Attention is given to the term IoT, which devices can be considered part of IoT, how these devices can communicate with each other and how autonomous decision making can be achieved. Following that, the interest is shifted towards Cloud Platforms, Cognitive Computing and Machine Learning, what are they and how can they influence IoT. Finally, existing Smart Home solutions are documented and how they use the aforementioned technologies.

2.1. Internet of Things

There is not a clear definition of the Internet of Things as many organizations, partnerships agencies and communities involved in defining technological standards and more, are in a disagreement, each providing its own definition [1]. For instance, the Institute of Electrical and Electronics Engineers (IEEE) describes IoT as:

“A network of items – each embedded with sensors – which are connected to the Internet.” [1, p. 10],

and the European Telecommunications Standards Institute (ETSI) as:

“Machine-to-Machine (M2M) communications is the communication between two or more entities that do not necessarily need any direct human intervention. M2M services intend to automate decision and communication processes.” [1, p. 12],

whilst the Internet Engineering Task Force (IETF) as:

“The basic idea is that IoT will connect objects around us (electronic, electrical, non-electrical) to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and co-operate each other to make the service better and accessible anytime, from anywhere.” [1, p. 19].

As it can be concluded from the definitions and descriptions mentioned above, IoT refers to any physical object/item/entity that can communicate with other objects through a network, exchange data and make decisions. Thus, IoT can safely be distinguished into three different areas, the devices (sensors, actuators, mobile phones, etc.), the communication network (e.g. the internet) and the decision making (computing), each described in the subsections to follow.

2.1.1. IoT - Devices

As mentioned earlier, a thing in the IoT domain can be any physical object that can communicate with another object and provide data and services over a network. In some cases, these objects could also be functional without a human intervention. These objects, things or devices can be any kind of sensors which can see, hear, talk, think (e.g. cameras, microphones, speakers, processing units) or act (e.g. actuators). In [10], IoT devices are divided into resource-rich and resource-constrained devices.

- **Resource-Rich Devices**

This type of devices are the ones that support the TCP/IP protocol suite by having the appropriate software and hardware to do so [10, p. 20]. Still, this does not mean that these devices are not constrained in terms of battery capacity and computational power. Because of these limitations certain protocols have been developed for data exchange in the application layer of the TCP/IP suite. These are the CoAP, MQTT, AMQP and others [10, p. 20], which are going to be described in more depth later on section 2.1.2.

- **Resource-Constrained Devices**

These devices, other than the limited battery capacity and computational power, they are also constrained in terms of software and hardware which results in the failure of supporting the TCP/IP protocol suite [10, p. 20]. Because of that, extra devices are needed that can help establishing a communication channel between them and the internet [10, p. 20]. The devices which act as a “middleman” between the resource-constrained devices and the internet can be simple gateways. As it will be discussed later on section 2.5, many Smart Home solutions have developed this kind of implementation.

Resource-rich and resource-constrained devices are both important in IoT and both types are going to exist in the future, since the selection between the two for an IoT implementation is based on the use case and the solution that is going to be developed. For instance, in an IoT solution where the devices need to consume as less energy as possible, the resource-constrained devices would be more suitable, since the communication technologies they use (e.g. Bluetooth Low Energy) are more energy efficient, as it is also documented later on section 2.1.2. On the other hand, if immediate internet access is one of the top priorities, then the resource-rich devices should be

considered in implementation. Other characteristics that should also be considered when choosing between the two types of devices are throughput, range, security, cost, speed and overall performance.

2.1.2. IoT - Network

The communication network area of the IoT is divided into two segments. The first describes the physical layer, i.e., the implemented technologies and standards (Wi-Fi, ZigBee, Bluetooth, etc.). The second layer is the application layer where used communication protocols are presented (CoAP, AMQP, DDS, etc.).

Physical Layer

The devices in an IoT environment can communicate with each other by using communication technologies over a network. There are many communication technologies for IoT solutions, some of them are Wi-Fi, IEEE 802.15.4, Bluetooth, Z-Wave and ZigBee. Each one with its own benefits and shortcomings in terms of power consumption, speed, range, application and cost [11, p. 2]. The following table, Table 1, summarizes all technologies based on their frequency band, data rate and range.

	Wi-Fi (802.11a/b/g/n/ ac/ad/ah) [10], [11], [12], [13]	IEEE 802.15.4 [10], [14]	Bluetooth (Low Energy) [10], [15]	Z-Wave [10], [16]	ZigBee [10], [11], [17]
Frequency Band	1 GHz 2.4 GHz 5 GHz 60 GHz	868 MHz 915 MHz 2.4 GHz	2.4 GHz	900 MHz	868 MHz 915 MHz
Data Rate	1 Mb/s – 6.75 Gb/s	20 Kbit/s, 40 Kbit/s, 250 Kbit/s	1 Mb/s – 52 Mb/s	40 – 200 Kbit/s	250 Kbit/s
Range	20 – 100 meters	10 meters	10 – 50 meters	100 meters	10 – 100 meters

Table 1: Communication Technologies Comparison.

In [10], comparisons between the above described communication technologies take place. More precisely, BLE outperforms the IEEE 802.15.4 and IEEE 802.11ah in power consumption [10, p. 14] yet still, between the IEEE 802.15.4 and 802.11ah, the first one is more energy efficient [10, p. 15]. However, the IEEE 802.15.4 was surpassed by IEEE 802.11ah in terms of throughput [10, p. 14]. Moreover, a comparison between Z-Wave and ZigBee indicated that ZigBee’s performance was not a match for Z-Wave’s, even though ZigBee has larger data rate, and that the Z-Wave’s implementation is more expensive than ZigBee’s [10, p. 15].

As it can be concluded, each technology has its advantages and disadvantages and whether to use one over the other highly depends on the use case they are going to be implemented and how well they perform in it. It also depends on the type of devices, whether they are resource-rich or resource-constrained (see section 2.1.1), and their hardware/software specifications and requirements.

Application Layer

Following the previous discussion, for a complete scenario of exchanging messages between IoT devices, communication protocols in the application layer need to be considered as well. Because of the limited processing power and battery capacity characteristics of these devices, different communication protocols other than the standard one, i.e., HTTP in the application layer of the TCP/IP protocol suite, have been developed. These are:

- The Constrained Application Protocol (CoAP),
- The Message Queue Telemetry Transport (MQTT),
- The Advanced Message Queuing Protocol (AMQP) and
- The Data Distribution Service (DDS).

Their characteristics are presented in the following table.

	CoAP [10], [18]	MQTT [10], [19]	AMQP [10], [20]	DDS [10], [21]
RESTful	✓	N/A	N/A	N/A
Transport	UDP	TCP	TCP	TCP /UDP
Security	DTLS	SSL	SSL	SSL/DTLS
Publish/Subscribe	✓	✓	✓	✓
Request/Response	✓	N/A	N/A	N/A

Table 2: Communication Protocols Comparison.

From Table 2, one can observe that some protocols share the same characteristics. However, their implementation is quite different, as documented in [10, pp. 8–10], and one protocol could perform better than another in different occasions. For this reason, whether or not to choose one over the other depends on the situation and the use case that is going to be implemented. It is also important to consider the type of devices, whether they are resource-rich or resource-constrained (see section 2.1.1), and their hardware/software specifications and requirements. A similar conclusion was derived in [10] as well, “each of these protocols may perform well in specific scenarios and environments. So it is not feasible to provide a single prescription for all IoT applications.” [10, p. 10].

2.1.3. IoT – Computing

The decision making, or computing, is the third and final part of the IoT domain. On this step, the IoT devices can consume the received data and make decisions. Although the idea of each IoT device having its own computation capabilities sounds logical, it would generate difficulties because of the limitation of these devices, as they were described in section 2.1.1. In that section, IoT devices were divided into two categories, resource-rich and resource-constrained, both limited on battery capacity and computational power. Thus, while basic functionalities could be implemented on these devices, larger and more complicated tasks should be performed elsewhere like microcontrollers, microprocessors, computers or even the Cloud.

Considering the resource-constrained devices, which require a gateway to connect to the internet (see section 2.1.1), adding an extra component for calculating the heavy tasks makes the whole architecture more complicated. As a result, two solutions can be considered. The first would be to have the gateway perform these tasks, which would require including some extra processing power in it. The second solution would be to assign these tasks to be computed in the Cloud. Both of these are feasible as stated in [10] and solutions exist which use a combination of both, as it is discussed later on section 2.5. In [10, p. 5], microcontrollers and microprocessors like Arduino [22], Raspberry Pi [23], Intel Galileo [24], etc. are described as hardware devices capable of supporting IoT applications. These devices accommodate the proper hardware and software to act as a simple gateway but also the “brain” behind an IoT solution. It should be noted though, that these devices are limited in computational power and if an IoT solution requires more, for instance providing intelligent services by using Big Data, Machine Learning or Cognitive Computing, then the Cloud can be utilized on performing these tasks [10, p. 5]. On the other hand, the resource-rich devices, which do not require a gateway, can directly communicate with the Cloud even for the simplest calculations.

By using the aforementioned solutions (microprocessors, microcontrollers, computers, Cloud) a user can access the IoT devices (e.g. from a webpage or a mobile application), check their status, give commands or even leave the devices make their own decision. The interaction between the user and the IoT devices could be one of the simplest forms, by using specific text/voice commands, buttons, sliders, etc. In addition, the decision making of these devices could be performed by a simple software. Despite of that, this interaction could be enhanced and made more natural to the user by using Cognitive Computing services and the decision making of the IoT devices could be improved by a more complicated software using Machine Learning. Both Cognitive Computing and Machine Learning are described in the ensuing sections.

2.2. Machine Learning

According to [25, p. 3], Machine Learning is defined as:

“Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data

or past experiences. The model can be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both.”

Specifically, as described in [25, p. 4], ML uses computer algorithms which are provided with data as an input to train themselves and build an ML model. This model contains the entire knowledge it has learned from the previous training. Then, the ML model can be used to understand further the existing data and extract new information or predict future outcomes by passing new data as an input. Machine Learning can be distinguished into three main learning techniques:

1. Supervised Learning,
2. Unsupervised Learning and
3. Reinforcement Learning.

Each learning technique has a different approach on how a computer can be trained, depending on the imported data and the use case to be implemented, along with specific algorithms that have been developed for each one. However, the primary focus is on Supervised techniques since the solution to be built by the end of this report is classified as such (see chapter 3). Moreover, Time Series Forecasting is also taken under consideration, since it can be used as a Supervised technique.

2.2.1. Supervised Learning

According to [26, p. 3], Supervised Learning requires a dataset that can be separated into two sets, training and testing. The training set includes examples of input data along with their target values. If the target values of the training set are numeric, then a Regression Supervised algorithm is used. Otherwise, if the target values are nominal, then Classification is used. Based on the training set, a Supervised ML algorithm tries to find correlation between data and build a model. Following that, the output of the trained model can be compared to the testing set and evaluate its performance. Regression and Classification are some of the categories to be discussed next along with their ML algorithms. However, it should be noted that more ML algorithms and variations of them might exist that could fall under these categories and are not researched because of the limited timeframe.

Regression

The Regression technique tries to predict a numeric value (output – dependent variable) based on the inputs (independent variables) which are going to be provided to the model [25, p. 9]. The dataset that is going to be imported into a Regression model can have different attributes [25, p. 9]. Based on these attributes, the model predicts a numeric value which is later evaluated compared to the desired value specified beforehand [25, p. 9]. Such Regression algorithms are:

- **Linear Regression** tries to correlate the dependent value (output) to the independent values (inputs) based on a linear model [26, Ch. 3]. It actually attempts to fit all the available data on a straight line and it can generate great results for data that can be illustrated as such. A variation of Linear Regression is the **Bayesian Linear Regression** [26, Ch. 3], which uses prior probability distribution as additional information for calculating the output.

- **Decision Trees** [27, Ch. 18] is a graph represented as a tree made out of **leaves** and **nodes**. Each node represents a rule on which the input data are evaluated and based on that, the end result (output) is reached. The output is represented as a leaf. The Decision Trees can consume a lot of memory and it takes a lot of time for training their models, but they can be very accurate [28, p. 526]. Some variations of Decision Trees specifically for regression techniques are:
 - The **Boosted Decision Tree Regression** [28, p. 525] which constructs a chain of trees, each learning from the errors of the previous one, and it can solve the overfitting problem.
 - The **Decision Forest Regression** [28, p. 525] also avoids the overfitting by constructing a large set of trees, each uncorrelated to the other. The average of all trees (the forest) produces one tree which is used as the model for predictions.
 - The **Fast Forest Quantile Regression** [28, p. 525] is useful when the quintiles of the predictive value are important for better understanding its distribution.
 Other than Regression, Decision Trees can also be used in Classification which is discussed later [26, Ch. 14].
- **Neural Networks** [27, Ch. 19] is an area of Machine Learning including algorithms inspired by the human brain and it consist of numerous **units**. Each unit is connected to each other through **links** and each link has a **weight** representing the data storage of the whole network. A unit can act either as an input or as an output and it is doing all the computations. Specifically for regression, the **Neural Network Regression** [28, p. 527] algorithm can be used.

Classification

Classification tries to find correlation between data and separate them into classes. The output of a Classification algorithm is nominal values that represent a class [25, p. 5]. Following the training of a Classification algorithm, the trained model should be able to receive an input and classify it into one of the desired classes, specified in advance [25, p. 5]. ML algorithms that support Classification are:

- **Logistic Regression** [26, Ch. 4] is an algorithm which tries to predict the output based on the probability of a certain set of inputs fitting to specific classes (output). This can be accomplished by fitting all input data into a logistic function, an S-shaped curve.
- **Decision Trees**, as it was discussed earlier, can be also used in classification [26, Ch. 14]. The Classification Decision Trees function the same way as in Regression, however, the result of their output is now a nominal value instead of a numeric one and the goal is to categorize the set of inputs under a specific class (output). In Classification, **Boosted Decision Tree** [28, p. 525] and **Decision Forest** [28, p. 525] algorithms can be used, as well as the **Decision Jungle** [28, p. 525], which is a variation of the Decision Forest algorithm and it can save memory but has the drawback of requiring more training time.

- **Neural Networks**, as they were described previously (units connected to each other through weighted links), they can also be applied to classification problems. Like Decision Trees, their output is also nominal instead of numeric and their purpose is to classify specific inputs to a specific output (class). Particularly for Classification, the **Averaged Perceptron** [28, p. 528] algorithm can be used. Despite of being a primitive version of Neural Networks, it is way faster in terms of training time, an advantage that could be of use for smaller, less complex problems.
- **Naive Bayes** uses the Bayes' rule which computes unknown probabilities of an event happening based on known conditions [27, Ch. 14]. The Naive Bayes model assumes that the features or attributes of the input data are independent to each other [26, Ch. 8]. This independency between the attributes might not be the case in the real world, that is why is called naive, but it can generate great results especially for data with large dimensionality [26, Ch. 8]. Based on the Bayes' rule, the **Bayes Point Machine** [28, p. 529] algorithm has been developed, which calculates in approximation the theoretical ideal average of linear classifiers.
- **Support Vector Machine (SVM)** calculates the boundaries between two or more classes by using a straight line with the margins between the lines and the data points to be as wide as possible from the closest data point [26, Ch. 7]. A variation of SVM is the **Locally Deep SVM** [28, p. 529] which is a nonlinear classifier. The way it functions is by using several smaller linear SVM methods and it is ideal when the linear SVM cannot produce enough accurate results.

2.2.2. Time Series Forecasting

The previous techniques, Regression and Classification, were developed based on the assumption of the input data being independent, however, in many situations, this might not be the case [26, Ch. 13]. For instance, in order to predict the weather, specific geographical location and the time of the day are required. Thus, making the input data dependent on time. For these reasons, when the input data are dependent and especially on time, Time Series Forecasting techniques have been developed.

Nevertheless, before start using a Time Series Forecasting algorithm, first the input data need to be analyzed. During the time series analysis of the data, certain methods can be used to decompose them into different components which can help identify patterns in the available dataset [29, Ch. 6]. These components are Seasonality, Trend, Cyclical and Error. **Seasonal** patterns occur when the data are affected by seasonal factors (e.g. yearly, monthly, daily, etc.) [29, Ch. 6], whilst **Trend** is linear or nonlinear long-term increase or decrease of the data over time and can be characterized as upward or downward [29, Ch. 6]. **Cyclical** pattern happens when the input data show escalations and drops that are not of a fixed period over a long period of time [29, Ch. 6]. **Error**, or the irregularity, is the remaining component after the decomposition procedure which contains anything else in the time series data [29, Ch. 6]. Such decomposition, as described above, can be either **Additive** or **Multiplicative**, meaning that all these components can be added or multiplied

together to form the original time series dataset [29, Ch. 6]. As it is documented in [29, Ch. 6], addition is preferable if the variation of the Trend pattern or the magnitude of the Seasonal pattern does not vary with the level of the time series. On the other hand, multiplication is more suitable if the variation of Trend or Seasonal patterns is proportional to the level of the time series dataset.

Some decomposition methods that can be used to identify the aforementioned patterns are the Moving Average and the Classical decomposition, but more sophisticated ones are the X-12-ARIMA and the STL [29, Ch. 6]. The **X-12-ARIMA** decomposition, as mentioned in [29, Ch. 6], is based on the classical decomposition method and it provides robust results for unusual observations. It can also handle additive and multiplicative decomposition. The drawback of X-12-ARIMA is that only handles data which are collected monthly or quarterly. On the contrary, the **Seasonal and Trend using Loess (STL)** decomposition works with a variety of seasonal patterns, not restricted to monthly or quarterly [29, Ch. 6], and supports seasonal pattern adjustments over long periods of time [29, Ch. 6]. However, it only works with additive decomposition and does not handle calendar variations automatically [29, Ch. 6]. Even though STL supports only additive decomposition, multiplicative can be implemented by using logarithmic calculations [29, Ch. 6].

Although there are many Time Series Forecasting algorithms that can be used following the decomposition techniques mentioned above, the most popular ones are the Exponential Smoothing and ARIMA. These are discussed next.

- **Exponential Smoothing** [29, Ch. 7] uses the average of past observations to make a forecast. These observations are weighted and the most recent ones have more weight than older ones. To achieve that, the **Error Trend Seasonality (ETS)** [29, Ch. 7] technique is utilized, where the components extracted from an earlier decomposition can be used additive or multiplicative. It should be stated that it is not necessary to use all the components, this highly depends on the input data. For instance, if a specific dataset does not show seasonality patterns, then the seasonality component can be left out. Thus, for each component of the ETS it can be assigned None, Additive or Multiplicative methods separately. An example of how this can be presented is the ETS (N, M, A), in which N stands for no error component, M for exponential trend where multiplication is used and A for linear seasonality in which addition is necessary. Other examples of ETS representations are the ETS(M,A,N), ETS(A,A,M), ETS(M,N,N) and so forth, each time depending on the decomposed patterns.
- **AutoRegressive Integrated Moving Average (ARIMA)** [29, Ch. 8] is made out of three different parts, the AutoRegressive (AR), the Integrated (I) and the Moving Average (MA), each focusing in different aspects of the ARIMA model. The **AR** component is responsible for calculating forecasts based on past observations, it looks like a linear regression and it is represented as $AR(p)$, where p is the term indicating the number of past observations to be used. The **MA** component uses forecast errors of the past, in a model which resembles a linear regression and is represented as $MA(q)$, where q is the number of previous periods of errors to be used. Last but not least, the **Integrated** component is represented as $I(d)$, which d refers to the number of transformations needed for a non-stationary time series to

become stationary. It is important for a time series to be transformed into a stationary, if initially it is not, because it will provide further information about the data and the correlation to each other. An **AutoCorrelation Function (ACF)** and a **Partial AutoCorrelation Function (PACF)** can be used for this purpose, to extract additional information, and following their inspection, the number of p and q terms to be chosen can be decided for the AR and MA models. Putting everything together, an ARIMA model can be presented as $ARIMA(p,d,q)$ and depending on the previous analysis, the p , d and q terms can be equal to different values. Examples of that are $ARIMA(1,1,0)$, $ARIMA(0,1,0)$, etc. It should be noted that the previous discussed $ARIMA(p,d,q)$ model is relevant to a time series which does not have any seasonal patterns. To take into consideration the seasonality, then extra terms are needed. These terms are the capital P,D,Q and low case m , resulting to $ARIMA(p,d,q)(P,D,Q)_m$. The P,D,Q terms represent the seasonal aspects of the AR, Integrated and MA components and the m refers to the number of periods in each season.

Implementing any of the aforementioned ML algorithms and finding the best parameters for each one, highly depends on the type of data. This process can be very complicated, time consuming, power intensive and memory demanding. However, several tools have been developed which can automate these processes, among them are some Cloud tools that are described later, section 2.4. These tools can be used to help automate the implementation of specific ML algorithms. Moreover, they provide the advantage of using remote memory and computational power. Then again, before going into the Cloud tools, the methods of how ML algorithms can be evaluated based on their accuracy are discussed next.

2.2.3. Evaluation

Having presented a few Machine Learning techniques and algorithms in the previous subsection, a question rises on how specific ones should be chosen in favor of others, as there is not a single algorithm which outperforms all others. To compare and choose an ML algorithm, certain techniques have been developed to evaluate each one and their forecast accuracy. These techniques are listed below. It should be noted that the following methods are focused on evaluating supervised ML algorithms.

- **Mean Absolute Error (MAE)** [29, Ch. 2] is a scaled-dependent error technique. It takes the average of the absolute difference between the predicted values and the original observations with all differences having the same weight. The result is dependent on the scale of the selected dataset and it cannot be compared between errors of different scales.
- **Root Mean Squared Error (RMSE)** [29, Ch. 2] is also a scaled-dependent error technique and it should be used to compare algorithms on a dataset of the same scale, like MAE. It is the square root of the average of squared differences between the predicted values and the original observations. RMSE gives a higher weight to large errors because each difference is squared before all differences are averaged. This is useful in predictions where large errors are unwanted.

- **Mean Absolute Percentage Error (MAPE)** [29, Ch. 2] as the name suggests, is a percentage error technique. It takes the average of the absolute difference between the original observations and the forecasted values divided by the original observations and all of that multiplied by 100. This makes a percentage error technique and it can be used to compare data which are on different scales. However, it can give values such as zero or infinite if the observation dataset includes values equal to zero or close to zero.
- **Mean Absolute Scaled Error (MASE)** [29, Ch. 2] is a technique independent from the scale of data, like MAPE, and it falls under the category of scaled error techniques. It is a more complicated technique that requires more calculations to be performed. This method is proposed as an alternative method to MAPE as it does not suffer from zero or infinite values.

For all techniques mentioned above, the lowest values indicate the most accurate ML algorithm, or the algorithm with less errors [29, Ch. 2]. Specifically for MASE, values lower than one indicate a better forecasting in comparison to an average naïve forecast, whilst values greater than one show a worse performance [29, Ch. 2].

2.3. Cognitive Computing

Following the discussion about Machine Learning, its techniques and algorithms, it can be seen the huge potential it has about solving difficult, real-life problems and enable new technologies as well. One of the technologies based on ML is Cognitive Computing.

Cognitive Computing was developed in order to provide a computer, a robot, or any electronic device with the appropriate knowledge around a specific operating field [30]. As stated in [9], *“Cognitive systems are probabilistic, meaning they are designed to adapt and make sense of the complexity and unpredictability of unstructured information. They can “read” text, “see” images and “hear” natural speech. And they interpret that information, organize it and offer explanations of what it means, along with the rationale for their conclusions. They do not offer definitive answers. In fact, they do not “know” the answer. Rather, they are designed to weigh information and ideas from multiple sources, to reason, and then offer hypotheses for consideration. A cognitive system assigns a confidence level to each potential insight or answer.”* [9, p. 6].

Following that, the first challenge for a Cognitive system would be to understand natural language and its semantics. To achieve that, more sophisticated ML algorithms are needed. According to [31, p. 1], traditional ML algorithms produce great results in numerical data, but they are not suitable for processing and understanding natural language where semantics play a crucial part in comprehending a sentence, a paragraph or a document [31, p. 2]. Because of this problem, in [31] a more sophisticated ML algorithm was presented which could understand the context of a sentence.

Nevertheless, not further research is conducted in such algorithms in the rest of this project, since there are already implementations of them by large, well-established companies (e.g. Amazon,

Google, Microsoft, etc.) and some of them are used later in chapters 3 and 4. Their provided Cognitive services can be used and accessed by Application Programming Interfaces (APIs) through their Cloud Platform (see section 2.4). Furthermore, before a Cognitive Computing system can understand the context of a sentence, first it needs to access this sentence. For this purpose, technologies like Speech-to-Text, Text-to-Speech, Computer Vision, Text Analytics, and many more are used. The Cognitive services provided over the Cloud are discussed in the ensuing section 2.4.

In conclusion, depending on the used Cognitive service, a device might need to perform all the aforementioned tasks which could be proved to be computational intense. However, by utilizing the Cloud, any device would be able to support and deliver Cognitive services to the end users. The Cloud and what services are provided is discussed next.

2.4. Cloud Platforms

As discussed earlier, the Cloud can be used by IoT devices in different solutions, from executing the simplest tasks to the more complicated ones. Although there are many platforms that support Cloud computing, only a few are researched as stated in chapter 1, section 1.3 - Limitations. However, before diving into the description of the platforms, a short introduction to the Cloud will follow. The purpose is to get an overview of how the Cloud can be used and which are the services that it can offer.

A Cloud Platform can be distinguished into three types of services [32, p. 13]. These are:

1. **Infrastructure-as-a-Service (IaaS):** As the name indicates, IaaS provides all the underlying infrastructure for software and applications to be built upon. Specifically, IaaS provides the necessary servers and databases where data can be stored. It also provides network security, including firewalls, and physical machines which host all the above. All that are illustrated in Figure 3 [33].
2. **Platform-as-a-Service (PaaS):** PaaS includes all the IaaS components but also development tools which can be used to build software, applications and/or services, see Figure 3 [33].
3. **Software-as-a-Service (SaaS):** SaaS provides already developed software or applications which can be used by users in their everyday life and they are hosted and executed in the Cloud, see Figure 3 [33].

