A Cloud architecture to integrate a Multi-Agent Smart Parking system

Milton Boos Junior

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Information Systems within the scope of a double diploma with the Federal Technological University of Paraná.

Work oriented by:

Prof. PhD Paulo Jorge Pinto Leitão

Prof. PhD Diego Roberto Antunes

Prof. PhD Paulo Alexandre Vara Alves

Prof. PhD Gleifer Vaz Alves

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2020-2021

A Cloud architecture to integrate a Multi-Agent Smart Parking system

Milton Boos Junior

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Information Systems within the scope of a double diploma with the Federal Technological University of Paraná.

Work oriented by:

Prof. PhD Paulo Jorge Pinto Leitão

Prof. PhD Diego Roberto Antunes

Prof. PhD Paulo Alexandre Vara Alves

Prof. PhD Gleifer Vaz Alves

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2020-2021

Dedication

I dedicate this work to everyone who inspired me to always do my best, and to all those who helped me in some way to pursue my dream.

Acknowledgement

First of all, I thank my parents, for all the support, affection and understanding. I will be forever grateful.

To my supervisors at the Polytechnic Institute of Bragança (IPB), Professor PhD Paulo Leitão and Professor PhD Paulo Alves, for all the welcome, guidance and the opportunity to participate in a project that gave me a wonderful learning experience. It was an unforgettable, once in a lifetime experience.

In addition, to my supervisors at the Federal Technological University of Paraná (UTFPR), Professor PhD Gleifer Alves, for the opportunity and help throughout the Dual Diplomacy program, and to Professor PhD Diego Antunes, who encouraged me and was always willing to help.

Finally, I would like to thank my friends Lucas Mendes and Leonardo Guth, who were with me at all times and showed me the true meaning of friendship, and also Lucas Sakurada, who offered significant support in this work.

Abstract

Smart parking systems are becoming a solution to recurring issues as the number of vehicles in traffic rises in major cities, which can be related to vehicle traffic congestion, unnecessary time spent searching for parking spots, and, consequently, environmental issues. The purpose of these systems is to help drivers who are searching for available parking spaces or who want to reserve for a specified period of time, quickly and, if possible, near the desired location. In this context, there are several modules within smart parking systems that may include cyber-physical systems, multi-agent systems, dynamic pricing and artificial intelligence. This dissertation presents the development of a smart parking system architecture, using Cloud-based technology to integrate a multi-agent system into a scalable, decentralized, adaptable and safe environment. The proposed architecture was tested using, as a case study, a web system developed for the management and analysis of smart parking lots, as well as an application for mobile devices, which allows users to interact with multiple functionalities available in this system. Results obtained demonstrate that the implementation of the proposed architecture offers an efficient communication between users who use parking resources and the multi-agent system responsible for the autonomy and intelligence of the parking system.

Keywords: Cloud Computing; Multi-Agent System; Internet of Things; Smart parking system.

Resumo

Os sistemas inteligentes de estacionamento estão se tornando uma solução para problemas recorrentes à medida que aumenta o número de veículos em trânsito nas grandes cidades, os quais podem estar relacionados ao congestionamento no tráfego de veículos, tempo desnecessário gasto na busca por vagas e, consequentemente, questões ambientais. O objetivo desses sistemas é auxiliar os motoristas que buscam por vagas disponíveis ou que desejam reservar por determinado período de tempo, de forma rápida e, se possível, próximo ao local desejado. Neste contexto, existem vários módulos dentro dos sistemas inteligentes de estacionamento que podem incluir sistemas ciberfísicos, sistemas multi-agentes, precificação dinâmica e inteligência artificial. Esta dissertação apresenta o desenvolvimento de uma arquitetura para sistemas inteligentes de estacionamento, utilizando tecnologia baseada em Nuvem para integrar um sistema multi-agentes em um ambiente escalável, descentralizado, adaptável e seguro. A arquitetura proposta foi testada utilizando, como um caso de estudo, um sistema web desenvolvido para gestão e análise de estacionamentos inteligentes, bem como uma aplicação para dispositivos móveis, que permite ao usuário interagir com as múltiplas funcionalidades disponibilizadas neste sistema. Resultados obtidos demonstram que a implementação da arquitetura proposta oferece uma comunicação eficiente entre usuários que utilizam os recursos e o sistema multi-agentes responsável pela autonomia e inteligência de um sistema de estacionamento.

