



IoT Solution for Rental Houses

Nuno Afonso Rebelo Patrício Freire dos Santos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Alberto Manuel Ramos da Cunha

Examination Committee

Chairperson: Prof. Manuel Fernando Cabido Peres Lopes
Supervisor: Prof. Alberto Manuel Ramos da Cunha
Member of the Committee: Prof. Miguel Filipe Leitão Parda

November 2021

Declaration I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

This work was created using \LaTeX typesetting language in the Overleaf environment (www.overleaf.com).

Acknowledgments

The study, design, implementation and writing of this Dissertation required investment. For this reason, I would like to thank Professor Renato Nunes for his support during this procedure, for this challenge, for his knowledge and expertise, which lead to the development of this Thesis. I would also like thank Professor Alberto Cunha for his guidance and recommendations.

I would also like to thank my parents for their friendship and support during this stage of my life and over these years. You are a pillar in my life and you have always been there for me.

To my grandparents, to my aunt and to my brother, for their care, for helping me surpass the obstacles in life and for backing me up.

Additionally, I would also like thank to Joaquim and Ricardo, for their friendship and their insights.

And last, to Ana. For all the late hours, for pushing me to do more and better and for always believing in me.

To each and every one of you, I would like to show you my deepest and most sincere appreciation. "Obrigado".

Abstract

The tourism growth led to an increase of short-term accommodation. However, the management of a property may be time consuming and difficult to achieve. This paper proposes a domotic system to assist people in performing house management tasks, integrating a smart lock, utility monitoring and appliance control in a smart home system.

The proposed solution uses various smart devices that are connected to sensors and actuators, being able to monitor and control a house's environment. Those devices use a Wi-Fi network to communicate to a home server, which is responsible for processing the data. Additionally, it offers a web interface to users, and connects to a central server that allows the management of multiple rental houses from a single point.

A prototype was developed that uses ESP8266 boards as smart devices. For the home server a Raspberry Pi was chosen.

The system was tested, in which users could send commands through the Raspberry Pi to the ESP nodes, in order to change an environment's property. Additionally, the introduction of a cache allowed a reduction of the message processing time by one eighth, when compared to the time it took for the server to access its internal storage.

Using the prototype, it was possible to conclude that the proposed system provides significant assistance in the management of short-term accommodation businesses.

Keywords

Smart House Rental; House Management; IoT; Smart Home; Smart Device; ESP8266.

Resumo

O aumento do turismo potenciou o aumento do alojamento de curta duração. No entanto, a gestão de um imóvel, neste contexto, pode apresentar algumas dificuldades. Este artigo propõe um sistema de Internet das Coisas para auxiliar pessoas a executar estas tarefas, integrando uma fechadura inteligente e dispositivos de monitorização de utilidades num sistema domótico.

A solução proposta utiliza vários dispositivos inteligentes que estão ligados a sensores e actuadores, sendo capaz de monitorizar e controlar o ambiente de uma casa. Estes dispositivos utilizam uma rede Wi-Fi para comunicar com um servidor doméstico, que é responsável pelo processamento dos dados. Além disso, este oferece uma interface web aos utilizadores e conecta-se a um servidor central, permitindo a gestão de uma ou mais propriedades através da Internet.

Foi desenvolvido um protótipo, utilizando um dispositivo inteligente, baseado no módulo ESP8266 e um Raspberry Pi para hospedar a aplicação do servidor doméstico.

O sistema foi testado pedindo a um grupo de utilizadores para aceder ao sistema, enviando comandos através da aplicação para os nós ESP. Além disso, a introdução da cache permitiu uma redução do tempo de processamento das mensagens em um oitavo, quando comparado com o tempo que o servidor levaria a aceder ao armazenamento interno.

Foi portanto possível concluir que o sistema proposto auxilia a gestão de alojamentos de curta duração.

Palavras Chave

Aluguer de Casa Inteligente; Gestão de Casas; IoT; Casa Inteligente; Dispositivo Inteligente; ESP8266.

Contents

1	Introduction	1
1.1	Motivation	3
1.2	Objectives	4
1.3	Document's Structure	5
2	Background	7
2.1	Protocols	9
2.1.1	Wireless Communication Protocols	9
2.1.1.A	Bluetooth Low Energy	9
2.1.1.B	Wireless Fidelity	10
2.1.1.C	ZigBee	11
2.1.1.D	Z-Wave	11
2.1.1.E	Synthesis	12
2.1.2	Application Layer Protocols	12
2.1.2.A	Hypertext Transfer Protocol	12
2.1.2.B	Constrained Application Protocol	13
2.1.2.C	Message Queuing Telemetry Transport	14
2.1.2.D	WebSocket Protocol	14
2.1.2.E	Transport Layer Security	15
2.1.3	Synthesis	16
2.2	Academic Projects	17
2.2.1	Door Authentication	18
2.2.1.A	Smart digital door lock for the home automation	18
2.2.1.B	Internet of Things Cyber Security: Smart Door Lock System	18
2.2.2	Utility Monitoring	19
2.2.2.A	IOT based application for monitoring electricity power consumption in home appliances	19
2.2.2.B	IoT Water Consumption Monitoring & Alert System	20

2.2.3	Synthesis	20
2.3	Products	21
2.3.1	Smart Devices	21
2.3.1.A	Door Authentication	21
2.3.1.B	Utility Monitoring	22
2.3.2	Controllers	22
2.3.2.A	NodeMCU	22
2.3.2.B	Raspberry Pi 3 B+	23
2.3.3	Integration Platforms	24
2.3.3.A	openHAB	24
2.3.3.B	Home Assistant	24
2.3.4	Synthesis	25
2.4	Summary	26
3	Architecture	27
3.1	Requirements	29
3.1.1	Reservation Management	29
3.1.2	Home Automation	30
3.1.3	Communication	31
3.2	System Entities	31
3.2.1	Smart Device	32
3.2.2	Smart Controller	33
3.2.3	Proxy	33
3.2.4	Gateway	33
3.2.5	Router	34
3.2.6	Home Server and Database	34
3.2.7	Cloud Server and Database	35
3.2.8	Web Browser	36
3.3	System Architecture	36
3.4	Communications	37
3.4.1	Controller-Server Connection	38
3.4.2	Inter-Server Connection	39
3.5	Summary	40
4	System Implementation	41
4.1	Smart Controllers	43
4.1.1	Smart Door	44

4.1.1.A	Components	45
4.1.1.B	Operation	45
4.1.1.C	Distributed Features	46
4.1.2	Smart Light	47
4.1.2.A	Components	47
4.1.2.B	Operation	48
4.1.2.C	Distributed Features	48
4.2	Home and Remote Servers	49
4.2.1	Reservation Management Tasks	50
4.2.1.A	House	51
4.2.1.B	User	51
4.2.1.C	Reservation	52
4.2.2	Home Automation Tasks	53
4.2.2.A	DeviceType and Properties	53
4.2.2.B	SmartDevice	54
4.2.2.C	Action and Event	54
4.2.3	Controller-Server Communication	55
4.2.3.A	SmartController	55
4.2.3.B	ControllerNotification	56
4.2.3.C	SmartDevice Caching Task	56
4.2.4	User-Local Server Communication	57
4.2.4.A	HTTP Connection	57
4.2.4.B	WebSocket Connection	57
4.2.5	Inter-Server Communication	58
4.2.5.A	Server	58
4.2.5.B	ServerNotification	59
4.3	Summary	60
5	Evaluation and Metrics	63
5.1	Automatic Lock Code Change	65
5.2	System Monitoring	65
5.3	Database Size	66
5.4	GUI Testing	67
5.5	Cache Processing Time	69
5.6	Summary	70

6 Conclusion	71
6.1 Conclusion	73
6.2 Future Work	74
Bibliography	77
Bibliography	79
A Access Point Configuration Templates	79
B Database Models Description	81
C Servers' Database Models Overlapped	85
D Servers' Website Navigation Scheme	89
E Web Page Views	91
F Smart Door Alerts	95
G User Testing Tasks and Times	97

List of Figures

1.1	International tourism, number of arrivals	3
2.1	Star and mesh network topology	10
2.2	MQTT's <i>Publish-Subscribe</i> pattern.	15
2.3	Application Protocols throughput comparison	17
2.4	Pavelić <i>et al.</i> 's communication between ESP client and application server	19
2.5	NodeMCU board and Raspberry Pi 3B+	23
2.6	openHAB Thing and Item concepts clarification	25
3.1	System's Architecture.	37
3.2	System layers and entities	40
4.1	Smart door controller's wiring diagram.	46
4.2	Smart light controller's wiring diagram.	49
4.3	<i>Reservation</i> overlapping situations	52
4.4	Door bell rang message flow.	61
5.1	Forced door event and action definition	66
5.2	Emailed alert content.	66
5.3	User test task time	68
C.1	Local and Remote Servers' database Model Relationship.	86
D.1	Website navigation scheme	90
E.1	<i>SmartDevice</i> Menu web page view.	92
E.2	Admin Website - <i>SmartDevice</i> list web page view.	92
E.3	Admin website - <i>Reservation</i> creation web page view.	93
F.1	Forced door alert in <i>DeviceType Menu</i>	96

F.2 Forced door alert stored notification. 96

List of Tables

4.1	Smart door message set and description	48
4.2	Smart light message set and description	49
5.1	Database object size, in bytes	67
5.2	Message processing time comparison, in seconds.	70
B.1	System models I.	82
B.2	System models II.	83
B.3	System models III.	84
G.1	Task description and time taken to complete each task, by the testers, in minutes and seconds	98

Acronyms

AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
BLE	Bluetooth Low Energy
CoAP	Constraint Application Protocol
DTLS	Datagram Transport Layer Security
EEPROM	Electrically Erasable Programmable Read-Only Memory
GSM	Global System for Mobile Communications
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
MQTT	Message Queuing Telemetry Transport
MQTTS	Message Queuing Telemetry Transport Secure
OSI	Open Systems Interconnection

RAM	Random Access Memory
REST	Representational State Transfer
SD Card	Secure Digital Card
SMS	Short Messaging Service
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
Wi-Fi	Wireless Fidelity
WSS	WebSocket Secure

1

Introduction

Contents

1.1 Motivation	3
1.2 Objectives	4
1.3 Document's Structure	5

The world population upsurge and worldwide development increased tourism, as the number of people travelling from a country to another also increases. According to World Bank Group's data, between 2009 and 2019, Portugal doubled the number of tourist arrivals and the European Union countries, combined, saw the number of tourists raise almost 40% ¹. Additionally, for the same period, the aggregated number of arrivals increased from 1.6 thousand million to 2.2 thousand million, for all countries in the world, as depicted on Figure 1.1. This globalization tends to increase short-term accommodation.

In December 2019, the virus SARS-CoV-2 spread around the world, which led to travel restrictions imposed by governments, thus reducing tourism. This Dissertation accounts that, despite these limitations, once these restrictions subside, the number of international arrivals will increase, following the aforementioned growth.

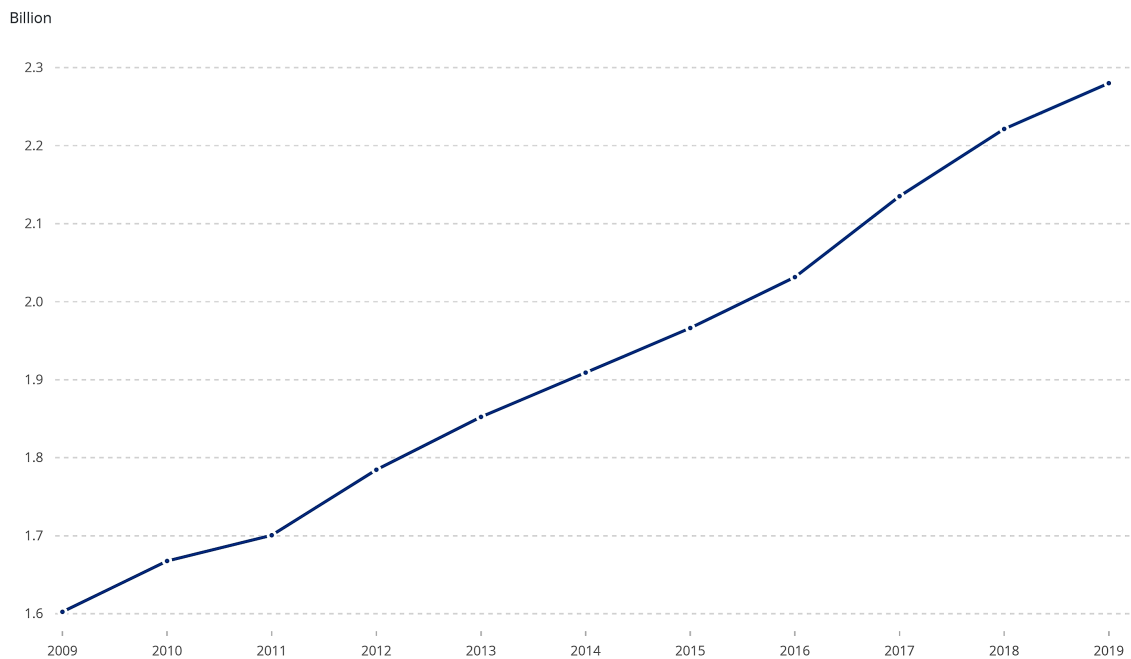


Figure 1.1: International tourism, aggregated number of arrivals, in thousand million (referred as billion), between 2009 and 2019 (adapted). ²

1.1 Motivation

The management of short-term accommodation requires a significant investment by a person responsible for the administration of a property, which may be the property's owner, a contracted person, or any other entity responsible for the asset. In the context of this document, this person shall be referred to as

¹World Bank. "International tourism, number of arrivals - European Union, Portugal". Available online at <https://data.worldbank.org/indicator/ST.INT.ARVL?end=2019&locations=PT-EU&start=2009>, last access in September 2021

²World Bank. "International tourism, number of arrivals". Available online at <https://data.worldbank.org/indicator/ST.INT.ARVL?end=2019&start=2009>, last access in September 2021

a house manager, or administrator.

Accommodation management can be a complicated task in situations where the administrator has more than one property to manage, and a reduced number of employees, or people, that could help perform this task. Customers arriving at the same time in different locations can be impeditive, as they might require the house key and the manager may not be available to open the door. Moreover, the administrator might want to access information, such as utility consumption during a client's reservation, and even turn off any appliances that have been left on, once the tenant leaves the property, without traveling to the it.

Technology's purpose, among others, is to help individuals and organizations to be more productive. The use of information technology can support the accommodation industry, assisting users to manage their properties.

The following concepts should be taken into consideration, as they are used throughout this document:

- An embedded computer is a small size, low-cost computer system, embedded in hardware [1]. These consist of a microcontroller equipped with, at least, a sensor or an actuator and communication interfaces;
- A smart device is a device, or an object, with an embedded system, which uses autonomy, context-awareness or connectivity to provide, at least, one user interaction feature [2].
- Internet of Things (IoT) is a concept introduced in the aggregation of these smart objects, interchanging data between each other, through wired or wireless communications, creating a network on which they operate together to reach common goals [3];

1.2 Objectives

The work presented in this Dissertation aims to provide comfort to the administrator, and to a client renting a house, by creating a smart house system. This is an IoT system integrated in a house environment, that provide context-aware actions, resulting in what appears to be intelligent behaviour (home automation). The term *domotics* can also be used in this context, referring to a smart house.

It is required that the system controls digital access to the property, enabling access to a registered user and restricting it to undesired people. To satisfy this requirement, the system must provide a feature that establishes different permissions to the users.

Additionally, this domotic solution needs to allow a person to monitor the house's utility usage, by metering the water, energy, and gas consumption, and it should also assist in the control of the house appliances, allowing, for example, to remotely power them on, prior to the client's arrival, and, after

their departure, verify if the client left one of these devices turned on, allowing them to be turned off when desired. By providing these solutions, the system aids the management of the property and helps improve the customer's stay.

Lastly, the system must authorize the administrator, through a user interface, to remotely manage all properties, by communicating with the house's devices, without compromising the system's security.

The work will have to take into consideration four aspects:

1. Connectivity — link established between devices with the purpose of sharing data;
2. Fault tolerance — enables the system to recover from event failures, such as reconnecting the involved entities upon power or connection loss, allowing it to continue its operation;
3. Scalability — architecture should be designed to support the system's expansion, integrating a variety of devices;
4. Safety and Security — Prevention and protection from both internal and external threats.

These are key on every IoT system, enabling the development of a trustworthy smart solution.

1.3 Document's Structure

The "IoT Solution for Rental Houses" project presentation is composed by the listed chapters, following this Introduction:

- In Chapter 2, project related concepts will be exposed, followed by projects and products which have significant importance in the development of the one suggested in this paper;
- Chapter 3 describes the approach proposed to solve the problem presented in this section, by considering the system architecture, the available components and the interactions established between them, in order to satisfy the proposed goal;
- The prototype created, comprised of two smart devices and two servers, and its functionalities are described in Chapter 4;
- The developed devices were used to evaluate the proposed project. The system evaluation and results, responsible for assessing the quality of the project, are given in Chapter 5;
- Lastly, Chapter 6, is used to present final remarks about the document.

2

Background

Contents

2.1 Protocols	9
2.2 Academic Projects	17
2.3 Products	21
2.4 Summary	26

State-of-the-art approaches that can be applied to this project are provided in this chapter. The work related to this proposal, is divided into the following categories: technologies, academic projects and commercial products. The first addresses communication protocols, the second handles scientific papers regarding developed projects and the last mentions products available on the market.

2.1 Protocols

The first section of this chapter focuses on protocols that operate on different layers of the Open Systems Interconnection (OSI) model and the Internet Protocol Suite, in particular wireless communication protocols and application layer protocols, important to establish a connection between devices in an IoT system.

As this section is larger when compared to the remaining, its subsections provide a dedicated segment, where a synthesis is provided, comparing the presented protocols.

2.1.1 Wireless Communication Protocols

Distributed systems require protocols that define a format, with a set of rules, for two or more entities to communicate between each other. Wireless protocols functions include breaking information, received from the network, into frames and transmitting each of these, by converting bits into signals. These rules are performed in reverse, when receiving data.

This subsection will address the most common wireless communication protocols used in IoT, with the objective of understanding the ones applied in this Dissertation.

In wireless networks, nodes connect to each other, creating a system that can be represented by a graph. The protocols described bellow, allow for the creation of star networks, where devices are connected to a central unit, and mesh networks, where devices are connected between each other, both illustrated in Figure 2.1.

2.1.1.A Bluetooth Low Energy

Bluetooth, a personal area network managed by Bluetooth Special Interest Group, uses $2.4GHz$ radio frequency to provide wireless communication between devices.

One device, the master, scans the frequency spectrum searching for a slave, a second device, which responds to the first with an acknowledgment. Once the signal is detected, the master sends the needed information for establishing a connection to the slave. After starting the connection, a star topology network is created, where the master may establish a connection with other devices. Each device is identified by a 48-bit address and can send up to 255 bytes of data on each packet.

To avoid interference from devices and other communication technologies, Bluetooth uses frequency-hopping spread spectrum. In this technique, the devices use random frequencies, defined by the master, within a specified range, to communicate. Bluetooth communication frequency changes between 79 different frequencies 1600 times each second.

Introduced as part of Bluetooth 4.0, Bluetooth Low Energy (BLE) is an improved version of this technology specifically conceived for the IoT. BLE allows a transmission rate of up to $1Mbps$, secured by a 128-bit Advanced Encryption Standard (AES) [4] and a maximum power consumption of $0.5W$. Therefore, BLE has been designed for systems where entities are limited by their processing power, memory size and power consumption, being applied to smartphone accessories and smart homes. In addition to a star topology network, supported by the classic version of Bluetooth, BLE also supports a mesh network, allowing for messages to travel for longer distances, by using other BLE supporting devices to propagate the data ¹.

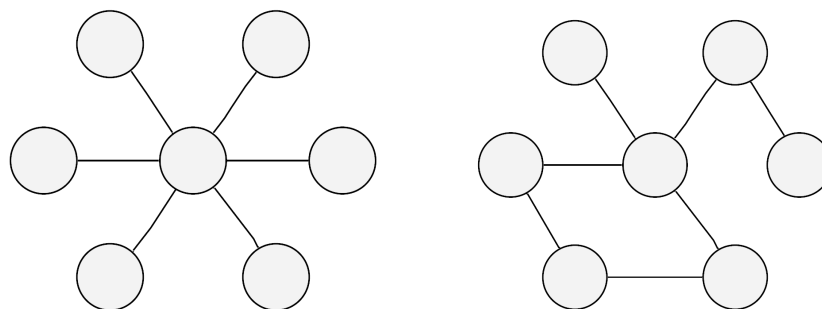


Figure 2.1: Star (left) and mesh (right) network topology.

2.1.1.B Wireless Fidelity

The most frequently used communication protocols, managed by Wi-Fi Alliance, is Wireless Fidelity (Wi-Fi) ². A Local Area Network (LAN) protocol family, based on IEEE 802.11, that connects distributed devices without a physical link, by using half-duplex connection through radio frequency.

Wi-Fi networks require a wireless router, a device responsible for creating the network and working as an Access Point (AP), allowing many devices to connect to it. Messages sent by a device connected to this network will go to this central unit, the router, which then forwards it to the intended receiver.

Introduced in 2009, IEEE 802.11n, or Wi-Fi 4 modifies the physical and the data link layers of the OSI model to improve previous versions of the protocol. This protocol uses radio frequencies near $2.4GHz$,

¹M. Woolley, "Bluetooth Mesh Networking", available online at <https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf>, last access in September 2021.

²Wireless Fidelity. "Discover Wi-Fi", available online at: <https://www.wi-fi.org/discover-wi-fi>, last access in September 2021.

supporting $5.0GHz$ as well, and a multiple-input multiple-output technique, utilizing several transmitters and receivers at the same time to increase the data throughput, guaranteeing a rate up to $600Mbps$ for a thirty-meter distance ³.

The hardware required for multiple-input multiple-output feature increases a devices power consumption. To reduce it, the protocol propagates data through time bursts, remaining idled for the remaining time.