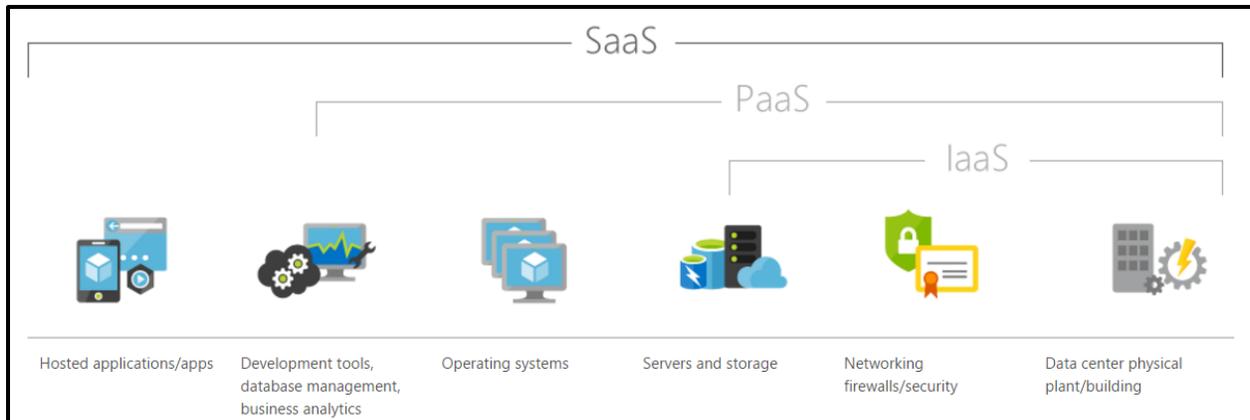


Figure 3: Cloud Platform.

All these Cloud services can be used separately or in a combination, depending on the solution a developer wants to offer. Following that, three different Cloud Platforms are researched, Microsoft Azure, IBM Bluemix and Amazon Web Services, and the services they provide. However, the research is limited on the Cloud services which are of interest of this report and are important for fulfilling the second objective about the Cloud in correlation to IoT, ML and Cognitive Computing (chapter 1, section 1.3).

2.4.1. Microsoft Azure

Microsoft Azure is a Cloud Platform which can offer many different tools and combines all the aforementioned services (IaaS, PaaS, SaaS) that developers can utilize [34]. Nevertheless, the most interesting one for this project is the PaaS part of it and Microsoft Azure provides plenty of tools under this service.

More precisely, a whole platform about IoT applications can be used [35], providing connectivity between many heterogeneous IoT devices through an IoT Hub [36, p. 8]. The functionalities of the IoT hub are explained in [36, p. 8]. First, each device needs to register itself to the hub and a unique identifier is given to it. By using this identifier, an IoT device can be authenticated and authorized to access the hub. Following that, the IoT Hub is responsible for sending, receiving and redirecting messages between the devices. Apart from the individual devices, a central component could be built which would act as the central unit for controlling all these devices. This central unit could be implemented in a microprocessor/microcomputer like a Raspberry Pi (see subsection 2.1.3) or even a smartphone. Additionally, different kinds of databases (SQL or NoSQL) can be used to store necessary information [36, p. 584] and plenty communication protocols (like the AMQP, see section 2.1.2) [36, p. 9] can be implemented for resource-rich and resource-constrained devices (see subsection 2.1.1).

Furthermore, Azure offers another platform called Machine Learning Studio [37], which can be used in conjunction with the rest of the services, e.g. access a database and retrieve the stored data [28, p. 25]. In this studio, the retrieved data can be analyzed before feed them in the available ML algorithms and train specific models. Azure implements plenty ML algorithms for supervised and unsupervised learning alike [28, p. 520] and it also includes Time Series Forecasting techniques

[28, p. 103]. Having trained a model, Azure Machine Learning Studio gives the capability to make it available to the internet through a web service and accessible by using an API key [28, p. 73]. This way, applications could send relevant data to this model and receive predictions based on these data. It should be noted that the Machine Learning Studio is very flexible and it can provide several models to be accessed under the same web service [28, p. 604], through a Request-Response Service (RRS) call [28, p. 25] or a Batch Execution Service (BES) [28, p. 25]. The difference is that the RRS can make one call at a time resulting to real-time predictions, while BES can make multiple calls at the same time to the web service which could take some time to calculate the predictions. Furthermore, because of the deep integration the studio has with the rest of the Cloud Platform, a deployed model can be easily adjusted or retrained [28, p. 664], if needed, and new data can be used for training, since a connection can be established with Azure but also non-Azure databases [28, p. 25].

Other services Azure can offer are the cognitive ones. Microsoft Cognitive Services [38] is another platform provided by Azure, where specific services can be access by API keys. These services include language understanding, speech recognition, computer vision, knowledge exploration, etc. [39]. A particularly interesting one is the Language Understanding Intelligent Service (LUIS) [39, p. 73], [40] that can be used to build conversational intelligence into any software application. By using LUIS, a service can be provided to users through a device, like a computer or a smartphone, which can understand language semantics. Thus, any user would be able to communicate with his/her device in natural language [39, p. 73]. The information given to LUIS is in text form, i.e. a sentence. Then, LUIS analyzes the provided sentence and it can identify its context. Subsequently, LUIS can respond to the user in text form, based on his/her initial request, and provide an answer [39, p. 73]. Going a step beyond that, by implementing speech recognition to the same device, i.e. using the Bing Speech API service, the user-LUIS interaction can take place entirely on voice alone [39, p. 28]. The aforementioned process describes an intelligent bot. Essentially, a bot is a software agent which can communicate with users and provide services to them without the presence of a human [41, Ch. 2].

There are several ways someone can access all these platforms and based on them different restrictions may be applied. The most common one is to start using Azure for the first month as a free trial and then continue with a free account or as a Pay-As-You-Go subscription [42]. If a developer uses a free account, he/she will have to face some restrictions on the provided services. For instance, using a free database has restrictions in terms of database type, as well as, storage capacity [43]. For Azure, the freely accessed database is an SQL one with 32 MB of storage. Another example is the IoT Hub which restricts the number of devices to be connected, up to 1000 [36, p. 261], and the amount of messages to be exchanged per day, 8,000 [44]. The Machine Learning Studio also comes with its own restrictions, if it is used with a free account. For example, only Request-Response Service (RRS) calls [45] can be used. As long as the Cognitive Computing platform is concerned, LUIS service is limited to 10,000 transactions per month [46] and the speech recognition service to 5,000 calls per month [47].

The same model (one-month trial, free account, pay-as-you-go) is followed by other Cloud Platform providers as well, like the ones to be discussed next, but with different kind of restrictions. All the Cloud Platforms are summarized and compared towards the end of this section on Table 3:

Cloud Platforms – IoT., Table 4: Cloud Platforms – Machine Learning. and Table 5: Cloud Platforms – Cognitive Computing.

2.4.2. IBM Bluemix

IBM's Bluemix platform, like Microsoft's Azure, offers several services. Among them there are an IoT platform, Cognitive Computing Services and Machine Learning. The IoT platform works the same way as the one described in Azure and it supports many IoT devices. As documented in [48], the connected devices can communicate with each other by exchanging messages and they can be accessed and controlled by a central unit. The central unit can be built on the Cloud, by using IBM's Message Hub, or to a Bluemix application implemented in a microprocessor/microcontroller, as the ones discussed in subsection 2.1.3 (e.g. Arduino). The communication between the devices can take place by using certain communication protocols, which some of them were described in section 2.1.2 (e.g. MQTT) and the data exchanged can be stored in a NoSQL database.

Additional services are the Cognitive Computing such as speech/vision recognition, language understanding, big data analysis and so forth. These can be accessed through Watson, by using API calls, and makes it easy for any type of devices to be enhanced with additional intelligence. Watson is IBM's intelligent service that can be used for creating an intelligent bot [49], similar to Microsoft's LUIS. Watson can interact with a user and understand the context of the conversation [49]. It can also be further enhanced with speech recognition, Watson Speech-to-Text [50] and Text-to-Speech [51], for a more realistic interaction.

Apart from the aforementioned services, IBM Bluemix offers Machine Learning services as well, although its implementation is quite different from Microsoft's Azure. As documented in [52], the data analysis and the training of a Machine Learning model must happen offline by using a specific software made by IBM, the IBM SPSS, which can retrieve a dataset as input from many different SQL and NoSQL databases [53]. Then, the trained model can be exported and uploaded to the Machine Learning service in the Cloud. From this point forward, the model can be accessed the same way as in Azure, by a set of REST API keys. Using these keys, applications could ask the model for predictions based on given data in real-time, but also by providing a batch of data and then waiting for their forecasts. A disadvantage of this method is that there is not a seamless integration of the ML service with the rest of the Cloud tools. This drawback can restrict the functionalities on some applications. Moreover, once a model has been deployed to the Cloud, there is not an easy way of retraining it, unless if it was designed to be retrained before uploading it [52]. On the other hand, the IBM SPSS software is widely used and a great variety of ML algorithms are provided, including Time Series Forecasting [54]. The greatest advantage is that it can automatically test several ML algorithms at the same time and in the end, suggest the one performed the best [54].

Like Microsoft Azure, IBM Bluemix is also accessible freely and fully functional for one month, but following that, some restrictions may be applied. These restrictions can be deducted if a Pay-As-You-Go subscription is used. For instance, if a free account is used, the IoT Platform is limited to support only 500 registered devices [48] and the messages exchanged between them are

constrained in 100 Megabytes per month [55]. Also, the free database offered for storing data from the IoT platform is a NoSQL limited to 1 GB of storage [56]. Moreover, the intelligent bot by Watson can only make 1,000 transactions per month [57] and the speech recognition is restricted to the first 1,000 minutes for Speech-to-Text [58] and first 1,000,000 characters for Text-to-Speech [59]. Regarding the Machine Learning platform, there is a limited number of ML models to be uploaded in the Bluemix platform [52] and the SPSS software used for training the models is not freely accessible after a 30-days trial [60]. A comparison of Bluemix and the other platforms can be seen in Table 3, Table 4 and Table 5.

2.4.3. Amazon Web Services (AWS)

Similar to the Cloud Platforms described earlier, Amazon Web Services (AWS) offer plenty and great Cloud tools such as an IoT platform, Machine Learning and Cognitive Computing. The AWS IoT platform and the Cognitive services work almost identical to the previous discussed platforms, Microsoft's and IBM's. For the IoT platform [61], again different heterogeneous devices can be registered and connected to the Cloud and exchange messages between them. The whole communication is handled by a Device Gateway [61, p. 1] located in AWS and the communication channels use specific protocols like MQTT [61, p. 1] (see subsection 2.1.2). The data coming from the different devices can be easily stored in SQL and NoSQL databases [62].

Moreover, AWS supports Cognitive services and intelligent bots can be implemented by using Amazon Lex [63] (utilizing Amazon's language understanding service), similar to IBM's Watson and Microsoft's LUIS. On top of that, Amazon Polly [64] can be used to enhanced Lex with speech recognition capabilities.

Not limited to those services, AWS offers a Machine Learning platform as well, which unfortunately is limited compared to the ones provided by the other two discussed platforms (Azure and Bluemix) in terms of algorithms variety. Amazon Machine Learning [65] is restricted to only three ML algorithms, which are all focused on supervised techniques [65, p. 15]. These algorithms are the Linear Regression for Regression problems and Logistic Regression and Multinomial Logistic Regression for Classification problems [65, p. 15]. However, AWS Machine Learning is much simpler to use with step-by-step guides and automatic selection of the most suitable ML algorithm [65, p. 1]. Amazon specific SQL and NoSQL databases can be used to import the initial data for training a model [65, p. 2] and applications can make real-time & batch requests [65, p. 34] by using specific API calls [66], after the model has been trained and deployed.

Finally, AWS can be accessed freely with some restrictions applied to its services for 12 months and then a Pay-As-You-Go subscription is required [67]. Restrictions to the IoT platform include 250,000 messages per month [62], which is a time-limited offer and then additional charges are applied. Following that, Amazon Lex is restricted to 10,000 text requests per month and another 5,000 speech requests per month, which again will be decreased if the trial expires. One more important limitation is that Amazon Machine Learning is only available under a paid account [62].

2.4.4. Differences between Cloud Platforms

To get an overview of the capabilities of each platform, a comparison was set with each other based on the earlier discussion. The following tables (Table 3, Table 4 and Table 5) were created in order to summarize all the relevant Cloud tools and the applied restrictions on their free services. Considering the proposed solution briefly described in chapter 1, the Cloud services each platform provides are more important for designing such a solution than the restrictions applied under a free account. However, the limitations of each free Cloud service become relevant when the prototype needs to be implemented, since a free account is intended to be used throughout the entire project.

The following table, Table 3, compares the previously discussed Cloud Platforms and specifically their IoT Platform in terms of:

- how IoT devices can be connected (**Cloud Gateway**),
- the maximum number of devices each platform can support under a free account (**Connected Devices**),
- the total number of messages to be exchanged per day for a free account (**Total Number of Messages per Day**),
- the supported communication protocols between devices (**Communication Protocols**) and
- the database each platform can utilize to store data from the IoT devices (**Database - Storage**), again by using a free access to the service.

Characteristics	Microsoft Azure	IBM Bluemix	Amazon Web Services
Cloud Gateway	IoT Hub	Message Hub	Device Gateway
Connected Devices	1,000 ¹	500 ¹	Not specified
Total Number of Messages per Day	8,000 ¹ per Day	3,333 ¹ per Day ²	8,333 ¹ per Day ³
Communication Protocols	AMQP / MQTT / HTTP	MQTT / HTTP	MQTT / HTTP
Database – Storage	SQL – 32 MB ¹ NoSQL	NoSQL - 1 GB ¹	SQL - 5 GB ⁴ / NoSQL - 25 GB ⁴

Table 3: Cloud Platforms – IoT.

¹ Free service. Can be increased if a paid account is activated.

² The information given by IBM is 100 Megabytes per month [55], which is around 100,000 Kilobytes per month. IBM also specifies that each message is 1 Kilobyte [55], this results in 100,000 messages per month, divided by 30 days it results in 3,333 messages per day.

³ The actual information given by Amazon is 250,000 messages per month, divided by 30 days makes it 8,333 messages per day. It should be noted that this is a limited time offer of 12-months [62] and then additional charges are applied.

⁴ Trial service (1 year). Can be increased if a paid account is activated, but also decreased after the trial expires.

From Table 3, it can be observed that, all three platforms offer similar functionalities. Moreover, Microsoft Azure should be selected if the solution to be implemented requires the AMQP communication protocol. However, IBM Bluemix should be avoided if the solution requires a SQL database instead of a NoSQL one. Furthermore, if a free account is used and the storage capacity is of importance, Azure will be limiting the prototyping process. On the other hand, if the database is not a crucial component of the solution, AWS or Azure are the better choice. Of course, whether to choose one platform over another also depends on the Machine Learning and Cognitive Computing services each one offers. These are described next.

The ensuing table, Table 4, compares each platform according to their Machine Learning capabilities and they are based on the following parameters:

- how Machine Learning functionalities can be accessed (**Platform**),
- what kind and how many ML algorithms are provided (**Machine Learning Algorithms**),
- if time series forecast techniques can be used (**Time Series Forecasting**),
- how a trained ML model can be accessed by other applications and services (**Web Service**),
- how many trained ML models can be provided under a single web service and for a free account (**Models per Web Service**),
- if already deployed ML models can be retrained (**Retrain Deployed Models**) and
- what kind of databases can be used to make a dataset available to each ML platform (**Supported Databases to Import Data**).

From that table, a meaningful distinction between the three platforms can be achieved by observing three parameters. The first parameter is the **Machine Learning Algorithms** each one provides. While Supervised Learning is offered by all platforms, AWS can implement only three. Alternatively, Bluemix and Azure provide a large variety of them. In addition to that, Unsupervised Learning algorithms are only available to Azure and Bluemix. On top of that, only Bluemix offers Reinforcement Learning algorithms. The next crucial parameter in the **Time Series Forecasting** algorithms, which AWS does not implement. Last but not least, the **Models per Web Service** is another important parameter, as it indicates the number of ML models that can be deployed as a web service. This number is not specified for AWS. However, Azure can have multiple ML models, whilst Bluemix only two. Whether to choose one over another highly depends on the needs of the solution to be designed. Nevertheless, if a free account is used, Azure should be selected as it has fewer restrictions.

Characteristics	Microsoft Azure	IBM Bluemix	Amazon Web Services
Platform	Azure Machine Learning Studio	Watson Machine Learning & IBM SPSS ⁵	Amazon Machine Learning ⁶
Machine Learning Algorithms	Supervised / Unsupervised (Multiple)	Supervised / Unsupervised / Reinforcement ⁵ (Multiple)	Supervised ⁶ (Restricted to 3)
Time Series Forecasting	Yes	Yes ⁵	No
Web Service	REST APIs (Real-time & Batch Requests ⁷)	REST APIs (Real-time & Batch Requests)	APIs ⁶ (Real-time & Batch Requests)
Deploy as a Web Service	Automatic	Manually	Automatic
Models per Web Service	Multiple	2 ¹	Not specified
Retrain Deployed Models	Yes	Only by design ⁵	Not specified
Supported Databases to Import Data	SQL / NoSQL Not restricted to Azure	SQL ⁵ / NoSQL ⁵ Not restricted to Bluemix	SQL / NoSQL Amazon only

Table 4: Cloud Platforms – Machine Learning.

At Table 5 below, the Cognitive services are compared. The focus is on:

- the available bots each platform has (**Intelligent Bot**),
- how many transactions a bot supports under a free account (**Intelligent Bot Transactions per Month**) and
- if speech recognition is supported (**Speech Recognition**).

⁵ Trial service (30 days).

⁶ Paid service.

⁷ Batch requests are only available with a paid account.

Characteristics	Microsoft Azure	IBM Bluemix	Amazon Web Services
Natural Language Understanding (Intelligent Bot)	LUIS	Watson Conversation	Amazon Lex
Intelligent Bot Transactions per Month	10,000 ¹ / Month	1000 ¹ / Month	10,000 ⁴ / Month (Text requests) 5,000 ⁴ / Month (Speech requests)
Speech-to-Text (Speech Recognition)	Bing Speech API	Watson Speech-to-Text	Amazon Lex
Text-to-Speech (Speech Recognition)	Bing Speech API	Watson Text-to-Speech	Amazon Polly

Table 5: Cloud Platforms – Cognitive Computing.

From the above table, it can be observed that the only distinction between the platforms is the amount of transaction per month allowed for free. Despite that, all three implement an Intelligent Bot that can understand the context of a conversation. Furthermore, they also provide services which can convert speech to text and vice versa.

Finally, it can be concluded that all Cloud Platforms offer more or less the same services, but there are differences on their implementation. These variations can be the main focus of whether one platform should be chosen over another, depending on the solution to be built by a developer along with his/her preferences and budget. Moving away from a Cloud-centric discussion, the next section is dedicated to Smart Home solutions and how IoT, Machine Learning, Cognitive Computing and the Cloud can shape them.

2.5. Smart Home Solutions

During this section, Smart Home solutions are discussed which are currently available on the market. However, an investigation is conducted as well on what researchers are currently working on to make homes even smarter. By the end of this section, there will be a clear understanding on how IoT, ML and Cognitive Computing are used in a smart home environment. Thus, answering the third objective - *An investigation is necessary to be conducted on how IoT, ML and Cognitive Computing technologies are currently used in a Smart Home environment.* (chapter 1, section 1.3) and finalize the study on the first research sub-question (chapter 1, section 1.3):

What is the current State-of-the-Art in Smart Homes considering the IoT, ML and Cognitive Computing?

The rest of this section includes commercial and research solutions for a Smart Home environment and how the Cloud, Cognitive services, Machine Learning and the Internet of Things are utilized. It should be noted that more solutions exist in the market and proposed by researchers, but not all of them can be documented here. As a result, only a few were selected which represent the rest, as they share similar functionalities, or provide a couple of unique services.

2.5.1. Commercial Solutions

In the following pages, selected solutions which are available to the market are documented. First, examples of *Complete Solutions* are examined. These offer an integrated Smart Home experience to their users and comprise of a smartphone application, a variety of IoT devices and a Hub. Secondly, *Individual IoT Devices* which are not part of Complete Solutions are described. These are devices capable of operating by themselves without the need of a central Hub. Finally, *Complementary Solutions* are documented. Devices included in such solutions are capable of controlling other IoT devices through Cognitive services. Such devices can be used to enhance the Complete Solutions with natural language understanding capabilities. On top of that, they can function as a bridge between the Complete Solutions and the Individual IoT Devices to create a more versatile Smart Home experience.

Complete Solutions

As discussed earlier, Complete Solutions are designed to offer a complete Smart Home experience by providing all the necessary components:

- a variety of IoT devices,
- a central Hub, where all devices are connected to, and
- a smartphone application, from which all device can be controlled.

Some examples of such solutions are the Samsung SmartThings [68], Bosch Smart Home [69] and Honeywell [70]. It should be noted that many more corporations exist that offer a complete Smart Home solution. However, since the core functionalities of each are quite similar and the time is not enough to document every solution, only Samsung SmartThings is examined next. The purpose for that is to gain a clear understanding on the services a Complete Solution can offer.

Samsung SmartThings is a Smart Home solution provided by Samsung [68] and it is composed of a central unit, a variety of IoT devices and a smartphone application [71]. The type of IoT devices Samsung offers are resource-constrained devices (see section 2.1.1), which are forced to be connected to an extra device for internet access [72]. This extra device is the central unit called Samsung SmartThings Hub [73], which is responsible for connecting all devices to the internet and allow communication from and to the smartphone application. However, the SmartThings ecosystem is highly integrated and products from other manufacturers can be used. These products include resource-constrained and resource-rich devices (see section 2.1.1) [72]. The SmartThings Hub connects to the smartphone application and to the internet through Wi-Fi and it communicates

with the IoT devices with Z-Wave, ZigBee and/or Wi-Fi [73]. An illustration of the basic components can be seen in Figure 4 [71] below.

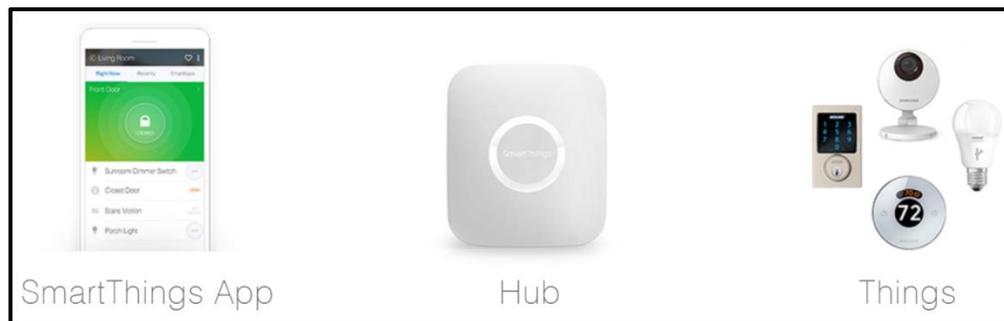


Figure 4: Samsung SmartThings.

By using the smartphone application, the user is capable of controlling each device separately. However, the user can also set predefined scenes/routines which, when activated, can control multiple devices at once [74]. A routine is a scenario which can happen during the user's everyday life. For example, an "I am back" scene lets the Smart Home know that the user is back home, so it should unlock the door, heat-up the living-room and turn on the lights. The devices each routine controls are defined by the user [74]. A routine either can be activated manually by the user or automatically by the time of the day, the geographical location of the user or a motion sensor [74].

IoT devices available from Samsung and also other manufacturers include motion sensors, water leak sensors, smoke sensors, outlets, light bulbs, speakers, thermostats, cameras, door locks, voice assistant devices and many more [72]. Moreover, repeater devices are also available which can extend the range of the SmartThings Hub for Z-Wave and ZigBee enabled devices [75].

Individual IoT Devices

These devices can operate by themselves, without the need of being part of a Complete Solution. Such devices are the Nest Thermostat [76], or the Philips Hue lighting system [77]. There are other companies that provide Individual IoT Devices as well, e.g. Withings [78]. Nevertheless, since the primary focus here is to understand how these devices operate, only one is documented next. Furthermore, as Nest Thermostat is one of the few devices offering Machine Learning capabilities, it is the one to be discussed.

The Nest Thermostat [76] falls under the category of the resource-rich devices which can be connected directly to the internet and communicate with other IoT devices. This device is responsible for heating up or cooling down the temperature of a specific room in a house [76]. The Nest Thermostat can be seen in Figure 5 [79].



Figure 5: Nest Thermostat.

The traditional ways of adjusting the temperature are to use directly the Nest Thermostat or the companion smartphone application [76], nevertheless, Google Home can be used to control the Nest Thermostat entirely by voice [80]. What makes this device special is being one of the few which utilizes Machine Learning algorithms and Cloud Computing services in order to figure out the daily schedule of the user and adjust the room temperature accordingly [81]. The disadvantage is that one device can only control the temperature for one room and more devices are required to cover the whole house. On the other hand, by having one Nest Thermostat to each room, it can adapt to it and to the people's preferences using that room, so a more personalized experience can be provided [82].

Complementary Solutions

These solutions comprise of devices which can function either as a bridge between the Complete Solutions and the Individual IoT Devices, or to enhance the Complete Solutions with natural language understanding capabilities. Examples of that are the Google Home [5] and Amazon Echo, or Echo Dot [6]. Even though more devices like the aforementioned ones might exist in the market, the core functionalities remain the same for each, to perform tasks through voice alone. Among these tasks are the control of Individual IoT devices, or devices included in Complete Solutions. For that reason, only one of them is described next, as the emphasis is on how a device like this functions and what services it provides.

Google Home is a voice assistant device powered by the Google Assistant, which among other functionalities, can be used to control heterogeneous devices installed in a house [5]. What makes it unique is that users can talk to this device and it can understand the context of what users are saying. This simplifies the process of controlling different devices, as no specific commands need to be remembered by the users and given to the device. This is possible because the Google Assistant [8] utilizes the power of the Cloud (Cognitive services, see section 2.3) to analyze voice commands. Once the device has received a command, it can execute it or ask the user for further clarifications by voice. Google Home is illustrated in Figure 6 [5] below.



Figure 6: Google Home.