Palavras-chave: Computação em Nuvem; Sistemas Multi-Agentes; Internet das Coisas; Sistemas inteligentes de estacionamento.

Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence. 2

API Application Programming Interface. 45

CPS Cyber-Physical Systems. 2, 7, 15

CPU Central Processing Unit. 20

CRUD Create, Read, Update and Delete. 46

DOM Document Object Model. 42

FaaS Functions-as-a-Service. 30

GCP Google Cloud Platform. 41

GPS Global Positioning System. 53

GPU Graphics Processing Unit. 10

HTML Hypertext Markup Language. 11, 42

HTTP Hypertext Transfer Protocol. xii, 10–12, 14, 24, 30, 35, 45, 46, 62

IoT Internet of Things. 7, 10, 12, 13, 61

IPB Polytechnic Institute of Bragança. 2, 15, 57

JSON	JavaScript Object Notation. 21, 27
JWT	JSON Web Token. 49
MAS	Multi-Agent Systems. 2, 3, 7, 8, 14, 15, 17, 25–27, 30, 32, 33, 35, 37–39, 51, 57, 61, 62
MQTT	Message Queuing Telemetry Transport. xii, 13, 14, 24, 25
PaaS	Platform-as-a-Service. 10, 17, 61
QoS	Quality of Service. 8, 13, 57, 62
RAM	Random-Access Memory. 20
RFID	Radio Frequency Identification. 7
TCP	Transmission Control Protocol. 12
UI	User Interface. 47
UN	United Nations. 1
UTFPR	Federal Technological University of Paraná. 2, 15
VPS	Virtual Private Server. 16, 17

Chapter 1

Introduction

Over the past few years, the considerable population increase in large cities has become more noticeable. According to the United Nations (UN) [1], around 2050, the world's urban population will reach 68%, exceeding the current rate of 55%. As a result of this population growth, these urban conglomerates tend to become messy and disorganized over time, putting the management of natural resources and energy at risk [2].

The urban vehicle traffic field is greatly affected by this disorganization. According to a study commissioned by IBM [3], approximately 30% of a city's traffic is caused by drivers actively searching for a parking spot. In London, drivers spend an average of 67 hours a year looking for onstreet and offstreet parking [4]. Therefore, it becomes essential the search for solutions that reduce the time spent by drivers on the main roads, preserving their time and reducing unnecessary traffic, resources and emission of gases that are harmful to the environment [5]. Although the creation of new spaces facilitates the reduction of traffic congestion, the ideal solution would be to facilitate the search for available parking spots, making the time spent in the middle of traffic significantly reduced.

As a consequence, smart parking technologies emerge to solve this type of problem. These systems allow better management of parking spaces, quick rental of vacant spots on public roads, or even reserve a space as needed in a private parking. Implementations in this field range from mobile applications that inform drivers about vacant spaces to park, to the use of Artificial Intelligence (AI) to determine which is the best spot for each user according to their needs.

Smart parking systems are composed of Cyber-Physical Systems (CPS), whose counterparts are computational processes such as software or applications, and physical technologies such as sensors, where both are integrated in the same network. Sometimes, such physical components can fail due to hardware, software or connection problems, causing the system to lose data by cutting off direct communication with the device. In this context, the Multi-Agent Systems (MAS) methodology is seen as an organized collection of autonomous components with particular objectives, which interact, negotiate and coordinate efforts among themselves to solve tasks. Therefore, the system becomes distributed and able to self-regulate in the event of physical failures. A smart parking MAS treat parking spots as agents that communicate with each other and negotiate which spot will be allocated to the driver's agent based on parameters such as price, location and number of available parking spots.

Polytechnic Institute of Bragança (IPB), in partnership with the Federal Technological University of Paraná (UTFPR), has been developing a smart parking project especially in the MAS context [6], along with other modules such as dynamic pricing and AI. It is important to note that currently these modules do not have a scalable architecture that allows data sharing between distributed agents and integration between modules.

The project aims to involve parking of different types of vehicles, primarily bicycles and cars, so that the user can perform a distributed request to the MAS, providing some preferences, such as location and maximum price. Based on this information, the system executes the entire decision and pricing process, through negotiations between agents and data collection by machine learning, rapidly returning these dynamic decisions to the user.