As billions of devices depend on Wi-Fi, particularly in people's houses and businesses, this protocol requires a security feature to ensure devices communicate securely within the local network. This is achieved using Wi-Fi Protected Access, a Wi-Fi Alliance algorithm based on AES, to encrypt data.

As technology improves, so does Wi-Fi, providing faster speeds while consuming reduced amounts of energy, when compared to previous versions of this protocol.

2.1.1.C ZigBee

Centered on IEEE 802.15.4, Zigbee uses $2.4GHz$ frequency to create a personal area network, enabling data share between devices. Built to improve IoT communication, it is a low cost and low power consumption protocol that uses a mesh network to deliver messages, providing higher propagation distances, by hopping messages through other ZigBee compatible devices ⁴. The need for other ZigBee devices presents an advantage, as it increases the signal, and a drawback, as their use becomes mandatory to extend the signal indoors.

A ZigBee network may contain up to 65,000 smart devices, requiring one coordinator. This entity responsible for creating the network and moderating the communication, by selecting its channel. These networks use a 128-bit symmetric encryption to transmit data, at a rate up to $250kbps$, providing a safe and relatively fast communication.

ZigBee's product certification allows devices from different manufacturers to communicate between each other. However, its cost is quite elevated, as it requires a fee for each product, therefore becoming a big disadvantage.

2.1.1.D Z-Wave

Similar to ZigBee, Z-Wave is a Sigma Designs protocol designed to be integrated into smart home devices, through low energy radio waves. By setting a wireless mesh network, this personal area network protocol allows other Z-Wave devices to broadcast the original message, granting a speed between 40 and $100kbps$. Z-Wave messages are limited to a four hop distance, which is not a disadvantage as it

³Intel Corporation, "Learn about Multiple-Input Multiple-Output", available online at <https://www.intel.com/content/www/us/en/support/articles/000005714/network-and-i-o/wireless-networking.html>, last access in September 2021.

⁴Zigbee Alliance, "Zigbee Technical Specifications", available online at <https://zigbeealliance.org/solution/zigbee>, last access in September 2021.

uses lower frequencies, around $0.9GHz$, providing Z-Wave messages less interference in comparison to other protocols, and travel distances up to $100m$ (meter) at a highly reduced power consumption⁵.

This protocol uses an *AES* – 128 message encryption, allowing peers to securely exchange messages between each other.

By certifying products, Z-Wave allows compatible communication between devices, with its biggest downside being the device cost, supported by the fact that this is a proprietary technology, requiring a license for protocol to be integrated in a device.

2.1.1.E Synthesis

Considering the communication technologies presented in this section, their aspects enhance the qualities of the project, to achieve a low power consumption, flexible system.

Despite having limitations, each protocol contains features that make its use preferable over other protocols. Wi-Fi provides security at higher speeds, ZigBee achieves larger distances by hopping messages, Z-Wave is characterized by having the lowest power consumption and BLE offers fast communication speeds while consuming reduced energy [5].

Selecting the right form of connectivity is an inevitable choice on IoT projects and, as Wi-Fi can cover large distances at increased data propagation speeds, with a relatively low power consumption. Additionally, it is currently being used in most habitations, therefore being the chosen protocol to integrate the proposed system.

2.1.2 Application Layer Protocols

Application layer protocols belong to the highest layers of OSI model and Internet Protocol Suite. These are used to transfer data between two entities, by defining the communication syntax rules used.

Below, a description of some of the most used application protocols in distributed systems and IoT is presented.

2.1.2.A Hypertext Transfer Protocol

In use since 1990, the Hypertext Transfer Protocol (HTTP) has been used in *client-server* architectures, where the first entity, a client, establishes a connection to the second unit, the server, to send a message, called a request. Upon receiving this message, the server processes it, sending a response back to the client.

⁵Z-Wave, "Learn about Z-Wave", available online at <https://www.z-wave.com/learn>, last access in September 2021.

These requests require a Representational State Transfer (REST) method and a Uniform Resource Identifier (URI), composed by the endpoint to which the message is sent and a path to an asset. Below, a list of some of the most commonly used methods, available in HTTP, is provided [6]:

- GET — retrieves an object, such as a web page, from a URI;
- POST — performs a change in the server. It may contain additional data with the objective of adding or updating an object;
- DELETE — sends a request to the server asking for the object identified by the URI to be deleted;

An HTTP message is also composed of a header, providing information regarding the request or the response, depending on the type of message, and a body which, when present, carries additional information (a payload).

To these requests, the server responds with a three digit *statuscode*, and, possibly, a payload. The *statuscode* type is defined by the most significant digit. There are several codes, replied in particular situations, where, for example, a 200 symbolizes a request successfully processed, and client error is specified by a 400.

2.1.2.B Constrained Application Protocol

Defined in RFC 7252, Constrained Application Protocol was introduced in 2014 as a request and response, machine-to-machine protocol. Especially designed for IoT systems, this REST-based model is similar to HTTP, as entities may send requests to an endpoint using *GET*, *POST*, and *DELETE* methods, to which the server replies using a *status code* and, possibly, a payload.

Intended to be used by constrained devices, microcontrollers with reduced processing power and memory, Constraint Application Protocol (CoAP) provides a low overhead and low bandwidth connection to devices, with the objective of decreasing their power consumption. With the use of User Datagram Protocol (UDP), CoAP is able to achieve faster communication speed than most IoT protocols, which use TCP, thus reducing a microcontroller's transmission cycle, consequently minimizing the device's power consumption.

UDP does not provide a reliable connection. To circumvent this obstacle, CoAP identifies messages using a 16-bit unsigned header, and a second 2-bit header, called *type*, indicating if a message expects confirmation, when set to 0, or the opposite, when set to 1. Only messages expecting confirmation should receive a reply, containing a *type* value of 2 indicating that the message was received successfully, or a 3 if there was an error while processing the message. For situations where an *acknowledgement* is required but not received in a specified time range, CoAP re-transmits the message at exponentially increasing intervals until a response is obtained, or a defined limit is reached [7].

Detection of non-confirmed messages is impossible to be achieved and needs to be handled by the application.

2.1.2.C Message Queuing Telemetry Transport

Message Queuing Telemetry Transport (MQTT) is a *publish-subscribe* protocol designed for the IoT. A *publish-subscribe* architecture requires three types of entities: a publisher, a subscriber and a broker. The publisher is responsible for generating and publishing data into the broker, using a topic, an MQTT resource, which can then send the message to a subscriber.

The broker is the only entity known by both publishers and subscribers, meaning these do not know which other devices are involved in the system [8].

This is a *publish-subscribe* protocol running over Transmission Control Protocol (TCP), in which the broker contains the list of available topics, allowing it to propagate a message to a subscriber, whenever data is published to a topic. This protocol allows for more than one instance to publish to and subscribe from the same topic.

Figure 2.2 provides an example of a simple *MQTT* system, containing a broker that broadcasts a message, published to *topic a*, to *Subscriber a* and *Subscriber c*, but not to *Subscriber b*, as the latest is not subscribed to the topic.

MQTT provides an agreement for the number of times a message is delivered to a subscriber. MQTT offers three levels of agreement, known as *Quality of Service*. Level 0 allows a message to be delivered at most once without requiring an *acknowledge*, level 1 states that a message is sent at least once, being its *acknowledge* mandatory, and level 2 allows a message to be delivered exactly once. This last level requires a four-way *handshake*, resulting in slower messages, in comparison to the other levels [9].

Compared to HTTP, it consumes less computational resources, thus being an advantage to operate with low-power devices. Additionally, the broker acts like the middle-man, being responsible for coordinating the system's information, preventing long-polling situations, where a server holds a client's message until the requested data is available.

2.1.2.D WebSocket Protocol

Proposed in December 2011, the WebSocket protocol, or WebSockets, enables a client to send information to a server and it also allows the server to freely push data to the client, while the connection is opened, thus providing a full-duplex communication [10].

WebSockets are reliable, meaning the protocol guarantees that the message is delivered unless one of the parties suddenly disconnects from the channel. This feature is provided by TCP, as this protocol relies on it.

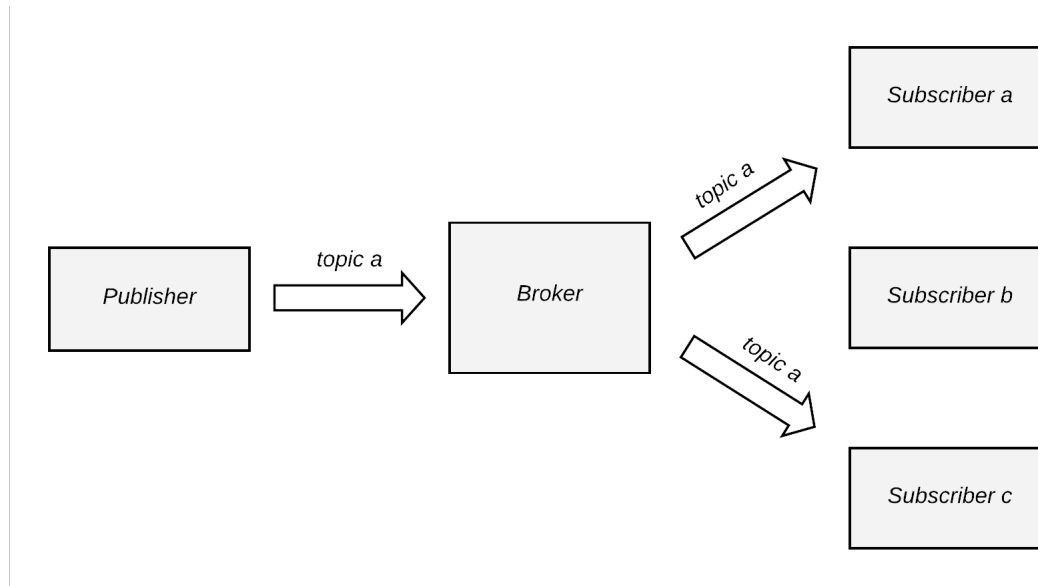


Figure 2.2: MQTT's *Publish-Subscribe* pattern.

WebSockets require an initial *handshake* between the client and the server, performed by an HTTP request, after which communication is kept open until one of the two entities disconnects. This *handshake* is the main disadvantage of the protocol, as it leads to a large overhead, whenever the communication channel is initiated. For this reason, it is not desired to use the protocol for situations where connections are opened for a single request, followed by constant re-connections.

The protocol's frame uses a 4-bit *Opcode* allowing, among other functions, for the server to recognise a connection termination. To detect it, the protocol uses a *ping/pong* message pair. *Ping* is a message sent by the server and repeated within a fixed period of time. This message expects a reply from the client, called *pong*, thus granting the server information about the latter's status, and *vice-versa*.

2.1.2.E Transport Layer Security

Transport Layer Security (TLS) is an application protocol which allows secure communication between two peers, by encrypting an entity's message. To provide this characteristic, communications are established during a *handshake* summarized by the following steps [11]:

1. *Client Hello* – The client opens a connection to the server, containing a list with the encryption algorithms supported and a large random number;
2. *Server Hello* – The server sends a response with the algorithm chosen to compute the symmetric key (allowing both client and server to perform the same operations), another random number, a session identifier (used to recognise the client), and a digital certificate (a file containing information relative to the server, including the its public key). This certificate is generated by a third entity, an

organization trusted by both client and server, signed using that entity's private key;

3. *Verification* – The client verifies the server's authenticity by validating the received certificate. This is done by processing the certificate's message, using a one-way hash function, and matching the result to the certificate's digital signature, decrypted using the entity's public key. This key may be embedded on the client's device, or accessible using an additional request to the trusted authority;
4. *Client Key Exchange* – The client generates a random key, the *pre-mastered-secret*, and sends it to the server encrypting the value with the latest's public key. This ensures only the server can decrypt the message, as no other entity contains the private key used to decipher the message.

After the last moment, both client and server may use the *pre-mastered-secret* and the two randomly generated numbers, sent in each of the *Hello* messages, to generate a symmetric key. This key is unique for this communication channel and is used to encrypt and decrypt the messages exchanged by both parties, ensuring only these two entities can visualize the information.

TLS requires a reliable, error free data stream, thus layering on top of TCP. TLS ensures protection against replication attacks because its transport layer protocol contains a sequence number, which can no longer be modified by a malicious entity.

TLS is application-protocol independent, meaning other application protocols, such as HTTP, MQTT or WebSockets, can protect data through it. These secured protocols are named Hypertext Transfer Protocol Secure (HTTPS), Message Queuing Telemetry Transport Secure (MQTTS) and WebSocket Secure (WSS), respectively.

Datagram Transport Layer Security (DTLS) is an alternative protocol used to secure UDP based connections, which will not be covered in this document. It is mentioned as CoAP, using UDP, requires DTLS to protect its data.

2.1.3 Synthesis

The presented application layer protocols are reliable and may provide encryption using TLS, or DTLS, in CoAP connections.

HTTP may be good for *client-server* applications, however it is not designed to be used in IoT systems due to its high overhead. For this reason, CoAP, MQTT and WebSockets can be used as an alternative. However, all of these present certain limitations as well. The first of these three protocols requires units to act as both client and server to achieve a full-duplex connection. Moreover, the requirement of a broker for the MQTT protocol may not be desired as it increases the expenses of a project. Finally, WebSockets main disadvantage is the high amount of server resources consumed by a connection left-open.

These three main IoT protocols' (CoAP, MQTT and WebSockets) packet transmission rate (throughput), τ , was compared for a connection where 100 packets were transferred. The authors concluded that, despite WebSocket's overhead, this protocol performs as good as CoAP and it is better than MQTT, when sending reliable messages over a long term connections [12]. Figure 2.3 provides this comparison, where CoAP non-confirmable requests are depicted in red and confirmable requests are illustrated in light green and MQTT's *Quality of Service* is denoted by *QoS*.

WebSockets presents a middle term protocol providing a fast message transmission, with low overhead which, allowing the client to send information to the server, and the latest to push state changing messages to the client, for as long as the connection between the two is open. For these reasons, WebSockets has been chosen to integrate the system's proposed in this document.

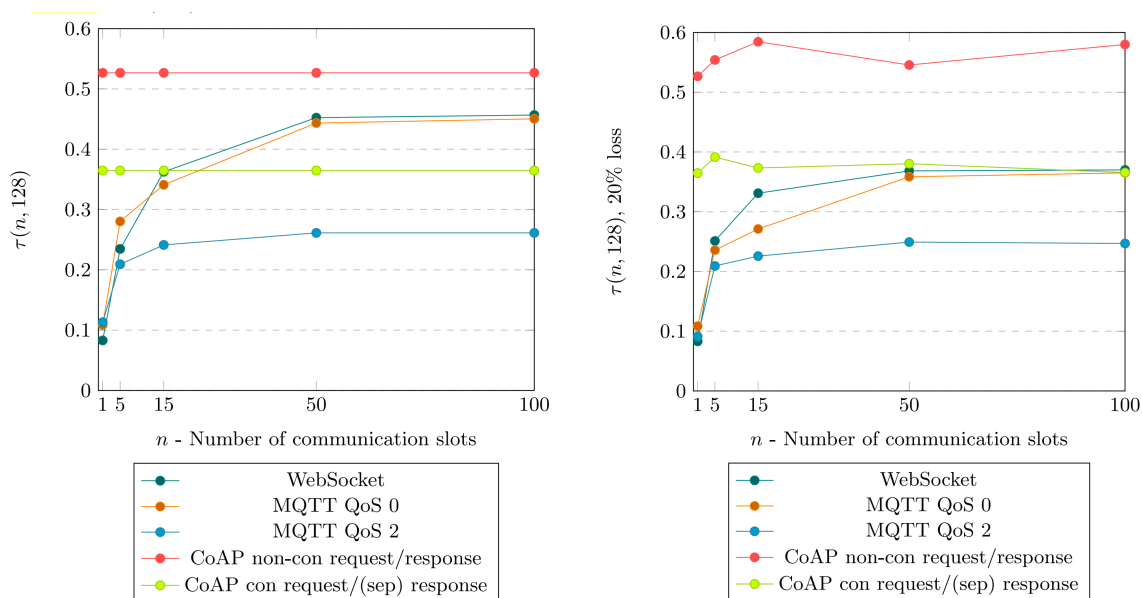


Figure 2.3: Application Protocols throughput comparison without packet loss (left) and with 20% simulated packet loss (right) [12].

2.2 Academic Projects

The second section of this chapter focuses on summarizing home automation projects, described in scholarly literature. Projects specifying door lock and utility monitoring systems, may assist on the management of a property, thus being described in their respective subsection.

2.2.1 Door Authentication

Door authentication allows a person with a specific key access a room, or a house. Traditionally, a physical key is used to open a door. This approach is not desired in property rental as it forces the item to be delivered to the tenant, by the administrator. Below, projects that try to solve this problem, by using an electronic solution, are presented.

2.2.1.A Smart digital door lock for the home automation

Y.T. Park, P. Sthapit and J. Pyun created a digital lock system to interact with a smart home [13]. Their project includes two forms of system authentication: a keypad, allowing for a door to be unlocked using a code embedded on the microcontroller, and Radio Frequency Identification to detect the presence of a valid user, in possession of a specific tag.

The communication between sensor nodes and a central controller is provided by *Zigbee*, reducing the system's power consumption. Moreover, the *Zigbee* module is connected to a set of sensors, such as a temperature and gas detectors, allowing the system to recognize fire hazards, using Short Messaging Service (SMS) to notify a person when these situations are detected.

The identification tags, despite being practical for the user, require the client to be in possession of one of these items to unlock the door. The need for a physical object raises a problem similar to the use of a key, as it requires for the administrator and the client to meet.

2.2.1.B Internet of Things Cyber Security: Smart Door Lock System

A project has been developed using an ESP8266 based microcontroller to control an electric lock [14]. This smart lock system provides Websockets communication between a microcontroller and a web server, which manages the system data, by storing it in a database. The last entities in this system are the user's terminals, accessing the server using a web or a mobile application.

Pavelić *et al.*'s work focuses on providing security to the system. The HTTP connection between the user terminal and the server is secured with TLS, and a WSS communication is provided to protect the connection between the microcontroller and the server. The system identifies several threats, which have been diminished with the used of this encryption protocol.

This solution allows for users to control several doors, by connecting multiple microcontrollers, to the same server. When one of these ESP clients is powered on, the microcontroller starts a TLS *handshake* followed by a WebSocket connection. This last protocol provides a persistent communication channel between the entities, allowing the server to push messages to the microcontrollers, opening the door, whenever a message is received from the user's terminal. The use of WebSockets' *ping-pong* feature allows for the server to detect sudden disconnects. Figure 2.4 provides an illustration of this

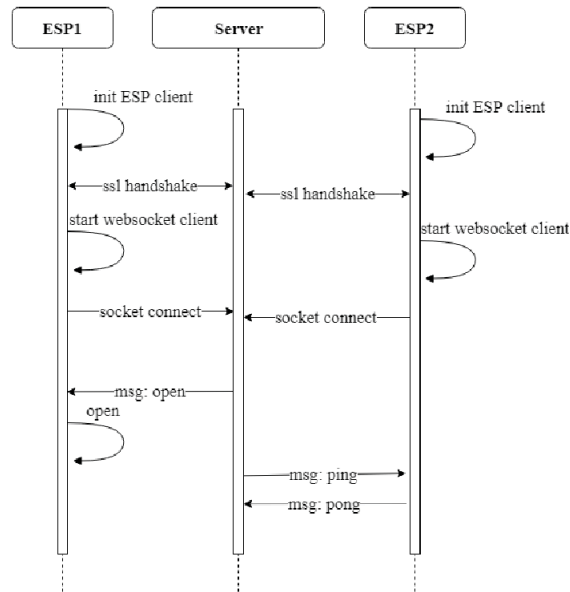


Figure 2.4: Pavelić *et al.*'s communication between ESP client and application server [14].

communication process, using two microcontrollers.

Users are forced to download a mobile application to access this system. As the system does not provide a push-notification feature, to alert the user whenever the system is at risk, the mobile application presents a disadvantage, as tenants renting a house may not wish to fulfil the requirement.

2.2.2 Utility Monitoring

Utility monitoring systems ensure users can supervise a house's environment, by measuring power, water and gas consumption. This is an important device that helps solving the limitation presented in the first chapter, as it provides information regarding the tenant's visit, allowing an administrator to identify the visitors' utility consumption, thus helping to manage house wastage.

This segment focuses on describing two projects designed to monitor a house's utility consumption.

2.2.2.A IOT based application for monitoring electricity power consumption in home appliances

This IoT project has been developed in using a smart device to calculate the power consumption of an appliance.

A sensor allows a microcontroller to detect both alternating or direct current being consumed by a house appliance, which is then uses a Wi-Fi connection to propagate the sensed data to a Cloud hosted server [15].

The Cloud entity aggregates the data sensed by the device and stores it in a database. This data is associated with a timestamp, allowing for a user to access the database and monitor the house's

environment over time, with the objective to lower the energy consumption.

The use of a Cloud service is the most interesting feature of this project, as it allows data to be remotely accessed at any moment. However, the system fully dependent on the Internet, meaning it cannot be monitored if the connection is lost. Using a server hosted inside the property could improve this project, allowing the system to store data, thus contouring this adversity.

2.2.2.B IoT Water Consumption Monitoring & Alert System

Che Soh *et al.* developed a system to monitor the water consumption of a house [16]. Their system uses a water flow sensor to meter the water consumption, which sends the data to a Cloud server.

A microcontroller board reads the data collected by the sensor and sends it to the server through Wi-Fi, using MQTT protocol. This Cloud hosted server allows the user to access the data, using a dashboard.

This project provides a notification system, in which the server is responsible generating and sending notifications to the user, in situations where the consumption surpasses a defined threshold. Both emails and SMS are used to deliver this alert message.

2.2.3 Synthesis

Four home automation academic papers have been described in this section, in particular smart lock and utility monitoring systems, as these present an important factor to take into consideration while designing a smart home solution for property rental.

Some of these systems provide an alert system, allowing the application to notify users, in critical situations. This feature was achieved using SMS, a technology based on Global System for Mobile Communications (GSM), allowing for short-sized messages to be sent to a person's telephone (as most of these personal devices use this protocol), or using emails. This last technology uses Simple Mail Transfer Protocol (SMTP) to send the data to an email server, requiring second protocol, Internet Message Access Protocol, for users to retrieve the message from the server.