Google Home is a perfect example of an IoT resource-rich device (see subsection 2.1.1) that can be connected to the internet directly and harness the power of the Cloud. Utilized Cloud services are the Cognitive ones, such as natural language understanding, speech recognition and knowledge exploration (see section 2.3).

2.5.2. Research Solutions

Up until now, commercial products have been researched about Smart Homes, some focusing in the overall Smart Home environment, e.g. Samsung, and others on how the Smart Home experience can be enhanced by using services like Cognitive Computing and Machine Learning, Google Home and Nest Thermostat respectively. Particularly, the latter is an area of interest for many researches as well and many publications emphasize on how the Smart Home experience can be improved.

In [83], many solutions are documented on how the habits of a user can be detected in the environment of a house by using data mining techniques and unsupervised ML algorithms. Following that, the authors of [83] proposed a solution based on fuzzy mining to model user routines in a house environment filled with many different types of IoT devices, e.g. light bulbs, actuators, thermostats, etc. The habits were detected by the traces a user leaves after interacting with a specific device [83]. By the end of [83], it was concluded that four main set of habits were detected with high confidence, these correspond to a morning routine, relax, washing dishes, and cooking/eating. However, further research is needed before providing services to the end user based on this solution.

In another published research, [84], a user behavior prediction solution for Smart Homes was developed by using ML supervised techniques and a variation of the Neural Network algorithm, the Backpropagation Neural Network (BPNN). The combination of BPNN with parallel programming in the Cloud produced amazingly good results on predicting user habits based on input data from the current status of heterogeneous IoT devices. Yet, not a clear description was provided on how the produced predictions will affect the home devices and if an autonomous operating house is possible.

Last but not least, in [4] an Improved Hidden Markov Model (IHMM) was used along with historical data to predict the behavior of people with disabilities in a Smart Home environment. Specifically, in this research, the time variant was considered as it can play a crucial part on the state of the home devices in the IHMM. Still, in this research, an answer was not given on how the predictions from the IHMM could be utilized from a Smart Home solution in a larger scale.

2.6. Remarks

In this chapter, several aspects of Smart Homes were researched, starting from the area of the Internet of Things, continuing to Machine Learning, Cognitive Computing and the available Cloud Platforms and their capabilities. In the end, it became more transparent how all these technologies are used in commercial available products and realized their potential through further research. Thus, accomplishing the first three objectives:

1. *Knowledge about IoT, Machine Learning and Cognitive Computing must be gained.*
2. *Research is required about other technologies such as the Cloud and how it can be used in relation to IoT, ML and Cognitive Computing.*
3. *An investigation is necessary to be conducted on how IoT, ML and Cognitive Computing technologies are currently used in a Smart Home environment.*

stated in section 1.3 - Problem Formulation and answering the first research sub-question:

What is the current State-of-the-Art in Smart Homes considering the IoT, ML and Cognitive Computing?

With the knowledge acquired so far, the rest research sub-questions can be answered, starting with the second one in the next chapter.

3. Analysis

This chapter sets the starting point on answering the second research sub-question stated in chapter 1 – section 1.3, which focuses on:

How a new solution can be designed for Smart Homes to achieve a self-operated house?

The chapter starts by proposing the Smart Home solution to be designed in this project. Then, it is compared to current solutions, documented in chapter 2, in order to comprehend how the proposed solution is different and what are its advantages and disadvantages. Moreover, the components of the proposed solution are identified in relation to technologies already discussed in chapter 2. Following that, a few scenarios are described, where the proposed solution can be used. These scenarios will help to extract use cases, which are later visualized in a Use Case Diagram. Finally, from the use cases, requirements specifications are extracted, functional and nonfunctional. These are used in chapter 4 to design the proposed solution and implement a prototype.

3.1. Proposed Solution

As briefly mentioned in chapter 1 – section 1.2, the proposed solution consists of three main components, as they were illustrated in Figure 1 at the same section. The first component is the IoT devices installed in a house, the second is a smartphone application accessing and controlling these devices and the third and final component is the Cloud, which acts as a bridge between the first two components and also provides further functionalities.

Going into more depth, the home devices include any device that provides a specific service/functionality to the house and it can be connected to the Cloud through a direct internet connection. Examples of these IoT devices are thermostats, light bulbs, door/window locks, boilers, coffee machines, fridges, TVs, etc. The Cloud provides a link between each IoT device and the smartphone application from where the user can see the current status of each device and change it according to his/her preferences. Apart from the IoT device – to – smartphone application connection, the Cloud provides speech recognition and natural language understanding capabilities to the application's user as well, with the purpose of enhancing the interaction experience between the user and the application.

In the Cloud, the status of each device, based on date and time, is stored for later use by the Machine Learning service. Having gathered a respectful amount of historic data from the user – home devices interaction, the ML service hosted in the Cloud can now train a model and use that to forecast the status of each home device for a specified time period. The amount of historic data to be used is later discussed in chapter 4, along with the specified time period for which the service should forecast values. Following that, the ML model can be deployed as a web service and accessed from the smartphone application. Then, the application can provide the ML model with a specific date and time and receive the predictive value of each IoT device. Finally, the smartphone application can set the status of each home device to the received forecasted values without user intervention. The name of this proposed solution is **Machine Learning-as-a-Service (MLaaS)** and a context diagram illustrating it can be seen in Figure 7 below.

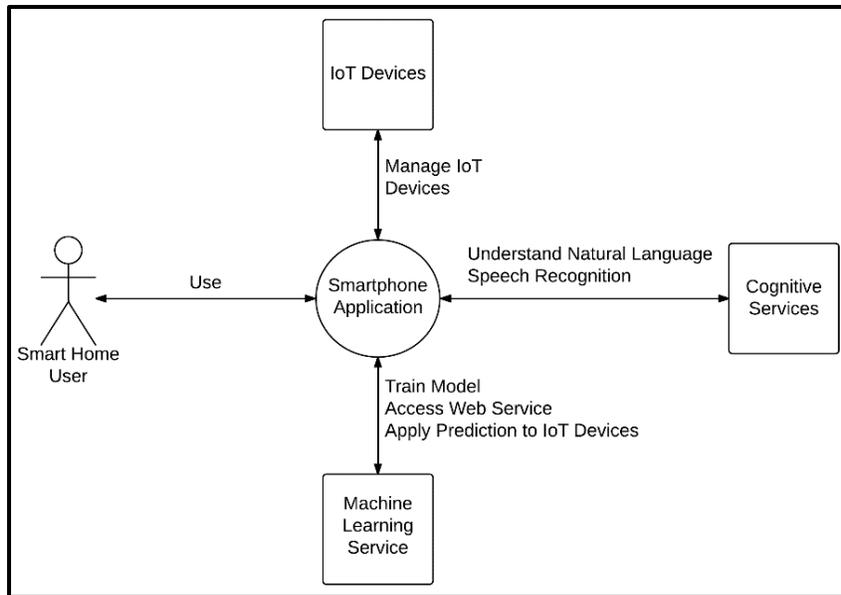


Figure 7: MLaaS – Context Diagram.

Figure 7 shows the use of the Cloud by the smartphone application and how Machine Learning and Cognitive services are exploited. Additionally, it illustrates the accessibility of the IoT devices to the application. However, an entity is missing from this diagram which is the user. To include the user, an extra figure is shown next. Figure 8 below illustrates a context diagram from the smartphone application perspective. There, the process of how a Smart Home user can access all the Cloud services from the smartphone application and control the IoT devices becomes visible.

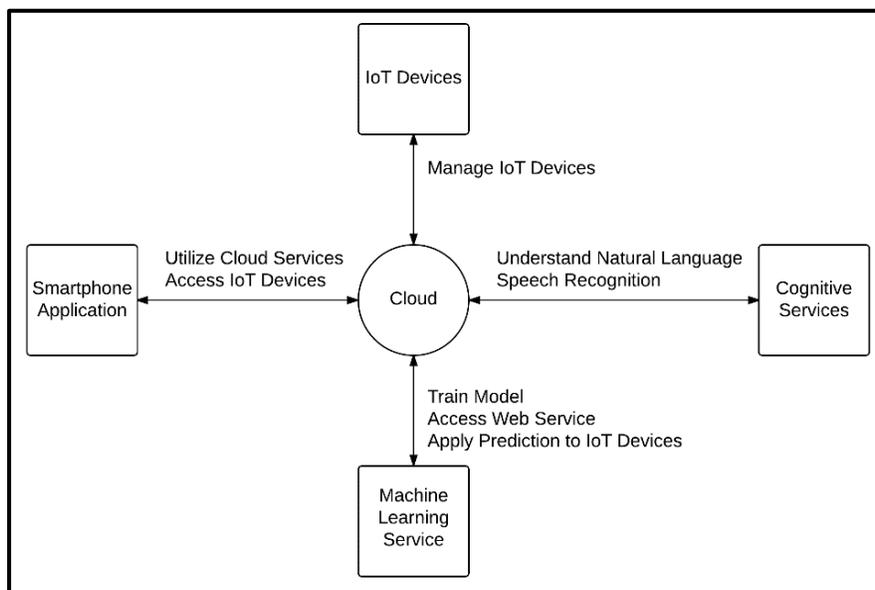


Figure 8: MLaaS - Context Diagram 2.

Machine Learning Service

Particularly for the ML Service, it is intended to use historic data for training an ML model in order to forecast future values of home devices based on the date and time, as also it was discussed earlier. In chapter 2 – section 2.2, three different Machine Learning techniques were described and an increased focus was given to the Supervised Learning techniques, since they are ideal for training models based on past values. Among them, Time Series Forecasting algorithms were described, like ARIMA and ETS, which are specifically used when the input data are dependent on time. As a result, the focus is to use these algorithms for designing and implementing the proposed solution (MLaaS).

The approach of using Time Series Forecasting algorithms for predicting values in a Smart Home environment has not been tried so far, to the best of this author's knowledge, and it is a different method compared to the more traditional ones seen by commercial products and researchers (section 2.5). Nevertheless, results from these types of algorithms are going to be compared to results from traditional Supervised Machine Learning algorithms, like Regression (e.g. Linear Regression, Decision Trees, etc.) and Classification (e.g. SVM, Naive Bayes), in order to determine if indeed Time Series Forecasting algorithms are the better choice. The comparison of the three different Supervised ML algorithms can be seen in chapter 4, section 4.3.

3.1.1. Differences with related work

Comparing the proposed solution to the existing ones described in chapter 2 – section 2.5, it can be observed that they share similarities but also some significant differences. Considering the commercial available products, Samsung SmartThings [68] ecosystem provides a variety of IoT products with different connectivity capabilities (Wi-Fi, ZigBee, Z-Wave) which can be controlled from Samsung's own smartphone application. However, the traditional user – application interaction is a downside which Google Home [5] can cover. By using a device like Google Home, a more user friendly experience can be provided because it allows its users to control the Samsung SmartThings [72] by natural language understanding voice commands. Still, a disadvantage remains as both of them need user input for controlling the home devices. Based on the research conducted on this report and at the best of this author's knowledge, this issue has not been resolved yet and it is not offered by any service or product in the market. The only exception is the Nest Thermostat [76], which utilizes Machine Learning to understand the daily schedule of a user and control the temperature of a single room in a house without user input. While Nest Thermostat is a product aiming towards home automation, it is limited in functionality and house region (controls only the temperature and for one room).

The solution in home automation based on user habits may come from researchers working with Machine Learning and how it can be used to detect user behavior. Researches in [4], [83] and [84] (discussed in chapter 2 – section 2.5) prove that identifying, detecting and predicting user habits is possible and some of them even provide high accuracy. However, none of them discuss how their solution can be deployed as a service or provided as a product to end-users in a large-scale. Authors in [4] did not discuss how the IHMM predictions can reach IoT devices in a house and how they can be utilized. In [83], it was concluded that further research is needed in order to

provide fuzzy mining predictions to end users. Last but not least, [84] did not discuss how the BPNN with parallel programming predictions can be used in conjunction to home devices and if an autonomous operating house is possible.

The results from the discussion above indicate that commercial services and products focus more on controlling several IoT devices in a house, whilst researchers' main concern is how to detect user habits and predict them. The proposed solution of this project fills this gap by combining attributes from both fields, commercial and research solutions, and builds on top of that to provide a unified-personalized experience to Smart Home users. How the proposed solution functions in different scenarios is discussed in section 3.2.

Diving into further details about Samsung SmartThings specifically, some technical benefits and drawbacks can be identified compared to the proposed solution. As it was described in chapter 2 – subsection 2.5.1, Samsung SmartThings supports many communication technologies, whether it is Wi-Fi, ZigBee or Z-Wave. This has the advantage of supporting a large variety of Smart Home devices which are already in the market. However, sometimes the SmartThings Hub is not capable of covering the entire house (depending on the size of the house, the floors and the room distance) and some devices may be out of range from the Hub. That is why repeater devices can be used [75], to make out of range devices reachable. Then again, this creates another problem. If the unreachable devices are many and scattered in a house, several repeaters are necessary to be used. The problem escalates even more if a room away from the SmartThings Hub has installed devices operating in all three communication technologies, then three different repeaters are needed. The same problem exists in other solutions similar to Samsung SmartThings.

To overcome this complexity, the proposed solution in this project supports only one communication technology. Since Wi-Fi is the one most used by users to connect devices like computers, laptops, smartphones and tablets to the internet, it was chosen as the one implemented in this solution. However, it can be argued if this is the correct choice in terms of energy efficient since many IoT devices are limited in battery capacity. On the other hand, and as discussed in chapter 2 – subsection 2.1.2, the 802.11ah standard could be used, especially for IoT devices, which is part of the Wi-Fi family and at the same time more energy efficient than other members of Wi-Fi. Still, further research is needed on which communication standard is the best choice since there are many which can provide better energy efficiency (e.g. IEEE 802.15.4). However, this research is beyond the scope of this project, as it was also stated in chapter 1 – section 1.3 – Limitations.

Since the communication technology used (Wi-Fi) provides direct internet connection to IoT devices, only resource-rich devices (see chapter 2 – subsection 2.1.1) can be considered in this solution. This gives the flexibility of not implementing a hardware hub, so all home devices are connected to the internet without an intermediate (hub) but only the Wi-Fi router. The benefit is that no extra devices are needed to be installed by the user, which brings the complexity level down. On the other hand, by not having a hub or a central unit in the house which can manage all home devices, the system would not function without internet connectivity, unlike Samsung SmartThings which can provide basic functionalities without internet because of the hub device [85]. This could be a major drawback since without internet the user would not be able of unlocking

the door of his/her house, or turning on the lights. This is an issue which must be addressed, however it is not in the scope of this project as it is assumed that there is always internet connectivity, see chapter 1 – section 1.3 – Limitations.

3.1.2. Security and Privacy

Having several IoT devices communicating with the Cloud and a smartphone application able to access and control these devices through the internet, raises security and privacy concerns. In [86], several Smart Home devices were tested and many vulnerabilities were detected. For instance, the popular light bulb series Philips Hue do not encrypt the data exchange between the smartphone application and the hub. This can lead in eavesdroppers to snoop in and check if lights are on or off. Thus, they will be able to determine human presence, or even control the light bulbs of an entire house. Another example is the Withings Smart Baby Monitor, which is vulnerable to “man-in-the-middle” attacks and the camera can be exploited by hackers. By doing so, hackers can invade the privacy of its users. These result in the necessity of strong authentication and authorization process, in order to make sure that only the owners of the house have access to the home devices. On top of that, a strong encryption is needed to ensure that people who eavesdrop the communication channel cannot compromise the security and privacy of people’s lives.

An interesting solution to these problems was given in [86], where a Cloud-based network-level security and privacy control for home devices was described. This is beneficial since it can be applied across multiple heterogeneous IoT devices, independent from manufacturer and communication technology. However, no further investigation is conducted during this report, as the proposed solution does not consider security and privacy to its design and implementation, due to time limitations (see chapter 1 – section 1.3). Yet, some suggestions are presented in chapter 5, section 5.2.

3.2. Scenarios & Use Cases

There are two basic scenarios for which the proposed solution is beneficial. The first scenario focuses on how a user can access and control IoT devices installed in his/her house and the second one emphasizes on how the proposed solution can be trained, predict and control the home devices without user intervention.

3.2.1. Scenario 1 – Controlling Home Devices

Before describing the scenario, first some assumptions need to be stated. Firstly, it is presumed that the user’s smartphone has internet connectivity. Secondly, the user has downloaded the application to his/her smartphone, he/she has created an account, authenticated/authorized to gain access to it, created a virtual house with floors and rooms and registered IoT devices to his/her account. Then, as mentioned in chapter 1 – section 1.3 – Limitations, it is presumed that all IoT devices are already installed in the house, on top of that, internet connectivity is present and each home device is linked to a specific room.

Having said that, the user can now see the current status of each home device per room. So, if the house has three rooms, the application displays these rooms, the devices installed in each one and the status of each device. Following that, the user is able of speaking or writing to the application, in natural language, the command which sets the device status. For instance, the user can speak or write “Turn on the heat in the bedroom at 23 degrees”. The application then checks if there is a device capable of controlling the heat in the specified room, e.g. a thermostat, and if yes sets the value to 23 degrees Celsius. Subsequently, the thermostat is responsible of setting the heat up to 23. If there is not a device capable of controlling the heat in that room, then the application informs the user. The same example applies to other devices such as light bulbs, locks, etc. It should be noted that if the user does not give a full command, specifying action, appliance, room and value, the application should ask the user to clarify them. For instance, if the user gives a command like “Turn on the heat” the application should ask the user to clarify room and desired temperature.

3.2.2. Scenario 2 – Machine Learning as a Service

As the user continues interacting with the home devices, their status is stored in a database to the Cloud, along with the date and time of each status change. After a certain amount of days (see chapter 4) the Machine Learning service should train itself on the historic data and produce a model capable of forecasting the status of each device based on the date and time. The model should be available to be accessed through the Cloud by the smartphone application, which could request a prediction of the status of each home device (the house status) in real time. Thereafter, the application could use the predictions to change the status of the home devices. This request and alteration of the house status should be automatic and repetitive in order to avoid user intervention and achieve home automation.

3.2.3. Use Cases

Following the explanation of the scenarios, use cases are derived from them which define the actions of the MLaaS system. Figure 9 illustrates the use cases in a diagram, where three actors are present, the Smart Home user, the Machine Learning service and the Cognitive Computing service. All these actors interact with each other through the use cases. It should be noted that the color of some use cases is different than others. The ones colored blue are a category of use cases which are not designed nor implemented through the course of this project because of time restrictions. However, they are important and that is why they are included in the use case diagram. The green colored category use cases are the ones designed and implemented in this project. Nevertheless, a short description for each use case follows the diagram. Furthermore, the use cases will provide necessary information in order to extract the requirements of the MLaaS implementation.

An important thing to be considered is the Smart Home user actor who is illustrated as one type user. However, in real life this might not be the case and more than one users could live in the same house managing the home devices. This can result in different types of users with different authorization access to only specific devices inside the Smart Home. While this is possible, it is not considered as such in this project, as also stated in chapter 1 – section 1.3 – Limitations.

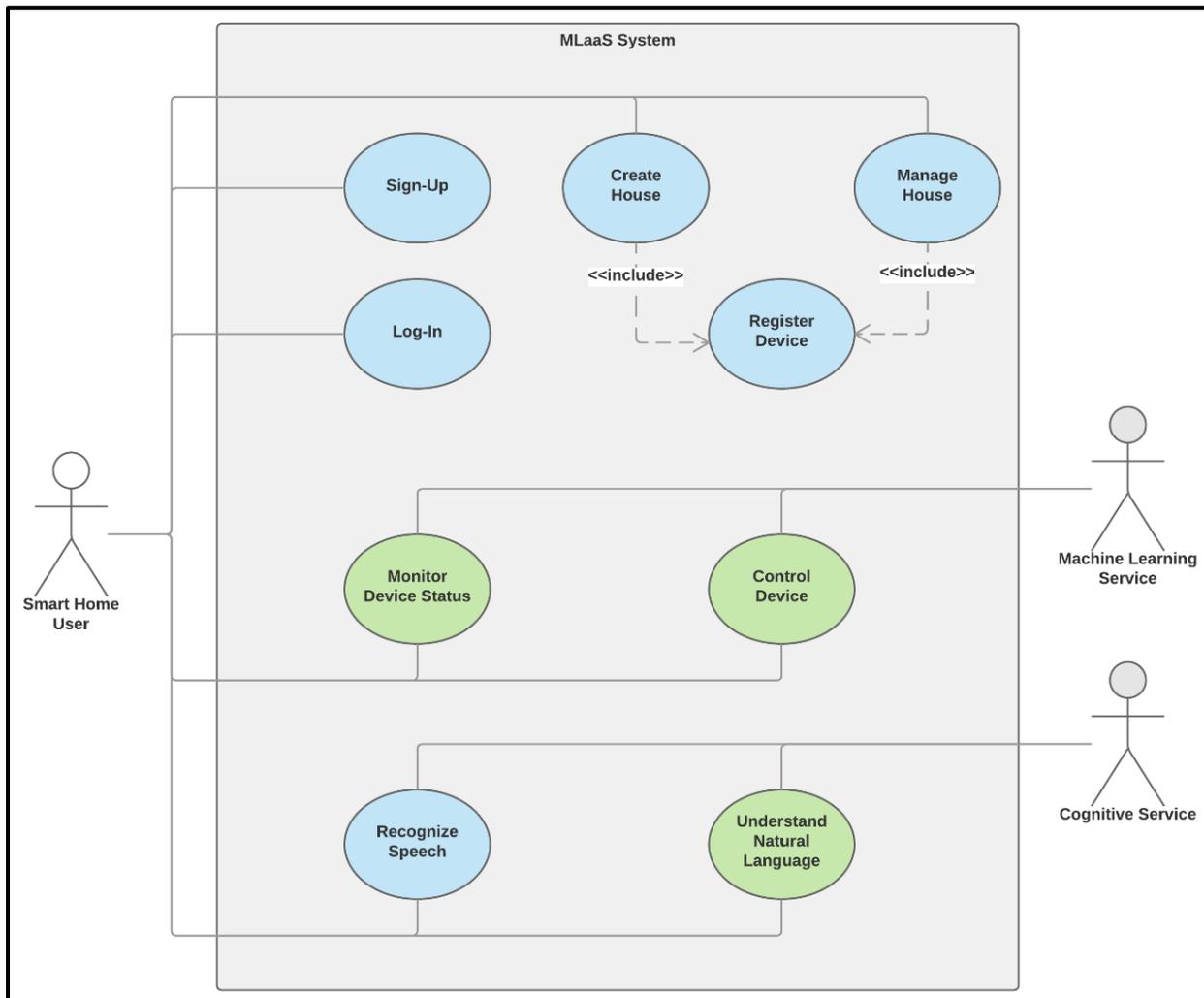


Figure 9: MLaaS – Use Case Diagram.

Sign-Up

This use case is responsible for registering a user to the system. By using the smartphone application, the user needs to create an account with the required information of an e-mail address and a password. The user must also give permission to the application to collect data, like the status of each IoT device installed in the house, in order to provide more personalized services, like the real-time prediction of the status of each home device.

Log-In

By providing an e-mail and a password, the user must be authenticated to the system and authorized. Following that, the user is now able of accessing and controlling the home devices.

Create House

In this use case, the user is able to create a virtual home representing his/her own. This can happen by creating individual floors and rooms. The user can also provide a name for a specific floor or

room. If no floors or rooms are created by the user, the application creates one floor and one room with default names (Floor 1, Room 1).

Register Device

The user must be able to link an IoT device to his/her account, in order to declare ownership and that he/she can only access and control it. Then, the user must link the IoT device to a specific room through the smartphone application.

Manage House

The user should be able of adjusting the house floors and rooms, so he/she can change their name, delete some of them or create new ones. The user should also be able of unlink an IoT device from one room and link it to another. Doing this, a home device can be moved from one room to another. Moreover, the user should have the ability of unlinking an IoT device from his/her account. This is useful when the user wants to replace a specific IoT device with a new one.

Monitor Device Status

The user must be able to see the current status of each device installed in the house per room. However, the status of each device must be stored (e.g. in a database) along with date and time information in order to be used by the Machine Learning service and train a model.

Control Device

The user must be able to change the status of all IoT devices inside a Smart Home. Furthermore, the ML service must be able to change the status of the home devices as well, following the training of the ML model based on previous stored data.

Recognize Speech

The smartphone application should be able to recognize speech. This means that the application should hear speech and convert it to text and receive text and convert it to speech. This is useful because the user could give a command in voice, this command can be converted to text which can be further analyzed by the Understand Natural Language use case. Once the analysis is completed, the smartphone application receives an answer in text and converts to speech, so that the user can hear it.

Understand Natural Language

The smartphone application must be able to understand natural language so it can communicate with the user in a more natural way. For example, the user could give a command to the application like “Turn on the lights.” by text or voice. Then, the application must understand that the user wants to turn on an appliance inside the house and this appliance is the lights. However, not specific room was given so it asks the user: “To which room?”. Following that, the user must speak or text a room name like “For the kitchen.”. Finally, the application knows the room and responds: “Turning on the lights to the kitchen!”.

3.3. Requirements Specification

Specifying the requirements of the MLaaS solution can provide valuable information on how the system should work, what are its functionalities and how the end-product should be designed [87]. These requirements can be divided into functional and nonfunctional.

Functional Requirements

The functional requirements can be used to describe the capabilities of the system, as well as, its performed functionalities. These requirements can be documented either as user-friendly/general or as developer oriented/detailed [87]. The functional requirements of MLaaS are derived from the discussion of the previous section (section 3.2) about the scenarios and use cases, and are displayed in Table 6 in a general form, with further details included when it is necessary.

Non-Functional Requirements

The nonfunctional requirements describe the constraints of the system on the services and functionalities it provides [87]. The MLaaS nonfunctional requirements are presented at Table 7.

Both, functional and nonfunctional requirements are prioritized based on the MoSCoW [88] (Must Should Could Will not) technique. The ones indicated as “Must” (M) are the focus of this project and specifically of chapter 4, where their design and implementation is described. The requirements marked as “Should” (S) or “Could” (C), while important for the system are beyond the scope of this project.