The implementation of this project requires a secure, fast, scalable and cost-effective infrastructure. Based on this, two options are likely to be implemented. The first would be to create its own infrastructure, taking responsibility for equipment cost, maintenance and implementation of techniques for system security.

On the other hand, the second option is based on the use of Cloud technologies and platforms, introducing flexibility to integrate different modules, providing end-to-end security and system decentralization. In addition, it allows fast scalability, without the need to know the technical specifications of hardware or to pay for resources that are not used.

Considering the above, this work explores the creation of a architecture model that integrates a smart parking MAS within a Cloud infrastructure, providing a user interface for interaction between vehicle's drivers and the agents that control physical assets, namely the controllers of access to the parking spots.

This dissertation is divided into 5 chapters, the first one presents introduction of related subjects and the context to which they apply, together with important concepts and objectives of the work.

Chapter 2 provides helpful information to the reader about fundamental concepts through a review of related work, discussing typical implementations and protocols needed for the development and understanding of different modules.

Next, chapter 3 describes the proposed Cloud-based architecture to be used by smart parking systems, explaining the modules used, their applications and individual objectives. It addresses the use of a Back-end platform for database management, a Publish-Subscribe module, which allows the exchange of messages between devices, event-based Cloud functions, and the use of client-side for user interaction.

Chapter 4 presents the development of the architecture described in Chapter 3, showing the technical steps of implementation, activation of functions and communication flow. To understand these processes end-to-end, explanations using diagrams is provided. The development of a mobile application and a web system are demonstrated as part of a case study for validation purposes.

To present the results, chapter 5 demonstrates different situations and functionalities tested for architecture verification and validation. Finally, chapter 6 concludes with a brief description of the work presented, along with future work suggestions.

Chapter 2

State of the art

This chapter introduces the state of the art relevant to smart parking work, along with important concepts related to the implementation of Cloud technologies. Through a literature review, works related to the study of smart parking environments are presented. Subsequently, applications in the Cloud computing field are demonstrated, as well as protocols commonly used in the communication between different components of these systems.

2.1 Multi-Agent Smart Parking

When it comes to large cities, the growing increase in vehicle circulation has been causing problems in several areas. These cities, however large they are, have limited space in relation to the number of vehicles, and thus, tend to experience congestion on large avenues and central regions [7].

At times when vehicle traffic is at its peak, there is also a high demand for parking spaces, and due to the limited space in public areas, as well as private ones, many drivers waste too much time looking for a place to park [8]–[10]. Therefore, another important problem is emerging, dealing with environmental issues. With a large number of vehicles burning fuels and creating a substantial impact on emissions, the need for measures to reduce such pollutants becomes evident.

Smart parking methodologies arise to solve this type of problem [8]. Frequently, simple information about the location of available parking spaces can decrease the time spent by drivers in traffic. In [11], the authors propose a system to help users know about the availability of parking spaces in real time, using mobile application technologies connected to the Cloud.