GSM ensures a fast, reliable and secure message delivery to a user's device, with the biggest disadvantage being its cost. SMTP allows for large messages to be broadcast to several users, most times free of charge. This last protocol is not encrypted, nevertheless TLS may be used to secure any sent message. However, SMTP biggest downside is the fact messages may not reach the recipient within short time periods, thus not being suited for critical alerts.

2.3 Products

A variety of smart products are used in houses to assist and simplify humans in day-to-day tasks.

The system proposed to assist in home rental management is designed to integrate a diversity of devices and can be expanded, in the future, to integrate many more.

This section is dedicated to the description of available market products designed to create a smart home, presenting smart devices which could assist in the management of a house, followed by a section describing platforms which allow user to integrate devices from different manufacturers.

2.3.1 Smart Devices

The IoT expansion into the home automation market led to the creation of domotic systems. The demand to have a smart house increased in the last years, and companies seized the opportunity to develop smart devices which can be integrated in a house.

This subsection focuses on presenting commercially available products that could be be useful in the context of house rental.

2.3.1.A Door Authentication

August and Google Nest sell products that allow people to access a property. As door authentication is an important feature for the system described in this document, a description of smart lock products sold by these companies follows.

August

An american company named August provides smart lock products, controlled using a wireless technology. The locks can be unlocked using a physical key, a keypad code or a mobile application, accessing the lock via Wi-Fi or BLE ⁶. The application allows house owners to provide permanent or scheduled door access to the property, and that person's access can be logged to an activity feed.

Google Nest

Google Nest produces a battery powered smart lock with a ten digit touch screen, for users to enter an access code. The lock is a low power device providing up to one year of battery power, possible by using ZigBee communications ⁷. This technology requires the device to be connected to a gateway, converting ZigBee into Wi-Fi, allowing users to create a temporary access code and to receive notifications when the state of the door changes, or a person fails to insert the access code.

⁶August, "How Smart Locks Work", available online at <https://august.com/pages/how-it-works>, last access in September 2021.

⁷Google, "Nest X Yale Lock ", available online at <https://store.google.com/us/product/nestxyalelockspeccs>, last access in September 2021.

2.3.1.B Utility Monitoring

Similarly to the previous segment, underneath follows a pair of companies that provide utility monitoring products, which could be integrated the system described by this document.

STMicroelectronics

STMicroelectronics intelligent meters to monitor gas, electricity and water consumption. The company provides single-phase and three-phase electricity meters capable of handling between 120 and 240V AC, gas and water meters which use ultrasonic or electromagnetic signals to measure the utility consumption⁸. These meters provide secure wireless communication using BLE or Zigbee. Alas, the meter price is extremely elevated.

Kamstrup

Kamstrup, develops electrical⁹ and water¹⁰ intelligent metering solutions for residential use, that transmit real-time consumption data to an app, allowing users to monitor the sensed data. Additionally, the application can provide alerts to a user using GSM.

2.3.2 Controllers

NodeMCU and Raspberry Pi 3B+, both depicted in Figure 2.5, are two controllers commercially available, which are often associated with domotic systems. The first is frequently used by enthusiasts to create sensors or actuators. A cluster of these can be integrated in a single system using a more powerful controller, such as the aforementioned Raspberry Pi.

2.3.2.A NodeMCU

The NodeMCU is a low power microcontroller board composed by an ESP8266 module, the ESP12e. This 3.3V board combines a processor, cable of running at 80 or 160MHz, with a 50KB Static Random Access Memory¹¹. Despite not having embedded physical memory, it is possible to emulate a 4096B Electrically Erasable Programmable Read-Only Memory (EEPROM), permitting data to be stored and accessed between reboots.

⁸Stm, "Electricity Metering", available online at <https://www.st.com/en/applications/metering/electricity-metering.html>, last access in September 2021.

⁹Kamstrup, "OMNIPOWER® Three phase meter", available online at <https://www.kamstrup.com/en-en/electricity-solutions/smart-electricity-meters/three-phase-meter>, last access in September 2021.

¹⁰Kamstrup, "flowIQ® 2200", available online at <https://www.kamstrup.com/en-en/water-solutions/smart-water-meters/flowiq-2200>, last access in September 2021.

¹¹Espressif, "ESP8266EX Datasheet", available online at <https://components101.com/sites/default/files/componentdatasheet/ESP8266-NodeMCU-Datasheet.pdf>, last access in September 2021.

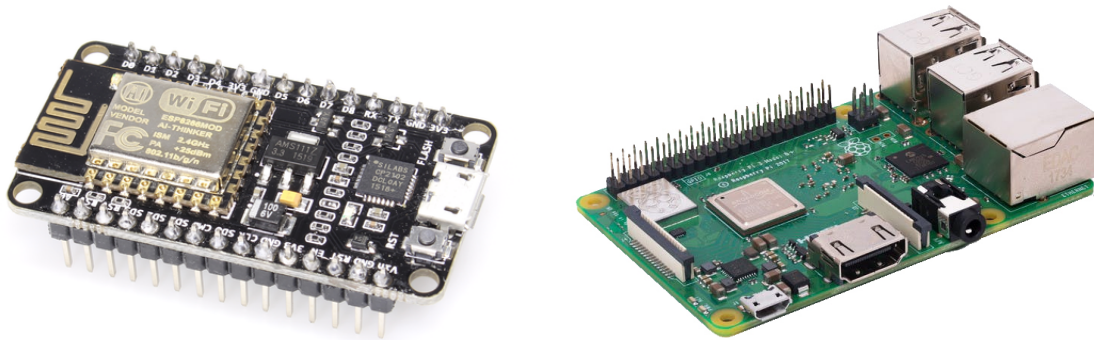


Figure 2.5: NodeMCU board (on the left)¹² and Raspberry Pi 3B+ (on the right)¹³.

Containing 11 General Purpose Input/Output (GPIO) pins, used to propagate electrical signals, allowing a controller to connect to a module used to perceive the environment, or to apply a change on it.

The ESP8266 has been developed by Espressif and provides the board a wireless communication system that supports Wi-Fi. The board has a micro Universal Serial Bus port that can be used to upload an executable script and to power the device.

The NodeMCU contains one of the fastest processor speeds accounting its reduced cost and power consumption. It provides several low power consumption modes, including a $10\mu A$ sleeping mode, and wireless communication, making it a great choice for controlling a smart device.

2.3.2.B Raspberry Pi 3 B+

The Raspberry Pi 3 B+ is a $5V$ single-board computer made by Raspberry Pi Foundation. Launched in 2018, this 40 GPIO pin board is composed by a processor capable of performing 1.4 billion cycles per second¹⁴. This computer provides a $1GB$ Random Access Memory (RAM) and a micro Secure Digital Card (SD Card) slot, which allows up to $256Gb$ of storage.

This single board computer has wireless connection, being able to support Bluetooth 4.2, BLE, IEEE 802.11.b/g/n/ac and cabled connections. Additionally, it has a relatively low power consumption given the processor's speed, therefore making it a robust choice for a local server running systems such as the ones described in 2.3.3.

¹²Make Magazin DE, "NodeMCU Amica", available online at <https://commons.wikimedia.org/wiki/File:Nodemcuamicabot02.png>, last access in October 2021.

¹³Raspberry Pi, "Raspberry Pi Model 3B+", available online at <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>, last access in October 2021.

¹⁴Raspberry Pi, "Raspberry Pi Model 3B+ Datasheet", available online at <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>, last access in September 2021.

2.3.3 Integration Platforms

The domotic industry is growing and it is expected to keep satisfying the market's desires. Smart device products cannot solemnly satisfy the user needs, as they address solutions for different problems, forcing people to purchase devices from distinct producers. Additionally, these individual devices communicate using different protocols and technologies, not working ensemble, thus requiring users to control them from different applications. This adversity led technology enthusiast to create their own solution, with the goal of controlling devices from a single application.

This subsection addresses two open-source projects, openHAB and Home Automation, whose objective is to integrate products using different communication protocols, from different manufacturers and technologies into one home automation solution. This allows a house to be automated with smart devices, which could help manage to rent a property.

2.3.3.A openHAB

Open Home Automation Bus, known as openHAB, is one of the most famous home automation softwares with a large community using it to automate their homes. This solution uses a server which provides users with a web and mobile application, on which they can monitor and control their house's smart devices. This application can be accessible from the house's LAN or remotely, using a Cloud.

This software provides remote access to *Things*, physical devices working as sensors and/or actuators that provide access to its properties, the *Channels*. The last can be altered by the user, using a Graphical User Interface (GUI), and controlling one or more *Items*. There are several other concepts in openHAB which will not be mentioned. These concepts can be better understood with the sketch provided by Figure 2.6, in which two lights, the *Items*, are controlled by the same device, the Thing. Both of these devices can have property, the Channel, indicating the light's state (whether it is turned on or off).

It is important to notice that openHAB 3.0.2, the version of this software at the time of this document's writing, has a very simple system for users, as it does not support user access restriction, meaning it does not provide authentication, nor it allows the application administrator to manage the system using a GUI. Lacking the configuration of user permission, openHab has an impractical solution for a system requiring new users to be added constantly, such as a house rental solution.

2.3.3.B Home Assistant

Another well know home automation system is Home Assistant. Being able to integrate more than 1500 solutions and technologies, this system, through the configuration of a data-serialization language files, grants actuators the ability to perform certain actions, triggered by a user described event. An example

of this would be turning on a thermostat whenever the room temperature drops below 20°C .

Despite being able to provide users with remote access to a house (via Internet), the use of configuration files, reassures that Home Assistant targets users with a background knowledge on information technology, a disadvantage in the context described in this document.

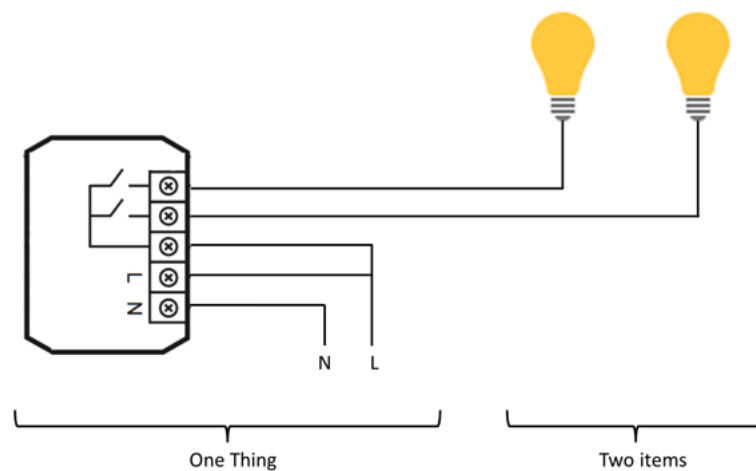


Figure 2.6: openHAB Thing and Item concepts clarification ¹⁵

2.3.4 Synthesis

This section mentioned the simplest components required for a home automation system available on the market. People can acquire smart devices, using several technologies, to access a variety of functions. Using a microcontroller, such as the aforementioned ESP8266, the devices can be accessed remotely and sense/actuate on the environment. Benefiting of an integration platform, devices can be combined into a single solution allowing users to access all the devices.

These open-source softwares, that can be hosted in a Raspberry Pi computer, providing freedom for a user to create a complex and scalable system, that integrates different technologies. Nevertheless, its implementation requires familiarization with programming, and other skills making their configuration complex and, therefore, undesirable for the most users.

As referred, another important disadvantage of such systems, which appear to be able to solve the house management problem, is the fact they do not provide a proper user authentication system, that could allow users with different permissions to access specific devices. A desired feature in home rental scenarios, as tenants should have access to a limited set of smart devices.

¹⁵OpenHAB, "Concepts", available online at: <https://www.openhab.org/docs/concepts/>, last access in September 2021.

2.4 Summary

In the *Background Chapter*, protocols, academic projects and commercially available products, all related to home automation systems have been mentioned.

Wireless communication protocols, BLE, Wi-Fi, ZigBee and Z-Wave have been introduced and analyzed. Despite increasing a device's power consumption when compared to the remaining protocols, Wi-Fi provides fast communication and users are familiar with its usage as it integrates most people's houses, covering the property's area. For this reason, it presents a robust choice when applied to home automation systems. Wi-Fi can be used for *client-server* applications, such as the one described in this document. For this reason, application layer protocols have been addressed. Domotic systems require smart devices to transmit real-time data to an application, which could then be consumed by a user. CoAP, MQTT and WebSockets can be used to connect the smart nodes to the application, being the last preferable over the others for persistent connections, as it provides a full-duplex connection between a client and a server, allowing data to be sent at any time in both directions. Additionally, HTTP, being the preferable protocol for web applications can be used for connecting a client browser to a system. Both protocol connections can prevent external people from eavesdropping a connection, by providing TLS encryption.

Two types of academic projects were discussed. Smart lock systems, which allowed users to access a property by controlling the door, followed by utility monitoring systems. Both of these relevant to a domotic system in which the management of a property is required, as the first can provide access to different users to the property, and the second allows administrators to monitor the property. Some projects utilize two complementary protocols, SMTP and GSM, and mobile application notifications allowing for users to be notified about system events.

Finally, a microcontroller has been presented, the ESP8266, which can be used to create smart devices such as the ones described in the academic projects section. Additionally, two integration platforms have been introduced: openHAB and Home Assistant, which can be hosted on Raspberry Pi, thus creating a home system which can be used to connect users to smart devices.

3

Architecture

Contents

3.1 Requirements	29
3.2 System Entities	31
3.3 System Architecture	36
3.4 Communications	37
3.5 Summary	40

The third chapter of this Dissertation presents a home automation system architecture, designed to solve the house rental and property management problem. For this reason, it includes a section in which the system requirements are presented, followed by another in which the most relevant communication protocols, selected for a real-time home automation system, are identified. Finally, the architecture of the system is described.

3.1 Requirements

Home automation systems, designed to assist a person to manage a property in short-term accommodation scenarios, are required to contain a set of specifications, with the goal of achieving the purpose they were designed for. On this section, the most relevant requirements identified to design a house rental solution are described. These specifications can be sorted into reservation management, home automation and communication tasks.

3.1.1 Reservation Management

Reservation management features represent the defined system traits that provide house managers the possibility to define and control the house tenant's reservations.

First, the solution should provide a web application, through which authenticated users may access, manage and control the system and its entities, thus storing the system's information. Access to the application's functionalities should be made unchallenging towards inexperienced users, being configured and set up with ease.

As this project is dedicated to property rental, it is crucial that the application allows client reservations to be managed, by associating a person to a property, between two dates.

Additionally, it is required that the system concedes privileged users (the property managers) the ability to control and monitor the house prior, during and after a client's rent period.

This system is designed for home owners. These may only wish to rent part of a property, such as a room, an entire house, or both. For this reason, the system should provide owners with the possibility to integrate several properties in a single solution, managing them from a single application.

Lastly, the application may be accessed by different types of users. For this reason, it is required that these have restricted access to the application, depending on their role. The property owner should have unrestricted access to the system, unlike the remaining users, that should be restricted to the houses they are assigned to. This is:

- A contracted house manager may be authorized to access the properties they supervise;

- A tenant's authorization must be limited to the house the person is renting, during their accommodation period;
- A staff member should also be restricted to the properties it maintains.

3.1.2 Home Automation

The system must also provide a set of features that allow managers to observe changes in a house's environment, such as verifying if an appliance has been left powered on. Moreover, owners may provide tenants with additional comfort before and during their accommodation period, by controlling the environment properties. An example would be to use a thermostat to regulate the temperature of a house, prior to a client's arrival.

A solution for this can be achieved with a domotic system, using smart devices to sense and actuate on the house's surroundings.

The management, monitoring and control of a houses smart devices should be provided by the application. However, some these devices should not be accessed by all users. For this reason, the permission level assigned to each user should also be used to restrict the access to a house's smart devices, ensuring, for example, that clients utilize a restricted set of features.

An example of a smart device whose integration in a home automation system for rental house management, as presented in the previous chapter, is a smart lock, allowing for tenants to have scheduled access to the property. Ideally, this should be provided using a code, assigned to each reservation, thus ensuring temporary authorization to the house, without the use of a physical item. Moreover, the lock should provide a fixed access code, so that the property's managers can enter the property using a fixed key, different from the clients'.

It is desired that the system is able to store data whenever a user defined event has occurred, ensuring house managers may observe changes in the house environment, throughout time. This is useful during a client's reservation, for example, to monitor the tenant's utility consumption.

The system must also provide a set of alert features, with the goal of notifying user after an event occurred, ensuring its detection and a rapid response, from any user, in case an event occurs, such as a device disconnection or a security hazard is identified. As discussed in the previous chapter, system notifications should be delivered, to the user, using the application interface, emails and SMS, as these technologies complement each other. Preferably, the owner should be able to chose when to use each, as notifications that represent critical situations should be sent using SMS, as these are delivered in near real-time, and emails ought to be used in informational circumstances, as SMS could become expensive. This feature is particularly important, as insufficient logging and monitoring is one of the biggest security risks for web applications ¹.

¹OWASP, "OWASP Top Ten", available online at <https://owasp.org/www-project-top-ten/>, last access in September 2021.

3.1.3 Communication

Communication between system entities is an important factor to take in consideration when designing a distributed solution. Hence, this subsection identifies the communication characteristics desired for an IoT system dedicated to assist solving the problem described in the beginning of this document, minimizing the costs to owner.

First, having identified the necessity to manage several houses in a single solution, it is desired that this task can also be achieved remotely, thus ensuring control over the system over the Internet, without requiring a manager to enter the property in order to configure, monitor and control the system.

As the manager has remote access to the property, messages between the system entities, should be made available in real-time, or close to real-time, as instantaneous communication is not possible due to message propagation and processing times.

In case a message fails to be sent or processed, the system should asseverate the message delivery, except if the request was generated by the user, in which case the system must ensure a method for the person to correct the problem. For example, messages generated when a user is creating a reservation or changing the state of an environment property, may be acknowledged and resent, whereas those informing a change in the house's environment, which are created by a smart device, such as an opened door, need to have guaranteed delivery, otherwise the message is lost and the user may never be aware of the event.

Security is also an important aspect to consider, when creating a home automation project, as the system controls a person's assets. For this reason, access to the system should be provided only to authorized users and machines, minimizing the vulnerabilities of the solution. Communication channels between electronic devices must be secured, preventing data to be exposed. Additionally, sensitive data, such as user passwords, the property's access codes, and others, must be encrypted, when stored in a database.

Lastly, as there is a variety of home automation devices available in the market, from different companies, offering diverse functionalities, the solution should provide owners with the possibility to incorporate devices regardless of their communication technology and messaging format. This may, for example, allow the owner to have a smart lock that communicates over WebSockets, a water meter that utilizes CoAP and even a thermostat that utilizes BLE to be controlled, thus creating an automation system tailored to owner's needs.

3.2 System Entities

To surpass the adversities presented, a home automation system has been designed, connecting numerous devices and utilities via the house's network. By taking advantage of Wi-Fi connections inside

a house, this system connects home automation instances using this wireless communication protocol, allowing for messages to be rapidly sent across devices, without requiring additional hardware to be obtained by the owner, as most properties already rely on this technology.

Being the reference protocol for web applications, HTTP is used to exchange information between a server and a web-browser. For this reason, any requests emitted by a user terminal to the server, with the goal of accessing the system's information, should be provided using this protocol. Additionally, the WebSocket protocol allows an entity to establish a link with another, listening for a new connection, ensuring messages can be exchanged in real-time, without requiring the server to defer a client's message until the requested information is available, thus preventing long polling latency problems caused by HTTP's large headers [17]. These protocols must take advantage of the cryptography application protocol discussed in the previous chapter (TLS), ensuring a message's integrity is maintained, therefore protecting the system from malicious entities, thus fulfilling the communication channel security requirement.

Taking into consideration these protocols, this section describes smart devices, providing interactions with the house environment, network units, responsible for creating communication channels for entities to exchange data, and two servers. One server is hosted inside a property, while another is hosted in the Cloud, providing a hybrid solution, in which a house can be constantly monitored, even if the connection to the Internet is lost, thus solving the presented problem.

3.2.1 Smart Device

The document describes an IoT system to assist in the management of a property, using smart devices to interact to the property.

A house's object or appliance can be turned into a smart device, provided that its tasks can be performed by an electronic component. For this reason, a smart device should contain, at least, one characteristic (or property) capable of interacting with the house's environment. Depending on the type of interaction, this property may be an actuator, producing a change on the environment, such as a light bulb's state, or a sensor, monitoring the environment, such as an energy meter.

This architecture should provide managers with the possibility of extending the application over time, by adding different types of smart devices to it, capable of monitoring and controlling several house appliances, thus allowing the property to be configured with respect to the owner's and tenants' necessities.

3.2.2 Smart Controller

In this system, any microcontroller entity responsible for the control of, at least, one smart device is called a Smart Controller. This entity should have Wi-Fi integration, in order to establish a connection to another entity, via a software interface that allows two applications to communicate. This intermediary is called an Application Programming Interface (API).

Smart Controllers are very limited, when compared to a house's server, as they have reduced memory, storage and processing power. These limitations, however, result in the devices consuming reduced energy, an important advantage for home automation systems.

In resemblance to an openHAB *Thing*, more than one device may integrate a single Smart Controller, for example, one microcontroller may be attached to an array of mechanical relays, each controlling the state of a light, thus leading the Smart Controller to connect to as many smart devices as the number of relays in the array. A relay is a switch that changes its state when a small voltage is provided (in comparison to the relay's circuit voltage).

3.2.3 Proxy

It is proposed that the devices use Wi-Fi to communicate a central entity in a house. These communication channels are provided by the WebSocket protocol. However, it should be noted that, as previously mentioned, smart devices are not required to utilize this application protocol to communicate. For this reason, the system should contemplate the use of a proxy. The proxy's task is to translate messages sent by Smart Controllers that do not use the WebSocket protocol, thus receiving messages from the controller and forwarding them to the house's central unit, using the WebSocket protocol. The reverse process is also provided by a proxy, when messages originate in the server. For this reason, the proxy acts as an API.