Functional Requirements				
Use Case	Nº	Name	Description	Priority
Sign-Up	F1	Register e-mail	The user should provide a valid e-mail address for authentication purpose.	S
	F2	Register password	The user should provide a strong password for authentication purpose.	S
	F3	Ask permission	The user should be asked to accept or reject the collection of data. These data are related to the status of home devices which can help train a Machine Learning model to be used later for real-time predictions.	S
Log-In	F4	Authenticate and authorize user	The user should be authenticated and authorized to access the system.	S
Create House	F5	Create floor	The user should create a floor and give a name to it. If no floors are created by the user, the application creates one floor with a standard name of 1 (Floor 1).	S
	F6	Create room	The user should create a room and give a name to it. If no rooms are created by the user, the application creates one room with a standard name of 1 (Room 1).	S
Register Device	F7	Link device to user account	The user should link an IoT device to his/her account to declare ownership and restrict access to his/her account only.	S
	F8	Link device to room	The user should link an IoT device to a specific room so that the system would be aware of which room's appliance this device controls.	S
Manage House	F9	Adjust floor settings	The user could be able of renaming a floor, delete it or create a new one.	C
	F10	Adjust room settings	The user could be able of renaming a room, delete it or create a new one.	C
	F11	Adjust device settings	The user could be able to unlink an IoT device from a specific room and link it to another. Also, the user could be able of unlinking a specific device from his/her account.	C
Monitor Device Status	F12	Display current device status	The user must be able to see the current status of each device installed in the house per room in his/her smartphone application.	M
	F13	Save past device status	The status of each device must be stored in a database along with date and time information.	M
Control Device	F14	Change device status from user	The user must be able to change the status of all IoT devices installed in his/her house.	M
	F15	Change device status from ML service	The Machine Learning service must be able to change the status of the home devices following the requested predictions from the smartphone application based on the date and time.	M
Recognize Speech	F16	Convert speech to text	The smartphone application should receive voice commands from the user and convert them to text for better analysis.	S
	F17	Convert text to speech	The smartphone application should convert text to speech, so it can "speak" to the user and pronounce different results.	S
Understand Natural Language	F18	Understand natural language	The smartphone application must understand natural language so it can provide a natural conversation with the user about controlling the IoT devices.	M

Table 6: Functional Requirements.

Non-Functional Requirements			
N _o	Name	Description	Priority
NF1	Smartphone Application Platform	The smartphone application must be available to all mobile platforms (Android, iOS and Windows 10 Mobile).	M
NF2	Password Strength	The password to be chosen by the user should include numbers, letters and symbols without a common sequence (e.g. 1234 or QWERTY) [89].	S
NF3	Internet Access	Internet access is vital for the whole system to be functional and must always be present.	M
NF4	Smartphone Communication Technology	Communication technologies like Wi-Fi (IEEE 802.11a/b/g/n/ac) and/or a cellular network (e.g. LTE) must be used to allow data exchange between the smartphone application, the Cloud and the IoT devices.	M
NF5	IoT Hub	A Cloud-based hub must be present where all IoT devices are connected to, can be accessed and controlled.	M
NF6	IoT Devices Communication Technology	An IoT friendly and IP-based communication technology like Wi-Fi (IEEE 802.11ah) must be used for the IoT devices to communicate with the Cloud (IoT Hub) and the smartphone application.	M
NF7	Communication Protocol	An IoT friendly communication protocol must be used to allow data exchange between home devices, the Cloud and the smartphone application (e.g. AMQP or MQTT).	M
NF8	Secure Communication	Data exchange between IoT devices, the smartphone application and the Cloud should be encrypted at all times to ensure high confidentiality, integrity and availability of personal data [90].	S
NF9	Machine Learning Algorithms	Regression, Classification and Time Series Forecasting algorithms must be implemented and tested to choose the one which performs best.	M

Table 7: Non-Functional Requirements.

3.4. Remarks

The focus of this chapter was the proposed solution, what technologies it utilizes from the ones documented in chapter 2 and how it is different from other solutions described in section 2.5. On top of that, some concerns were raised and briefly discussed about the security of the system and the privacy of its users. Following that, two scenarios were examined and how the proposed solution fits on these, resulting in extracting several use cases. With the help of the use cases, functional and nonfunctional requirements of the system were derived. Thus, fulfilling this

project's fourth objective (chapter 1 – section 1.3): *Specification requirements are essential for the design of the proposed solution and the implementation of the proof of concept.*

Based on the analysis which took place in this chapter and with the help of the requirements specification (see Table 6 and Table 7), it is now possible to start designing and implementing the proposed solution (MLaaS) and answer the second research sub-question (chapter 1 – section 1.3).

4. Design and Implementation

An initial discussion about answering the second research sub-question took place in chapter 3 and it is going to be finalized in this chapter. Therefore, the focus here is on designing the solution proposed in chapter 3 and complete the second research sub-question:

How a new solution can be designed for Smart Homes to achieve a self-operated house?

Concurrently, an implementation of a proof-of-concept is documented, which is the answer to the third and final research sub-question:

How a proof of concept can be developed based on this new solution for Smart Homes?

The starting point of this chapter is the design choices made to create a complete system architecture for a full-scale solution, according to the functional and nonfunctional requirements from the previous chapter. Following that, an implemented architecture is designed as well, since there are some restrictions on the technologies used. These restrictions are discussed throughout the Implementation and Experimental Assessment sections. The implemented architecture is used to develop a prototype which is the proof that the proposed solution is feasible. Finally, after the proof-of-concept development documentation, a testing phase of the prototype follows.

4.1. Design Choices

According to the previous chapter and the nonfunctional requirements described in Table 7, there are some which constitute the core of the proposed solution and must be considered during the design and implementation of MLaaS. These are:

- **NF1: Smartphone Application Platform**, which defines that the smartphone application must be available to all mobile platforms (Android, iOS and Windows 10 Mobile). However, the constrained time-frame does not allow to develop the application for all platforms and only a Windows 10 Mobile application is considered at this point. Another reason for choosing the Windows 10 Mobile platform is that a smartphone with this Operating System (OS) is available during the development process.
- **NF3: Internet Access**, which describes that access to the internet must always be present, for the system to function correctly. This can be ensured by implementing communication technologies for the smartphone application and the IoT devices.
- **NF4: Smartphone Communication Technology**. Wi-Fi (IEEE 802.11a/b/g/n/ac) and/or a cellular network (e.g. LTE) must be present in order to provide internet connectivity to the smartphone application and communicate with the Cloud and IoT devices.
- **NF5: IoT Hub**. All IoT devices must be connected to a central point from where they can be accessed by the smartphone application and controlled. According to the proposed solution (see chapter 3 – section 3.1), this hub is a Cloud-based hub and must be implemented in the Cloud. In chapter 2 – section 2.4, three different Cloud Platforms were

researched and as it is summarized in Table 3: Cloud Platforms – IoT., all of them can support a Cloud-based hub. Looking at the same table, some differentiations can be seen between the three platforms but they are not critical at this point since they all offer the same functionalities in different price ranges.

- **NF6: IoT Devices Communication Technology.** This nonfunctional requirement necessitates an IP-based communication technology which is applicable to IoT solutions. An example is the IEEE 802.11ah (part of the Wi-Fi family), as also discussed in chapter 2 – section 2.1. This technology will establish a communication channel between the IoT devices, the Cloud and the smartphone application.
- **NF7: Communication Protocol.** The protocol used for the communication between the IoT devices, the Cloud (IoT Hub) and the smartphone application must be one of the following: CoAP, MQTT, AMQP or DDS since they are IoT-friendly (chapter 2 – section 2.1). However, according to Table 3 (chapter 2 – section 2.4) the available protocols from the researched Cloud Platforms are AMQP, MQTT and HTTP, which restricts the choices to either MQTT (Azure, Bluemix, AWS) or AMQP (Azure).
- **NF9: Machine Learning Algorithms.** The selected Cloud Platform must provide Regression, Classification and Time Series Forecasting algorithms in order to test and compare them. Following that comparison, the best performing algorithm will be chosen for deployment. This requirement limits the available choices of the researched Cloud Platforms (chapter 2 – section 2.4), since AWS does not support Time Series Forecasting algorithms according to Table 4: Cloud Platforms – Machine Learning. Out of the two remaining ones, Microsoft Azure and IBM Bluemix, and as they were described in chapter 2 – section 2.4, Azure provides a more seamless interaction between training an ML model and deploying it as a web service, since no extra software is required and everything can be performed in the Cloud. This can provide the advantage of automating the entire process. Unlike Bluemix, which requires a specific software to be downloaded and installed locally. For these reasons, Microsoft Azure is chosen as the Cloud Platform to be utilized in the design and implementation of MLaaS.

In conclusion, MLaaS comprises from a Windows 10 Mobile smartphone application, the Microsoft Azure Cloud Platform and several IoT devices. Based on these choices, the next subsection is focused on the system architecture and the interactions between each entity.

4.1.1. System Architecture

In Figure 10 the complete system architecture is illustrated, based on the aforementioned design choices. Nevertheless, due to some restrictions in the Azure Cloud Platform and its free services, the actual implementation of the architecture is slightly different (see Figure 11). Both architectures, Complete Architecture and Implemented Architecture, are discussed next. The complete architecture describes the full-scale solution, which can be developed when there are not restrictions to the available resources (Microsoft Azure paid account). Whilst the implemented

architecture focuses on the prototype developed to prove the concept of the proposed solution (MLaaS).

Complete Architecture

Figure 10 below shows the complete system architecture which includes the IoT devices, the smartphone application, the Cloud and its entities and the interactions between them. Each interaction is represented using a different color and details about them can be seen in the bottom right of the image.

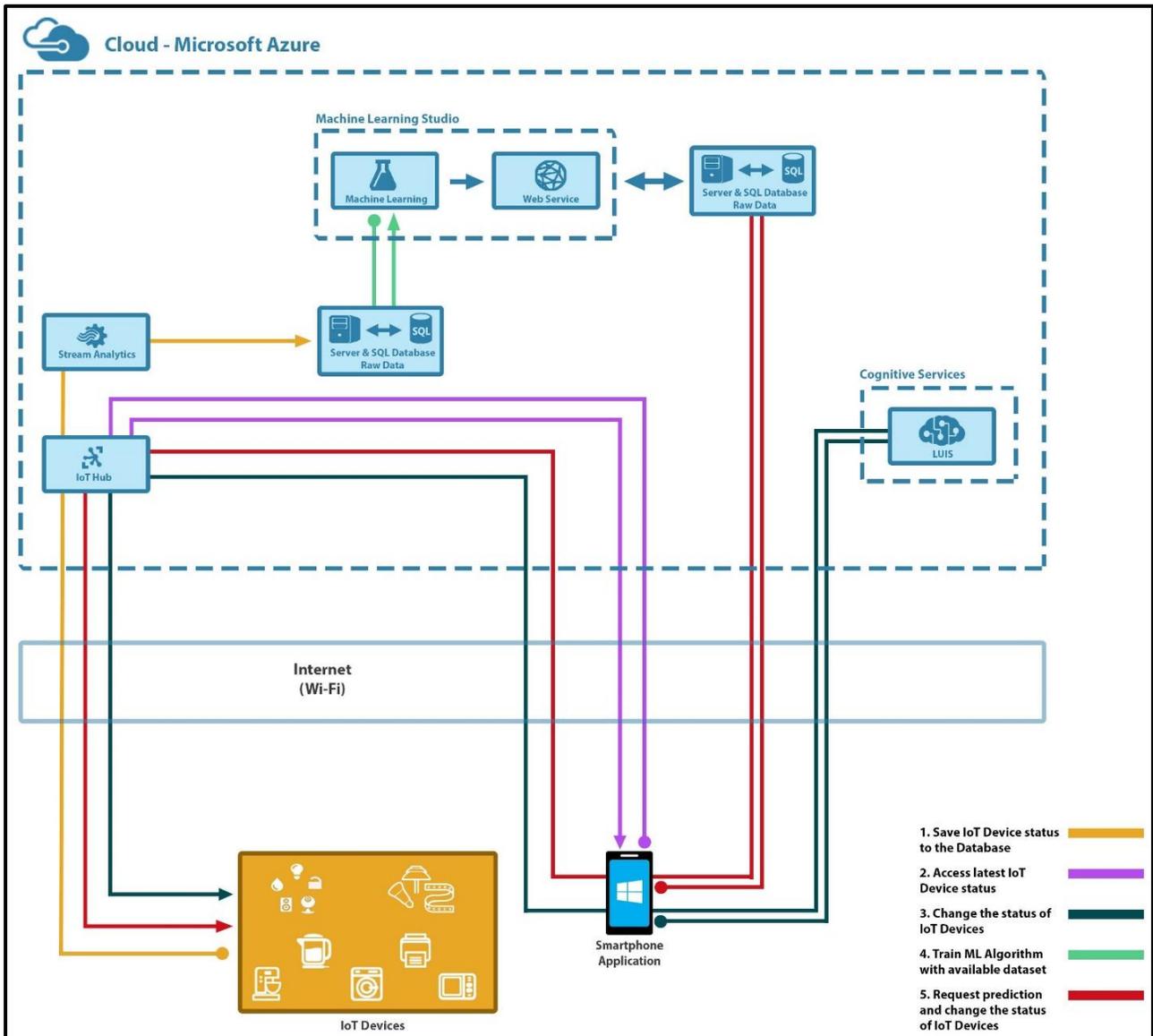


Figure 10: MLaaS Complete Architecture.

1. The first interaction is illustrated by the color **yellow** and it shows how the IoT devices store data (their status) in a database to the Cloud. Each IoT device communicates with the IoT Hub located in the Cloud. Then, the IoT Hub transfers the related information to another entity called Stream Analytics, which is responsible for saving the incoming data to a database. These data are going to be used later during step 4 by the ML service.
2. The second step (colored **purple**) shows how the smartphone application can access the latest IoT device status. During this step, the user is able to see the status of each device per room. For instance, the user can check if the kitchen's lights are on, or if the heat is on in the bedroom and in what degrees.
3. During this step (colored **dark green**), the smartphone application changes the status of a selected IoT device upon user request. The user can text or speak to the application in natural language. The user request is sent to LUIS where analysis can take place and understand it. Following that, LUIS responds back to the application which can control a specified IoT device. To better understand this interaction an example follows. The user texts or speaks "Turn on the heat in the bedroom at 23 degrees". This command is sent to LUIS which can understand the user action (turn on), the appliance (heat), the room (bedroom) and the requested value (23). LUIS responds back to the application the action, appliance, status, room and then, the application must contact the IoT Hub and relay that information to the specific IoT device responsible for controlling the heat in the bedroom and adjust its value.
4. The fourth step (**light green** color) indicates how the Machine Learning service can access the historic data of all IoT devices saved during the first step, pre-process them, train an ML model and deploy it as a web service.
5. Prior to the final step, a server must be present to continuously request predictions from the web service. These predictions are stored in a database for each IoT device separately. Following that, step 5 (**red** color) shows how the smartphone application retrieves the stored predictions from the database and adjusts the status of all IoT devices through the IoT Hub.

These five steps also fulfill all the "Must" functional requirements stated in Table 6 (chapter 3 – section 3.3). **F13: Save past device status** is achieved during the first step and **F12: Display current device status** through step 2. These two conclude the *Monitor Device Status* use case. Both, **F14: Change device status from user** and **F18: Understand natural language** are achieved in step 3, thus partially satisfy the *Control Device* use case and complete in full the *Understand Natural Language* respectively. Last but not least, **F15: Change device status from ML service** is accomplished during steps 4 and 5. This completes the other half of the *Control Device use case*.

Implemented Architecture

The implemented architecture can be seen in Figure 11, which maintains the same interactions, colors and functionalities. However, some of them are designed differently. As explained earlier, this is due to resource restrictions that apply with the free version of Azure. For instance, as seen in Table 3: Cloud Platforms – IoT., the freely accessible database is based on SQL and its capacity is limited to 32MB. At this point, it was unknown the amount of data generated by the implemented IoT devices. In order to avoid any database restrictions, in terms of capacity, a local server and database were implemented in a computer. This choice could provide hundreds of Megabytes, even Gigabytes, depending on the internal storage of the host computer. This decision influences steps 1, 2 and 4. How these steps are reorganized is discussed next.

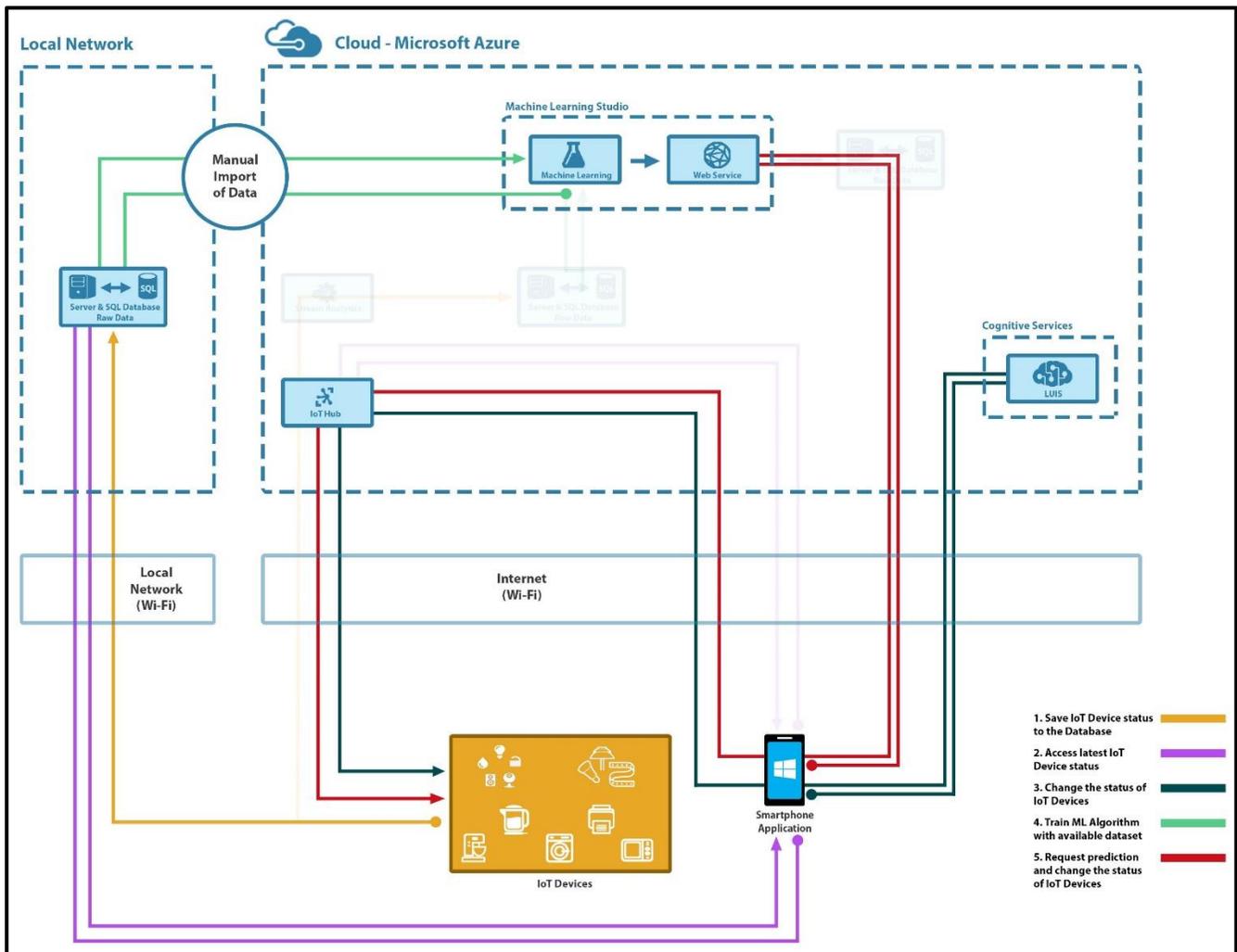


Figure 11: MLaaS Implemented Architecture.

Before moving in the discussion about the implementation steps shown in Figure 11, it should be noted that the complete architecture is shown as transparent entities and interactions.

1. Step 1 (**yellow** color) shows the way that IoT devices save their status in a database. This time the database is local and communications are performed through a local network. However, it still maintains the same communication technologies and protocols. In this step, the IoT Hub, Stream Analytics and Cloud database entities are not included. Nevertheless, the IoT Hub is still used by steps 3 and 5 as discussed next.
2. Because of the changes in the first step, in this step (colored **purple**) the smartphone application needs to communicate with the local database in order to retrieve the latest status of all IoT devices.
3. Step 3 (**dark green**) is not affected by the changes and it remains the same as in the complete architecture.
4. The fourth step (**light green**) is affected by the migration to a local database from the one maintained in the Cloud. Now, a manual export of the historic data must take place from the local database, along with a manual import of the same dataset to the Machine Learning service. This is the only difference between the complete and the implemented system during this step. The rest functionalities remain the same, i.e., data preprocessing, training of ML models and deploy one of them as a web service.
5. Step 5 (**red**) is also implemented differently, since there was not enough time to implement the server continues requests and save the predictions in a database. Thus, the smartphone application is directly communicating with the web service in order to get a prediction. Then, the generated forecasts are used to adjust the status of all IoT devices through the IoT Hub.

4.2. Implementation

Following the design choices, the implementation of the proposed solution is documented in this section. The starting point is the Azure IoT Hub and the way IoT devices can communicate through it with other entities in the architecture. Next, the software implementation of simulated IoT devices is documented, along with the development of the smartphone application. It is also important to discuss how the database was structured.

4.2.1. Azure IoT Hub

The IoT Hub in Azure Cloud is the first component to be built, as it is the bridge between the IoT devices and the rest of the architecture. In [36, p. 16], a well written tutorial is presented by Microsoft on how an IoT Hub can be initialized. Moreover, Appendix 02 includes the steps followed to setup the Azure IoT Hub for this project. Once the hub has been created, it can be accessed by a connection string which includes the *Host Name*, the *Shared Access Key Name* and the key itself, *Shared Access Key*. This connection string is unique and it can be used by any

device which wants to establish a connection with the IoT Hub, either it is a home device or the smartphone application.

The IoT Hub offers many functionalities. Among others, the hub supports registering new IoT devices, allow data to be exchanged from the smartphone application or the IoT devices to the hub (**Device-to-Cloud communication**) and vice versa, from the hub to the IoT devices and the smartphone application (**Cloud-to-Device communication**). These are a few of the core functionalities, which can be used to build a communication channel between the IoT devices and the smartphone application. The use of main functionalities will be discussed in the following subsections (4.2.3 and 4.2.4). Further details about the Azure IoT Hub and its extra functionalities, as well as, its security measures are available in [36].

4.2.2. Database

The database used for implementing the proposed solution is a local one. It is based on Wampserver [91] which provides a local Apache web server [92] and a MySQL database [93]. Following Wampserver's installation process, some changes were necessary to allow communication between the Server/Database and an external device, i.e. the smartphone. The modification process is illustrated in Appendix 02.

Following the database setup process, a local database called *SmartHome* is created. This database includes two tables, termed *Devices* and *StreamData*. The *Devices* table contains information about the devices themselves, such as the device id, type, description and assigned room. The *StreamData* table includes the device id and the device values over time.

4.2.3. IoT Devices

As stated back in chapter 1 – section 1.3, one of the limitations is the use of simulated IoT devices instead of real ones. This is because there was no immediate access to hardware devices since resources and time was a limiting factor. Instead, software applications were developed, which would act as real devices and simulate their behavior. The software applications also give the flexibility of developing various types of IoT devices (e.g. thermostats, light bulbs and locks), which could be proven difficult to find their counterparts as hardware devices. Moreover, the communication technologies (e.g. Wi-Fi) and protocols (e.g. MQTT) used in the simulated devices can be easily implemented in real devices as well, since they are widely used in IoT solutions.

The simulated devices were developed in a windows environment (Windows 10) utilizing the Universal Windows Platform (UWP). UWP is an application platform that developers use to build one application which could be implemented by all devices running Windows 10 (computers, laptops, smartphones, etc.) [94, p. 14]. While this advantage does not benefit the simulated devices, since in a real-life scenario they would not exist, it is highly useful for the smartphone application. That is because it is not limited into smartphones and it can be used across a variety of devices running Windows 10. This will provide the possibility to the user of controlling all the home devices from a Windows 10 smartphone, but also from a Windows 10 computer or laptop.

The implemented IoT devices depend on the number of rooms and home appliances the user has in his/her house. During this project, only one person was successfully approached. This person's computer was fulfilling the hardware and software requirements, i.e. to have a Windows 10 computer/laptop where the simulated devices could operate. On top of that, this person was willing to use the simulated devices during the course of four working days, in order to gather data about his/her daily life and routine. The number of rooms this person's house have and the types of devices installed in it are illustrated in Figure 12.

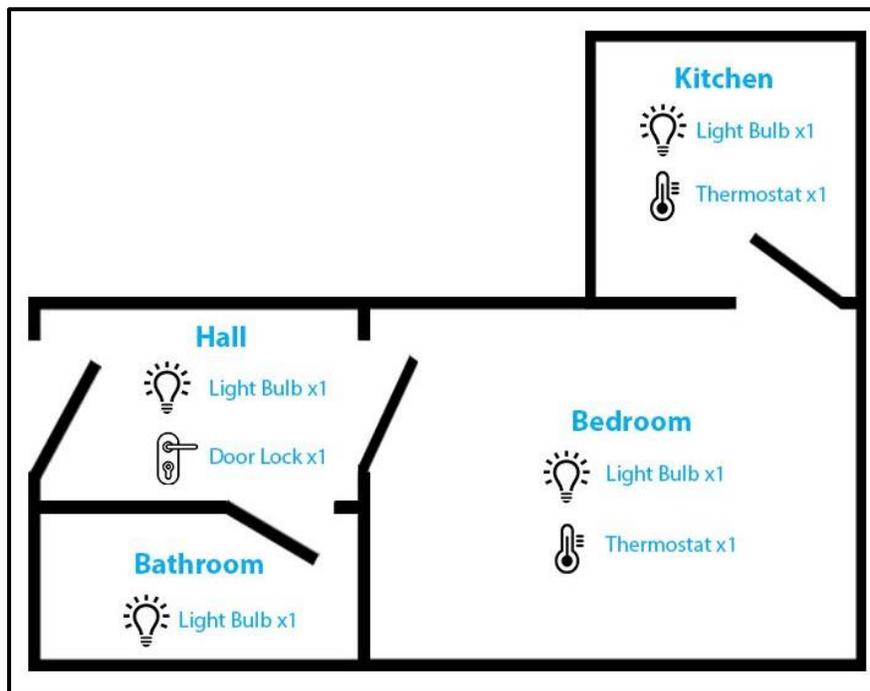


Figure 12: User House – Rooms and IoT Devices.