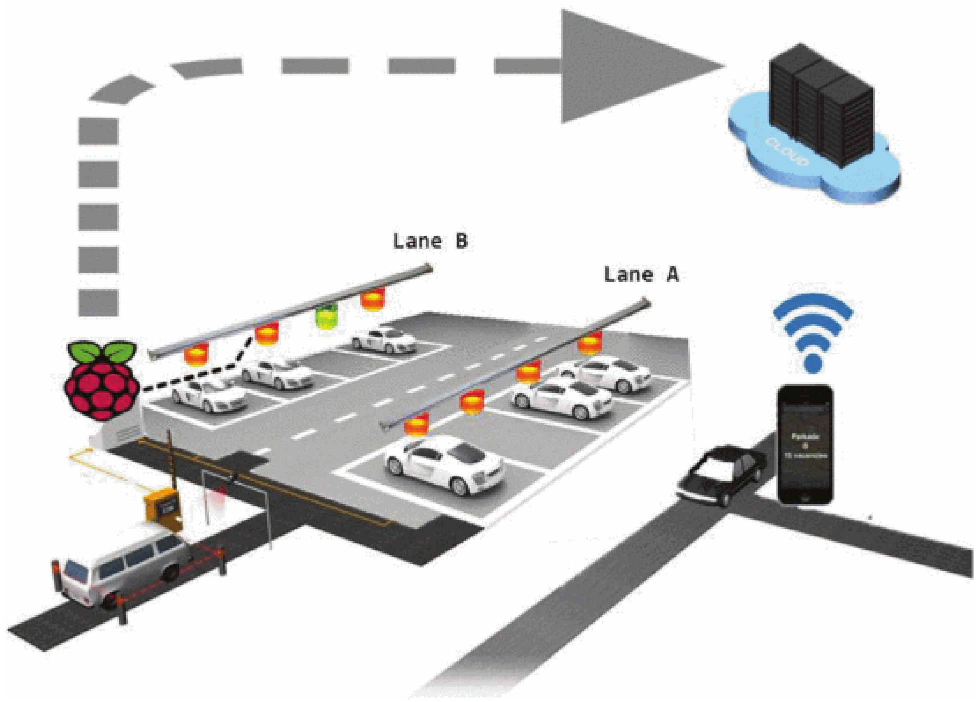


Figure 2.1: Smart-parking based on availability [11]

These systems are not efficient enough, as they do not have modules to improve the use of resources in an analytical way. In some cases, they can successfully help drivers to find a desired spot, but on the other hand, it can cause congestion to be redirected to other areas and thus make the traffic worse [12].

Some reservation-based systems focus on the use of parking spaces, allowing drivers to request reservations anytime, offering a wider range of usage possibilities. In these, dynamic prices are applied based on the general real-time availability of parking lots [12].

In [10], a smart parking system with reservation-based methodology is also proposed, where the user can establish requirements for reservations, such as the combination of proximity to a location and also parking cost, while the system ensures that its total capacity is used efficiently.

Sometimes, problems related to the management of these spaces are encountered, which may cover the physical area of the system, with sensors and microcontrollers, or even with digital control of the environment. In the study [13], through the use of Radio Frequency Identification (RFID) technology, it is provided solutions for some common problems found in smart parking management systems.

It is important to highlight the need for systems that can be scalable, as well as cities that need this kind of implementation. One approach that has gained popularity in this field is the use of MAS [6], [9], [14]–[16]. This approach facilitates communication between different components in a distributed way, where each agent acts in an autonomous and decentralized way, having different tasks within the system as a whole [17], [18].

Components or entities that perform actions on the system are called agents, and in the context of smart parking, these are divided into two fundamental groups, being drivers and spots [6]. Driver agents represent the users of the application, who can perform actions such as requesting reservations and searching for available spots. On the other hand, spot agents represent parking spaces, which can perform direct actions to the physical space, as well as communicate with other spots to negotiate and allocate reservations.

In the majority of multi-agent approaches, cooperation between agents is carried out by information sharing and enforcing constraints using rule-based procedures [19]. The ability to operate even if one of the components fails because of malfunctioning is one of the benefits when using MAS within a smart parking architecture, because it provides a distributed communication that handles normalization of the system in such cases, as well as multiple parallel requests.

A study by [6] presents an agent-based CPS for smart parking systems, addressing how software agents are interconnected with physical asset controllers using Internet of Things (IoT) technologies. This work covers not only smart parking for cars, but also for bicycles as alternative vehicles.

In [14] is proposed a platform for dynamic parking negotiation based in MAS. The implementation of the agents utilize classical artificial intelligence planning methods to plan activities and resolve any foreseen conflicts. Through negotiation, the agents engage in message sharing, exchanging and evaluating each other's proposals, then modifying their own proposals until all agents are satisfied.

The study [20] explores a integration between mobile agent technology with multi-agent systems to build a application that facilitates the search for available parking spaces, negotiation of parking fees, reservation of parking spaces, and the derivation of optimal paths.

In the work of [21], a MAS capable of allocating spaces in parking lots is modeled and developed based on a centralizing agent, called *manager*, which is responsible for managing these spaces. The system also makes use of degrees of confidence, as well as the waiting time factor that influences the process of allocating those spaces.

2.2 Cloud Computing

Cloud computing is evolving as a new computing model designed to offer dynamic and on-demand computing environments for users, providing a fast, secure and customizable service [22], [23]. In this chapter, Cloud computing model is analyzed through various aspects, such as concepts, distinct features, including an introductory review of state-of-the-art technologies.

A computing Cloud is a set of network enabled services, providing scalable, Quality of Service (QoS) guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way [24]. In other definition, it is refered to be both the applications delivered as services over the Internet, as well as the hardware and systems software in the data centers that provide those services [25].