Additionally, these units reduce the number of connections to the central entity, permitting the house system to grow in size, without impacting the its performance.

3.2.4 Gateway

Wi-Fi is the communication protocol proposed to integrate this system. However, it is suggested that the project includes IoT gateways, responsible for connecting a network, whose Smart Controllers utilize a different communication protocol, to the Wi-Fi network, allowing the controllers to communicate with the house's central API.

These entities should utilize WebSockets to communicate to the system's main API, thus transferring messages from devices that do not use Wi-Fi to the server, allowing the house manager to add any type of smart device to the system, independently of the protocol used. If the application protocol used in the

Wi-Fi network is not the WebSocket protocol, then a proxy would also be required between the gateway and the server.

The introduction of gateways, similar to proxies, ensures the system's scalability, thus completing another requirement.

3.2.5 Router

The system proposed in this document relies on a Wi-Fi connection for entities to exchange data with each other. These need a Wireless Local Area Network in order to communicate. The entity which provides this network, is the router.

The router is responsible for creating an AP, generating a secure wireless network, and manages the Wi-Fi packets flow within this network, forwarding a packet sent from entities to another.

Lastly, this unit may also serve as a home gateway, redirecting packets sent from the Internet to the house server, via Wi-Fi, and *vice-versa*, as this is the only entity in the house's network that is connected to the outside. This assists in satisfying the requirement in which a house should be accessed remotely.

3.2.6 Home Server and Database

Lying within a property's limits, the Home, House, or Local Server is an independent unit that provides users with an application, in which they can oversee the system, allowing the management of reservations and system entities. Additionally, the application allows the control and monitoring of the house's appliances.

As data is required to be persisted on the server, this unit utilizes a connection to a dedicated database. The main objective of this second entity is for the server to store information relative to the property, allowing it to save information created by privileged users, including client reservation data and any relevant information to create a smart home.

The stored details should be used to authenticate both users and microcontrollers, ensuring only authorized entities are connected to the system. This feature should be handled by the server, which should also provide users the possibility to add this information to the system. It should be noted that a Home Server's database should only store information related to that property, preventing data to be shared among houses. For example, assuming two properties *A* and *B*, the database hosted in home *B* should not have access to *A*'s accommodations, as this data is not required in the second database, reducing the databases' size. Additionally, this storage constraint minimizes data that might be vulnerable, if an application flaw is exploited, in case a malicious user rents the property.

The Home Server is the central unit of the house as it connects all entities. It is responsible for controlling the smart devices inside a property, and connecting a user, within the house network, to their

controllers. This instance should send data collected by the sensors to the users and propagate any requests by the user to the corresponding actuator. In both situations, the server should test if the data propagated data could trigger off a user defined action. This is particularly important to detect undesired changes on the environment, with the objective to notify the user of this occurrence.

This server works as an API, called by both Smart Controllers, proxies and gateways with the objective of connecting the entire network to the users.

3.2.7 Cloud Server and Database

Responsible for integrating several homes into a single solution, the Cloud provides users the capability of remotely controlling and configuring the components of a house, therefore fulfilling one of the most important project requirements: allowing a manager to access the house without being connected to its LAN.

Using the Cloud's server, it should be possible for privileged users to access a house's data, ensuring it is management from afar. By dedicating a database to this server, users may access a property's data without requesting information from the House Server, reducing the number of messages, exchanged between the instances, and their size. Aside from increasing the total system storage, this trait requires databases to be synchronized, which may lead to collisions when database objects are changed in both systems, while the connection between the servers is lost. For this reason, the application must handle a strategy to contour this problem.

The system has been designed to work without requiring the connection between the Local and the Remote Servers, meaning after a change sensed by a device, the controller sends the notification to the House Server. This last entity should be able to store any important information regarding this change.

This server is directly connected to the Internet. For this reason, particular care has been taken into consideration when designing this connection architecture. The remote server waits for a House Server to establish a link between the two instances, using WSS, allowing them to exchange data between each other. It should be reinforced that, the security of this connection is of extreme importance, meaning the server needs authentication protection from both local servers and users, as only permitted entities must access the system.

Web-based information systems must be designed to minimize potential vulnerabilities, particularly a home automation system, as it contains details regarding a property's occupancy and allows the house to be controlled. For this reason, this server must not process data from an unauthorized connection, as it may compromise the entire system.

In similarity to a Home Server, this entity acts as an API, managing connections between the House Servers and a user.

3.2.8 Web Browser

A person may access the system using the web application, provided by one of the system servers, through a web browser.

As a Home Server does not listen for requests from outside of the property's network, the user may only access a house's information through the Remote Server, meaning a client cannot directly establish a connection to the property from outside of the property's network.

A person may access a system entity by establishing a connection through the house's server or the Cloud instance. After this moment, it should be possible for the person to utilize any other system entity, using a Wi-Fi connection.

Only an authorized user must access a house system. It is absolutely crucial that the connection between a browser and a server provided with authentication services, as only authorized users must access the system.

3.3 System Architecture

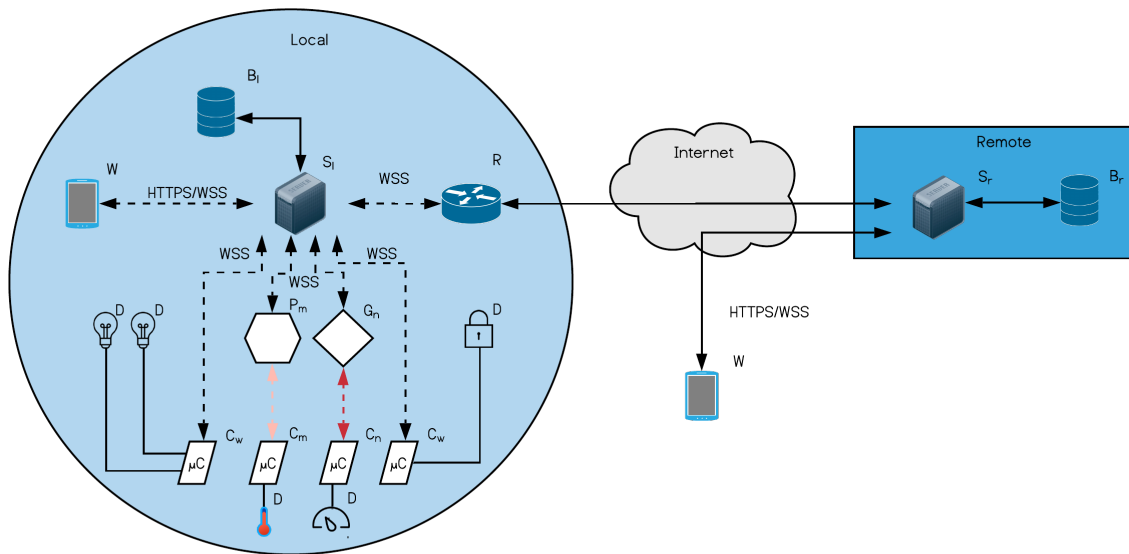
A system's architecture represents the model's structure, the behaviour and the relationship between the entities involved in that system. These architectures may be composed by a perception, a network and an application layer [18]. The perception layer interacts with the environment, actuating and/or collecting data from it. The application layer is composed by all applications which integrate an IoT solution. The network layer, is a middle layer, as it connects devices, linking the perception and application layers, by transmitting data from one entity to another.

It is proposed that a property contains smart devices, connected to a microcontroller. The first interacts with the house's environment, generating data, which is collected and processed by the second, and then sent to the house's central unit, a server. A house's server gathers data from several controllers, processes it and can even send it to a cloud hosted server.

Figure 3.1 provides an overview of the designed system architecture. On the left, drawn in light blue, the components of a smart home (local entities) can be found. A server (S_l), connected to a database (B_l), and Wi-Fi using controllers (C_w), composed by any type of smart devices (D , depicted with a lock, a meter, a thermometer and light bulb icons). All of these, connected by the router (R).

The system also allows the owner to integrate Wi-Fi supported controllers, that do not use WebSockets, thus requiring a proxy to connect to the Home Server. The controller, C_m , transfers data to the server, using an application protocol $m! = WebSockets$, thus requiring a proxy, P_m , to connect to it.

Similarly, the owner may introduce controllers that utilize other communication protocols. The controller, C_n , identifies a controller that transfers data using a communication protocol n , such as BLE, ZigBee, etc (excluding Wi-Fi), which requires a gateway, G_n , for that same protocol.



Connections	Smart Devices (D)	Other Entities	
— Wired/Wireless	Smart Meter	Smart Controller (C_w, C_m, C_n)	Router (R)
- - - Wi-Fi	Smart Lock	Local (S_l) / Remote (S_r) Server	Remote database (B_r)
- - - Non-Wi-Fi	Smart Light	Web Browser (W)	Gateway (G_n)
- - - Non-WSS	Smart Thermostat		Proxy (P_m)

Figure 3.1: System's Architecture.

In darker blue, on the right side of the image, the system's remote entities are designed. These are composed by a server (S_r) and a database (B_r), ensuring properties can be managed and monitored from outside of the house's network.

The architecture also contemplates the presence of a user, via a client web browser (W), which can be connected to the house's server using the property's LAN, or to the Cloud Server, by accessing the Internet. In both situations, data is exchanged using HTTPS or WSS.

This layered architecture allows managers to extend the application by adding different smart devices to it and access them from a single application, the Cloud Server.

3.4 Communications

Three entities are required when two people communicate with each other: a sender, a message and a receiver. It is required that these two people share a set of rules, allowing the sender to formulate the message in a way the receiver will understand it. One example of these guidelines is a language, which

requires entities to understand the meaning of each word and, using grammar, they may construct a sentence.

Machines also need to determine rules to communicate with each other. It has already been defined that the system will communicate over Wi-Fi, using a WebSocket Secure connection.

As the protocols defined for this system rely on TCP, which ensures WebSocket messages cannot be lost, provided the connection remains indefinitely open, it allows devices to send information without requiring an acknowledgement from the receiver. However, entities should not depend on this characteristic, as the connection may be lost after a message is sent but not yet received, or processed. For this reason, in communications between two machines, the system should rely on confirmation messages, *acknowledges*, to ensure the message was processed by the receiving peer, allowing for messages to be re-transmitted after sudden disconnections.

This section focuses on defining the communication format between a Smart Controller and the Home Server, and between this last entity and the Remote Server.

3.4.1 Controller-Server Connection

Controller-Server Connection, as the name suggests, represents the set of messages exchanged between the controllers and the house's central unit, the server. These follow an HTTP based pattern, where each text message dispatched is referent to a smart device's property. For this reason, messages require the identification of the controller's smart device, its property and the value changed, or to be changed, depending on the type of property.

All messages sent from the controller may contain a timestamp, composed of a date and time, and used to give extra information to the API and, subsequently, to the user, about the moment in which the environment change has occurred. This is particularly important for messages that have been delayed or failed to be sent, ensuring the user has access to this information. Whenever the controller connects to the server's API, it should request for the current date and time, allowing the first entity to inform the interface of the moment an event was detected in the house.

Messages are composed by, at least three fields, and, at most, five. These are separated using a forward slash character, thus mimicking a URI's resource path. This format is shown below:

method/device/property(/value)(/timestamp)

Where:

- *method* indicates the action being perform, such as an HTTP method;
- *device* is unique alphanumeric value which represents the device being altered, or the server, when requesting for data. A controller has as many values for this field, as smart devices it controls;

- *property* represents, through alphabetic characters, the device's property;
- *value* constitutes a set of alphanumeric characters associated to a property. This may indicate the a sensed value, when the message is sent by the controller, or the value to which an actuator should be set to, if the message is received by the entity;
- *timestamp* – this optional field is composed of digits representing the moment an event has occurred. Should follow the format year-month-dayThour:minutes:seconds.

To these requests, the API must respond with an acknowledgement, ensuring the controller is informed that the interface processed the sent data, in case the request performed a change in the Home Server. Similarly, whenever the controller requires information from the server, such as the current time, the server should respond with a single value, simplifying the entire process.

Following this message format ensures devices using other application protocols to be added to the system, as most protocols already follow this convention. For example, CoAP, being based on HTTP, uses URIs and MQTT's topics are also identified by this format. This facilitates protocol conversion, ensuring the system can be expanded.

3.4.2 Inter-Server Connection

Inter-Server Connection is the communication channel created between the house and the remote server.

This set of messages transfer more information than those exchanged using the aforementioned Controller-Server Connection. After comparing the two most used data interchange formats, JavaScript Object Notation (JSON) and Extensible Markup Language, it is suggested that the messages use the first as it is a "lightweight, text-based format, defines a small set of formatting rules for the portable representation of structured data" [19]. It has been chosen because it is easier to understand and visualize by a person, it is shorter in size and, therefore, faster to process, using fewer resources when compared to Extensible Markup Language [20].

Transferring complex data in larger amounts, messages within this channel are required to migrate changes in a house's environment to the Remote Server, and to synchronize both entities' databases. For this reason, they have to identify the type of object being transferred, the action being performed and, when required, the object's data. As such, it is proposed that a request performed by one of these server resembles an HTTP request, containing a method, followed by a resource identifier and, possibly, a payload. The server which receives this message should provide a response, containing a status code, which allows the requesting server to acknowledge if its message was successfully processed, and a payload, providing extra information.










Entities	Layers
 	Application
  	Network
   	Perception

Figure 3.2: System layers and entities

3.5 Summary

This chapter identified the main requirements of the system used to design the proposed solution. A description of the entities used to satisfy this problem is presented, followed by the suggested architecture, to create a smart home solution to assist renting a property.

Figure 3.2 represents the system architecture layers and their composition, using the system entities depicted previously, in the System's Architecture. figure.

The system entities have different responsibilities. Smart Controllers manage one or more smart devices, interacting directly with the environment, collecting data from, or actuating on it, thus belonging to the system's perception layer.

The heart of a house, the Local Server, provides an API for the users to control the house's entities, allowing the manager to access and manage the system. Belonging to the application layer of this system, the server is connected to a database, responsible for storing information about the system. Additionally, as to ensure a house can be constantly monitored, this server is responsible for sending alerts to users, so that critical situations in a house can be detected.

Using a WebSocket connection, the controller may establish a connection to the server's API, via a Wi-Fi network created by a router, allowing the home environment to be managed. As managers may want to connect Smart Controllers that do not use neither WebSockets, nor Wi-Fi, the design contemplates the integration of proxies and gateways, connecting devices that do not communicate via these protocols to the server. These three devices (router, proxies and gateways) are responsible for connecting all entities, thus belonging to the network layer of the system.

Finally, a Cloud hosted server provides an API for a Home Server to connect to, using the WebSocket protocol, it is possible for the entire system to be supervised in real-time, using a single application, accessible through the Internet, regardless of the number of houses a person is responsible for managing.

4

System Implementation

Contents

4.1 Smart Controllers	43
4.2 Home and Remote Servers	49
4.3 Summary	60

The described architecture has been proposed to solve the aforementioned problem. With the goal of testing this system, a prototype has been developed in which two microcontrollers, a House Server and a Cloud Server, both with databases, have been included.

The forth chapter of this Dissertation focuses going one step further and describing the implementation decisions taken, in order to create the system described chapter 3, detailing some of the its characteristics, to satisfy the mentioned functional requirements. It starts with a section dedicated to the developed Smart Controllers, followed by a another detailing the servers' implementation and behaviour. The two servers provide similar functionalities, as they share information about the system, thus performing identical tasks. For this reason, they are detailed in the same section, with differences between the two being highlighted.

4.1 Smart Controllers

This project focuses on creating a smart house system for assisting people in the process of renting a private property. Its main focus is not the implementation of smart objects. Nevertheless, this section describes the controllers and, consequently, the smart devices used in order to asseverate the quality of the system.

The Smart Controllers mentioned in this section were created using the NodeMCU board and its code was written in C language, using the Arduino Integrated Development Environment ¹.

Despite the fact that this project is composed by each entity of the system, in particular the servers, the devices were implemented with the purpose of being used in property rental scenarios. For this reason, the controllers' design considered that the devices could operate on their own, without requiring a connection to an API.

To ensure the controllers can operate on their own, a configuration state has been implemented, where Smart Controllers create a password protected wireless AP, allowing one client terminal to establish a connection to it. A controller can enter this state, provided that a specific button is pressed within 5s after this unit has been powered on.

Upon detecting a successful connection to the AP, the NodeMCU creates a captive portal, which a user may configure the module, allowing it to work on its own or using a distributed feature, in which it communicates with an API. This second mode requires the user to define the LAN's authentication credentials. Using the AP, it is also possible to set the API's Internet Protocol (IP) address, ensuring the controller can be added to different systems. These configuration values are stored in the microcontroller's EEPROM and obtained from it every time the the device is powered on, allowing the values to

¹Esheridan, "Integrated development environment", available online at <https://www.arduino.cc/en/software>, last access in September 2021.

be restored in case of power loss. Configurations specific to each device are provided in the device's segment of this section.

If the remote mode is enabled, each controller uses a header, containing an access token, to authenticate itself in the system, when connecting to an API. After this connection is established, the microcontrollers requests the API for the current date and time, allowing this entity to inform the interface of the moment an event was detected in the house. Following this request, the controller informs the API of the current state of each device, as these might have changed prior to this channel having been created. Additionally, whenever an event occurs in the environment, it triggers a procedure in which the occurrence is added to a message queue and then sent to the API. The queue is used to store the sequence of events detected, and their time, allowing for the controller to re-send the queued data, by the order the events were detected, in case it disconnected from the API. Sent notifications require an *acknowledge*, removing the event from the queue, by the order they were sent, thus ensuring the API processes a message at least once. TCP guarantees that the n^{th} sent *acknowledge* corresponds to the n^{th} sent message, provided that the receiver processes messages by the order they were received.

Two Smart Controllers have been created. A smart door, with smart lock features, thus controlling the access to the house. This device has been created to test and demonstrate how the system reacts to the start and end of a rent period. The second, a smart light, was designed so that external users could test the application's interface, as mentioned in Chapter 5.

It should be noticed that the controllers can operate even if the connection to the Wi-Fi or to the API cannot be established, in which situation, when configured to do so, it will attempt to create this connection, at periodic intervals, until it succeeds.

Bellow, details are provided for the two controllers, with a segment dedicated to the its components, another to its behaviour and a third which describes features related to the device's operations when connected to an API.

4.1.1 Smart Door

The smart door was created for a person to access the inside of a house, using a code, and to sensor changes on a door, in order to notify an external entity. The created device can supply current to an electric latch, unlocking a door.

It can also store of up to three codes, allowing managers to define more than one code. For example, a master code can be used by the manager and temporary code by a tenant, which can be enabled during at the start of its reservation and revoked afterwards.

4.1.1.A Components

This device been developed using two sensors and two actuators: a keypad, a reed switch, a relay and a red-green-blue light emitting diode.

The first sensor, the keypad, is composed of 16 push buttons: 10 digits, the first four letters of the roman alphabet, an asterisk (*) and a hash sign (#).

The light emitting diodes are used by the controller to signal the user, exhibiting a green color whenever the door is unlocked, and a red color in situations where the inserted code was invalid. The module for these diodes contains a 150Ω resistor, for each light emitter, to reduce the voltage received by it.

The first actuator is a mechanical relay, used for simulating an electric bolt, unlocking a door when current is supplied to it. According to the relay's datasheet ², the relay switch requires about $0.36W$ to be closed, which cannot be provided by the NodeMCU, having a maximum current of $12mA$, per GPIO pin. By taking advantage of an optocoupler (a light operated switch), embedded in the relay's module, and by using an external power source, it is possible to isolate the microcontroller from the higher voltage, thus protecting the controller in case there is a problem within the relay. In this project, $5V$ have been used to power this electric switch.

The last module of the device, the reed switch, closes a circuit in the presence of an external magnetic field. The switch contacts touch in the presence of a magnet, closing the circuit, and allowing the controller to perceive the state of the door, by detecting the presence of current. By positioning a magnet in the door and the sensor on its frame, the controller is able to detect if the door is opened, or closed. This module requires a connection to ground, with a $1k\Omega$ resistor, to prevent invalid readings.

Figure 4.1 illustrates the wiring used to integrate the four modules in the controller and the door latch. These components are not to-scale.

4.1.1.B Operation

A smart door task starts by sensing the buttons pressed on the keypad. To prevent situations where the key is pressed for a large amount of time, leading to invalid readings, the button is only considered pressed after being released.

All buttons can be used to create an access code, with the exception of 'B', as this button is reserved, leading to a total of 15 characters. The device requires codes with a size of 4 characters, as it would allow for 50625 different access combinations.

Using the controller's AP, it is possible for a user to determine up to three access codes, that can be inserted in the keypad to provide access to the property. These codes, whenever updated, are stored in the ESP's EEPROM allowing the values to be persisted in case the device is turned off.

²SONGLE, "SONGLE RELAY", available online at <https://components101.com/sites/default/files/componentdatasheet/5V%20Relay%20Datasheet.pdf>, last access in September 2021.