Figure 12 illustrates that the user's house has four rooms, with different IoT devices setup in each one. The first room is the bedroom that includes one thermostat and one light bulb. The second room is the kitchen with the same type of IoT devices as the ones found in bedroom. The third room is the hall, represented with one door lock and a light bulb. Finally, the fourth room is the bathroom with only one light bulb. To summarize, there are 4 light bulbs, 2 thermostats and 1 door lock applications that must be developed in order to simulate all the IoT devices. However, for the implementation of the aforementioned IoT devices, two types of applications need to be developed. The first type is an application which can act as a thermostat and can take values from 5 to 30, representing temperature measured in Celsius degrees. The second type is an application acting as a light bulb or door lock, which can take values 0 or 1. These represent if a light bulb is off (0) or on (1). The same applies for the door lock, 0 for unlocked and 1 for locked. Having developed these two types, they can be easily reproduced in order to match the amount of IoT devices needed for this scenario. Thus, the documentation of the simulated devices is restricted to only two types.

Type 1: Thermostat

This device type is responsible for adjusting the temperature of a specific room. It should provide the functionality of adjusting the temperature, as well as, displaying it in a screen. Also, the thermostat should send its status to the local database in order to be stored. Last but not least, it should allow the user to control it from a smartphone application. This device can be seen in Figure 13. In that figure, the device details and its current status (17 degrees Celsius) are shown. The available slider can be used to change the device status within a range of 5 to 30 degrees

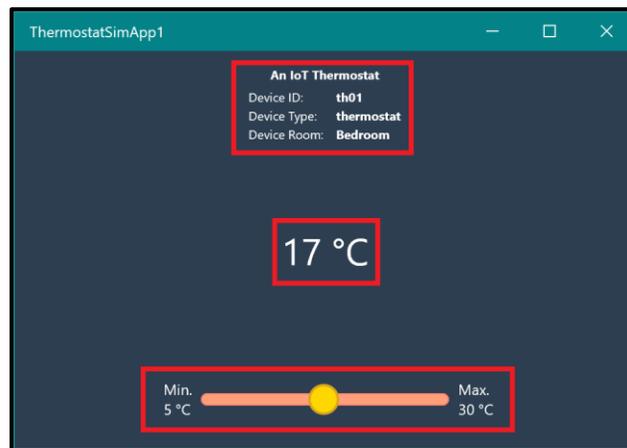


Figure 13: IoT Device – Type 1.

The device details include:

- the **Device ID**: a unique identifier for each device,
- the **Device Type**: what kind of device it is, e.g. a thermostat, light bulb, or lock device, and
- the **Device Room**: indicates in which room this device is assigned to.

This device can be easily replicated by changing the device details with primary focus on changing the Device ID.

Type 2: Light Bulb/Door Lock

Type 2 is used to turn on/off the lights or lock/unlock the door lock in a specific room. It should allow its user to perform the aforementioned functionalities, as well as, see the change of status. Moreover, the status should be stored in the local database for later use and it should allow the user to change the status from a smartphone application. The device is illustrated in Figure 14 with its two representations, the left side represents a light bulb (off) and the right side a door lock (locked). For both representations, there is a toggle button in the middle which can change the device status.

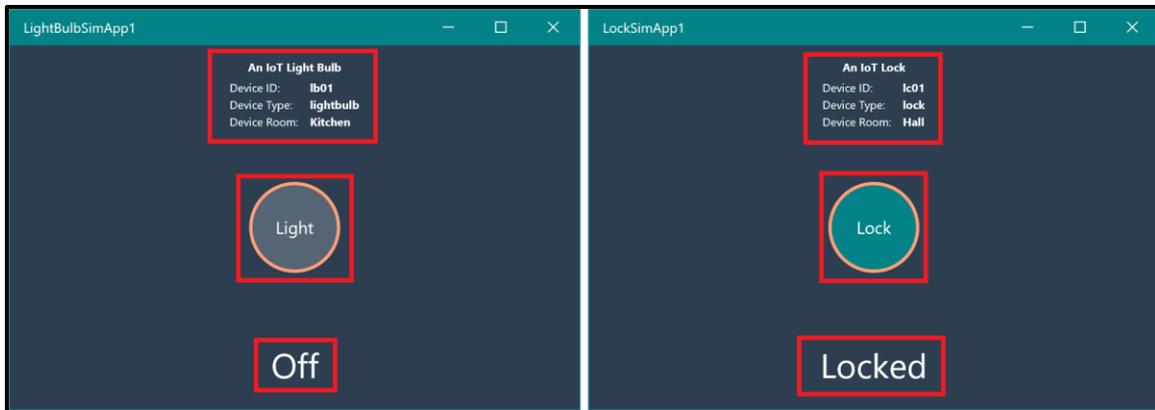


Figure 14: IoT Device – Type 2.

Similar to type 1, the device details include:

- the **Device ID**: a unique identifier for each device,
- the **Device Type**: what kind of device it is, e.g. a thermostat, light bulb, or lock device and
- the **Device Room**: indicates in which room this device is assigned to.

This device can be easily replicated by changing the device details with primary focus on changing the Device ID.

Because the code behind is similar for both types, it is documented only once. The documented code is revolved around the main functionalities of the devices, like the registration of the device to the IoT Hub, the Cloud-to-Device communication and the storage of device status to the local database. The complete code of all IoT devices is available in Appendix 03.

Registration

Starting with the registration process, each IoT device uses the aforementioned connection string (section 4.2.1) in code line 38 of Figure 15, to connect to the IoT Hub. Once connected, it tries to register itself as a new device with its unique Device ID, code line 42. If the registration is successful, a symmetric key is generated which includes the device's primary key, code line 49. By using that key, the device can authenticate itself to the IoT Hub and start communicating. If the registration is unsuccessful, which means that there is already a device with that key, then it is assumed that the same device tries to register again, since each device id is unique, thus, bypassing the registration in the IoT Hub and only return the primary key, code line 46 and 49.

```

35 // Register Device to IoT Hub.
36 public static async Task<string> AddDeviceAsync(string deviceId)
37 {
38     RegistryManager registryManager = RegistryManager.CreateFromConnectionString(hubConnectionString);
39     Device device;
40     try
41     {
42         device = await registryManager.AddDeviceAsync(new Device(deviceId));
43     }
44     catch (DeviceAlreadyExistsException)
45     {
46         device = await registryManager.GetDeviceAsync(deviceId);
47     }
48     //Debug.WriteLine(device.Authentication.SymmetricKey.PrimaryKey);
49     return device.Authentication.SymmetricKey.PrimaryKey;
50 }

```

Figure 15: IoT Device – Registration.

Some potential security issues might exist in that code which could be resolved with extra security measures by the IoT Hub, like using X.509 Certificates or Access Tokens [36, p. 272]. However, the secure communication of the whole solution is out of the scope of this project.

Cloud-to-Device Communication

The communication between the Cloud (IoT Hub) and the IoT devices is shown in Figure 16. This part of the code is responsible for relaying any command received in the Cloud (IoT Hub) and targeted to a specific device, to the device itself. For example, any command given by the user to his/her smartphone, it reaches the Azure IoT Hub through a Device-to-Cloud communication (see section 4.2.4) and then, by using the Cloud-to-Device communication it reaches the specific home device.

```

67 // Cloud to Device Communication
68 public static async Task<string> ReceiveCloudToDeviceMessageAsync(string deviceId, string deviceSharedAccessKey)
69 {
70     // Connction string for device communication
71     string deviceConnectionString = "HostName=" + hubHostName + ";DeviceId=" + deviceId +
72     ";SharedAccessKey=" + deviceSharedAccessKey;
73
74     var deviceClient = DeviceClient.CreateFromConnectionString(deviceConnectionString, Microsoft.Azure.Devices.Client.TransportType.Amqp);
75
76     while (true)
77     {
78         var receivedMessage = await deviceClient.ReceiveAsync();
79
80         if (receivedMessage != null)
81         {
82             var messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());
83             await deviceClient.CompleteAsync(receivedMessage);
84             return messageData;
85         }
86
87         await Task.Delay(TimeSpan.FromSeconds(1));
88     }
89 }

```

Figure 16: IoT Device – Cloud-to-Device Communication.

The code lines 70-72 show the connection string used to connect a device to the IoT Hub. This string contains the name of the IoT Hub, the device's id and its primary key. The device's id and primary key are used for authenticating the device to the IoT Hub. Then, code line 74 performs the connection using the AMQP protocol. It should be noted that the MQTT protocol could be

used as well. Once the connection has been established, the device keeps listening for any incoming messages, lines 76-88. Then, the received messages can be used to set the device status.

Store Device Status

Before each device stores its value to the local database, first it needs to save some vital information about it, like its unique id, what type of device it is, in which room it is assigned to and some optional description. All these data are saved to the table: *Devices*, which is located in the database. This can be seen in Figure 17.

```
37     }
38
39     MySqlCommand registerDevice = new MySqlCommand("INSERT INTO Devices (deviceId, deviceType, deviceDescription, deviceRoom) VALUES (\\" +
40         deviceId + "\", \\" + deviceType + "\", \\" + deviceDescription + "\", \\" + deviceRoom + "\")", connection);
41
```

Figure 17: IoT Device – Register to Local Database.

However, before a device can store any information to the database or even register to it, a connection to the local server is necessary. This is possible by using a connection string (Figure 18, code line 67), which includes information about the server IP address the database is hosted, the database name to be used and authenticate to it by using the database username and password. Having a connection to the local database and all the devices registered to it, then, the devices can store their statuses (deviceValue), along with their ids (deviceId) and a timestamp (dTime) by using the SQL command in lines 79-80 (Figure 18). The timestamp is represented as “yyyy-MM-ddTHH:mm:ss” (e.g. 2017-05-19T16:45:52). The information about the value of each device and the timestamp are saved in table: *StreamData* into the database.

```
59     // Send data to DB.
60     public static void deviceToDB(string deviceId, double deviceValue, string dTime)
61     {
62         // Change the character encoding.
63         EncodingProvider encode;
64         encode = CodePagesEncodingProvider.Instance;
65         Encoding.RegisterProvider(encode);
66
67         using (MySqlConnection connection = new MySqlConnection(connectionString))
68         {
69             try
70             {
71                 connection.Open();
72             }
73             catch (Exception)
74             {
75
76                 Debug.WriteLine("There was a problem connecting to the database. Maybe is not active?");
77             }
78
79             MySqlCommand registerDevice = new MySqlCommand("INSERT INTO StreamData (deviceId, deviceValue, dTime) VALUES (\\" +
80                 deviceId + "\", \\" + deviceValue + "\", \\" + dTime + "\")", connection);
81
```

Figure 18: IoT Device – Store Device Status.

Concluding the development of the IoT devices, now a connection can be established from the home devices to the Azure IoT Hub, receive messages from a smartphone application and store their status. The flow of the whole process can be seen in Appendix 04 as sequence diagrams.

4.2.4. Smartphone Application

The smartphone application was developed using the UWP in Windows 10. As briefly discussed in section 4.2.2, once an application has been developed in UWP, it can be used by any Windows 10 device. This provides the flexibility of controlling the IoT devices from a Windows 10 smartphone, computer or laptop. The application must be able to retrieve the latest status of each home device, allow the user to change the status of all the registered devices and change the status of these devices by itself as well, by utilizing the predictions from the Machine Learning Service.

Before diving into the core functionalities of the application, it should be noted that its UI is designed according to specified rooms shown in Figure 12 (subsection 4.2.2), including four fixed rooms and three different sensors for each, see Figure 19. This is part of the delimitations mentioned in chapter 1 – section 1.3, along with the user experience the application provides. Further research is needed to design and develop an application which can add or remove rooms and sensors dynamically, based on user/house needs.

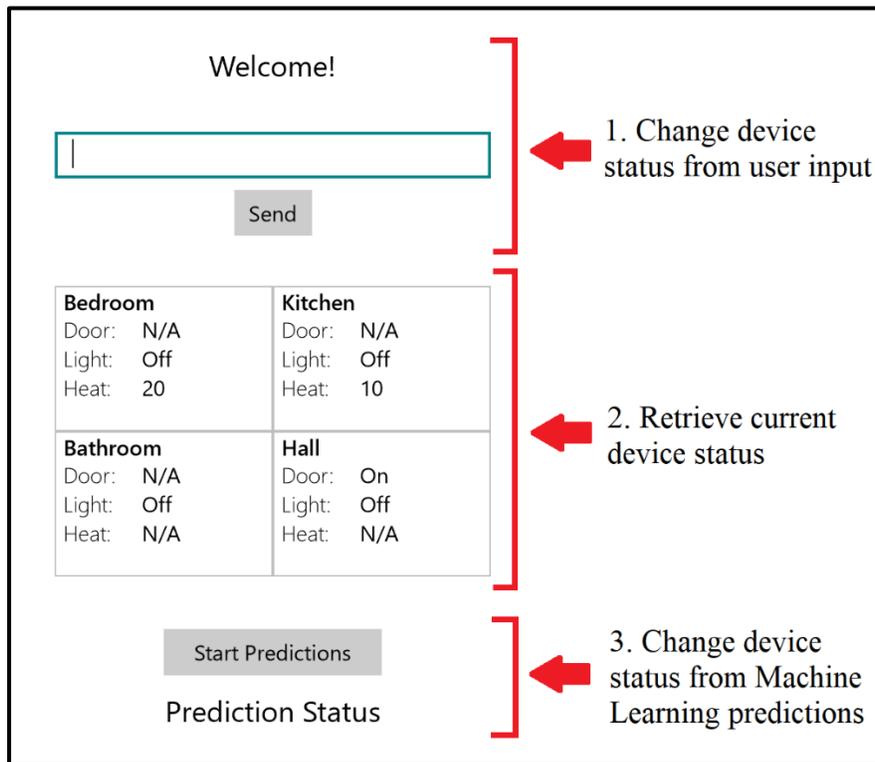


Figure 19: Smartphone Application.

Figure 19 illustrates the developed smartphone application. On the top, there is a welcome message with a textbox where the user can write a command in natural language. Then, the four rooms with their installed home devices follow, while their current status is displayed. Finally, a button is available for starting the predictions for each home device. Each one of these functionalities are discussed next, along with their corresponding code. The following code is focused on the most important parts of these three functionalities and the whole application code is available in Appendix 03.

1. Change device status from user input

The first arrow in Figure 19 shows a welcome message, a textbox and a button. The textbox is used by the user to write the command of changing the status of a home device in natural language and submit it by pressing the “Send” button. An example of such a command is: “Turn on the heat to 23 in the kitchen”. The application must analyze the given command and understand that the user wants to turn on the heat in the kitchen at 23 degrees. This analysis is done by the Cognitive Computing service (LUIS) and further details about it can be seen in subsection 4.2.5. However, at this point it is important to know that LUIS will return a message to the application with the intent of the user (turn on), the appliance (heat), the room (kitchen) and the value (23). By having this information, the application can find the home device associated with the heat in the kitchen and adjust the value to 23. This flow is available as a sequence diagram in Appendix 04.

To connect with the Cognitive Service, LUIS, the Predict method is used (Figure 20), where an **application id** and **subscription key** are necessary (code lines 99 and 100). These are made available from the Cognitive Computing service (more about it can be found in subsection 4.2.5). By using the aforementioned id and key, a connection to the LUIS client can be achieved (code line 106). Following that, a text command can be fed to LUIS (code line 107) and analyze it by calling the processRes method (code line 108).

```
97 public async void Predict()
98 {
99     string appId = "████████████████████";
100    string subscriptionKey = "████████████████████";
101    bool preview = true;
102    string textToPredict = textBox.Text;
103    //string forceSetParameterName = txtForceSet.Text;
104    try
105    {
106        LuisClient client = new LuisClient(appId, subscriptionKey, preview);
107        LuisResult res = await client.Predict(textToPredict);
108        processRes(res);
109        Debug.WriteLine("Predicted successfully.");
110    }
111    catch (System.Exception exception)
```

Figure 20: Smartphone Application – Connect to Cognitive Service.

The processRes method receives the results computed in LUIS. If LUIS was not provided with all the necessary information, i.e. the intention, the appliance, the room and the value, it continues the conversation with the user and asks him/her to specify the missing information. For instance, if the user writes “Turn on the lights”, the processRes should prompt the user the following question “To which room?”. Once all the required information is available, the processRes method is responsible for finding the IoT device which is responsible for the lights in the specific room and contact the IoT Hub in the Cloud for changing its value. Finally, the user should also be notified of the success or not of this action. The entire code of the smartphone application is available in Appendix 03.

2. Retrieve current device status

The second arrow in Figure 19 illustrates all IoT devices per room and their status. Each room is shown as a different box and includes three different devices: a door, a light and a heat. It is not

necessary for each room to have all these devices installed. This can be seen by the N/A values indicating that specific devices do not exist in specific rooms. The devices installed in a room have their respective values (e.g. On, Off, 20 °C, Locked, etc.).

Initially the application must connect to the local server by using a connection string. This includes the IP address of the server, the name of the database, where the information is stored, and a username/password for authentication purposes. Following that, the application requests the information about all IoT devices, as shown in code line 46 – Figure 21, from the *Devices* table. Then, a while loop (code line 52) iterates the returned information per row (since one row corresponds to one device) and retrieves the device id, type and assigned room.

```
45
46 MySqlCommand retrieveDevices = new MySqlCommand("SELECT * FROM Devices", connection);
47
48 try
49 {
50     using (MySqlDataReader reader = retrieveDevices.ExecuteReader())
51     {
52         while (reader.Read())
53         {
54             // get the results of each column
55             deviceId = (string)reader["deviceId"];
56             deviceType = (string)reader["deviceType"];
57             deviceRoom = (string)reader["deviceRoom"];
```

Figure 21: Smartphone Application – Retrieve Devices.

By using the retrieved information, now the IoT devices can be bound to the UI based on the room each device is assigned to and according to the type of each device. As it can be seen in Figure 22, a for loop iterates through all rooms with the purpose of finding the one in which the recently retrieved device from the database belongs to. As soon as the right room has been found, the device is assigned to it according to the type. For example, if the device represents a light bulb, then the device type equals to “lightbulb” and it is assigned as such.

```
67 for (int i = 0; i < rooms.Count; i++)
68 {
69     if (deviceRoom == rooms[i].Name)
70     {
71         if (deviceType == "lock")
72         {
73             rooms[i].Door = deviceId;
74         }
75         else if (deviceType == "lightbulb")
76         {
77             rooms[i].Light = deviceId;
78         }
79         else if (deviceType == "thermostat")
80         {
81             rooms[i].Heat = deviceId;
82         }
83         exists = true;
84         break;
85     }
86 }
```

Figure 22: Smartphone Application – Bind Devices to UI.

Now that all devices have been retrieved from the database and are bound to the UI, individual requests per device id of their values can be asked from table: *StreamData* of the same database (see Figure 23).

```
157 MySQLCommand retrieveDevices = new MySQLCommand("SELECT deviceValue FROM streamdata WHERE deviceId = \" + deviceId +  
158 \" ORDER BY id DESC LIMIT 1", connection);
```

Figure 23: Smartphone Application – Retrieve Recent Status.

The flow of retrieving the recent status of IoT devices can be seen as a sequence diagram in Appendix 04.

3. Change device status from Machine Learning predictions

The final development process that needs implementation is the alteration of the status of all IoT devices based on the Machine Learning predictions, illustrated in Figure 19 - third arrow. The process of how the Machine Learning Service was built is documented in subsection 4.2.6. Even so, it is important to know at this stage that the Machine Learning Web Service deployed (see subsection 4.2.6) can be accessed by an API key. Then, an Http request can be made (Figure 24, code line 19) with a date-time input. This input is represented as “yyyy-MM-ddTHH:mm:ss” (e.g. 2017-05-21T20:47:52). Thereafter, the ML service computes the predictions and replies back to the application the forecasts for all IoT devices in JSON format. This response includes the ids of the devices and the predicted values. Consequently, the application is responsible for reaching the IoT Hub (Cloud) and send the values each home device should have. Then, the IoT Hub must communicate with each IoT device and send the predictive value. This sequence of events is illustrated in Appendix 04 as a sequence diagram.

```
19 using (var client = new HttpClient())  
20 {  
21     var scoreRequest = new  
22     {  
23         Inputs = new Dictionary<string, List<Dictionary<string, string>>>() {  
24             {  
25                 "input1",  
26                 new List<Dictionary<string, string>>(){new Dictionary<string, string>(){  
27                     {  
28                         "dTime", dTime  
29                     },  
30                 }  
31             }  
32         },  
33     },  
34     GlobalParameters = new Dictionary<string, string>()  
35 }
```

Figure 24: Smartphone Application – Connect to Machine Learning Web Service.

4.2.5. Cognitive Computing Service

In the previous subsection, it was discussed how the Cognitive Computing service LUIS can receive a command in text form, analyze it and produce an output in JSON format. However, here

it is examined how LUIS works internally, how it analyzes a command and how it reaches an output.

LUIS can be accessed online [40] and it can be used to create intelligent bot apps. The most basic components of a LUIS bot are the intents and the entities. An intent states the intention of the user over an entity. For instance, if the user wants to lock the door, the intent is the lock action and the entity is the door. There can be more than one entities at the same time, an example is the phrase: “Lock the door in the hall”. The intention is the same but there are two entities, the first being the door (equipment entity) and second one the hall (room entity). Following these examples and based on the available IoT devices discussed in 4.2.3, the Smart Home Bot consists of 6 different intents and 4 entities. The thermostat IoT devices use intentions such as adjust the temperature, turn on and turn off. The entities include an Appliance (e.g. heat), a Room (e.g. bedroom) and a Temperature (e.g. 23 degrees). The turn on and turn off intentions are also used by the light bulb home devices together with the entities Appliance (e.g. lights) and Room (e.g. kitchen). The lock IoT devices use the lock and unlock intentions with the Equipment entity (e.g. door) and Room (e.g. hall).

When all required intents and entities have been specified, a general training can be performed in the bot and test its results. Figure 25 below illustrates a one-click training process, by pressing the button “*Train Application*”. The training is performed in the background and it is not visible to the developer, neither the ML algorithms used for this process. Once the training is completed, the developer can test the bot app by writing different phrases, e.g. “*turn the heat on*” (see Figure 25 – left side). During this process, it is visible if the bot app understood correctly the intention and the entities involved (see Figure 25 – right side).

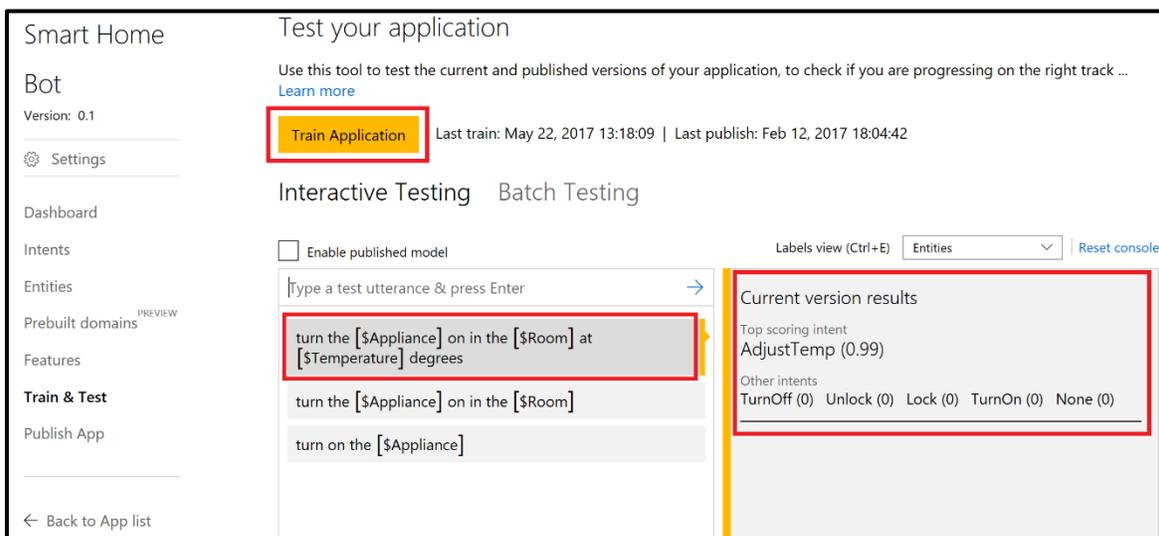


Figure 25: LUIS – Training and Testing.

If the results are ideal, i.e. if the bot can successfully understand a sentence, the bot app can be published and accessed using an *API* and a *subscription* key. With these keys, the smartphone application is allowed to contact the Smart Home Bot, provide text commands and get results back. The results are formulated in JSON format and provide the intention and the entities involved. The entire process of how a LUIS bot can be built and trained is documented in Appendix 05.

Up until this point, the software development phase has been completed. This allows to move from stage 3 (Software Development) of the project’s methodology to stage 4 (Data Gathering), see Figure 2, chapter 1 – section 1.4. However, the collection of a dataset is documented later in section 4.3 and this section continues with the development of the Machine Learning Service (stage 5).

4.2.6. Machine Learning Service

The Machine Learning Service can be accessed online from the Azure Machine Learning Studio [37]. An illustration of the Studio and how it functions is presented in Appendix 06. Before an ML service can be deployed and accessed by other services or applications, as documented previously (subsection 4.2.4), several underlying components need to be considered. These are illustrated in Figure 26 and are implemented in the next section, 4.3. Experimental Assessment.