Having approached some definitions, the study [26] addresses five elements that define the main objectives of Cloud computing services.

- On-demand self-service: Systems can use computing resources according to demand without the need for human interactions with the service provider.

- Broad network access: These computing resources can be accessed from a wide range of client applications across different platforms, such as desktops, tablets and smartphones.

- Resource pooling: Computational resources of a cloud service provider are ’pooled’ together in an effort to serve multiple users. As a result, physical computer resources are abstracted and become ’invisible’ to users who do not usually have access or knowledge about technical specifications of these.

- Rapid elasticity: Ability to provide scalable services according to the users’ immediate need, without limit of use.

- Measured service: Providers use measurement techniques to make available information concerning the usage of various services for each individual customer.

Based on these concepts, many applications have been developed using the services potential offered by Cloud providers. Platforms that provide these services are becoming an excellent alternative for academic organizations, or even private businesses with less budgets to host and run online applications [27].

In [28], a software testing environment is proposed, using Cloud computing technology and virtual machines with fault injection facility. This application can configure environments and perform tests automatically, as well as inject faults automatically into hardware devices on a virtual machine.

A work by [29] designs a Cloud-based health care service that performs real-time monitoring of user health data for effective management of chronic illness. This approach implements a system that uses advanced sensory devices to gather large data sets of users from different locations. For effective retrieval, updates and fast transfers as and when necessary, a Cloud-based storage repository is used.

To take advantage of Cloud computing power, [30] proposes moving computationally demanding object recognition processes within drones to a remote compute cloud. This technique removes the need to use high-end Graphics Processing Unit (GPU) that require too much power and weight.

The security aspect is very important in environments that use components based on Cloud services and has been addressed by several studies [31]–[33]. Therefore, instead of implementing manually, many users choose to use platforms that provide this type of service, ensuring data security and reliability.

There are studies that make the analysis and comparison between these platforms, functioning as Platform-as-a-Service (PaaS), taking into account factors such as implementation, computational processes speed, scalability, security and pricing [34]–[38].

2.3 Protocols

Some communication protocols commonly used in cloud-based systems are discussed in this section. For this context, the section will briefly describe HTTP, that is generally used for client-server communication, WebSocket, which can be used to allow users across different platforms to access database resources in real time, and finally MQTT, a protocol frequently used in decentralized IoT applications.

2.3.1 HTTP

The HTTP is a stateless communication protocol between client and server, i.e., the client-side does not maintain an active connection with server-side. Instead, data is exchanged by sending requests and receiving responses [39].

It is defined as a half-duplex protocol, where both sides cannot communicate simultaneously, with a one-way message exchange flow. [40]. Even though it is a stateless protocol, there are approaches that show the use of persistent connections for multiple data exchanges [41].

The request message consist of some attributes, such as the protocol version, a method to indicate the desired action to be performed, the path of the resource to be fetched, a header containing additional information, and optionally a body, which is commonly used with some methods. There are methods with different objectives when it comes to the desired action towards the server, of which the main ones are described according to [42].

- GET: requests a representation of the specified resource, only used to request data.
- POST: uses the request's body to submit an entity to the resource.
- PUT: replaces all current representations of the resource with the request body.
- DELETE: deletes the specified resource.

The responses also have a standardized structure, which, like the requests, contains the protocol version and a header, in addition to a status in code and message form, representing the success or not of the requested action. Finally, an optional body containing the requested resources is present.

HTTP protocol usage is commonly present when accessing websites [43], in which the browser makes a request using the GET method to obtain an Hypertext Markup Language (HTML) file. As soon as the server returns a response, the browser displays the file in the form of a web page.

In order to obtain data in near real time, a method called HTTP polling is used [45]. This method consists of making requests constantly in a short period of time, making new information to be obtained quickly. However, this method can cause server overload, due to the large number of requests made by multiple users [46].

2.3.2 WebSocket

As opposed to HTTP, WebSocket provides full-duplex communication [47]. Without first being asked by the client, it is possible to provide a standardized way for the server to transfer data to the client and allow messages to be sent and received while keeping the connection open [48].