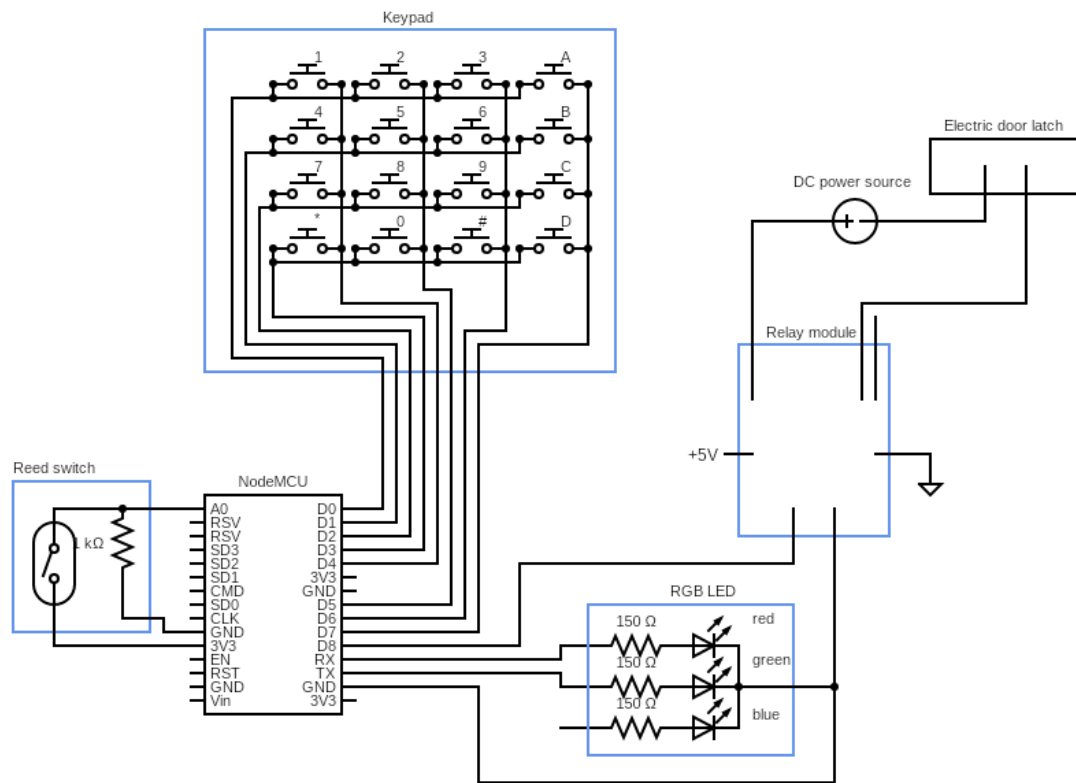


Figure 4.1: Smart door controller's wiring diagram.

Upon detecting the insertion of a correct code, the controller uses an electric signal to trigger the relay for a period of 3s. In case the introduced key fails to match the stored codes, after three attempts, the keypad is temporarily disabled, thus delaying a person trying to insert random codes, in the hope of finding the correct one.

It is also worth mentioning the inserted code is reset whenever 10s have passed without a button being pressed, preventing codes to be left half written.

4.1.1.C Distributed Features

The described characteristics are available whenever the device operates on its own. When connected to an external entity, the number of features the device can provide to a user increases, as it can receive commands from or emit them to an API.

The controller can send different notifications the interface, whenever the door changes its state. It sends different messages, depending on the access code used to unlock the door. This can be used to inform an owner about who entered the property. The device may also detect and send warning messages, when the door has been opened without an access code having been introduced. This is

described as the door being "forced open", representing a critical situation, requiring immediate intervention from a manager.

There are two additional situations the device can detect, which may be harmful for the system. First, it may sense if a door was left open, which happens when the the sensor does not detect a closed door within one minute after it has been opened. The second warning generated is created when the maximum number of attempts (3) to insert an access code is reached.

Additionally, the connection to another entity allows a functionality where the door may be unlocked remotely. This property may present a security flaw, as it allows the door to be unlocked at will. For this reason, it is required that a person presses the bell ring button, the keypad's reserved button ('B'), to use this functionality. Once opened or ten seconds after the button is pressed, the door can no longer be opened remotely, unless the button is pressed another time, thus ensuring it is only unlocked when someone is at the property's entrance.

Lastly, the three access codes may be changed remotely.

Appendix A can be consulted to observe the smart door's process flow.

Table 4.1 depicts all possible messages exchanged by the smart door, providing a description of each of them. It indicates the options for the *method*, *device*, *property* and *value* of the message, following the format mentioned in Chapter 3: *method/device/property(/value)(/timestamp)*. To facilitate the comprehension of this format, an example for a message indicating the door being opened using the its primary access code, at noon of the first day of the year 2021, would be:

POST/DOOR/State/Open/2021 - 01 - 01T12 : 00 : 00

As previously mentioned, the controller's configuration state can be used to enable or disable these notifications.

4.1.2 Smart Light

A smart light device is used to operate a light bulb. During this subsection, the implementation of a smart light that allows users to control a bulb's state is described.

4.1.2.A Components

The device's actuator is a relay, used to close or open a light bulb's circuit, thus turning it on or off, respectively. The device also contains a sensor, a switch, which can be used to manually control the light's state, similar to a house's switch. This circuit also uses of a $1k\Omega$ resistor prevent invalid readings. The connection between the microcontroller, the relay and the switch is represented by Figure 4.2, which is not to-scale.

Direction	Method	Device	Property	Value	Description
Outgoing	GET	Server	Time	---	Obtain current time from API
	POST	Door	State	Open	Door opened using the primary code
				Open2	Door opened using the secondary code
				Open3	Door opened using tertiary code
				OpenWeb	Door opened by remote server
				Closed	Door closed
			Warning	ForcedOpen	Door Opened without a code
				LongOpen	Door left open for too long
				MaxAttempts	Keypad inserted code failed 5 consecutive times
			Bell	Rang	The door bell button was pressed
Incoming	POST	Door	Bolt	Unlock	Unlocks the door
			PrimaryCode	{value}	Changes the primary code to {value}
			Secondary Code	{value}	Changes the secondary code to {value}
			TertiaryCode	{value}	Changes the tertiary code to {value}

Table 4.1: Smart door message set and description

4.1.2.B Operation

This device's operation is fairly simple to describe. The NodeMCU opens and closes the light bulb's circuit, using the relay, depending on the state of the switch.

When the circuit is open, the device senses the presence of current, flowing from the 3.3V pin to the controller's digital pin. Alternatively, when the circuit is closed, the controller does not detect current, as it flows to the ground pin. If the value sensed in instance t differs from the value sensed at $t+1$, then the device alters the state of the light, through the relay.

4.1.2.C Distributed Features

There are only two family of messages used by this smart device, both relative to the device's single property, its state.

Table 4.2 demonstrates these four messages and their description. There are two equal pairs of messages. The first is triggered when the light changes its state, in which the controller notifies the API of this occurrence. The second type should be used by the same interface, to notify the controller that the state of the light should be changed, allowing a user to remotely control light.

It is important to notice that, when operating without receiving remote commands, the relay's closes the light bulb circuit whenever the switch closes the device's sensing circuit. Upon receiving commands from the interface, these circuits will no longer remain synchronized, as the relay state changes, but the

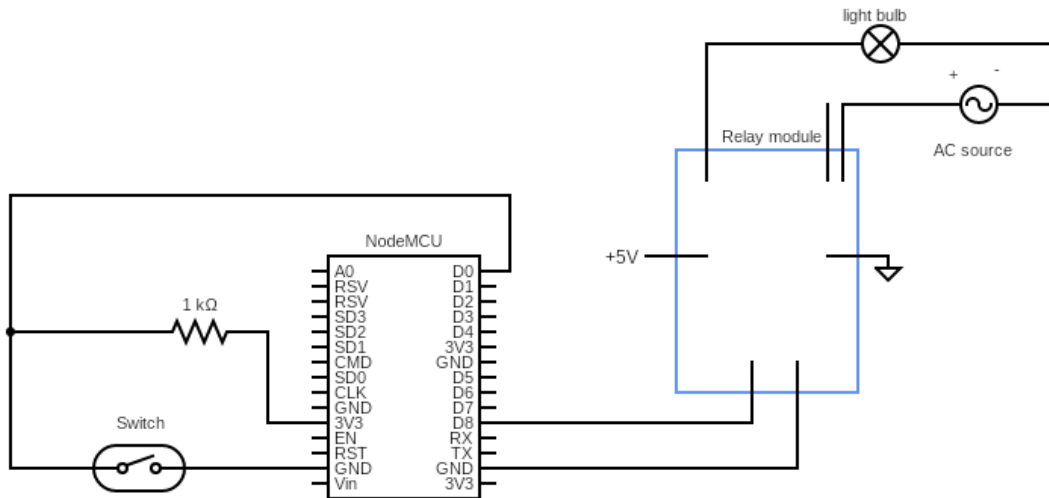


Figure 4.2: Smart light controller's wiring diagram.

Direction	Method	Device	Property	Value	Description
Outgoing	GET	Server	Time	---	Obtain current time from API
	POST	Light	State	On	Light turned on
Off				Light turned off	
Incoming	POST	Light	State	On	Turn on the light
				Off	Turn off the light

Table 4.2: Smart light message set and description

switch's does not. For this reason, the light bulb's output value cannot be deducted from the switch's state. By storing the switch's and the light's current state in the EEPROM, it is possible for the device to always present the correct light state, even in situations where the switch is changed while the controller is turned off.

4.2 Home and Remote Servers

The Home Server is connected to every entity on the system, managing requests from both users and machines and the Cloud Server allows a manager to remotely access one or more properties. This section describes these two entities most important system objects, some of their attributes and their relevance towards the fulfilment of the system requirements.

As both entities required the implementation of an application, first an introduction to the softwares used to assist in the development of the interfaces are is provided:

- Django Web Framework, as the name indicates, is a software that assists in the development of web applications. It provides an architecture pattern composed by *Models*, *Views* and *Controllers*. *Models* represent the objects of the project, making it responsible for handling the data structures, *Views* deal with the information representation presented to the user, via a GUI, and the *Controllers* are the communication channels utilized by the users to interact with the *Models* ³. The use of this framework reduces the implementation time taken to develop the application which servers requests from machines and users.
- SQLite is a relational database management system, desired when developing projects that utilize microcontrollers and single-board computers, as it is embedded in the application ⁴. Its fast engine and its reduced memory consumption library are ideal for the mentioned devices, as their hardware is constrained. For this reason, this relational database has been chosen to store the servers' data.

To host the applications, two entities have been chosen. The Raspberry Pi 3B +, introduced in the *Products* section of Chapter 2, has been used to host a house's application, created in Python programming language with the assistance of the aforementioned technologies. Similarly, the Cloud Server uses SQLite and Django Web Framework, but it was hosted on a laptop with 1.8GHz CPU, 16Gb of RAM and 1Tb of storage, connected to the Internet. This was a very powerful computer for the running the application, but it simulates an application hosted in the Cloud provider, avoiding the costs associated with a Cloud provider.

It should be noted that each instance has an attribute, a primary key named *id*, that allows the system to uniquely identify a table's object. However, most of these contain a unique text property, which the server uses to present said object to the user, as it is easier for a person to identify a smart door named "Entrance door", than the smart device with *id* 22.

Additionally, both servers handle sensitive data, such as user passwords, which, if obtained by a malicious user, may endanger the entire system. As such, their values, in the database, are not stored in plain text. Instead, they are saved using a one-way hash function, preventing the value's decryption. Any exceptions will also be mentioned in this section.

Appendix B, containing a description of the system *Models*, and Appendix C, providing the designed database architecture, contain information that can be useful throughout this section, to assist in the comprehension of the system objects and how these satisfy the system requirements.

4.2.1 Reservation Management Tasks

Tasks relative to the management of a reservation are one of this system most important characteristics.

³Django, "FAQ: General", available online at <https://docs.djangoproject.com/en/3.0/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>, last access in September 2021.

⁴SQLite, "What Is SQLite?", available online at <https://www.sqlite.org/index.html>, last access in September 2021.

Reservations require a user to be associated to a house within a period of time. The following segments describe how the applications allow the system to manage reservations, providing information towards their utility to the system.

4.2.1.A House

The system is required to provide owners with the possibility to access and supervise several properties.

This object's main objective is to relate a system object, such as a *Reservation*, to its corresponding house, ensuring data can be associated with the property they belong to.

4.2.1.B User

A *User* object is the server's representation of a person registered in the system. It contains a *username* attribute which, when combined with another attribute, the *password*, grants a user access to the application. These objects may also store a phone number and an email, fields which assist in the house management rental context, as it allows the system and a house manager to contact a client.

Bellow, the four *User* types and their authentication limitations are described below:

- *Administrator*, or *Admin*, is person with permission to freely navigate through the application without restrictions. It has been designed to be attributed to the system owner, being store in every database of the system, ensuring a user of this type can access the application in every house;
- *Manager*, another privileged account type that can be used to manage the system. A person with this type of account has more limitations than an *Admin*, as it can only access the data relative to houses it manages, therefore being ideal for a person contracted to manage the property. In addition to the Cloud, data relative to this type of account is stored only in the databases of the houses the user manages.
- *Client*, a more restricted account type, has been created to represent a person renting the house. The access to the application must be limited to the house the user is renting, during its reservation time;
- *Staff Member*, another limited account, similar to a *Client*, but designed for a person whose job is to assist in maintaining the property.

These four types allow the applications to satisfy the requirement to have restrict access for different users.

4.2.1.C Reservation

Reservation objects are used to represent the allocation of a person to a property, within a two dates. Two types of *Reservations* can be created: *Client Reservation* and *Shift*. A client's reservation is represented by the first, while the second denotes a staff member's shift. During this Dissertation, the term *Reservation*, is used to address both of these objects.

Reservation objects are kept in the Remote Server's database and in the house to which it belongs. Similarly, data relative to clients and members of staff is shared to a Home Server, once a *Reservation* for that property is created. This characteristic contains data to the houses it belong to, reducing the information shared between instances.

Currently, nor two clients, nor two workers may have a simultaneous *Reservation* for a house. However, there might be one *Shift* and one *Client Reservation* occurring at the same time.

There are four scenarios where two reservations may overlap. Figure 4.3 represents the four scenarios using two hypothetical *Reservations* of the same type and for the same house, where st_n and et_n represent the *Reservation n*'s start and end times (composed of a date and time), respectively. Assuming that *Reservation1*, illustrated in blue, is being inserted in the system, and that *Reservation2*, represented in black, already exists on it, it is possible to conclude that in every scenario st_1 occurs before et_2 and et_1 after st_2 , with $st_n < et_n$.

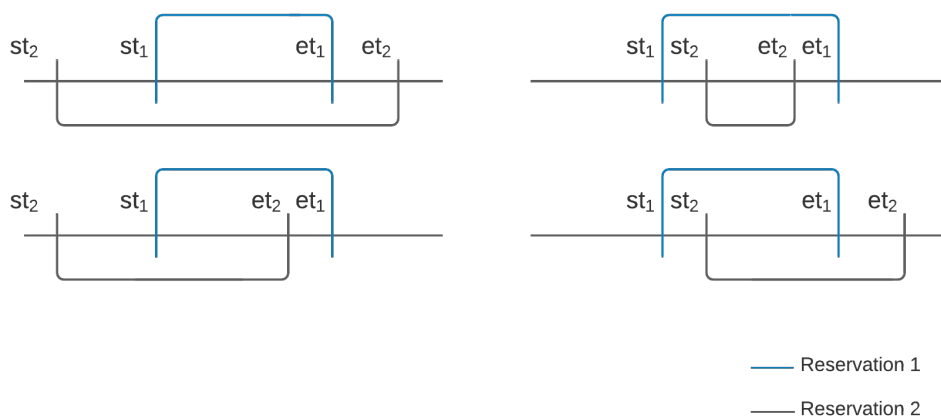


Figure 4.3: *Reservation* overlapping situations

Additionally, *Reservations* have one attribute, *canceled*, used for disabling the actions related to a reservation, without deleting it from storage. The system allows for several *Reservations* to be assigned to a property in a given period of time, provided that at most one is not canceled.

Lastly, *Reservations* have a code associated to them, which can be defined by a privileged user, or randomly generated following a set of defined rules. This code is used to provide users access to certain

domotic features of the system, explained in 4.2.2. Two consecutive *Reservations* for the same property cannot have the same code, preventing two users to share the same access to said features.

4.2.2 Home Automation Tasks

The system is also designed to be used for home automation purposes. Instances related with the representation of smart devices are described bellow, describing domotic characteristics of the system, ensuring the houses can be monitored and controlled.

4.2.2.A DeviceType and Properties

DeviceType is the term chosen to represent a family of devices with the exact same properties. This class allows users to insert several identical smart devices to the system, as these contain the same characteristics.

These contain at least one *Property*, corresponding to the device's properties, which may act as sensors, actuators or both. *Properties* also contain a field, *name_in_controller*, used by the Home Server, to recognize a device's property in messages sent by the Smart Controller. Similarly, using the attribute *name*, the application identifies the same property, in communications with a user.

Currently, a *DeviceType* supports three types of *Properties*:

- A *DiscreteProperty* represents a property whose values are countable and can be described by a finite set. They are categorical, such as the state of a light bulb which can be assume the values "On" and "Off". Each of these is represented by the object *Value* ;
- A *ContinuousProperty*, defines numeric data within an interval, comprised between a minimum and a maximum values, with a determined resolution. Measurements, such as the sensed energy consumption, can be introduced in the system using this *Model*;
- A *CodeProperty*, whose value is composed by a set characters, is utilized to restrict access to specific appliances, which require an authorization key to be used, such as a smart lock's access code. This *property* can only be associated to an actuator, and may be directly associated to a *Reservation's* code, allowing users to have temporary access to certain devices.

It is required that the system filters which *Properties* can be accessed by each user, as some device functionalities should not be accessible by all people. This feature can be determined using the *User's* type.

4.2.2.B SmartDevice

A house's smart device can be defined using the object *SmartDevice*. An instance of this type is stored in the database of the property to which it belongs. The same happens to all objects that inherit this one.

Whenever a *DeviceType* is added to the Home Server, this entity's application creates a database table with as many columns as the number of properties defined in *DeviceType*. Additionally, a row is added to this table whenever a *SmartDevice* joins the application, meaning each value (defined by a row and a column) corresponds to the latest value sensed by a device, or set by a user, depending on property's type. Upon request, these values can be accessed by the application, allowing a user to perceive the house's environment.

This configuration constitutes an important characteristic of the prototype, as it allows users to define different types of devices, thus ensuring the system's scalability. Additionally, this table can be accessed by the application, assisting users to monitor the house.

4.2.2.C Action and Event

The last objects described for this home automation system are *Actions* and *Events*. As the name indicates, these allow an action to be performed, by the system, upon the detection of an occurrence (event).

There are two types of *Events*, both requiring a condition and a value, or threshold:

- A *PropertyEvent*, detects a change on a smart device's property. Whenever the House Server receives a message from a Smart Controller, sensing a change in the environment, it compares the message's value with the one defined by the user condition. This also applies for actuators, whenever a user changes a device's property;
- A *DateTimeEvent*, in contrast, utilizes the server's time as a value, being used to detect a defined moment.

In both situations, if the *Events* condition test is verified, the server triggers an action. These conditions allow the server to identify equalities and inequalities (greater or equal to, greater than, different, lower or equal to, lower than) between the user defined value and the one detected by Home Server. Additionally, there is a specific condition that allows the server to always detect an event, ensuring an action is always triggered independently of the detected value.

Bellow, two types of actions are described.

CodePropertyAction

A *CodePropertyAction* is used to change all *CodeProperties* in the house, using its current *Reservation*, so that a tenant can utilize a code to utilize a certain device. When a *Reservation* is created, a specific

type of *DateTimeEvent* is generated, used to detect the start of the *Reservation*, triggering this action and setting the house's *CodeProperties* to the *Reservation's* code. The same happens for its end, allowing the access to be revoked. This is particularly important to automatically change, for example, the smart door's access code.

NotificationAction

When defining the system requirements, it has been concluded that users must be provided with the possibility to monitor changes in a house environment. The *NotificationAction*, when triggered, satisfies this requirement, by generating a notification, alerting a user (*Client*, *Manager* and/or *Admin*), via email and through the application, whenever an event is detected. Additionally, this occurrence can be stored in the database, allowing a house to be monitored over time. For example, a *PropertyEvent* can detect if the implemented smart door's controller sent message detecting a door having been forced open, leading to an email being sent to the owner, allowing the event to be investigated. Similarly, the user may be notified if a Controller-Server or an Inter-Server Communication is lost, ensuring this situation can be detected and fixed at will.

4.2.3 Controller-Server Communication

Controller-Server Communication channel is composed by messages sent by the Home Server to a Smart Controller and *vice-versa*. The first, being generated by the server system, triggered by an *Event*, or converted by the computer as a result of a message send by a user and the second being created by the controller itself. For this reason, this subsection first mentions *ControllerNotification* objects and their importance for messages sent by the server.

4.2.3.A SmartController

Suggested by the name, Smart Controllers are represented in the applications using the *SmartController* model.

As it is required that all microcontrollers provide trusted authentication, users are required to define a controller's IP address and access token, which the server stores in its database. Whenever a Smart Controller tries to establish a connection to the server, it shares this information in the HTTP *handshake* headers. The Home Server compares these values with the ones stored, thus accepting or declining the connection.

4.2.3.B ControllerNotification

A Smart Controller can have several *ControllerNotifications* assign to it. This object has two tasks. First, it saves any state changes in a Smart Controller, allowing a user to know whenever the entity was connected and disconnected from the API. Second, it is the main unit responsible for transferring data from the House Server to a controller, allowing the first to store the content of a message, ensuring it can be resent until an *acknowledgement* is received, thus deleting the object.

A *ControllerNotification* is only stored if the message was generated by the system, such as those created by *CodePropertyActions*, ensuring the application always delivers these messages, thus ensuring the access code is changed before and after a *Reservation*. Additionally, this feature prevents messages sent by a user, with the intend to change a device's property, from being sent out of time, such as an appliance being turned on hours after a user performed this request, due to a connection lost.

The Home Server processes messages received from the controllers, based on the message format mentioned throughout this document, *method/device/property(/value)(/timestamp)*. Currently, it only processes requests containing a *POST* method, associated to a change on the environment. The *GET* method requesting the server's current time: *GET/Server/Time*; is an exception, as it is used to synchronize the Smart Controller's times.

4.2.3.C SmartDevice Caching Task

The House Server handles a reduced number of users but may contain a large number of Smart Controllers. Some of these controllers contain smart devices which sense the environment, sending messages to the Local Server, which then stores this value on its database.

A storage unit read speed is large when compared to its write speed. To prevent bottleneck situations, where controllers send data faster than the server can write it to its storage, a structure has been implemented in the application's process, to temporarily store these values without saving them directly in the database, as the computer's RAM is much faster.

Upon receiving a valid value, the server stores it in the structure, acting as a cache. Once every 30s, this data is saved in the database, allowing a user to have access to this information. This approach may lead to data losses if the application suddenly terminated, but it reduces the workload associated with the database write operations.

To prevent the program from run out of memory, this cache has a maximum size of 100 devices. Whenever this limit is reached, the program prunes the structure, storing all values sensed in the database.