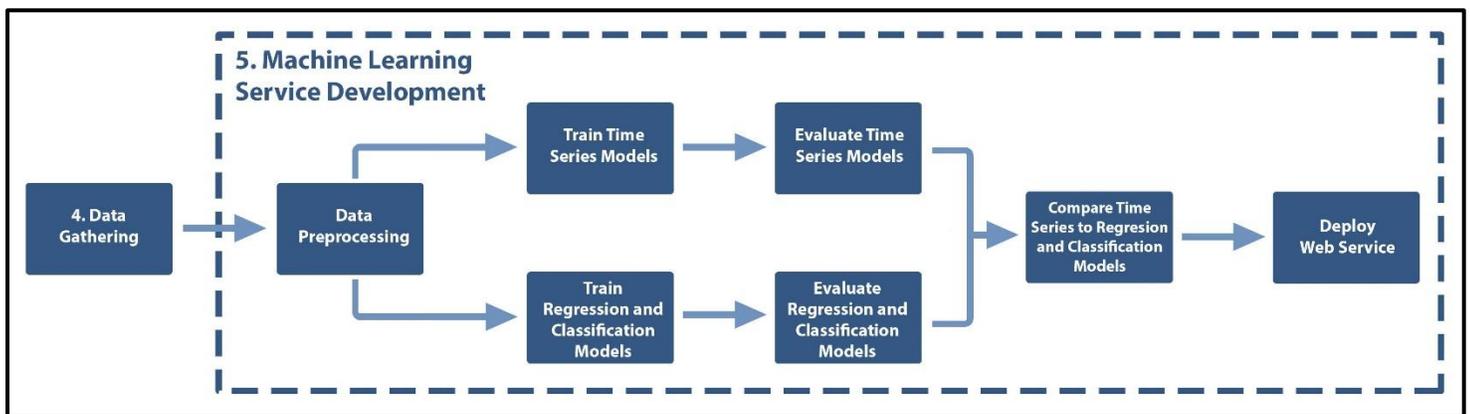


Figure 26: Methodology - Step 5.

1. First and foremost, the gathered data should be preprocessed and prepared, before feeding them into an ML algorithm. During this step, data can be visualized to get a better understanding. They can also be preprocessed by treating their missing values and remove noise.
2. Secondly, the resulting dataset from the previous step is used to train various Time Series ML models, along with a combination of Regression and Classification models. The reason why Regression and Classification models are used together is because there are data which are different in nature. Further details about this are encompassed in section 4.3, subsection 4.3.2.
3. The next step is to evaluate each trained ML model separately in order to find the one performed best. By the end of this step, a single Time Series Forecasting model must be selected along with a single Regression model and a single Classification model.
4. Then, the performance of the Time Series Forecasting model is compared to a combination of the Regression model, regarding a specific type of devices, with the

Classification model, for the rest of the IoT devices. Further details about this are available in section 4.3, subsection 4.3.2

5. Finally, the ML model yielding the most accurate predictions about sensor output is deployed as a web service. This web service is later used by the smartphone application, as seen in subsection 4.2.4.

4.3. Experimental Assessment

In this section, all the performed experiments are documented, from the data gathering to the deployment of the web service. Initially, it is described how the dataset was gathered. Then, several experiments were conducted in Azure ML Studio based on the available dataset. The first experiment is the data preprocessing, followed by the training of Time Series Forecasting, Regression and Classification ML models. Furthermore, all ML trained models are evaluated and the best performing one from the Time Series Forecasting algorithms is compared to the best Regression and Classification model. Finally, the top-performing ML algorithm from this comparison is deployed as a web service.

4.3.1. Dataset Acquisition

So far, the software development process has been completed and a dataset can be gathered. By using the smartphone application, the Smart Home user is capable of adjusting the status of all IoT devices according to his/her preferences. By doing so, two different datasets were collected over the course of four working days. The only difference between the two datasets is the sample rate, i.e. 1 second and 1 minute. Next, the decision process is discussed on why these sample rates were selected.

The IoT devices were designed to save their status to the database over a certain amount of time (sample rate). From Table 3 it can be observed that each of the considered Cloud platforms (chapter 2 - section 2.4) supports a certain amount of messages to be exchanged between the IoT devices and the Cloud-based IoT Hub per day. Specifically for Azure, messages are limited to 8,000 per day. Taking as an example the sample rate of 1 minute, this would mean 5.55 messages per minute, since 1 day has 1,440 minutes. This is not enough, as the available IoT devices are seven, according to Figure 12. Thus, the required minimum number of messages to be exchanged is 7, without considering the messages from the smartphone application to a single IoT device in order to change its status. By increasing the sample rate to 2 minutes, 11.11 messages are supported. This sample rate fits the requirements of 7 messages in addition to 4 extra ones that can be used to adjust the temperature of an IoT device. Similar restrictions apply to AWS, since it supports 8,333 messages per day, and even more to Bluemix, as it only supports 3,333 messages per day.

However, up until this point, it is not safe to assume if a dataset with a sample rate of 2 minutes can provide better results over a dataset with a 1-minute sample rate or 10-minute. Because of that, it was decided to perform experiments with different sample rates, in order to determine which one can be more beneficial. Furthermore, since a local database is used in the prototype and the status of all IoT devices is stored there, bypassing the IoT Hub, a dataset can be gathered with a

sample rate as low as 1 second. Nevertheless, because of time restrictions, it was not possible to perform experiments with multiple datasets over different sample rates. For this reason, only two experiments were conducted, including a dataset with a sample rate of 1 second and another one of 1 minute. Still, experiments with additional sample rates should be conducted in the future, as it is also mentioned in chapter 5.

An example of the gathered dataset is illustrated in Figure 27. Figure 27 (a) shows the user’s preferences of the bedroom’s temperature with values from 5 to 30. This variation represents Celsius degrees. Figure 27 (b) shows the kitchen’s light status, with values 0 or 1. The value zero represents the “Off” state of the light bulb, whilst the value 1 represents the “On” state. Both of them derived from a dataset with a sample rate of one minute. Still, there are 5 more IoT devices with their own line plots which are not illustrated here. These are available in Appendix 07, along with their corresponding line plots about the dataset with a sample rate of 1 second.

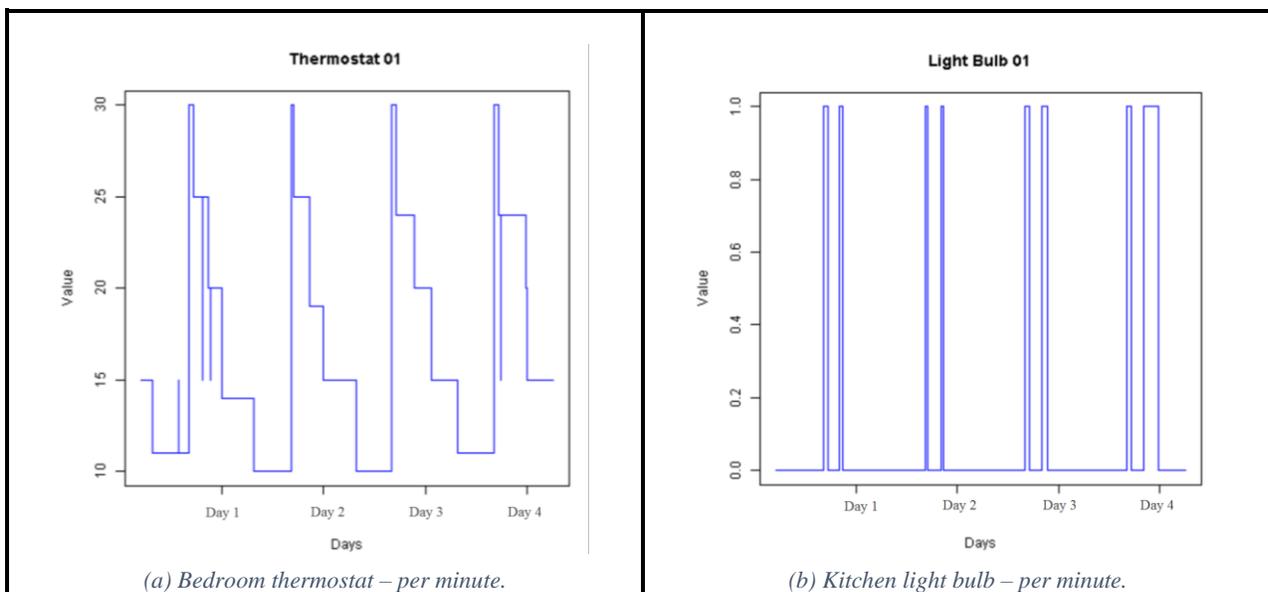


Figure 27: User Preferences.

4.3.2. Machine Learning Experiments

From the previous subsection, it was decided that two different datasets are going to be used in the ML experiments, one with a sample rate of 1 second, and another one with a sample rate of 1 minute. In order for these datasets to be used, they need to be uploaded to Azure ML Studio. Following that, the two datasets can be visualized and retrieve extra information about them. Based on that information, the datasets can be preprocessed before using them in any ML algorithm. By the end of all the experiments, among other results, it should be concluded which sample rate, every second or minute, gives better results. However, further experiments should be conducted with different sample rates (e.g. every ten minutes) in order to determine the best one. This is also mentioned as future recommendation chapter 5.

Data Preprocessing

Figure 29 illustrates the first dataset, every second. There, a small sample of the whole dataset can be seen, with the total number of rows and columns. By selecting a single column, in this case the th01 IoT device, a BoxPlot is generated which indicates the values the column has, Figure 29 (b). Also, some statistics are shown about the selected column, Figure 29 (a).

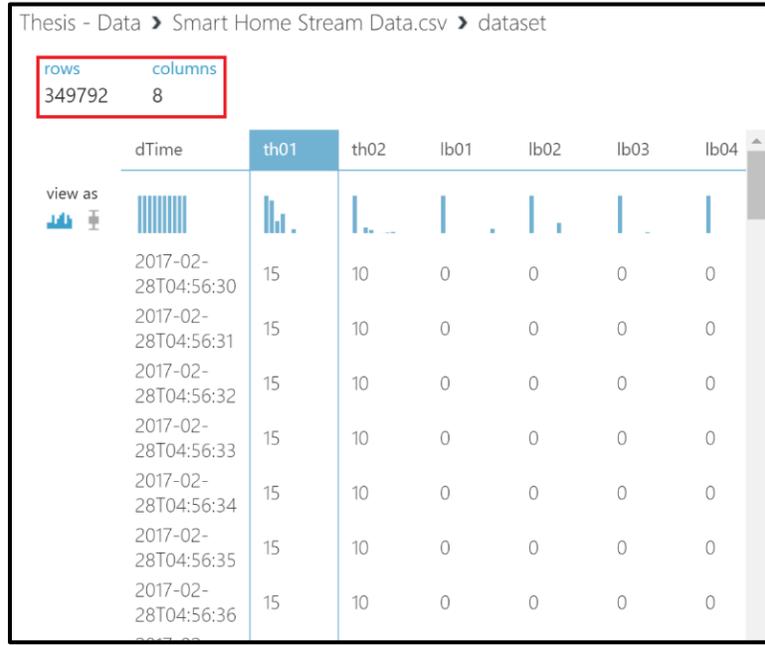
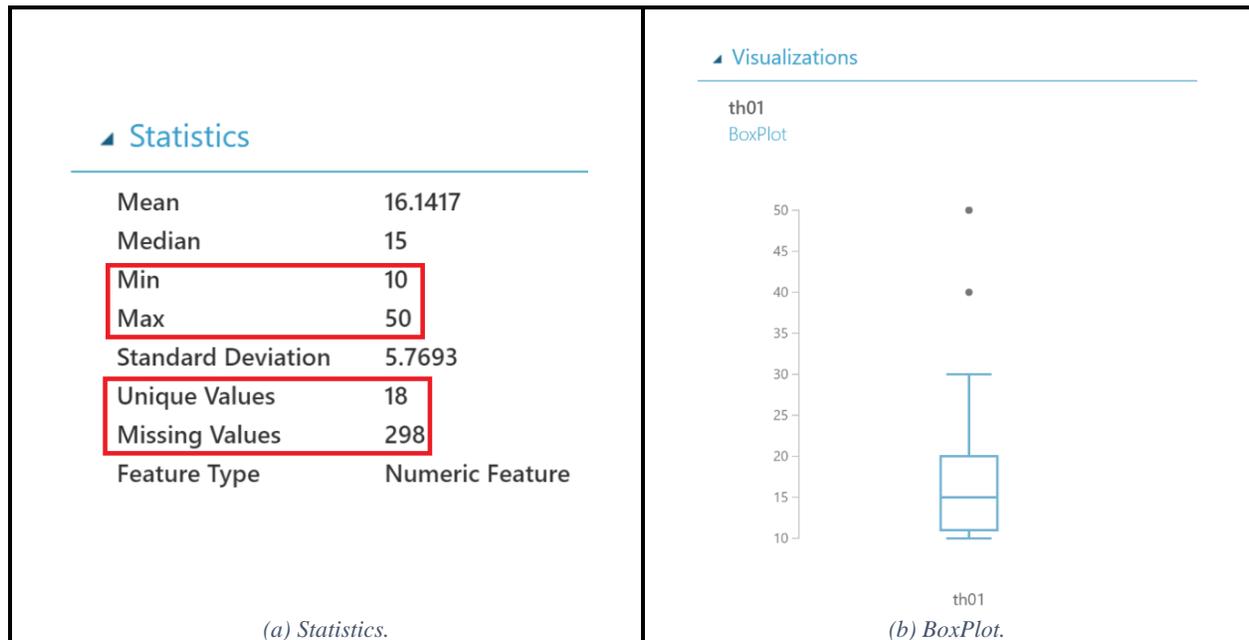


Figure 28: Azure ML Studio – Dataset every second 1.



(a) Statistics.

(b) BoxPlot.

Figure 29: Azure ML Studio – Dataset every second 2.

Based on the visualized BoxPlot and Statistics, it can be seen that there are some values which are unusual. As stated before, the heat IoT devices have a value range from 5 to 30, however, the BoxPlot shows that there are two values beyond the maximum 30 (40 and 50), which is also visible in the Statistics that shows a maximum value of 50. These should be excluded from the ML training process, or replaced with the maximum value 30, since a heat IoT device would never set the temperature over 30 degrees. In this case, all values above 30 are replaced with 30 as it is assumed that this is an error which occurred in the communication channel and the value should have been 30.

Additionally, from Figure 29 (a), it can be seen that there are 298 missing values. These should be either deleted or replaced with other values. It was decided to replace the missing values because if they were to be deleted, entire rows would be erased from the dataset including valid values from other IoT devices. The replacement method can be argued and further research is needed in order to decide the best one, however, the mode technique was chosen in this case, so all missing values were replaced with the one that appears most often in the dataset. The same techniques were used to the remaining columns (IoT devices), for both second-based and minute-based datasets. In particular, for the devices which store 0 or 1 as their values, it was noticed that there is an error of values exceeding the maximum 1. All these were replaced with 1.

Following the results of the preprocessed data, Figure 30 illustrates the heat IoT device th01 line plot per second (a) and per minute (b). It can be seen that the per minute line plot is smoother than the per second one. Nevertheless, at this point it cannot be decided which one is better since the per second line plot, while jittery, contains more information that could provide better prediction results. This decision will be made later, when evaluating the trained ML models.

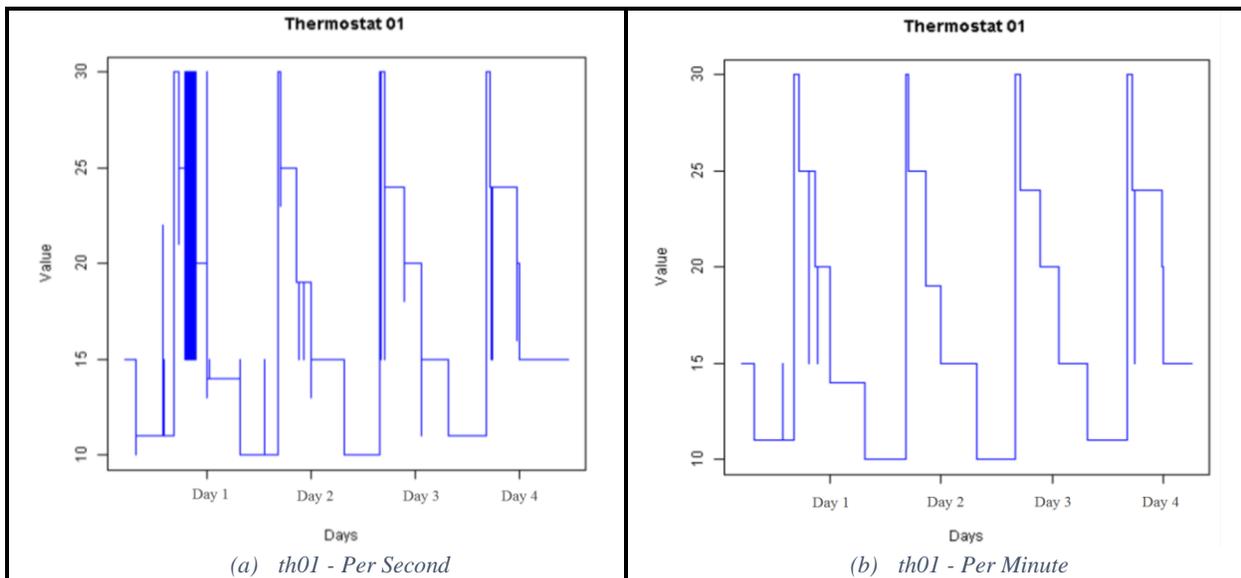


Figure 30: Azure ML Studio – th01 Line Plot.

In contrast to the th01 device, the lb01 per second and per minute line plots, Figure 31 (a) and Figure 31 (b) respectively, are similar and distinctive observations cannot be made.

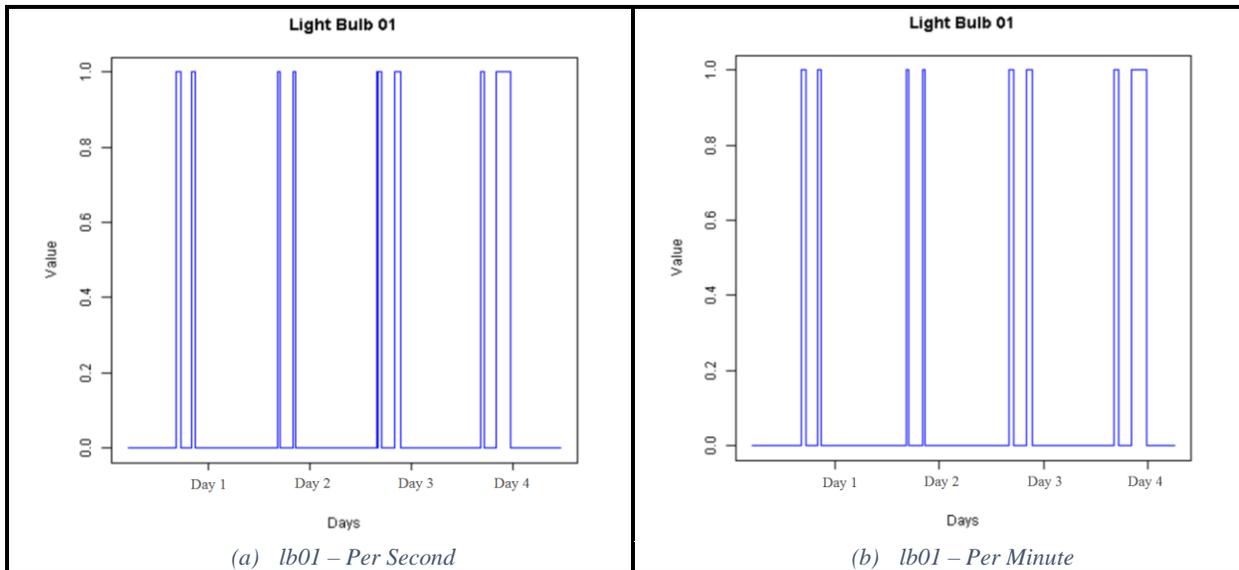


Figure 31: Azure ML Studio – lb01 Line Plot.

The rest of the IoT devices share similar line plots, the th02 with th01 and the lb02, lb03, lb04, lc01 with lb01. For this reason, they are not shown here, but they can be found in Appendix 07.

Time Series Forecasting Models – Training

Since no distinction has been made so far between the per second dataset and the per minute one, the training of the ML models is continued for both available datasets.

The next experiment is about the training of Time Series Forecasting models. There, two different algorithms were used, ARIMA and ETS. While Azure ML Studio does not provide implementations of Time Series Forecast algorithms, as it does with Regression and Classification, it offers a functionality in which R code can be implemented. By using specific functions of R code, ARIMA and ETS models can be built. In [95, p. 38], the `forecast()` function is documented in R language, which can be used for implementing Time Series Forecasting Models. This function takes as arguments a time series model for which predictions need to be calculated, e.g. the values of one device over time, the number of periods for forecasting, e.g. the next day, and the method, ETS or ARIMA. In this case, out of the four days dataset, the three were used for training and the remaining one for testing.

However, before feeding a time series model to the forecast function, a decomposition method can be performed on it, if seasonal or trend information exist. As it can be seen in the previous figures, Figure 30 and Figure 31, there is a pattern in the data follow which occurs every day. More precisely, regarding the per second dataset, the pattern occurs every 86,400 seconds and for the per minute dataset every 1,440 minutes, since 1 day equals to 1,440 minutes and also to 86,400 seconds (1 day = 24 hours, 1 hour = 60 minutes, 1 minute = 60 seconds). Therefore, there is a seasonal pattern in both datasets. To take advantage of this information, the STL (Seasonal and Trend using Loess) decomposition method can be used, which allows seasonality other than monthly or quarterly as stated in chapter 2, subsection 2.2.2. R language supports STL with the

use of the `stl()` function [95, p. 53]. Now, the time series model needs to be used in the `stl()` function and the result of this can be used in the `forecast` function, instead of the initial time series model. An example of creating a training model for the th02 IoT device using the ETS technique can be seen in Figure 32 below.

```
34 # Build th02 forecasting models
35 train.ts <- ts(train$th02, frequency = horizon, start = date.info(train))
36 train.stl <- stl(train.ts, s.window="periodic")
37 train.model <- forecast(train.stl, h = horizon, method = 'ets', ic = 'bic', opt.crit='mae')
38 forecast.th02_stlEts <- train.model$mean
39 forecast.th02_lo95_stlEts <- train.model$lower[,1]
40 forecast.th02_hi95_stlEts <- train.model$upper[,1]
```

Figure 32: Azure ML Studio – th01 ETS Training Model.

The same procedure is followed for the rest IoT devices, one time for the ETS algorithm and one time for the ARIMA, for both datasets, per second and per minute. The Azure ML Studio implementations of ARIMA and ETS for all IoT devices is available in Appendix 06.

Regression and Classification Models – Training

Despite the Time Series Forecasting techniques, ARIMA and ETS, Regression and Classification algorithms can be also considered to predict future values based on time. Regression is used for the thermostats (th01 and th02 IoT devices), since the targeted prediction is a number with a range of 5 to 30. On the other hand, for the rest of the IoT devices (lb01, lb02, lb03, lb04, lc01) Classification is used, as the targeted prediction value is nominal and specifies the class it belongs to. About the light bulbs, 0 represents the **Off** class, whilst 1 the **On**. Regarding the lock device, the value 0 stands for the **Unlocked** class and the value 1 for the **Locked** one.

Before feeding the two datasets (per second and per minute) to Regression and Classification algorithms, the date and time attribute needs to be modified. So far, the date/time was represented as “yyyy-MM-ddTHH:mm:ss” (e.g. 2017-05-19T16:45:52) and considered one attribute. However, since Azure Regression and Classification models cannot process this attribute in that form, it was necessary to split into several others. For the per second dataset, the date/time is divided into 6 different attributes: year, month, day, hour, minute and second. Regarding the per minute dataset, the distinct attributes are 5: year, month, day, hour and minute. By doing so, a more precise sense of date and time can be given during the training of ML models. Then, for each device an ML model is created. Appendix 06 includes figures illustrating the building blocks for all Regression and Classification algorithms, regarding both datasets (per second and per minute), and for all IoT devices.

Several Regression and Classification algorithms are provided by the Azure ML Studio. The following table, Table 8, summarizes all of them [28].

Regression ML Algorithms	Classification ML Algorithms
Boosted Decision Tree	Boosted Decision Tree
Decision Forest	Decision Forest
Fast Forest	Decision Jungle
Neural Network	Neural Network
Linear Regression	Logistic Regression
Bayesian Linear Regression	Bayes Point Machine
	Support Vector Machine (SVM)
	Locally Deep SVM
	Averaged Perceptron

Table 8: Azure ML Studio – Regression & Classification Algorithms.

All algorithms presented at Table 8 are used during the training of Regression and Classification models. It should be noted that each algorithm has specific settings which can be adjusted, like the minimum or maximum number of leaves a Boosted Decision Tree can have. Nevertheless, the default settings were used for all algorithms and further research is needed in order to examine if an algorithm can perform better with different settings.

4.3.3. Comparison between Time Series, Regression and Classification Models

As soon as the training has been completed, an evaluation takes place by using the **accuracy()** R function [95, p. 4] for all models (Time Series, Regression, Classification). This function calculates error measures, like the ones discussed in chapter 2 – subsection 2.2.3 (MAE, RMSE, MAPE, MASE). In this case, two error measurements were calculated, the MAPE and MASE.

MAPE was used for the thermostats, while MASE for the light bulbs and lock devices. By using MAPE, it is easier to understand the prediction error, since it is presented as a percentage. On the other hand, as already discussed in chapter 2 – subsection 2.2.3, MAPE does not give accurate results for datasets that include values equal to zero or close to zero. This affects the light and lock devices, as many of their values are 0. Because of that, a different error measurement should be chosen. Specifically, MASE was selected in order to measure the prediction error of the light and lock devices, since it does not suffer from zero or infinite values compared to MAPE. A MASE value lower than one indicates that the specific algorithm generates better predictions than a naïve or a seasonal-naïve algorithm. On the other hand, a MASE value greater than 1 shows that a naïve/seasonal-naïve algorithm probably would perform best. Moreover, the algorithm with the lowest MASE value can be selected as the one producing less errors (see chapter 2, subsection 2.2.3).

The following table, Table 9, shows the Mean Absolute Percentage Error (MAPE) for the thermostats (th01 and th02), concerning the per second and per minute datasets. The purpose is to

compare the performance of Time Series to Regression models. By doing so, it can be determined which one performed best and for which dataset. It should be noted that the average (mean) MAPE value of all devices is presented in the table and the algorithm with the lowest percentage value has the least errors. Moreover, the lowest error value is indicated with a bold font. A table with the MAPE values for each device separately is available in Appendix 08.