WebSocket protocol is an application-level protocol that allows web applications to

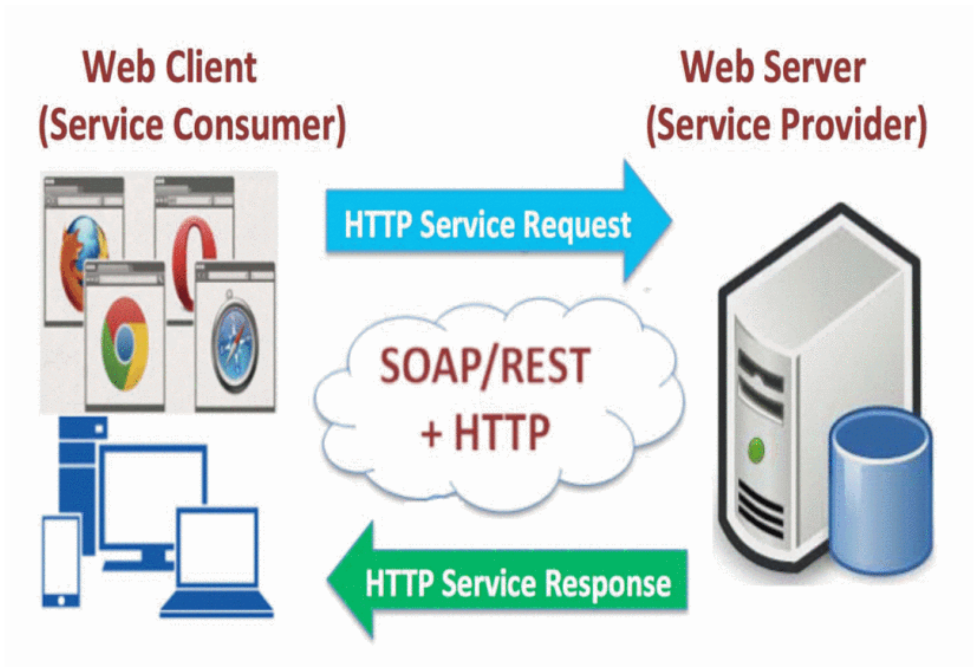


Figure 2.2: HTTP used to access websites [44]

maintain a connection based on the Transmission Control Protocol (TCP) sockets standards, causing information to be transmitted by both sides of the connection simultaneously [49]. This protocol is suitable for Web-based that transfers near real-time data, usually required in games, chats, multimedia, and remotely controlled systems[50].

Through a process known as the WebSocket handshake, the connection between client and server is established [46]. The communication starts with the client submitting an HTTP request to the server, which includes a header attribute informing the server that the client needs to establish a WebSocket connection. The server then agrees to and communicates a regular response notifying success. Thus, the handshake is done, allowing a WebSocket link to replace the original HTTP connection.

A study by [52] analyzes its application in IoT environments, such as smart homes, by monitoring and managing energy consumption in real time. In the same environment, [53] compares the use of the Websocket protocol with polling methods, in which the superiority in Websocket performance is concluded.

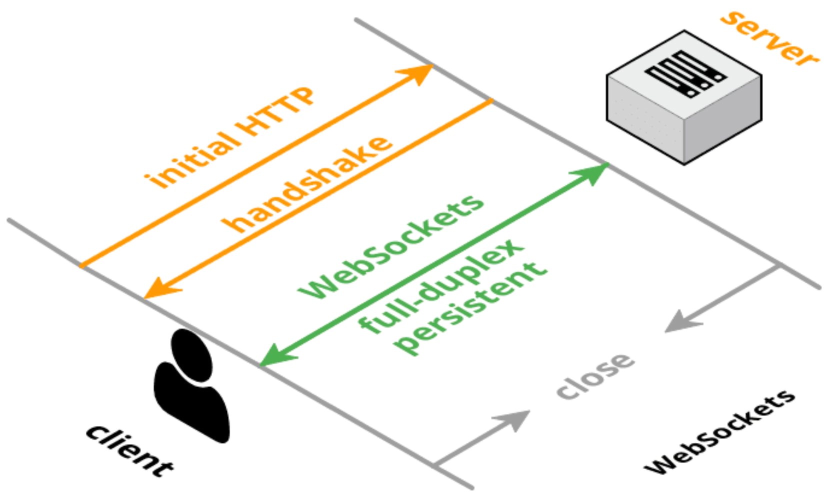


Figure 2.3: WebSocket handshake representation [51]

2.3.3 MQTT

MQTT [54], [55] is a communication protocol that allows messages to be transferred by one-to-one, one-to-many and many-to-many devices. It is commonly used in IoT applications, due to its flexibility and lightness when compared to other means of communication that require greater computational power.