4.2.4 User-Local Server Communication

A user may connect to the Local Server using only HTTP and WebSocket protocols with TLS protection, as the server rejects any others. The first is utilized to request data from the server, or apply a change on it, and the second applies to exchange real-time messages between both parties.

4.2.4.A HTTP Connection

The HTTP connection is established by a user, when accessing the Local Server's application, and provided so that the user can access the system, through a website.

This channel allows users to navigate through the web application, by performing requests to a server's resource, via URIs. To these requests, the server provides a web-page structured with Hyper-Text Markup Language, Cascading Style Sheets and JavaScript.

Users are required to login into the application, using the authentication pair associated with their account. To an approved login, the server responds with the *DeviceType Menu* and a header called a *session cookie*, containing a unique associated with the user's session. This value is sent by the user in each subsequent request, allowing the server to recognise the account, without requiring the person to perform a login in each web-page.

The *DeviceType Menu* contains a list of the system *DeviceTypes*, and can be used by the users to request for the house's smart devices. To this request, the server responds with the *SmartDevice Menu*, presenting users with said devices, filtered by the user permissions. Both of these web pages, as well as the remaining ones provided by the application, for a user to be able to access the system's functionalities, is shown in Appendix D

Privileged users (*Administrators* and *Managers*) may, additionally, use this communication link to access the *Admin Website*. This is a section of the web application in which users can navigate through a series of web-pages, each assigned to, at least, one system object. These pages provide users the possibility to create, update, view and delete the objects, therefore allowing them to manage the application and the house, thus fulfilling another system requirement.

The applications' web-page templates are presented in Appendix E and the navigation structure in Appendix G.

4.2.4.B WebSocket Connection

This second communication channel is also available for all types of users, allowing them to send data to the server, which can also push messages to the connected user.

The WebSocket connection has two functionalities, described below:

- First, it is used by both parties to send data, allows them to exchange domotic messages between each other. This means the user can ask the server to change a smart device's property and that the server may inform the user when a change is detected in the house's environment. Both of these depend on the user's type, preventing it from freely controlling or monitoring the property;
- Second, it is utilized by the server to push data generated by *NotificationActions*, creating alerts to a user when an event is triggered, ensuring they receive the alert in real time, provided that they are connected to the application.

This connection is initialized by the user's web browser, using JavaScript written code, which is delivered by the server, whenever the user requests a resource.

It should be noted that the use of this connection, when utilized to access the house's smart devices, is limited to the *SmartDevice Menu*. In contrast, alerts need to be delivered, independently of the web-page the user is utilizing, therefore being required throughout the website.

4.2.5 Inter-Server Communication

This subsection describes the communication channel between the two prototyped servers, by introducing the *Server* object. Additionally, it explains the communication channel between the two servers, presenting the object responsible for this link, named *ServerNotification*. Lastly, it details how the two applications synchronize their databases, describing how the system handles database conflicts.

4.2.5.A Server

A *Server* object, as suggested, represents a server entity in the system. In a Local Server it identifies the Remote Server, therefore being limited to one object. In contrast, the Remote Server can have as many House Servers, as the number of properties in the system.

The House Server requires the Remote Server's IP address to establish a connection and an access token. It is required that a user introduces this information in the Cloud Server's application to ensure the first entity can authenticate itself, when connecting to the second's API.

When the *Server* object is added to the Home Server, it informs this entity that a connection should be established to the Remote Server, allowing them to communicate using secured WebSockets. Similarly, this connection can be canceled.

If this communication is broken, the Local Server will attempt to reconnect at periodic intervals. During this time, users connected to the house's LAN may access the Home Server, provided that the disconnection was caused by a reason external to it, such as a connection loss to the Internet. Similarly, the Remote Server may be contacted by a user, through the Internet. However, in both situations,

messages between the two instances cannot be exchanged, meaning the remote access to a house's environment is not permitted.

4.2.5.B ServerNotification

At the resemblance of a *ControllerNotification*, a *ServerNotification* has two purposes. First, it saves any state changes in a server, allowing a user to know whenever one of these entities was connected or disconnected from the the other. Second, it is the object used to represent messages exchanged between two servers, when this distributed mode is enabled.

These objects can handle two tasks. First, they allow for the propagation of user requests, to a Smart Controller, and *vice-versa*, using the House Server as a message broker. Additionally, as this is a hybrid system, with the purpose of monitoring the house even if the connection between the House and Cloud Servers is broken, it is required that the system databases are synchronized. As such, *ServerNotification* objects are responsible for this process, ensuring the data is updated in both servers.

Bellow, follows an explanation of each of these tasks.

Request Propagation

Request propagation arises when a user asks the Remote Server for information, which the server cannot provide, sending it to the House Server, through the WebSocket channel.

This situation occurs when users access the *SmartDevice Menu*, requesting the Remote Server for access to a house's smart devices. As the Remote Server cannot provide the values sensed by the devices, it queries the Home Server for this information, to which the second replies with the data kept in its cache and database.

The WebSocket connection established by the user to the Remote Server is used to detect how many web-browsers are connected to the *SmartDevice Menu*, which, among others, can be consulted in Appendix E. While there is at least one user connected to it, the Home Server will send the values sensed by the house's devices, to the Remote Server, which will then be forwarded to the user.

When the last users exits the *SmartDevice Menu*, the Remote Server notifies the Home Server of this event, thus canceling the data propagation.

Messages sent by the user to the Remote Server, with the objective of changing a house's actuator, are also performed in the *SmartDevice Menu*, therefore travelling from one server, to another, in order to reach the device's Smart Controller.

Synchronization Messages

As mentioned, *ServerNotifications* are also responsible for ensuring the system databases are kept up to date.

These requests are stored, via *ServerNotification* objects, in the servers' databases, until a response is obtained. Whenever the connection is reestablished after a disconnection, all unconfirmed messages are resent by each server, by the order in which they were created, thus ensuring the synchronization process, as messages are delivered at least once. If a response is received, confirming a successfully processed request, the *ServerNotification* is deleted from storage.

In addition to the request's data, these messages contain a timestamp, representing the moment in which the system was changed. This ensures an application can compare a received *ServerNotification*, with one received, thus choosing the most recent. In case the two time values are the same, the system gives privileged to changes performed in the Remote Server.

If, after sending one of these messages, the applications cannot automatically recover from a conflict, which may arise when, for example, two overlapping reservations are created, then the reply provides enough information to resolve this situation, allowing both objects to be changed or deleted, by privileged, using one application.

This last solution is not ideal, as a users correcting a conflict, may raise a collision with a third object. However, ensures the problem can always be rectified.

4.3 Summary

In this chapter, a prototype for the designed solution has been described, composed by three types of entities: a Smart Controller, a Home Server and a Cloud Server.

Two Smart Controllers have been created using a NodeMCU, each containing one smart device. The most important device for this system was a smart door, capable of controlling the access to a property, through an access code. This device's controller can connect to an API, in this case, the Home Server, allowing the access codes to be changed remotely. Additionally, the device can sense the state of a door, notifying this state to the house's server, so that the property can be monitored.

The Home Server was implemented using a Raspberry Pi, responsible for hosting an application that, depending on a user's permissions, provides them with the possibility to manage the system entities and to control a house's smart devices. Furthermore, the server can generate notifications, allowing the a person to monitor the property over time.

To access the system through the Internet, the Remote Server has implemented using a personal computer. This server operates similarly to the Local Server, using that entity as an intermediary, to communicate with the property's devices.

Figure 4.4 provides a simplified process of the entities communication steps, with the implemented smart door, the two servers and a user connected to the system, through the Internet. It exemplifies a situation in which the door bell was rang, at instant T_0 , as it allows to show a request being sent from the

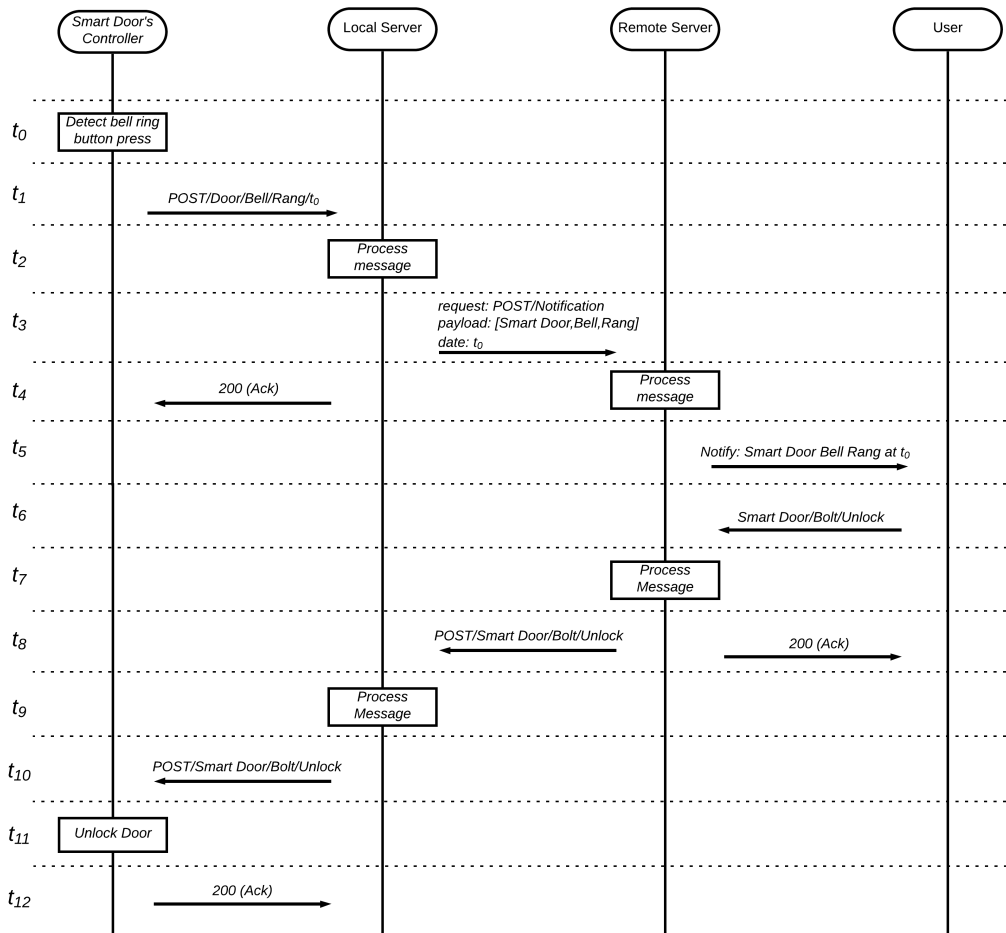


Figure 4.4: Door bell rang message flow.

Smart Controller to the servers, alerting the user. Additionally, this figure also shows a message being sent from the user, at instant T_6 back to the controller, representing the feature of remotely unlocking the door.

The developed entities ensure that a user can manage client reservations and control smart devices while constantly monitoring the property, thereby providing an IoT solution to assist in the management of rental houses.

5

Evaluation and Metrics

Contents

5.1 Automatic Lock Code Change	65
5.2 System Monitoring	65
5.3 Database Size	66
5.4 GUI Testing	67
5.5 Cache Processing Time	69
5.6 Summary	70

System testing is a very important task that can be used to evaluate the quality of a project, by analysing its features and its limitations on a real-world scenario. The five categories which have been used to evaluate the described system are depicted in this chapter.

5.1 Automatic Lock Code Change

It is desired that the system smart door authorizes a client to enter the property during a limited time period, the client's reservation.

To access this functionality, a *DeviceType*, representing the prototyped smart door, has been added to the system, containing two *CodeProperties* (one master key, with the value "0000", and another associated to the house's current reservation).

Additionally, a *SmartController* and *SmartDevice* instances have been added to the system, representing the property's main door (controller and device), that enables a tenant's access to the house.

Lastly, a *Reservation* object has been introduced in the Local Server, using the access code "1234". This reservation was restricted to a short period, beginning at 11 hours and 30 minutes of 11/6/2021 and finishing 5 minutes after its start. Usually, this does not happen in the real world, but it prevents long waits, facilitating the test.

It was detected that, at 11h25 of reservation's day, only the master key could unlock the door, as the reservation's code would not switch the relay. Few seconds after 11h30, both codes have been introduced in the device's keypad, activating the relay. Once the reservation time has elapsed, the initial situation was again verified, in which the tenant's code could not unlock the door, unlike the master key, which triggered the relay.

This test, provided the conclusion that after creating a *Reservation* instance, the Home Server automatically added two *TimeEvents*, associated with a reservation start and end date, responsible for triggering a code changing *PropertyAction*, in the beginning and at the end of that reservation. Moreover, it provided the conclusion that notification messages were delivered to the Smart Controller, therefore allowing to conclude that the main task of this project has been achieved.

5.2 System Monitoring

The requirements used to design the proposed system take into consideration that the generation of notifications, to alert users when an event occurs in the house environment, has been identified as being important.

To assess verify if the property owner was alerted in these situations, the smart door created has been used to detect the event of a door being forced open. For this reason, a *DiscreteProperty* has been

Add property event	Add notification action
Device type: SmartDoor	Event: Warning value is equal to Door Forced Open
My property: Warning	<input checked="" type="checkbox"/> Notify admins
Condition: equal to	<input type="checkbox"/> Notify house managers
Value: Door Forced Open	<input type="checkbox"/> Notify house client
	<input checked="" type="checkbox"/> Store in memory

Figure 5.1: Forced door event and action definition

added to the previously introduced smart door. Additionally, a *PropertyEvent* was created to detect the value "Forced Open" for that property, and a *NotificationAction* was associated to the event, as depicted in Figure 5.1. This action would create a notification in the system databases and notify the *Admins*.

The operation of opening a door has been simulated, by separating the implemented smart door's reed switch and a magnet, without in introducing the access code. This device detected this change on the environment, thus creating a message to the Home Server. The server reacted to this request, by sending an email to the *Administrator* account used to test the system, as illustrated by Figure 5.2.

Two other formats of this alert are provided in Appendix F. One displayed in real time to users accessing the web application and the other containing a view of this notification stored in the server's database.

Dear user,

Property Door:

Warning State value set to Door Forced Open on 25-Oct-2021, 18:45:13

Best regards.

Figure 5.2: Emailed alert content.

This test led to the conclusion that the system satisfies the requirement of alerting the user when an event has been detected in the property, as the three implemented notification types have been received.

5.3 Database Size

This system databases are scalable. It is required to know how many instances can be added to the system's storage, to ensure a house's data can be stored using a Raspberry Pi's SD Card.

Objects containing text attributes are expected to be more memory consuming, than those whose

Reservation Management Tasks				
Instance Model	House	User	ManagerHouse Relation	Reservation
Object Size	168	339	16	170

Home Automation Tasks - Events					
Instance Model	PropertyEvent	Notification Action	Device Notification	DateTimeEvent	CodeProperty Action
Object Size	33	25	561	25	24

Communication Tasks						
Instance Model	Controller Notification	Object Mapping	Server Notification	Server Token	SmartController	Server
Object Size	546	33	1068	144	181	59

Home Automation Tasks - Devices							
Instance Model	DeviceType	Discrete Property	Continuous Property	CodeProperty	HouseCode Rule	Value	SmartDevice
Object Size	40	153	193	157	82	80	88

Table 5.1: Database object size, in bytes

fields are composed by numeric values, as it can be seen in Table 5.1, providing the memory size occupied by the system's object instances, in bytes. These values can also be obtained from the databases' entity relation schemes, assuming *datetime* objects are stored using the current epoch (number of seconds since 1970).

ServerNotification instances consume large amounts of memory, when compared to any other instance, as it may require large payloads to be stored (requests and responses for synchronization messages). As these objects, when kept in storage, are deleted upon successfully delivering its message, this storage can be reduced, unless the connection between the channels is lost.

Considering the worst case scenario in which all objects required *1Kb* of memory, an approximate value of the largest object's size, and *4Gb* available on an *8Gb* SD-card (the recommended minimum size for a Linux based operative system, on the Raspberry Pi ¹), then the server would be able to store around 4 million records.

Additionally, a Raspberry Pi is compatible with a high range of SD-card sizes, allowing for users to select the its storage, based on their needs.

5.4 GUI Testing

Graphical Interface testing is an important task when conducting the analysis of a web application, as it can be used to evaluate how people interact with the system.

A group of 10 people have been selected to assess the prototype's interface, in particular the *Cloud*

¹ubuntu, "How to install ubuntu server on your raspberry pi," available online at <https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview>, last access in October 2021.

Server's, as it allows testers do remotely access the web application. This group consisted of people ranging between the age of 22 and 68 years old, with three of them being involved in the home rental industry.

Table 5.3 contains the time taken to complete the 14 tasks, divided by their groups, for each tester. Additionally each group's average time, for the 10 users. Appendix G contains detailed information regarding these tests, including the tasks corresponding to each group, their description and the time taken, by the users, to perform each of these.

Task Set	Task Time										User Average
	Tester1	Tester2	Tester3	Tester4	Tester5	Tester6	Tester7	Tester8	Tester9	Tester10	
Website Access Tasks Time	00'51"	00'53"	01'02"	00'41"	00'50"	00'43"	01'05"	01'00"	00'42"	00'52"	00'52"
Reservation Management Tasks Time	04'19"	06'22"	04'19"	07'45"	07'08"	07'20"	09'19"	05'23"	05'02"	07'36"	06'27"
Home Automation Configuration Tasks Time	07'54"	12'50"	12'13"	09'05"	14'09"	12'59"	12'11"	08'56"	13'28"	08'48"	11'15"
Database Synchronization Task Time	02'25"	02'18"	02'13"	02'41"	01'59"	03'37"	01'58"	02'12"	02'17"	03'04"	02'28"
Home Automation Tasks Time	01'32"	02'44"	01'04"	01'06"	00'55"	00'55"	02'04"	00'51"	01'15"	01'04"	01'21"
Website Sign Out Task Time	00'19"	00'09"	00'11"	00'11"	00'12"	00'14"	00'12"	00'18"	00'12"	00'09"	00'13"
Total Time	17'20"	25'16"	21'02"	21'29"	25'13"	25'48"	26'49"	18'40"	22'56"	21'33"	22'37"

Figure 5.3: User test task total time, measured in minutes and seconds

Login-in and login-out tasks have been performed. Additionally, the edition of a *House*, creation of a *Reservation* and deletion of a *User* have been requested. With the exception of the second, these tasks have been performed in reduced amounts of time, as the users were accustomed to perform similar tasks in other websites, as demonstrated by the execution times.

The configuration of instances related to the home automation task were more demanding, as most users were not accustomed to the terminology used in the project.

Additionally, a database conflict scenario has been created, so that users managed to correct the problem. This was considered a very complex task, as it was required for users to understand the problem in order to fix it.

For two times the users were asked to change the state of the smart light. On the first, the average task time was 18 seconds, being reduced to 5 seconds afterwards. This depletion indicates users got familiarized with the action, which could demonstrate the task could be achieved in small amounts of time. It is, for this reason, possible to conclude that the interface used for a smart device's control can be performed without difficulty.

More than 10 minutes were required, on average, for these people to configure a device. Despite the fact that this was a very complex task, as it required users to insert several values, it should be noted that 6 minutes and 55 seconds was a large duration, as all values were provided. Additionally, most users commented that they could not understand the difference between a *SmartDevice* and a *DeviceType*, as some of the attributes for a *Property* object. Some users also thought it would be better for icons to be used when creating a *DeviceType*, facilitating the system comprehension and improving the speed with which the tasks could be performed, as images are more intuitive than plain text.

The main advantage of this test, is that it allowed to evaluate most characteristics implemented in the prototype. Moreover, having the users established a connection to the Cloud Server, and performing requests through it to the Raspberry Pi, allowed to test the database synchronization features. Lastly, as the testers managed to perform the tasks which required them to change a smart light's state, it can be concluded that requests could, additionally, be sent from the Remote Server to the ESP node, and *vice-versa*.

It should be noted, however, that the number of people involved in these tests was reduced and that it does not contain a sufficient percentage of people involved in the home rental industry, in particular for short term accommodation, mainly due to restrictions imposed by the portuguese government, due to SARS-CoV-2. For this reason, the tests should be conducted with a larger sample of individuals, especially with people associated with this type of business.

5.5 Cache Processing Time

The cache used to store device's sensed data is an important feature of this system, as most SD-cards have reduced read and write speeds. For this reason, its evaluation is of severe importance.

Using the implemented smart light, 500 messages have been sent to the Raspberry Pi. These messages simulated a change in the light's state, alternating the sensed value between *On* and *Off*, forcing a change in the server, thus allowing to test the cache introduced in the system.

Two scenarios have been performed to test the server's message processing time, in seconds. In the first, a single *SmartDevice*, connected to a microcontroller, was introduced in the application and, in the second, 100 *SmartControllers*, each containing a device, have been simulated to send a total of 500 messages.

Three tests have been conducted for each scenario. In one, entitled "Cache Read & Write", this structure was accessed to perform test if the message was valid (read) and to update the sensed value (write). Another test, "Database Read & Cache Write", used the database to verify the validity of the received message, and the sensed valued was stored in the cache. Finally, "Database Read & Write" was a test in which the cache was not used, thus representing the experiment baseline. Table 5.2 contains,

N Devices	Cache Read & Write	Database Read & Cache Write	Database Read & Write
1	1,32	4,32	9,11
100	1,52	4,29	9,18

Table 5.2: Message processing time comparison, in seconds.

for the three tests conducted in the two scenarios, the total processing time of the 500 messages.

By analysing these times it is possible to perceive that, independently of the number of smart devices, the server would always process their messages in similar times. However, it is clear that the use of a hashing data structure to prevent database accesses significantly reduced the message processing time, due to its $O(1)$ amortized time complexity, when compared to both tests in which the database was accessed, affected by the SD-card's read and write speeds and context switch required to access the storage unit.

5.6 Summary

To evaluate this project, five tests have been conducted.

One tested the system's capacity to provide users, with different sets of permissions, access to the house, ensuring a person managing the house could entry the property at all times, while a client has the same possibility for a restricted period of time.

The notification messaging characteristic of the system has been successfully tested, ensuring a user can monitor the house environment, even when disconnected from the web application.

A third test analysed the worst case scenario for the prototyped system's object size and the importance of its use in a computer with restricted storage.