Machine Learning Algorithms	Mean Values of MAPE (Per Second)	Mean Values of MAPE (Per Minute)
ETS	5.56%	5.55%
ARIMA	5.56%	5.49%
Boosted Decision Tree	7.76%	9.06%
Decision Forest	6.52%	10.17%
Fast Forest	6.77%	7.23%
Linear Regression	23.34%	25.03%
Bayesian Linear Regression	23.23%	23.32%
Neural Network	17.86%	23.41%

Table 9: Mean Values of Thermostats – MAPE.

From Table 9, it can be observed that there is not a large differentiation between ETS and ARIMA for both datasets. For the dataset with 1 second sample rate, the two algorithms have an identical MAPE of 5.56%. The reason why this happens can be traced back to the training of these models. Once the training of each ML model has been completed, the generated forecasted values have many decimal digits. For instance, the predicted values of a certain timeframe for the thermostats, regarding both ETS and ARIMA, are float numbers. However, thermostats can only accept integer values, thus, these numbers were rounded. This results in identical error measures. Nevertheless, in cases where the difference is greater than a few decimal points, the error measurements varies. This is visible to the per minute dataset, where the ARIMA model generated less errors than ETS. All things considered, the per minute dataset for ETS and ARIMA produced less errors than the per second one. Regarding the Regression models, it can be observed that the per second dataset performed better than the per minute. Still, the percentage error of Regression models is greater than ETS and ARIMA. Concluding, the model that performed best is ARIMA with a 5.49% error.

The next table, Table 10, shows a similar comparison as the previous one, but for the light and lock devices. As discussed earlier, these devices are compared separately from thermostats, since the error measure MAPE produces zero or infinite values when applied to datasets that include zero or close to zero values. For this reason, for both light and lock devices, the MAPE is replaced with MASE (Mean Absolute Scaled Error) error measurement. Furthermore, the lowest error values are indicated in a bold font.

Machine Learning Algorithms	Mean Values of MASE (Per Second)	Mean Values of MASE (Per Minute)
ETS	1744.76	31.94
ARIMA	1744.76	31.94
Decision Jungle	1729.54	33.24
Decision Forest	1782.52	32.04
Boosted Decision Tree	1711.18	34.78
Logistic Regression	2489.04	39.50
Support Vector Machine	1988.9	37.85
Locally Deep SVM	2061.70	38.67
Bayes Point Machine	2166.92	40.32
Neural Network	2130.20	39.77
Averaged Perceptron	2171.45	40.12

Table 10: Mean Values of Light Bulbs and Lock – MASE.

By examining Table 10, it can be observed that ETS and ARIMA models produced identical error values for all light and lock devices. The reason for this is the same as the one described for the thermostats, that forecasted values were rounded before calculating the errors. Furthermore, it can be noticed that the per minute dataset generated significantly fewer errors than the per second one for all ML models. Specifically, ETS and ARIMA models performed best with a 31.94 MASE error. Still, the MASE for all ML models is way above 1, which indicates that a naïve or a seasonal-naïve algorithm may be the better choice, compared to the tested ML models. Unfortunately, the time was not enough to conduct further experiments with such an algorithm. Nevertheless, a naïve and a seasonal-naïve model need to be implemented in future experiments.

Considering the results so far, a per minute dataset should be used along with an ARIMA ML model for the deployed web service. This decision was made because ARIMA performed best regarding the thermostats, th01 and th02, (see Table 9) and it is among the top-performed models for the remaining IoT devices, lb01, lb02, lb03, lb04 and lc01 (see Table 10). The same result can be reached by examining other error measures like MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error). These data are available in Appendix 08.

The following figure, Figure 33, depicts the performance of the trained ARIMA model in predicting the status of the second thermostat (th02) and the door lock (lc01) for the fourth day. The blue line represents the initial dataset and the red line is the fourth day prediction according to ARIMA.

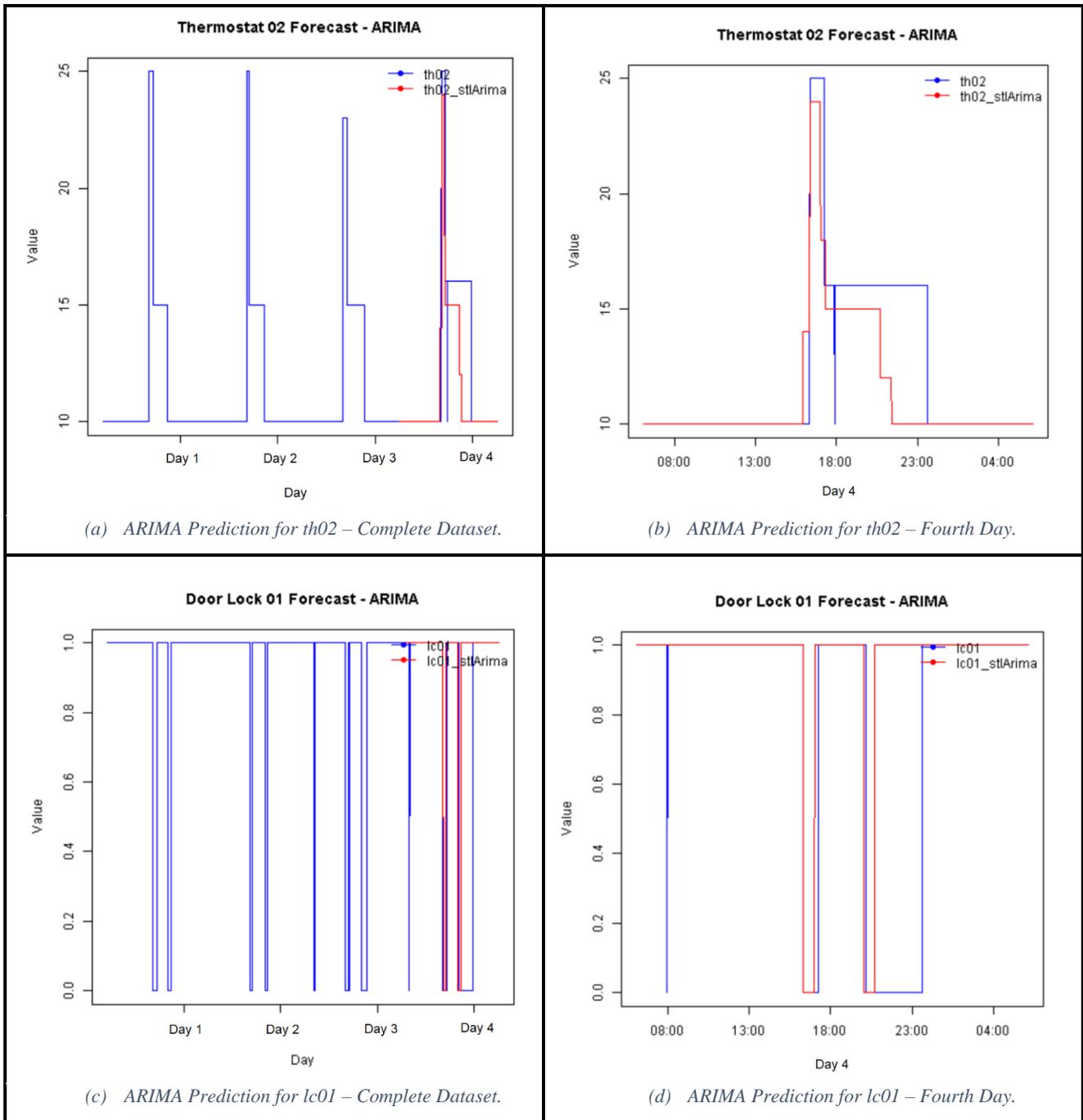


Figure 33: Azure ML Studio – ARIMA Prediction for th02 and lc01.

Figure 33 (a) illustrates, as a blue line, the gathered dataset per minute for the second thermostat, over the course of four working days. Essentially, it is the user’s preference of the heat in the kitchen. It can be seen that its lowest value is 10 and the highest 25. As a red line is represented the ARIMA predictions of the fourth day. A closer look of that day is visible at Figure 33 (b). In that figure, it can be observed the accuracy of ARIMA’s prediction for that day, by comparing the red line (ARIMA predictions) to the blue (user preferences). Similar observations can be made by looking Figure 33 (c) and (d) but for the door lock device.

As stated before, an R function was used to automatically train and produce ARIMA and ETS ML models. Moreover, the default settings were used for all Regression and Classification algorithms. If adjustments are made to Regression and Classification algorithm's settings, and a trial and error method is used to manual train Time Series models, the outcome could change and a different algorithm could perform best. Nevertheless, this is not the primary focus of this project, to find the best performing Machine Learning algorithm over time, but to prove that the proposed solution described in chapter 3 is feasible according to the available technologies discussed in chapter 2. On top of that, further experiments should be conducted on how the tested ML algorithms would behave in dataset with different sample rates (e.g. five minutes) and for a longer period of time (e.g. 2 weeks, instead of four days), including weekends. These experiments are going to be the focus of a future research.

4.4. System Prototype Deployment & Testing

By the end of section 4.3, the ARIMA model was proven to be the best performing one and was chosen to be deployed as a web service. However, during the deployment process, it was detected that in order for a Time Series algorithm to provide correct prediction results, it is needed to forecast values continuously. The time passed from training all the ML algorithms and evaluating them created a gap between the date/time of the training dataset and the deployment of the web service. This gap was enough to affect the predicted results. This problem can be overcome if the web service starts predicting values from the last observed date/time included in the training dataset. However, the time restrictions did not provide the flexibility to implement this. Therefore, it was decided to deploy a Regression/Classification model, which provides the flexibility of predicting values not bound in a continuous timeframe. It is important to state that, the Regression/Classification web service was deployed only to prove the proposed solution and the ARIMA should be used in a full-scale solution. The selected Regression and Classification models are the Regression Fast Forest for the thermostats and the Classification Decision Forest for the light bulbs and door lock device. An example of a deployed web service in Azure ML Studio and its building blocks is available in Appendix 06.

As soon as a web service has been deployed, Azure Machine Learning Studio offers tools that can be used to test it before using it. Figure 34 demonstrates this process. In Figure 34 (a), the "Test" button can be pressed and then a dialog box appears. Figure 34 (b) shows this dialog. By providing a date and a time, formatted as "yyyy-MM-ddTHH:mm:ss" (e.g. 2017-04-12T16:20:30), a prediction can be requested. Following that, the requested prediction is processed. Since a Request-Response Service (RSS) is used, the request is processed in real-time and almost immediately provides results. These are illustrated in Figure 34 (c) for all IoT devices in a JSON format.

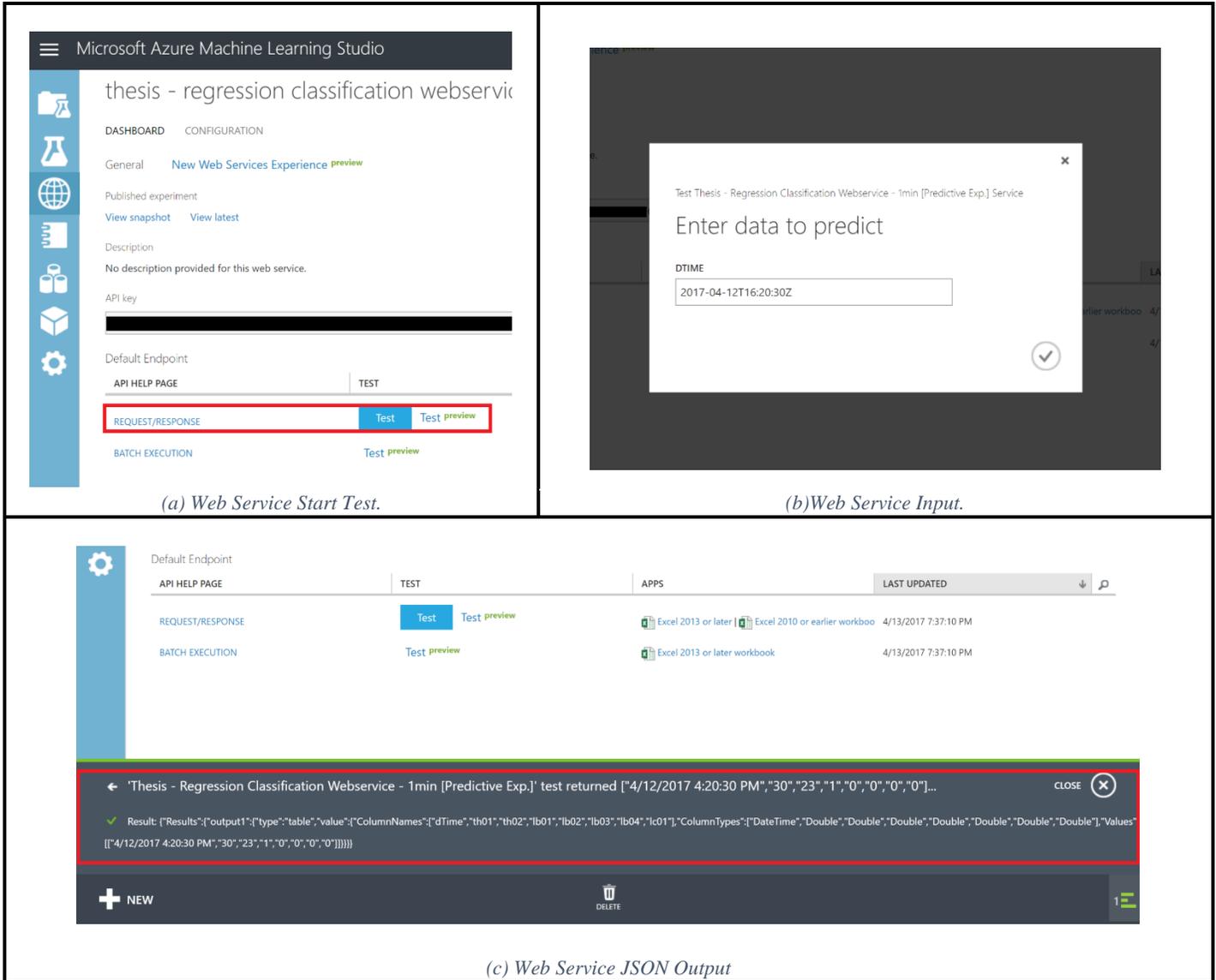


Figure 34: Azure ML Studio – Web Service.

4.4.1. Prototype Testing

Thus far, all the components of the proof-of-concept have been implemented. In this section, a test phase is documented on how the smartphone application interacts with the IoT devices. The starting point is illustrated in Figure 35, where the current status of all IoT devices can be seen. The focus is the lb03 device, which represents a light bulb in the bathroom. In Figure 35 the current status of lb03 device is Off. At a certain time within a day, the user wants to turn on the light in the bathroom. The end result is illustrated in Figure 36, in which the lb03 device is On.

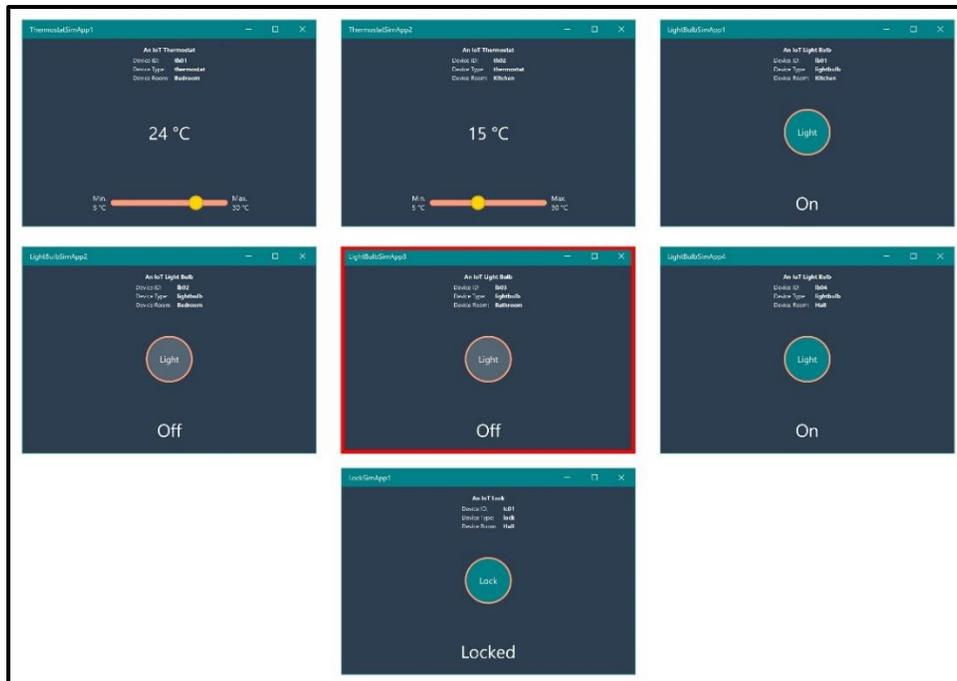


Figure 35: IoT Devices – Change lb03 Status 1.

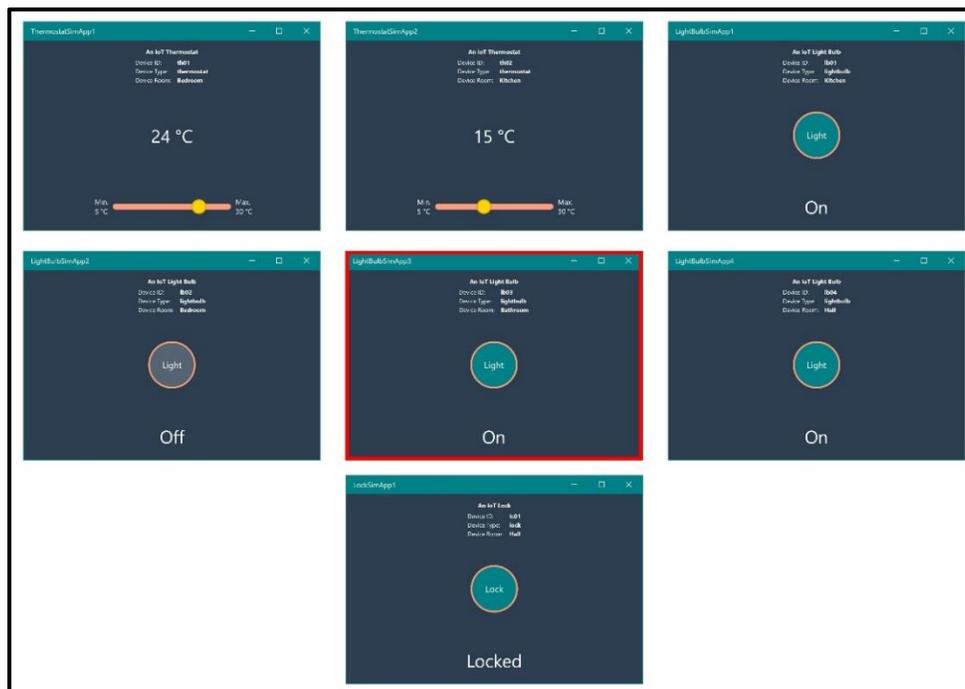


Figure 36: IoT Devices – Change lb03 Status 2.

In order to achieve this result, turning on the bathroom light, the user needs to use his/her smartphone. Figure 37 shows how the smartphone application is used by the user. At first, the user can see that the bathroom light is Off. Following that, he/she writes a command in natural language “Turn the lights on”. However, the user did not provide the room in which the light should turn

on, thus the application asks “*To which room?*?”. Then, the user responds with “*bathroom*” and the application gives its final response “*Turning on the lights in the bathroom!*”. Finally, the user can see now that the light in the bathroom is turned On.

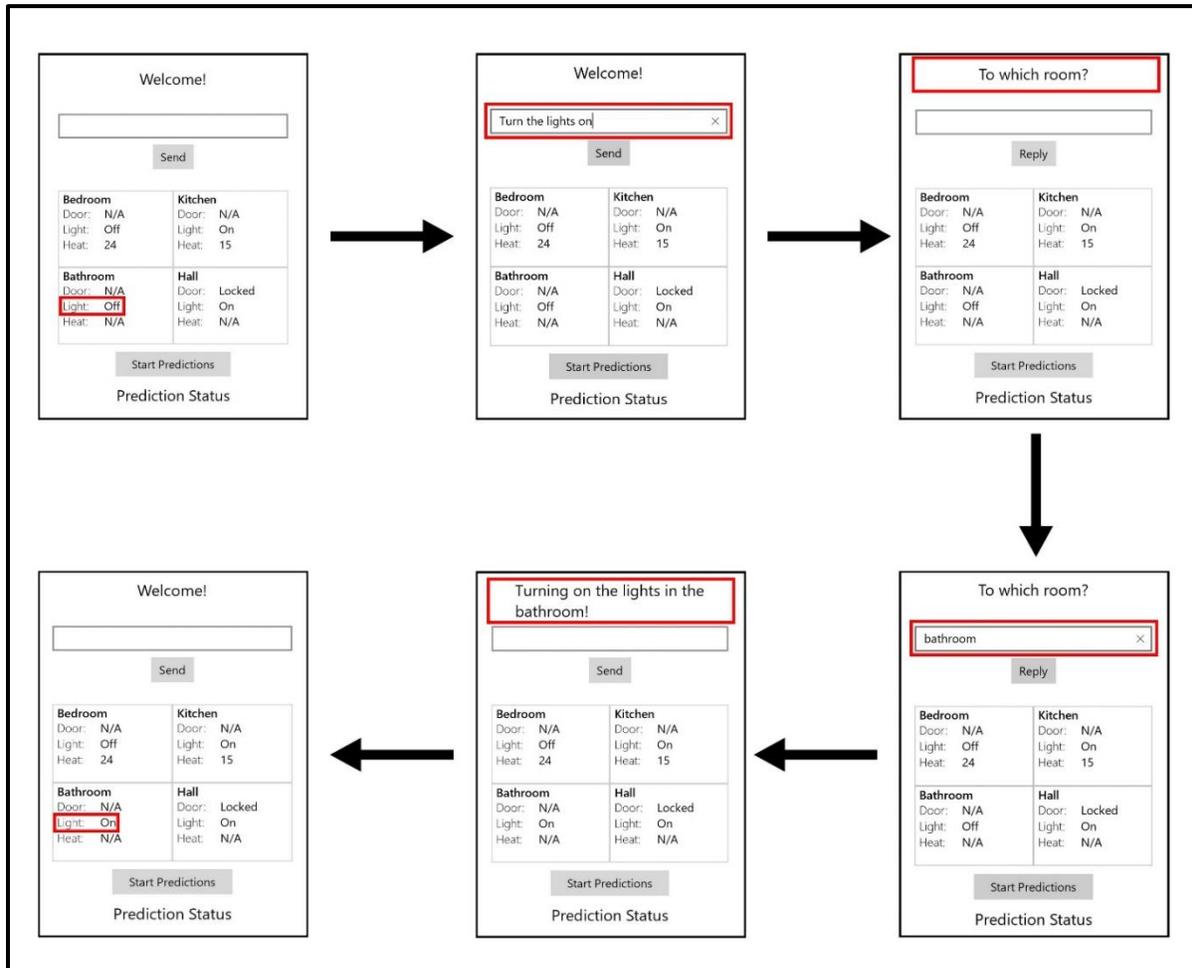


Figure 37: Smartphone Application – Change lb03 Status.

A similar interaction can take place between the user, the smartphone application and any other IoT device. Moreover, each IoT device stores its status in a database per minute over the course of four working days. As soon as the four days have passed, the gathered dataset is used to train the Regression and Classification ML models, Fast Forest and Decision Forest respectively. Once the ML models have finished training, the user can press the “*Start Predictions*” button in the smartphone application, Figure 38 – second red box, and the application starts requesting predictions from the web service. At the same figure, the first red box illustrates that a new prediction has been received and the application is adjusting the new values based on these forecasts. The third red box is included for developing purposes, in which the response from the web service can be seen in JSON format. This response contains the date and time the application requested a prediction for and the forecasted values of all IoT devices. Based on that response, the status of all IoT devices is adjusted.

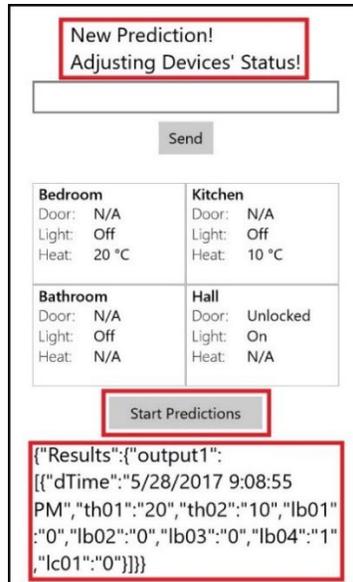


Figure 38: Smartphone Application – Change Devices' Status.

The following figure, Figure 39, illustrates the IoT devices and their status changed according to the previous JSON response.

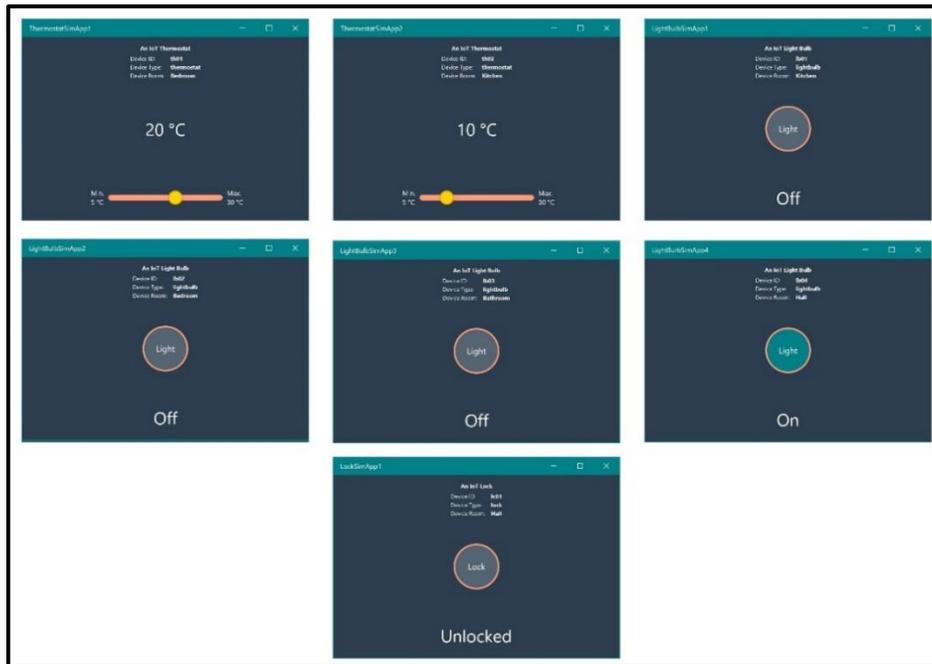


Figure 39: IoT Devices – Change Devices' Status.