Its communication is based on a central broker, responsible for forwarding messages between multiple devices [56]. These devices can act as senders of messages, being called Publishers, or as recipient, called Subscribers.

The exchange of data is done through messages linked to specific topics, in which interested devices can subscribe, establishing a listening relationship for new messages. Therefore, whenever a message is published, all devices that are subscribed to the topic will be informed.

In order to ensure the reliability message delivery, MQTT communications provide three QoS levels [56], [58]. The lowest level has the highest speed in delivering messages, while the highest level has the greatest guarantee that messages will be delivered.

- Level 0: A message is delivered only once and no acknowledgement of receiving is

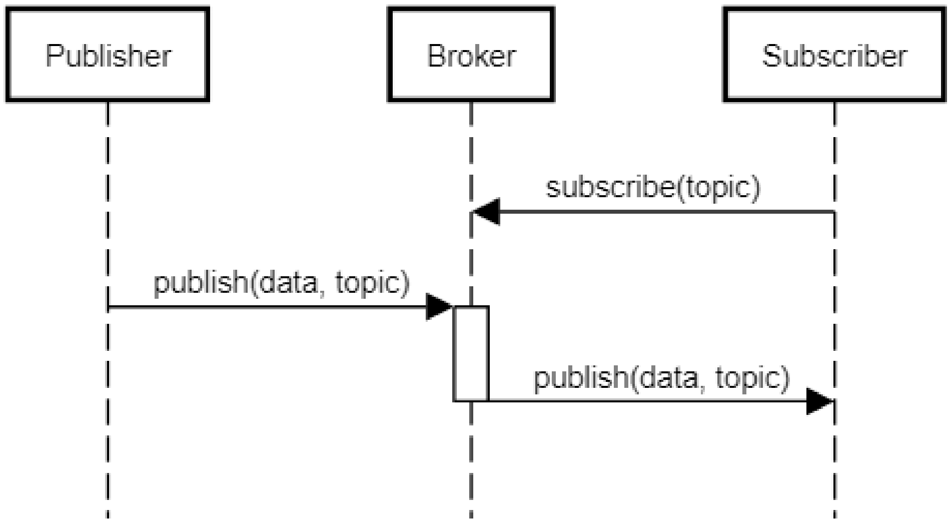


Figure 2.4: MQTT communication [57]

required.

- Level 1: Each message is sent at least once and an receipt acknowledgement is needed.

- Level 2: A four-way handshake method is used for the delivery of messages.

In the smart parking context, MQTT protocol fulfills the purpose of communication between system's agents, allowing a fast, light and distributed exchange of messages in both ways. Similar to the WebSocket protocol, communication occurs without the need for a request from the side interested in the information, allowing the MAS to obtain new data without the computational cost of long polling as in the HTTP method. In addition, it reduces data loss by ensuring message delivery, making the system less prone to communication failures.

Chapter 3

Proposed Cloud-based Architecture

This chapter presents an overview of the proposed smart parking architecture, briefly contextualizing the multi-agent system to be integrated and the reasons behind the choice of using Cloud technologies. The structuring of the architecture is also present, describing the role of each component that constitutes the system, focusing on the importance of interaction between the user interface and its respective driver agent. The present work uses as reference the MAS for smart parking proposed in [6].

3.1 Architecture Overview

One of the main objectives of the proposed architecture is to allow user interaction with the multi-agent system elaborated in the project between IPB and UTFPR, mentioned in the previous chapter [6]. In summary, this system performs the interconnection between physical components and computational resources, featuring a CPS, using an approach based on intelligent and autonomous agents that enable the decentralization of the system. In the figure 3.1, the architecture represents the connection between user interface and driver agents.

In this method, connectivity on the same network is necessary to maintain communication between spot agents and physical components. Thus, the scalability provided by MAS becomes limited by the local connection.