Additionally, the GUI has been tested, asking for several users to perform a set of tasks, ensuring the proper functionality of the interface.

Lastly, the cache structure developed was evaluated, by comparing the Local Server's processing time, for messages emitted by Smart Controllers.

6

Conclusion

Contents

6.1 Conclusion	73
6.2 Future Work	74

Upon reaching the last chapter of this Dissertation, some final remarks are presented in the Conclusion, followed by a section containing future improvements for the developed system.

6.1 Conclusion

The administration of short-term accommodation necessitates a substantial time and effort commitment on the part of a person wishing to rent a property, requiring the management of client reservations, as well as other administrative tasks.

To surpass this difficulty, a home automation system has been proposed to assist a person on managing a house, by reducing some of their work, independently of that person being the property's owner, or a person hired to execute the task.

The decision of creating a domotic system, allows smart devices to sense the environment and actuate on it. These devices use the house's LAN to establish a Wi-Fi connection to a server instance, running inside the same property's network and isolating it from the Internet. The communication channel between devices and server is provided using WebSockets, allowing a both entities to transfer data at the same time.

Using a web application, authorized users may access the smart devices' sensed data through the house's central processing unit. Similarly, they may also request the server for messages to be sent to the devices, altering an environment property. In both situations, available information and actions may be restricted to users, depending on their permissions. An advantage when compared to currently provided solutions.

The system data is stored on a database so that data can be persisted in case of power loss. However, as smart devices may send messages to the server at high rates, the system keeps the data sensed in the server's process memory, allowing messages to be acquired and processed within shorts periods of time.

Additionally, the system provides the possibility to add a cloud hosted server, which communicates, through WebSockets, with the property's server, providing the administrator real-time remote access to the house's entities. Moreover, the system allows for many house servers to be connect to the cloud hosted instance, ensuring a user can manage several properties, from afar.

A prototype has been developed, in which two smart devices have been created using a NodeMCU, one controlling a person's access through the house's entrance door and another for controlling a light bulb's state. These devices communicated with a server hosted on a Raspberry Pi, collecting data from them and providing it to a person, connected to the system, through the Internet. Furthermore, this last server was developed so that users could send commands through the Raspberry Pi to the ESP nodes, in order to change an environment's property.

Upon conducting a set of tests that demonstrated the system can be configured using both servers, allowing users to remotely control the house appliances. Additionally, despite having a non-intuitive interface, these tests shown the system can rapidly process messages sent by the smart nodes and that it can be integrated on most properties, as the Raspberry Pi with a Wi-Fi connection provides a reduced cost solution, capable of hosting the system's central unit.

Above all, the test in which the NodeMCU accepted the access code associated with a reservation, during that reservation's time, and revoking the code afterwards, assessed that this system's main functionality has been correctly implemented.

In sum, it is possible to conclude that this extensible system, satisfies the identified requirements, thus providing a significant assistance in the management of short-term accommodation businesses.

6.2 Future Work

The project developed should be improved so that the system can fulfill the requirements that were accomplished and the demands of house rental management systems.

First and foremost, as identified by evaluating this system web application, the GUI should be improved so that users can configure the home automation devices with ease. Additionally, presented below are several features which can be added to this system to better assist people renting a house.

The system does not yet allow for one house to accommodate different clients at the same time. A feature enabling this solution should be implemented in future versions of this system, as one house may have several rooms, allowing each room to be rented to a different client.

Future versions of this system should also contemplate the detection of house events, using a logical function, to automatically perform a change in the house's environment. This is, allow a complex condition to be detected, such as, for example, turning on an air conditioner one hour prior to a client's arrival at the property.

Improve the alert functionalities of the system. The system should also send notifications to the house managers, whenever an entity is disconnected from the system, allowing the person to correct the problem, if necessary. Additionally, to ensure alerts are received, by users, in real-time, a SMS should be implemented, as a complement for the notifications provided by electronic mail.

Two smart devices have been created in this project. However, there is a variety of these objects available in the market, that could be useful in the context of this project. Smart thermostats, utility meters, movement and noise sensors, smoke and carbon monoxide detectors are a few examples of devices that, when integrated in a house's environment, could provide additional comfort to both owners, tenants and even to neighbours. The incorporation of these types of devices should be contemplated in the future. Moreover, proxies and gateways should be added to the system, ensuring that different

smart devices can be integrated in a house, improving the system's extensibility.

Finally, the system should be able to detect unusual occurrences in the house's environment, such as a door being constantly open, allowing the users to be informed of these events. The use of machine learning techniques should be applied, taking advantage of the large amounts of data the system generates, allowing it to better aid a person in the management tasks.

Bibliography

- [1] P. Xiao, *Introduction to Arm® Mbed™*. John Wiley & Sons, Ltd, 2018, ch. 1, pp. 1–22, available online at: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119364009.ch1>, last access in September 2021.
- [2] M. Silverio-Fernández, S. Renukappa, and S. Suresh, “What is a smart device? - a conceptualisation within the paradigm of the internet of things,” *Visualization in Engineering*, vol. 6, no. 1, p. 3, May 2018, available online at: <https://doi.org/10.1186/s40327-018-0063-8>, last access in September 2021.
- [3] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A Survey,” *Comput. Netw.*, vol. 54, no. 15, p. 2787–2805, Oct. 2010, available online at: <https://doi.org/10.1016/j.comnet.2010.05.010>, last access in September 2021.
- [4] C. Gomez, J. Oller Bosch, and J. Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” *Sensors (Basel, Switzerland)*, vol. 12, no. 1, pp. 11 734–53, dec 2012.
- [5] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, “Internet of Things (IoT) communication protocols: Review,” in *2017 8th International Conference on Information Technology (ICIT)*, 2017, pp. 685–690.
- [6] H. Nielsen, R. T. Fielding, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.0,” RFC 1945, May 1996, available online at: <https://rfc-editor.org/rfc/rfc1945.txt>, last access in September 2021.
- [7] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” RFC 7252, Jun. 2014, available online at: <https://rfc-editor.org/rfc/rfc7252.txt>, last access in September 2021.
- [8] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks,” in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, 2008, pp. 791–798.

- [9] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6.
- [10] A. Melnikov and I. Fette, "The WebSocket Protocol," RFC 6455, Dec. 2011, available online at: <https://rfc-editor.org/rfc/rfc6455.txt>, last access in September 2021.
- [11] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008, available online at: <https://rfc-editor.org/rfc/rfc5246.txt>, last access in September 2021.
- [12] V. Sarafov and J. Seeger, "Comparison of IoT Data Protocol Overhead," in *Proceedings of the Seminars of Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM)*, 2018, pp. 7–14.
- [13] Y. T. Park, P. Sthapit, and J.-Y. Pyun, "Smart digital door lock for the home automation," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, 2009, pp. 1–6.
- [14] M. Pavelić, Z. Lončarić, M. Vuković, and M. Kušek, "Internet of Things Cyber Security: Smart Door Lock System," in *2018 International Conference on Smart Systems and Technologies (SST)*, 2018, pp. 227–232.
- [15] K. Luechaphonthara and A. Vijayalakshmi, "IoT based application for monitoring electricity power consumption in home appliances," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, p. 4988, 12 2019.
- [16] Z. H. Che Soh, M. S. Shafie, M. A. Shafie, S. Noraini Sulaiman, M. N. Ibrahim, and S. Afzal Che Abdullah, "IoT Water Consumption Monitoring Alert System," in *2018 International Conference on Electrical Engineering and Informatics (ICELTICs)*, 2018, pp. 168–172.
- [17] P. Saint-Andre, S. Loreto, S. Salsano, and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP," RFC 6202, Apr. 2011, available online at: <https://rfc-editor.org/rfc/rfc6202.txt>, last access in September 2021.
- [18] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 336–341.
- [19] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 8259, Dec. 2017, available online at: <https://rfc-editor.org/rfc/rfc8259.txt>, last access in September 2021.
- [20] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: A case study," *22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009*, pp. 157–162, 01 2009.



Access Point Configuration Templates

The implemented smart door's Access Point allows a user to configure this device's microcontroller.

Through this mode it is possible to connect the controller to the Wireless Local Area Network (a), by choosing one of the scanned networks and introducing the corresponding password, and to define the Application Programming Interface's network address (b), so that the device can connect to it.

Additionally, this configuration interface allows a user to change the access codes responsible for unlocking the door (c) and to define which set of messages are sent to the remote entity (d).

The templates for these four functionalities mentioned above are shown in this appendix, each identified by the mentioned letters (a,b,c and d).

Project: IoT Solution for Rental Houses
SmartDoor

Remote active time: 169s

By disabling WiFi, all wireless connections will be lost including access from a central unit. This can only be rectified by resetting the device.

Caution: this will reset the device remote access

Disable WiFi

wlan password

Submit

Cancel

a

Project: IoT Solution for Rental Houses
SmartDoor

Remote active time: 241s

Interface IP address

Submit

Cancel

b

Project: IoT Solution for Rental Houses
SmartDoor

Remote active time: 242s

Primary key

Submit

Cancel

c

Project: IoT Solution for Rental Houses
SmartDoor

IP: 192.168.1.85
Remote active time: 233s

- Send door state
- Send door forced open
- Send door left open
- Ring Bell
- Control door remotely
- Send limit of attempts surpassed
- Authentication code limit surpassed

Submit

Cancel

d

B

Database Models Description

This Appendix contains the system models, ordered alphabetically, and their description. Additionally, it provides each of these model's attributes, referring if they are required, or not, providing details regarding their uniqueness and a description of each one of them.

Model Name	Model Description	Attribute	Required	Unique	Attribute Description
CodeProperty	Represents a Smart Device's access code property	id	TRUE	TRUE	Identifies the object in the database
		device_type_id	TRUE	FALSE	Property's DeviceType identifier
		name	TRUE	For DeviceType	Identifies the property for a user
		name_in_controller	TRUE	For DeviceType	Identifies the property for a Smart Controller
		description	TRUE	TRUE	Description of the property
		user_permission	TRUE	TRUE	Defines the access users have to the property
		characters	FALSE	FALSE	Set of possible characters
		min_size	FALSE	FALSE	Code minimum size
		max_size	FALSE	FALSE	Code maximum size
Code Property Action	Changes the value of Reservation's Code Properties to the Reservation's access code	code_type	FALSE	FALSE	Defines if the code is belongs to Reservation, or it can created by a user
		id	TRUE	TRUE	Identifies the object in the database
Continuous Property	Represents a Smart Device's numeric property	event_id	TRUE	TRUE	Identifies the event that triggeres this action
		id	TRUE	TRUE	Identifies the object in the database
		device_type_id	TRUE	FALSE	Property's DeviceType identifier
		name	TRUE	For DeviceType	Identifies the property for a user
		name_in_controller	TRUE	For DeviceType	Identifies the property for a Smart Controller
		description	TRUE	FALSE	Description of the property
		user_permission	TRUE	FALSE	Defines the access users have to the property
		min_value	TRUE	FALSE	Numeric property minimum value
		max_value	TRUE	FALSE	Numeric property maximum value
Controller Notification	Represents the communication unit between Home Server and Smart Controllers	step	TRUE	FALSE	Numeric property incrementation step
		unit	FALSE	FALSE	Numeric property unit of measurement
		id	TRUE	TRUE	Identifies the object in the database
		date	TRUE	FALSE	Notification's date and time
		message	TRUE	FALSE	Notification's data transferred
		status	TRUE	FALSE	Specifies if a message has been sent
DateTimeEvent	Detects a change a moment in time	method	TRUE	FALSE	Specifies if a message is used to POST or DELETE data
		controller_id	TRUE	FALSE	Notification's controller identifier
		id	TRUE	TRUE	Identifies the object in the database
		condition	TRUE	FALSE	Defines the event condition. Can be lower than, lower or equal to, equal to, greater or equal to, greater than a value, or detected regardless of it
		value	TRUE	FALSE	Defines the date and time to be detected
Device Notification	Saves a change occurred in a house's enviroment	date	TRUE	FALSE	Notification's date and time
		message	TRUE	FALSE	Value sensed by the smart device or set by a user
		status	TRUE	FALSE	Represents if an alert has been sent to the users
		device_id	TRUE	FALSE	Identifies the device that sensed the environment or which a user changed
		user_id	FALSE	FALSE	Identifies the user that perform the change
		action_id	TRUE	FALSE	Identifies the action that triggered this notification
DeviceType	Represents a set of identical Smart Devices	id	TRUE	TRUE	Identifies the object in the database
		name	TRUE	TRUE	Identifies the device type for a user

Table B.1: System models I.

Model Name	Model Description	Attribute	Required	Unique	Attribute Description
Discrete Property	Represents a Smart Device's categorical property	id	TRUE	TRUE	Identifies the object in the database
		device_type_id	TRUE	TRUE	Property's DeviceType identifier
		name	TRUE	For DeviceType	Identifies the property for a user
		name_in_controller	TRUE	For DeviceType	Identifies the property for a Smart Controller
		description	TRUE	TRUE	Description of the property
		user_permission	TRUE	TRUE	Defines the access users have to the property
House	Represents an owner's house	id	TRUE	TRUE	Identifies the object in the database
		name	TRUE	TRUE	User identifier
		city	FALSE	FALSE	House's city
		country	FALSE	FALSE	House's country
		address	FALSE	FALSE	House's street address
HouseCodeRule	Defines the rules for generating a random access code	id	TRUE	TRUE	Identifies the object in the database
		house_id	TRUE	TRUE	House identifier
		characters	TRUE	FALSE	Set of possible characters
		min_size	TRUE	FALSE	Code minimum size
		max_size	TRUE	FALSE	Code maximum size
ManagerHouse Relation	Identifies the houses each manager can access	user_id	TRUE	For House	User identifier
		house_id	TRUE	FALSE	House identifier
Notification Action	Generates a notification when an PropertyEvent occurs	id	TRUE	TRUE	Identifies the object in the database
		event_id	TRUE	TRUE	Identifies the event that triggers this action
		notification_type	TRUE	FALSE	Defines if a the notification is sent to Admins, Managers, Clients, or simply saved in the database
ObjectMapping	Maps an object from the Remote Server to a House Server	id	TRUE	TRUE	Identifies the object in the database
		server_id	TRUE	FALSE	Server identifier
		local_id	TRUE	FALSE	Object's id in the House Server
		remote_id	TRUE	FALSE	Object's id in the Remote Server
		model_index	TRUE	FALSE	Object's model
PropertyEvent	Detects a change in a Smart Device's property	id	TRUE	TRUE	Identifies the object in the database
		condition	TRUE	FALSE	Defines the event condition. Can be lower than, lower or equal to, equal to, greater or equal to, greater than a value, or detected regardless of it
		property_id	TRUE	FALSE	Identifies the property that changed
		value	TRUE	For Property	Defines the value to be detected
Reservation	Represents a client's reservation or staff's shift	id	TRUE	TRUE	Identifies the object in the database
		house_id	TRUE	FALSE	House identifier
		user_id	TRUE	FALSE	Reservation's user identifier
		code	FALSE	FALSE	Encrypted Reservation access code
		start_date	TRUE	For House	Reservation's start date and time
		end_date	TRUE	For House	Reservation's end date and time
		canceled	FALSE	FALSE	Defines if a reservation has been canceled
		reservation_type	TRUE	FALSE	Defines if it is a client's reservation or a shift
Server	Represents a System's Server	id	TRUE	TRUE	Identifies the object in the database
		house_id	FALSE	TRUE	House identifier
		ip_address	TRUE	For House	Server's IP address
		active	FALSE	FALSE	Indicates the server's state
		nAttempts	FALSE	FALSE	Controls the number of login attempts
		block_expiration_date	FALSE	FALSE	Defines the date and time until which the server is blocked

Table B.2: System models II.

Model Name	Model Description	Attribute	Required	Unique	Attribute Description
Server Notification	Represents the communication unit between servers	id	TRUE	TRUE	Identifies the object in the database
		date	TRUE	FALSE	Notification's date and time
		message	TRUE	FALSE	Notification's data transferred
		status	TRUE	FALSE	Specifies if a message needs to be sent or has been rejected by the receiver
		method	TRUE	FALSE	Specifies if a message is used to GET, POST or DELETE data
		server_id	TRUE	FALSE	Notification's server identifier
		is_create	FALSE	FALSE	Defines if a POST message is creating an object
		model_index	TRUE	FALSE	Object's model
		obj_id	TRUE	FALSE	Object's id in the server
ServerToken	Represents a server's access token	id	TRUE	TRUE	Identifies the object in the database
		server_id	FALSE	TRUE	Token's server identifier
		token	TRUE	FALSE	Encrypted server token
SmartController	Represents a house's Smart Controller	id	TRUE	TRUE	Identifies the object in the database
		house_id	TRUE	TRUE	House identifier
		name	TRUE	For House	Identifies the controller for a user
		token	TRUE	FALSE	Encrypted Smart Controller's authentication token
		ip_address	TRUE	For House	Server's IP address
SmartDevice	Represents a house's Smart Device	active	FALSE	FALSE	Indicates the server's state
		id	TRUE	TRUE	Identifies the object in the database
		controller_id	TRUE	FALSE	SmartController identifier
		device_type_id	TRUE	FALSE	DeviceType identifier
		name	TRUE	For House	Identifies the device for a user
User	Represents a system user	name_in_controller	TRUE	For Smart Controller	Identifies the device for the Smart Controller
		id	TRUE	TRUE	Identifies the object in the database
		username	TRUE	TRUE	User identifier
		password	TRUE	TRUE	Encrypted user's authentication password
		email	FALSE	TRUE	User's email address. Utilized for alerts
		phone	FALSE	TRUE	User's telephone number. Utilized for alerts
		user_type	TRUE	FALSE	Defines the user permissions
		has_remote_access	FALSE	FALSE	Defines if a user can access the system remotely
blocked	FALSE	FALSE	Defines if a user can access the system		
nAttempts	FALSE	FALSE	Controls the number of login attempts		
block_expiration_date	FALSE	FALSE	Defines the date and time until which the user is blocked		
Value	Represents a categorical value of a Discrete Property	id	TRUE	TRUE	Identifies the object in the database
		property_id	TRUE	FALSE	Identifies the DiscreteProperty
		name	TRUE	For Discrete Property	Identifies the value for a user
		name_in_controller	TRUE	For Discrete Property	Identifies the value for a Smart Controller

Table B.3: System models III.



Servers' Database Models Overlapped

This appendix contains the both servers' databases Model Relationship represented in black. Additionally, models that are only related to the Home Server are depicted in light blue, while in dark blue are does related to the Remote Sever.