This test concludes the Design and Implementation chapter, as its focus was to design a new solution for Smart Homes by taking a different approach, 2nd research sub-question, and implement a proof-of-concept based on that solution, 3rd research sub-question.

5. Conclusion

This project's main focus was to propose a Smart Home solution dedicated on providing a personalized service to its users. To achieve that, the proposed solution utilizes many technologies to identify and then, emulate a person's daily routines inside a house. The technologies used are the Internet of Things, Machine Learning and Cognitive Computing. This is also stated in the main research question:

How the Internet of Things, Machine Learning and Cognitive Computing technologies can be used to design and develop a unified personalized service in a Smart Home environment for a purely autonomous house?

To answer that question, a solution was proposed, designed and developed in this project that utilizes all the aforementioned technologies. A summary of that solution follows.

The inhabitant of a house can use a smartphone application from which all IoT devices installed in the house can be controlled. The interaction between the Smart Home user and the IoT devices is based on voice or text commands in natural language. The application utilizes the Cloud to understand the user commands (Cognitive Computing) and then, it executes them by adjusting the values of the IoT devices through a Cloud-based IoT Hub. The Machine Learning comes into play, when a certain amount of historic data has been gathered through the interactions between the user, the smartphone application and the IoT devices. This dataset has a timestamp, which indicates the date and time the user performed an action. Based on this dataset, several Supervised Machine Learning models were trained, tested and evaluated in order to find the one which performed best under these conditions. Following that, the best performing ML model was deployed as a web service. The smartphone application can contact the aforementioned web service to request future predictions. Finally, these predictions are used to automatically adjust the value of all IoT devices without user intervention.

5.1. Discussion

In order to design and implement a Smart Home solution as the one described earlier, a research was necessary on the State-of-the-Art of existing technologies and Smart Home solutions (see chapter 2). First, the IoT area was investigated. During that research, two different types of IoT devices were detected, the resource-rich and resource-constrained. Moreover, the communication technologies (e.g. Wi-Fi, Bluetooth, ZigBee, etc.) and protocols (e.g. MQTT, AMQP, etc.) currently used in IoT services were examined. Last but not least, the location where the entire computing of an IoT solution is implemented was explored (e.g. microprocessor, the Cloud, etc.). All this information is essential, as different design choices can eliminate some of the IoT device types, communication technologies, protocols and processing units. In the proposed solution, the resource-rich IoT devices were selected along with a Wi-Fi connection over the AMQP protocol (chapters 3 and 4).

Following IoT, the Machine Learning field was researched and especially the Supervised Learning techniques (see chapter 2). However, since the data used in the proposed solution are dependent

on time, Time Series Forecasting techniques were also studied (see chapter 2). During that research, several Regression, Classification and Time Series Forecasting algorithms were detected. These were later used in chapter 4 to train different ML models. Furthermore, after evaluating each and compare its results to other models, the best performing one was selected for a web service deployment. Although the overall best performing algorithm was proven to be the Time Series ARIMA, it was not the one to be deployed due to time restrictions applied on this project. Nevertheless, the time was enough to deploy a Regression Fast Forest model for the thermostats and a Classification Decision Forest model for the light bulbs and door lock device. Having an ML model deployed as a web service, the application can request future predictions for all the installed IoT devices in the house and adjust their value accordingly. Moreover, the Cognitive Computing services were researched (chapter 2) in order to realize their capabilities and use them in the smartphone application. From all the provided services, the Text-to-Speech, Speech-to-Text and Natural Language Understanding were considered in the full-scale solution (chapter 3), yet the time was enough for implementing only the Natural Language Understanding service (chapter 4).

Another important field needed investigation was the Cloud Platforms (chapter 2), which can provide services needed for designing and implementing the proposed solution. While several exist, only three were researched (Amazon Web Services, IBM Bluemix, Microsoft Azure), because of the limited time, and only one was chosen; Microsoft Azure. While many reasons exist to justify this choice, the main one was its Machine Learning Studio versatility, adaptivity and flexibility (see chapter 4, section 4.1). Following the investigation of existing technologies, Smart Home solutions were also researched in chapter 2. There, different products and services were examined, from commercial ones, currently available in the market, to services proposed by researchers. These were taken under consideration when designing this project's proposed solution (chapters 3 and 4).

By the end of chapter 2, the first research sub question was answered:

What is the current State-of-the-Art in Smart Homes considering the IoT, ML and Cognitive Computing?

Chapter 3 was focused on analyzing the results from chapter 2 and together with chapter 4, they were responsible for providing an answer to the second research sub question:

How a new solution can be designed for Smart Homes to achieve a self-operated house?

The rest of the 4th chapter was emphasized on the third and final research sub question:

How a proof of concept can be developed based on this new solution for Smart Homes?

By answering all the research sub questions, the main one can be resolved:

How the Internet of Things, Machine Learning and Cognitive Computing technologies can be used to design and develop a unified personalized service in a Smart Home environment for a purely autonomous house?

5.2. Future Recommendations

This section focuses on suggestions that are interesting and going to be looked upon in the future. These recommendations include aspects of the project which were not considered in the design of the full solution, neither in the implementation of the prototype, due to time constraints. On top of that, some recommendations are made which could improve the whole proposed solution and the provided services.

Internet Connectivity, Communication Technologies and Protocols

Starting with the delimitations stated in chapter 1 - section 1.3, a Smart Home solution that requires internet even for minor operations might not be ideal. The reason for that is the absence of internet connectivity that might occur within a day. Thus, it should be considered how the proposed solution could function without internet connection. A solution to this problem would be to control all IoT devices manually as well and save the data generated from this interaction into the local memory of each device. When the internet connectivity is restored, the saved interaction can be sent to the Cloud. By doing so, the user can control home devices even without internet connectivity and still, the Machine Learning service would have the required dataset to train an ML model. Furthermore, experiments with physical IoT devices should be performed, instead of the simulated applications. These experiments will provide insights on the performance of the chosen communication technologies (Wi-Fi) and protocols (AMQP). Thus, from these experiments it can be decided if the selected technologies and protocols are ideal, or if others should be considered.

Security and Privacy

Since a lot of personal data about the user are stored in the Cloud, the security and privacy are of concern. Even though many communication technologies and protocols provide their own security measures, a thorough examination of the proposed solution should be conducted. Through this, all security risks should be identified and propose countermeasures. Some of them would be the end-to-end encryption of data, from the Cloud to the smartphone application and every IoT device, and from the smartphone application and each IoT device to the Cloud. Another vulnerable point is located in the encryption key exchange between entities, used for encrypting the communication channel. No matter how strong the encryption is, if the key used to encrypt the data is discovered by an attacker, the whole system becomes defenseless. While methods exist on how to exchange encryption keys, further investigation is needed to choose the best one, or create a new one from the ground up.

Moreover, the user should be able to see what kind of data are collected by the proposed solution and at any time he/she could delete those or stop providing data to the service. Of course, this will affect the services provided to the user and he/she needs to be notified about that. Last but not least, the proposed solution must comply with the European General Data Protection Regulation [90], since it will be forced across Europe by 25th of May, 2018 [96].

IoT Devices

In chapter 1, it was assumed that all IoT devices are already installed in a house. However, the installation process of registering a new home device under a specific user and his/her house should also be considered. On top of that, this process must be secure enough in order to avoid irrelevant people from installing and registering new devices. Additionally, the service built during this project used IoT devices fixed in number and type. In the future, it should be researched how the service could become more flexible and include several home devices and diverse types. Following that, this service should provide an easy way to the user of adding or removing IoT devices from his/her account and house. These changes probably will affect the implemented Machine Learning service, since it was also built upon a fixed number of home devices and types. Thus, it should be considered how the ML service can be adapted into these changes.

Dataset

Considering the Machine Learning experiments conducted in chapter 4, the dataset provided for training all the ML models was limited to a timeframe of 4 working days. Within each day, an observation was made every second or every minute. Nevertheless, the gathered data were enough to identify user routines and eventually emulate them, which also proves the feasibility of the proposed solution. However, a larger dataset (e.g. of 14 days) with a different sample rate (e.g. 5 minutes) could provide necessary information for training more accurate ML models, thus providing better predictions. As a result, several experiments can follow with diverse datasets in order to find the optimal data acquisition parameters, i.e. observed days and sample rate.

Furthermore, it would be interesting to see how extra inputs in the dataset, during the training of ML models, could affect the accuracy of future predictions. In the implemented Machine Learning service, the training of all ML models was based on one IoT device at a time. However, further experiments should be conducted on training ML models per room status or even the entire house. This means that in order to predict the status of a single home device, if the current status of all devices included in the same room or even the house is provided, it could generate more accurate predictions. Moreover, if weather data are also incorporated in the dataset along with the user/house location, it could enhance the precision of the forecasts even more. The weather data can provide information about the outside temperature, which can be helpful especially for the devices responsible for controlling the heat. The user/house location can be used to set up a geofence around the house and track when the user leaves from or arrives at it. This information could really help devices such as door locks and lights. These are experiments that should be conducted in the future to determine the ideal input data during the training of ML models.

Last but not least, the proposed solution was designed around a dataset that includes preferences for a single user. Further investigation is needed on how this solution would behave if more than one people live in the same house. This would be challenging, as the preferences of one user may contradict the preferences of the other, but also interesting to implement a service capable of considering the preferences of all users inside a house.

Machine Learning Service

All the ML algorithms used in this report were initialized with their default settings. However, further research is needed in order to examine if all algorithms can perform better with adjusted settings and if some, which previously did not perform well, could provide better predictions. Moreover, naïve and seasonal-naïve algorithms need to be considered as well, since the results from the Machine Learning experiments (chapter 4, section 4.3) indicated that could provide more accurate predictions. Furthermore, an enhancement of the ML service would be to allow the deployed ML model to retrain itself with new data. This can be achieved with the help of the smartphone application. If the application detects that the provided predictions contradict the user preferences, it could request the ML web service to retrain the ML model. An idea of how this can be implemented is presented next.

Smartphone Application

A welcome enhancement of the proposed solution would be the possibility to retrain the deployed ML model as described earlier. This can be proven beneficial as the changing of the environmental temperature and the natural sunlight can change according to the current season. This change can influence the preferences of the user. For instance, during summer time, the user would like a cooler home and would turn on the house lights later in time. The smartphone application should be able to detect these changes, as user preferences, and should provide the ML model with new data to retrain itself. By doing so, the proposed solution could become more flexible and adjust its future predictions accordingly. Finally, the smartphone's UI and UX should be tested in collaboration with end users and adjustments must be made according to their feedback.

6. References

- [1] R. D. Minerva Roberto, Biru Abyi, “Towards a definition of the Internet of Things (IoT),” *IEEE Internet Initiat.*, 2015.
- [2] A. Attwood, O. Abuelmatti, and P. Fergus, “M2M Rendezvous Redundancy for the Internet of Things,” in *2013 Sixth International Conference on Developments in eSystems Engineering*, 2013, pp. 46–50.
- [3] F. M. Al-Turjman, “Towards smart ehealth in the ultra large-scale Internet of Things era,” in *2016 23rd Iranian Conference on Biomedical Engineering and 2016 1st International Iranian Conference on Biomedical Engineering (ICBME)*, 2016, pp. 102–105.
- [4] E. Wu, P. Zhang, T. Lu, H. Gu, and N. Gu, “Behavior prediction using an improved Hidden Markov Model to support people with disabilities in smart homes,” in *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2016, pp. 560–565.
- [5] Google, “Google Home – Made by Google.” [Online]. Available: <https://madeby.google.com/home/>. [Accessed: 14-Apr-2017].
- [6] Amazon, “Echo & Alexa - Amazon Devices - Amazon Official Site.” [Online]. Available: <https://www.amazon.com/b/?ie=UTF8&node=9818047011>. [Accessed: 14-Apr-2017].
- [7] Google, “Features – Google Home – Made by Google.” [Online]. Available: <https://madeby.google.com/home/features/#?filters=home>. [Accessed: 14-Apr-2017].
- [8] Google, “Google Assistant - Your own personal Google.” [Online]. Available: <https://assistant.google.com/>. [Accessed: 14-Apr-2017].
- [9] J. E. Kelly Iii, “Computing, cognition and the future of knowing. How humans and machines are forging a new age of understanding,” 2015.
- [10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [11] M. Sain, Y. J. Kang, and H. J. Lee, “Survey on security in Internet of Things: State of the art and challenges,” in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, 2017, pp. 699–704.
- [12] IEEE, *IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, vol. 2012, no. March. 2012.
- [13] W. Sun, M. Choi, S. Choi, W. Sun, M. Choi, and S. Choi, “IEEE 802.11ah: A Long Range 802.11 WLAN at Sub 1 GHz,” *J. ICT Stand.*, vol. 1, pp. 83–108.
- [14] I. C. Society, *IEEE Standard for Low-Rate Wireless Personal Area Networks (WPANs)*, vol. 2015. 2015.

- [15] Bluetooth SIG, “BLUETOOTH CORE SPECIFICATION Version 5.0 Vol 0,” vol. 0, no. December, p. 154, 2016.
- [16] ITU, “Recommendation ITU-T G.9959 Short range narrow-band digital radiocommunication transceivers – PHY, MAC, SAR and LLC layer specifications,” 2015.
- [17] Z. Alliance, “ZIGBEE SPECIFICATION,” 2008.
- [18] Z. Shelby, K. Hartke, and B. C., “RFC 7252 - The Constrained Application Protocol CoAP,” 2014.
- [19] A. Banks and R. Gupta, “MQTT Version 3.1.1 Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Latest version: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>,” *OASIS Stand.*, no. October, p. 81, 2014.
- [20] “OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. 29 October 2012. OASIS Standard. <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf>,” *OASIS Stand.*, 2012.
- [21] OMG, “Data Distribution Service (DDS),” no. April, pp. 1–20, 2015.
- [22] Arduino, “Arduino.” [Online]. Available: <https://www.arduino.cc/>. [Accessed: 24-Apr-2017].
- [23] Raspberry, “Raspberry Pi - Teach, Learn, and Make with Raspberry Pi.” [Online]. Available: <https://www.raspberrypi.org/>. [Accessed: 24-Apr-2017].
- [24] Intel, “The Intel® Galileo Board | IoT | Intel® Software.” [Online]. Available: <https://software.intel.com/en-us/iot/hardware/galileo>. [Accessed: 24-Apr-2017].
- [25] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed. MIT Press, 2014.
- [26] Bishop M. Christopher, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [27] Russel J. Stuart and Norvig Peter, *Artificial Intelligence A Modern Approach*. .
- [28] Microsoft, “What is Machine Learning on Azure?,” 2016.
- [29] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. Melbourne, Australia: OTexts, 2013.
- [30] V. Raskin, “Semantics as the basis of truly cognitive computing,” in *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMII)*, 2017, pp. 000011–000014.
- [31] E. Khan, “Machine Learning Algorithms for Natural Language Semantics and Cognitive Computing,” in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1146–1151.
- [32] W. Voorsluys, J. Broberg, and R. Buyya, “Introduction to Cloud Computing,” in *Cloud Computing*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011, pp. 1–41.
- [33] Microsoft, “What is SaaS? Software as a Service | Microsoft Azure.” [Online]. Available:

- <https://azure.microsoft.com/en-us/overview/what-is-saas/>. [Accessed: 25-Apr-2017].
- [34] Microsoft, “Microsoft Azure: Cloud Computing Platform.” [Online]. Available: <https://azure.microsoft.com/en-us/>. [Accessed: 07-May-2017].
- [35] Microsoft, “Azure IoT Suite | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/suites/iot-suite/>. [Accessed: 07-May-2017].
- [36] Microsoft, “Azure solutions for Internet of Things (IoT Suite) Microsoft Docs.”
- [37] Microsoft, “Microsoft Azure Machine Learning Studio.” [Online]. Available: <https://studio.azureml.net/>. [Accessed: 07-May-2017].
- [38] Microsoft, “Microsoft Cognitive Services.” [Online]. Available: <https://www.microsoft.com/cognitive-services>. [Accessed: 07-May-2017].
- [39] Microsoft, “What is Cognitive Services?”
- [40] Microsoft, “LUIS: Homepage.” [Online]. Available: <https://www.luis.ai/home/index>. [Accessed: 07-May-2017].
- [41] B. W. Schermer, “Software agents, surveillance, and the right to privacy: a legislative framework for agent-enabled surveillance,” Laiden University, 2007.
- [42] Microsoft, “Pricing Overview - How Azure pricing works | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/pricing/>. [Accessed: 07-May-2017].
- [43] Microsoft, “Pricing - SQL Database | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/sql-database/>. [Accessed: 07-May-2017].
- [44] Microsoft, “Pricing—IoT Hub | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/iot-hub/>. [Accessed: 07-May-2017].
- [45] Microsoft, “Pricing - Machine Learning | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/machine-learning/>. [Accessed: 07-May-2017].
- [46] Microsoft, “Pricing - Language Understanding Intelligent Services API | Microsoft Azure.” [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/cognitive-services/language-understanding-intelligent-services/>. [Accessed: 07-May-2017].
- [47] Microsoft, “Microsoft Cognitive Services - Bing Speech API.” [Online]. Available: <https://www.microsoft.com/cognitive-services/en-us/speech-api>. [Accessed: 07-May-2017].
- [48] IBM, “Getting started with Watson IoT Platform.” [Online]. Available: https://console.ng.bluemix.net/docs/services/IoT/iotplatform_task.html#iotplatform_task. [Accessed: 07-May-2017].
- [49] IBM, “Conversation | IBM Watson Developer Cloud Doc.” [Online]. Available: <https://www.ibm.com/watson/developercloud/doc/conversation/index.html>. [Accessed: 07-May-2017].
- [50] IBM, “Speech to Text | About Speech to Text | IBM Watson Developer Cloud Doc.”

- [Online]. Available: <https://www.ibm.com/watson/developercloud/doc/speech-to-text/index.html>. [Accessed: 07-May-2017].
- [51] IBM, “Text to Speech | About Text to Speech | IBM Watson Developer Cloud Doc.” [Online]. Available: <https://www.ibm.com/watson/developercloud/doc/text-to-speech/index.html>. [Accessed: 07-May-2017].
- [52] IBM, “Getting started with Machine Learning.” [Online]. Available: <https://console.ng.bluemix.net/docs/services/PredictiveModeling/index.html?pos=2>. [Accessed: 07-May-2017].
- [53] IBM, “IBM Knowledge Center - Database Source Node.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.modeler.help/database_overview.htm. [Accessed: 08-May-2017].
- [54] IBM, “IBM Knowledge Center - Modeling Nodes.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/kc_gen/com.ibm.spss.modeler.help_com.ibm.spss.modeler.help_toc-gen25.html. [Accessed: 08-May-2017].
- [55] IBM, “IBM Watson IoT - Watson IoT Platform Pricing.” [Online]. Available: <https://www.ibm.com/internet-of-things/platform/pricing/>. [Accessed: 25-Apr-2017].
- [56] IBM, “IBM Cloudant for Bluemix Public - New Pricing Model.” [Online]. Available: <https://www.ibm.com/blogs/bluemix/2016/07/cloudant-public-improved-pricing-model/>. [Accessed: 07-May-2017].
- [57] IBM, “Conversation | IBM Watson Developer Cloud.” [Online]. Available: <https://www.ibm.com/watson/developercloud/conversation.html#pricing-block>. [Accessed: 07-Apr-2017].
- [58] IBM, “Speech to Text | IBM Watson Developer Cloud.” [Online]. Available: <https://www.ibm.com/watson/developercloud/speech-to-text.html#pricing-block>. [Accessed: 06-Apr-2017].
- [59] IBM, “Text to Speech | IBM Watson Developer Cloud.” [Online]. Available: <https://www.ibm.com/watson/developercloud/text-to-speech.html>. [Accessed: 06-Apr-2017].
- [60] IBM, “IBM SPSS Modeler.” [Online]. Available: <https://www.ibm.com/dk-en/marketplace/spss-modeler>. [Accessed: 08-May-2017].
- [61] Amazon, “AWS IoT - Developer Guide.”
- [62] Amazon, “AWS Free Tier.” [Online]. Available: <https://aws.amazon.com/free/>. [Accessed: 08-May-2017].
- [63] Amazon, “Amazon Lex - Developer Guide.”
- [64] Amazon, “Amazon Polly - Developer Guide.”
- [65] Amazon, “Amazon Machine Learning - Developer Guide.”
- [66] Amazon, “Amazon Machine Learning - API Reference.”

- [67] Amazon, “Amazon Web Services (AWS) - Cloud Computing Services.” [Online]. Available: <https://aws.amazon.com/>. [Accessed: 08-May-2017].
- [68] Samsung, “SmartThings. Add a little smartness to your things.” [Online]. Available: <https://www.smarthings.com/>. [Accessed: 08-May-2017].
- [69] Bosch, “Smart Home | Bosch Global.” [Online]. Available: <https://www.bosch.com/products-and-services/connected-products-and-services/smart-home/>. [Accessed: 08-May-2017].
- [70] Honeywell, “Honeywell | Your Home.” [Online]. Available: <https://yourhome.honeywell.com/>. [Accessed: 04-Jun-2017].
- [71] Samsung, “SmartThings Support.” [Online]. Available: <https://support.smarthings.com/hc/en-us>. [Accessed: 08-May-2017].
- [72] Samsung, “SmartThings - Products.” [Online]. Available: <https://www.smarthings.com/products>. [Accessed: 08-May-2017].
- [73] Samsung, “SmartThings - Hub.” [Online]. Available: <https://www.smarthings.com/products/samsung-smarthings-hub>. [Accessed: 08-May-2017].
- [74] Samsung, “Routine Defaults – SmartThings Support.” [Online]. Available: <https://support.smarthings.com/hc/en-us/articles/210613803-Routine-Defaults>. [Accessed: 08-May-2017].
- [75] Samsung, “A Guide to Wireless Range & Repeaters.” [Online]. Available: <https://blog.smarthings.com/iot101/a-guide-to-wireless-range-repeaters/>. [Accessed: 12-May-2017].
- [76] Nest, “Meet the Nest Learning Thermostat | Nest.” [Online]. Available: <https://nest.com/thermostat/meet-nest-thermostat/>. [Accessed: 08-May-2017].
- [77] Philips, “Philips Hue.” [Online]. Available: <http://www2.meethue.com/en-gb/>. [Accessed: 04-Jun-2017].
- [78] Withings, “Withings.” [Online]. Available: <https://www.withings.com/eu/en/>. [Accessed: 04-Jun-2017].
- [79] Nest, “Nest Learning Thermostat - Install & Explore | Nest.” [Online]. Available: <https://nest.com/thermostat/install-and-explore/>. [Accessed: 08-May-2017].
- [80] Nest, “Works With Nest.” [Online]. Available: <https://store.nest.com/works-with-nest/>. [Accessed: 08-May-2017].
- [81] Nest, “Nest Labs Introduces World’s First Learning Thermostat | Nest.” [Online]. Available: <https://nest.com/fr/press/nest-labs-introduces-worlds-first-learning-thermostat/>. [Accessed: 08-May-2017].
- [82] Nest, “Using multiple Nest Learning Thermostats in the same home.” [Online]. Available: <https://nest.com/support/article/How-does-Nest-work-if-I-have-multiple-Nest-Learning-Thermostats-in-the-same-home>. [Accessed: 08-May-2017].

- [83] M. Dimaggio, F. Leotta, M. Mecella, and D. Sora, "Process-Based Habit Mining: Experiments and Techniques," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCCom/IoP/SmartWorld)*, 2016, pp. 145–152.
- [84] G. Xu, M. Liu, F. Li, F. Zhang, and W. Shen, "User behavior prediction model for smart home using parallelized neural network algorithm," in *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2016, pp. 221–226.
- [85] Samsung, "Local processing." [Online]. Available: <https://support.smarthings.com/hc/en-us/articles/209979766-Local-processing>. [Accessed: 12-May-2017].
- [86] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home IoT devices," *2015 IEEE 11th Int. Conf. Wirel. Mob. Comput. Netw. Commun. WiMob 2015*, pp. 163–167, 2015.
- [87] I. Sommerville, *Software engineering*. Addison-Wesley, 2007.
- [88] Agile Business, "MoSCoW Prioritisation | Agile Business Consortium," 2014. [Online]. Available: <https://www.agilebusiness.org/content/moscow-prioritisation>. [Accessed: 13-May-2017].
- [89] M. Burnett and D. Kleiman, *Perfect Passwords*. Technical Editor, 2006.
- [90] European Parliament, "General Data Protection Regulation," *EUR Lex*, vol. 11, no. April, pp. 1–341, 2015.
- [91] WampServer, "WampServer, la plate-forme de développement Web sous Windows - Apache, MySQL, PHP." [Online]. Available: <http://www.wampserver.com/en/>. [Accessed: 21-May-2017].
- [92] Apache, "Welcome! - The Apache HTTP Server Project." [Online]. Available: <https://httpd.apache.org/>. [Accessed: 21-May-2017].
- [93] MySQL, "MySQL." [Online]. Available: <https://www.mysql.com/>. [Accessed: 21-May-2017].
- [94] Microsoft, "What's a Universal Windows Platform (UWP) app?"
- [95] R. Hyndman, M. O'Hara-Wild, C. Bergmeir, S. Razbash, and E. Wang, "Forecasting Functions for Time Series and Linear Models," 2017.
- [96] "Home Page of EU GDPR." [Online]. Available: <http://www.eugdpr.org/>. [Accessed: 02-Jun-2017].