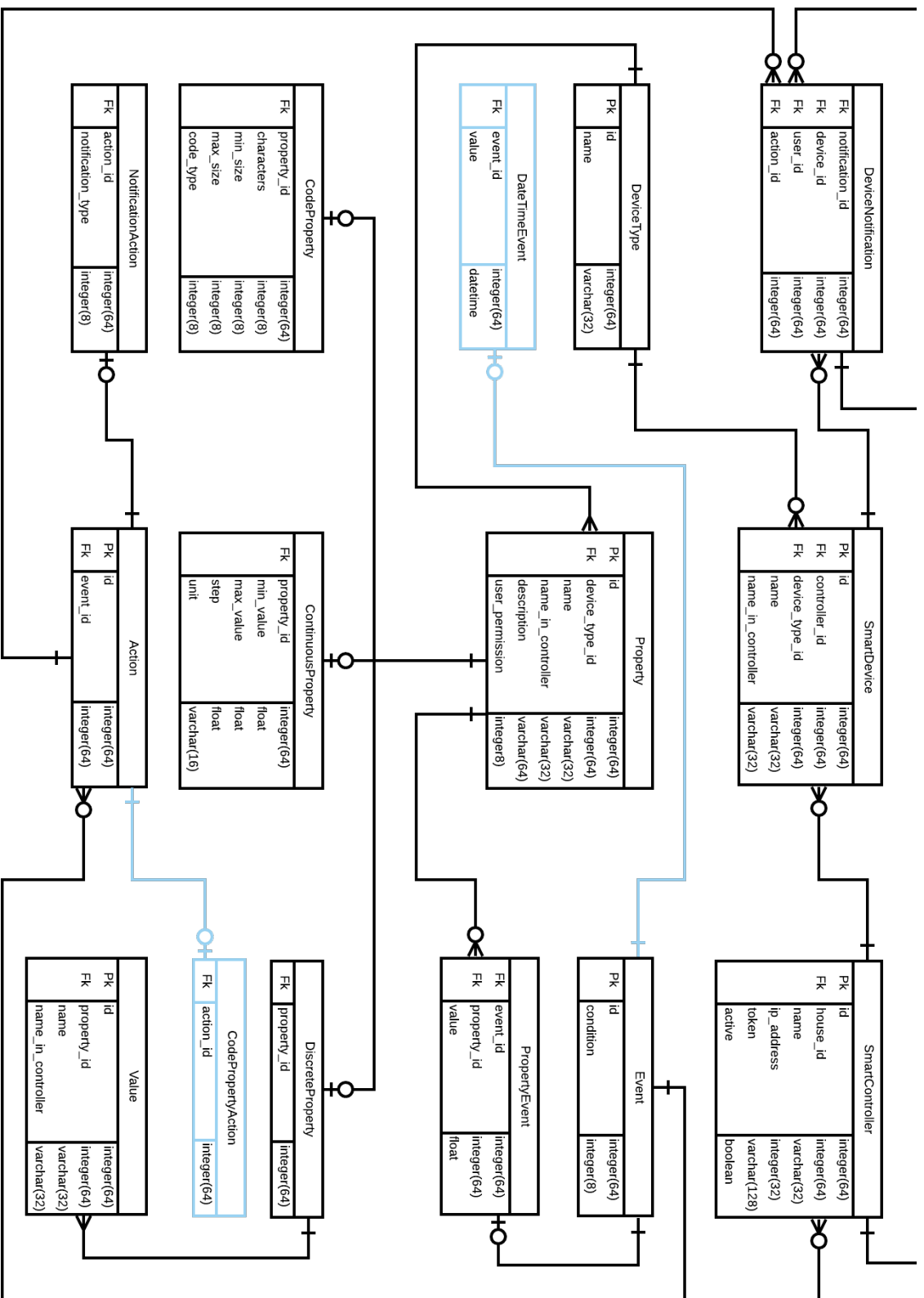
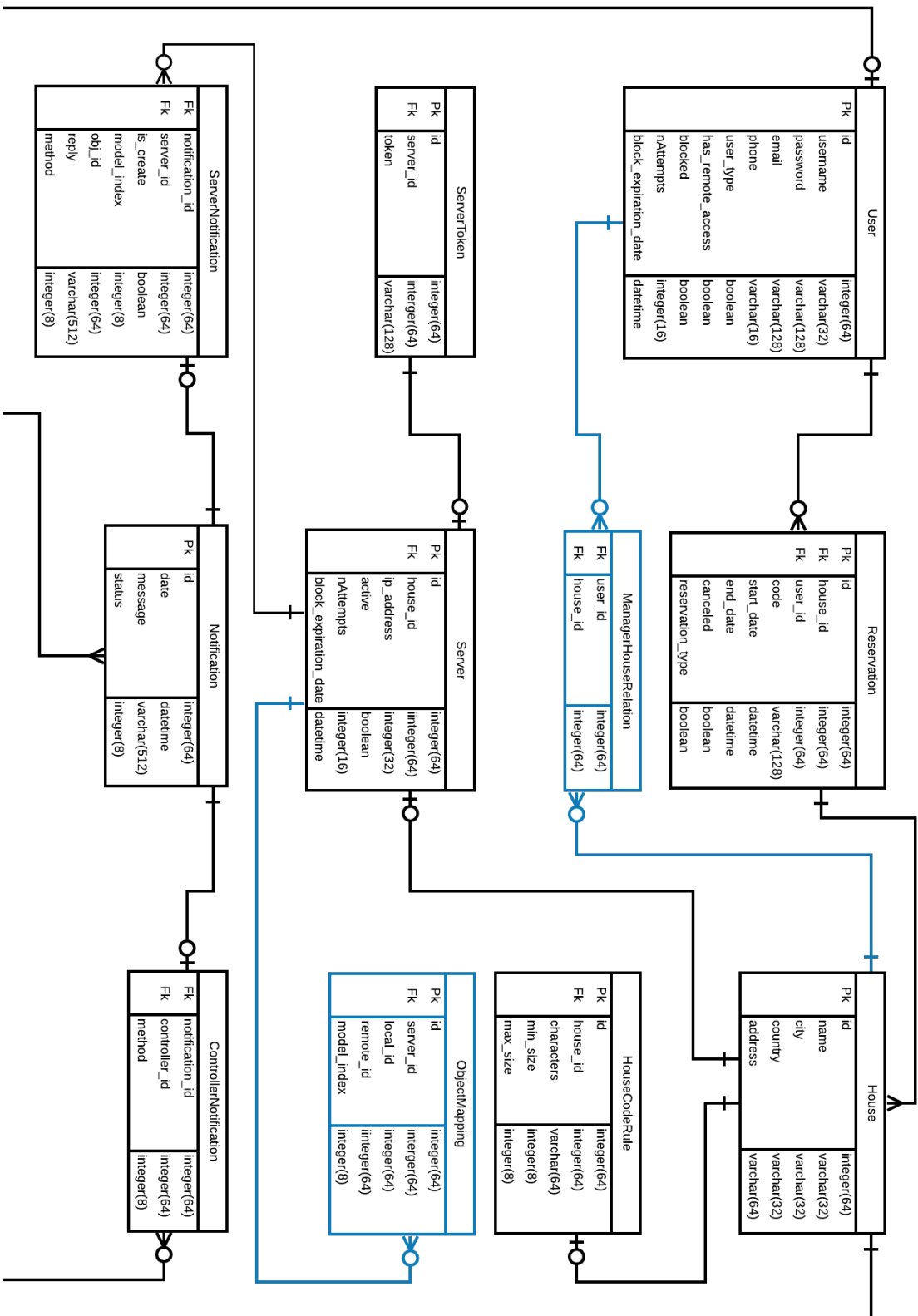


Figure C.1: Local and Remote Servers' database Model Relationship.





Servers' Website Navigation Scheme

This appendix provides the web-site navigation scheme provided by both web applications. On the top, the Home Server diagram can be observed, while the bottom figure contains the same diagram for the Cloud Server, with the only difference between the two being the *House Menu*. When accessing a house's smart devices, this menu allows a user to define the house to which they belong to.

Additionally, the two diagrams contain a subsection called the *Admin Web-Site*, representing the set of web pages which can only be accessed by a privileged user, as it allows the management of the entire system.

From *Main Admin*, of the restricted set of web pages, the user may decide which system object to be viewed, created, edited, or deleted, by accessing the group of pages corresponding to that object. Each of these groups is defined using the *Model* name,

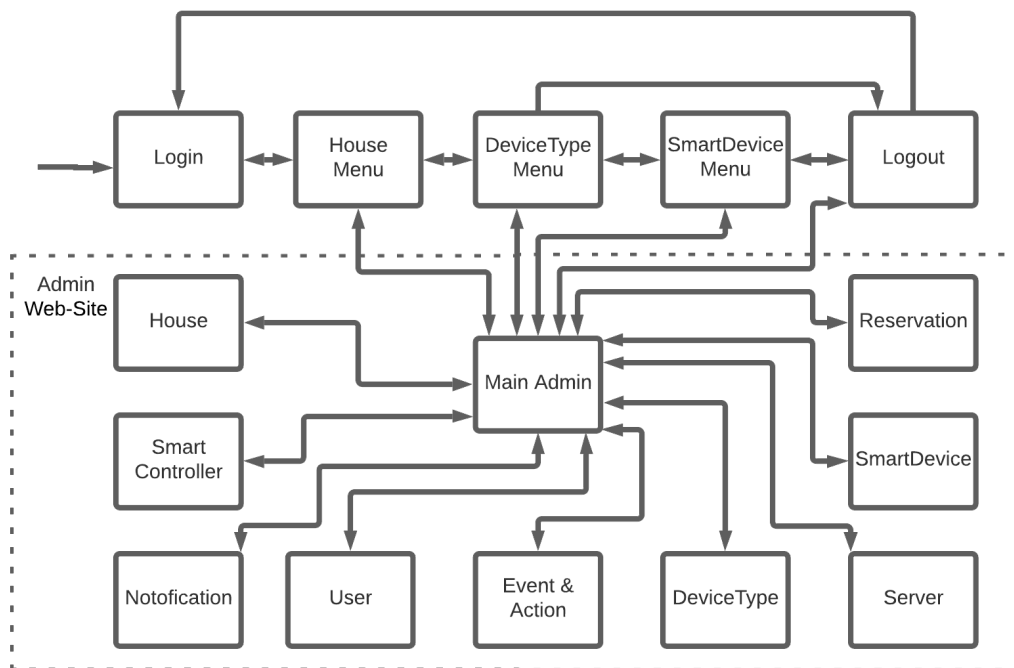
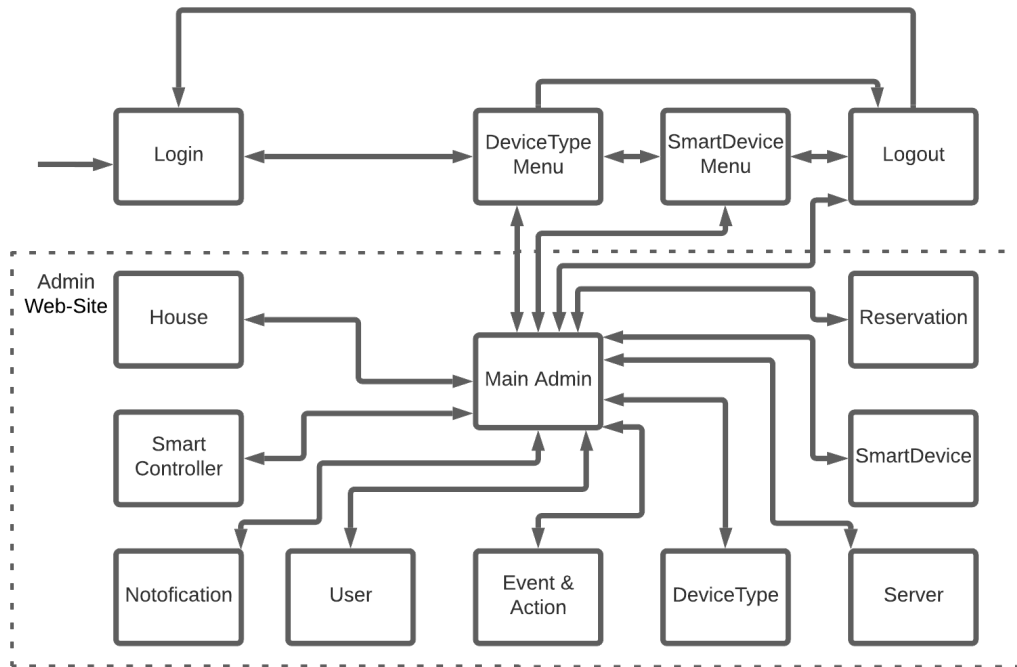


Figure D.1: Website navigation scheme for Home Server (on top) and Cloud Server (on bottom).



Web Page Views

This appendix provides three of the application web pages. Figure E.1 illustrates the *SmartDevice Menu* through which users can monitor and control the devices, in particular, smart lights. Additionally, a most administrative web pages are identical, two views accessible through the *Admin Web Site* are provided. The first, Figure E.2, provides an illustration of a web page listing all objects related to a model, in this case, *SmartDevices*, while Figure E.3 contains the web page used for users to create a *Reservation* object. In this case, a reservation for client *John Doe*, with the access code "1234" between the first and the second of January of 2022.

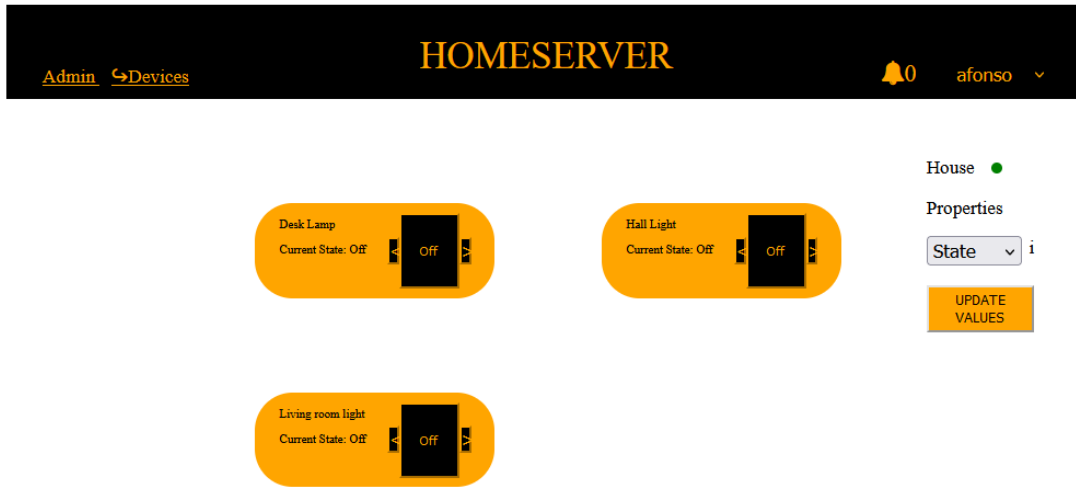


Figure E.1: SmartDevice Menu web page view.

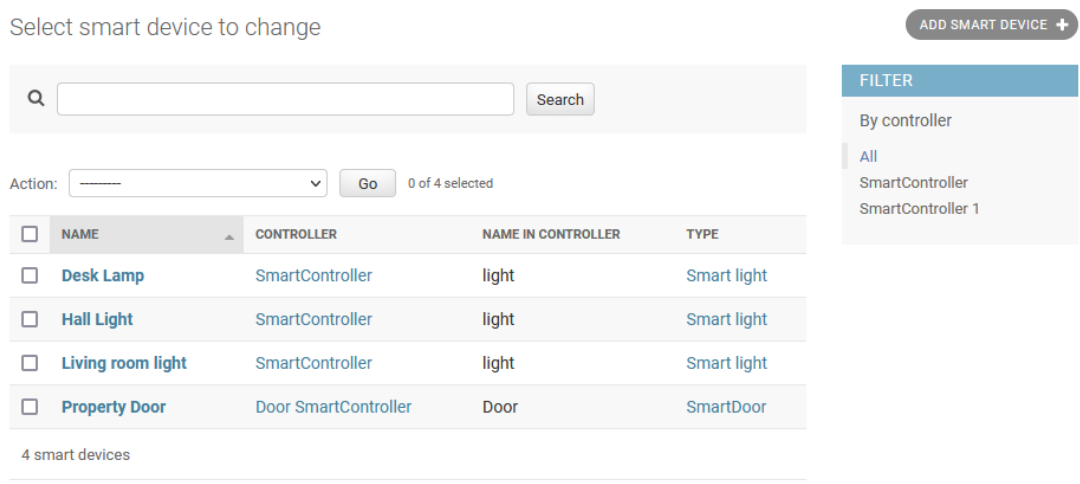







Figure E.2: Admin Website - SmartDevice list web page view.

Add Reservation

Info

Client:   

House:  



Access

Reservation code:

Generate random code

Canceled

Dates

Start date (UTC): Date: Today | 
Time: Now | 



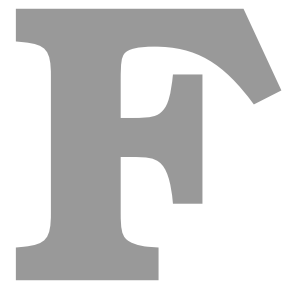
End date (UTC): Date: Today | 
Time: Now | 

Figure E.3: Admin website - Reservation creation web page view.



Smart Door Alerts

This Appendix provides two figures representing the alerts sent to the user, by the system.

Figure F.1 displays the application *DeviceType Menu*, in which an alert with the message "Property Door: Warning State value set to Door Forced Open [25-Oct-2021, 18:45:13]" has been sent to the user, using the WebSocket connection. Similarly, the Figure F.2 represents the same alert, stored in the Home Server's database, allowing the notification to be kept in the system until deleted by the user.

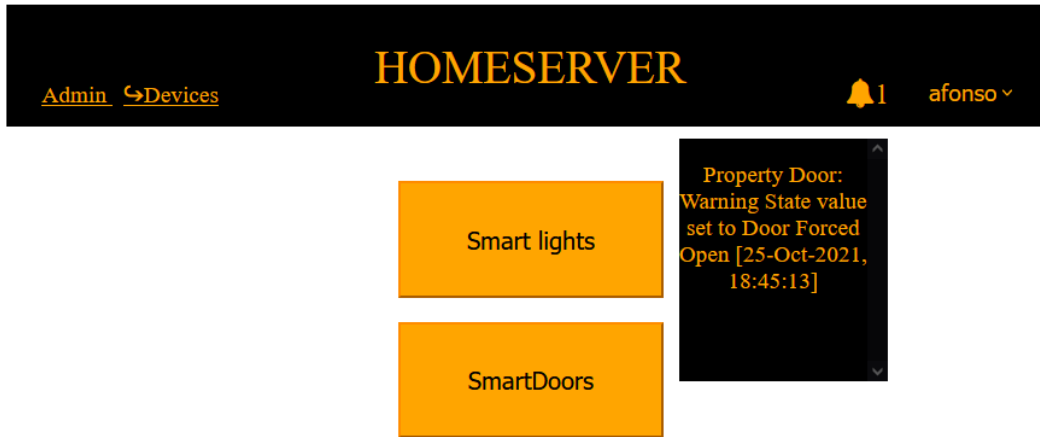


Figure F.1: Forced door alert in *Device Type Menu*.

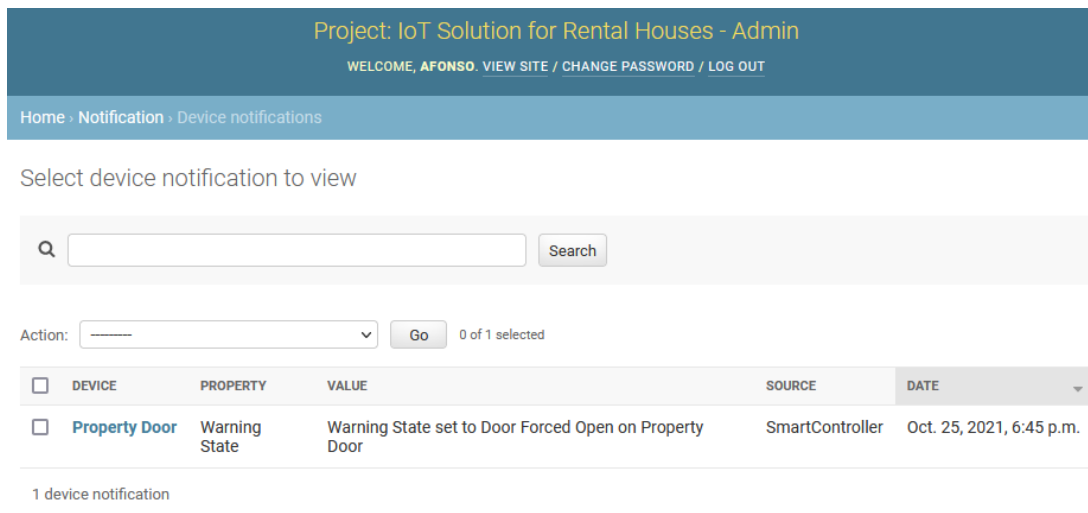


Figure F.2: Forced door alert stored notification.



User Testing Tasks and Times

This appendix provides the list of 14 tasks given to the 10 users, with the objective of evaluating the website's GUI.

Figure G.1 lists each of these tasks sorted by the order with which they were performed. It contains a column with the task name, its description, including the values required for the testers to insert in the system in order to complete the corresponding task. and the time taken by the testers to complete each tasks. Additionally, the column entitled "Extra Information" contains details given to the testers, to assist them in understanding what they were asked to do. It should be noted that the value for the first task is masked, as it included the computer's external IP address, which should be kept private, for security reasons.

#	Name	Task	Values	Description	Task Time for Each Tester														
					Tester1	Tester2	Tester3	Tester4	Tester5	Tester6	Tester7	Tester8	Tester9	Tester10					
Website Access Tasks																			
1	Navigate to server url	xxx.xxx.xxx.113	Access the website	0011"	0029"	0036"	0013"	0019"	0015"	0015"	0021"	0013"	0015"	0021"	0013"	0015"	0015"	0023"	
2	Login into the website	username: "manager" and password: "Manager_1"	Website authorization	0023"	0017"	0023"	0021"	0009"	0014"	0033"	0018"	0015"	0023"	0018"	0015"	0023"	0015"	0023"	
3	Access Admin Website	---	Website used to configure the system	0017"	0007"	0003"	0007"	0022"	0014"	0017"	0021"	0014"	0017"	0021"	0014"	0017"	0021"	0014"	
Reservation Management Tasks																			
4	Edit the system's only House	change city from "Lisboa" to "Lisbon"	Represents your property	0034"	0026"	0037"	0121"	0211"	0101"	0033"	0131"	0037"	0001:43	0131"	0037"	0131"	0037"	0001:43	
5	Delete User	Username: "Jane Doe"	Represents a user	0034"	0044"	0100"	0120"	0055"	0058"	0113"	0123"	0044"	0051"	0123"	0044"	0051"	0051"	0051"	
6	Create ClientReservation	"John Doe" in the property named "House", between 30/12/2021 and 2/1/2022 with noon as check-in and check-out time. Generate the code randomly	Represents the client reservation during a period of time	0311"	0512"	0242"	0504"	0402"	0521"	0733"	0229"	0341"	0502"	0229"	0341"	0502"	0341"	0502"	
Home Automation Configuration Tasks																			
7	Create a SmartController	Associate it to the house you have edittd. IP address: "192.168.1.100" Name "SmartController.1" token: "12345678"	Represents a controller handling several smart devices	0121"	0227"	0132"	0242"	0331"	0311"	0231"	0301"	0311"	0144"	0301"	0311"	0144"	0311"	0144"	
8	Add a DeviceType	"DiscreteProperty" named "State" and description: "controls the light state". It contains two "Values": "On" and "Off". All of these should have the same values for the fields "name" and "name in controller". This "Property" should be accessible by all types of users	Represents a house's light. Any light from the same seller, which can be controlled using this application	0519"	0901"	0842"	0452"	0721"	0809"	0725"	0418"	0858"	0501"	0725"	0418"	0858"	0501"	0501"	
9	Create a SmartDevice	Name: "desk lamp". Use the created device type and controller. Use "Light" for the attribute name in controller	Represents a smart device	0114"	0122"	0159"	0131"	0317"	0139"	0215"	0137"	0119"	0203"	0137"	0119"	0203"	0119"	0203"	
Database Synchronization Task																			
10	Fix SmartController data collision	Warning shown at the top of the screen. Force the home server data into the cloud server.	This happens because the server on the house and the remote server are out of sync	0225"	0218"	0213"	0241"	0159"	0337"	0158"	0158"	0212"	0212"	0212"	0217"	0212"	0217"	0304"	
Home Automation Tasks																			
11	Navigate to the main website	You are currently on Admin Website which is only used to configure the system	Website used to control the houses' devices	0106"	0218"	0046"	0039"	0029"	0037"	0131"	0037"	0049"	0040"	0131"	0037"	0049"	0049"	0040"	
12	Smart Device Control 1	Turn On the "desk lamp" light	Control the smart light	0020"	0019"	0015"	0022"	0020"	0013"	0026"	0009"	0019"	0019"	0026"	0009"	0021"	0019"	0019"	
13	Smart Device Control 2	Turn Off the "desk lamp" light	Control the smart light	0006"	0007"	0003"	0005"	0006"	0005"	0005"	0005"	0005"	0005"	0005"	0005"	0005"	0005"	0005"	
Website Sign Out Task																			
14	Logout	---	Sign off	0019"	0009"	0011"	0011"	0012"	0014"	0012"	0018"	0012"	0009"	0018"	0012"	0018"	0012"	0009"	

Table G.1 : Task description and time taken to complete each task, by the testers, in minutes and seconds