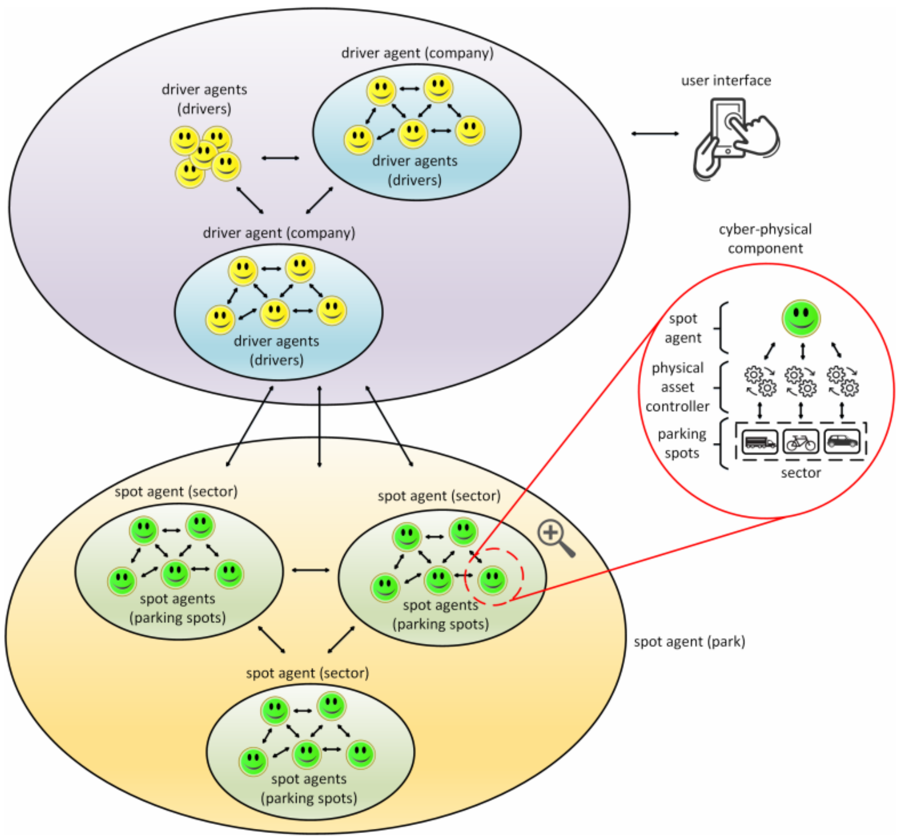


Figure 3.1: Multi-agent system architecture [6]

To get around this problem, it is necessary to integrate this system on a platform that allows communication between both parts, without this limitation occurring. Based on this, some options can be considered below.

- Dedicated Server: This option is the implementation of a private server, dedicated only to the system. The server does not share resources between multiple clients, allowing the client to access all available performance. However, it is the most expensive and difficult option to implement, as the cost of hardware and maintenance, as well as technical issues of software and operating systems must be considered.

- Virtual Private Server (VPS): When using this service, the system will be hosted on a server that shares its resources with other systems, but provides a virtual environment for each one. This option allows the administrative control of the own environment, but not the server as a whole. However, because it is a centralized server, systems that require high computational demand at specific times can be limited by the infrastructure.

- Cloud Computing: Unlike the methodologies above, this option does not depend on just one physical server. Cloud Computing allows an unlimited number of servers that act as a unified system. Therefore, the system is hosted in a virtual subsection, just as VPS, but the resources can be used by multiple servers at the same time. Its appeal consists of flexibility, ease in scaling processes and implementation, and payment according to the use of resources.

The use of Cloud Computing technology is an option compatible with this project in order to better deal with scalable applications, provide a better cost-benefit ratio and abstract hardware specifications.

The accumulated experiences demonstrate that the development of efficient Cloud-based architectures positively affects the scalability of complex systems. For the development of a good architecture, above all, it is fundamental to first consider that users must have a properly adequate experience regardless the amount of data flow. Based on this, it is important that the scalability of the system allows the user experience to be always pleasant.

In a smart parking context, such an architecture should provide fast and secure communication between driver and the MAS. For this reason, it is interesting to use Cloud platforms that have a high level of security and availability, and thus, abstracting concepts that will not be explored in this work. Such platforms are provided as a service, more specifically as a PaaS, which automatically scales up computing resources to match the usage patterns of users, causing billing to vary according to demand.

The use of a Cloud service provider platform allows easy scalability of applications and control over the use of resources. Based on the platform used in this work [59], [60], it is possible to notice the potential to support multiple connections of a large number of users. The limitations of its services are based on the size of requests and above all on

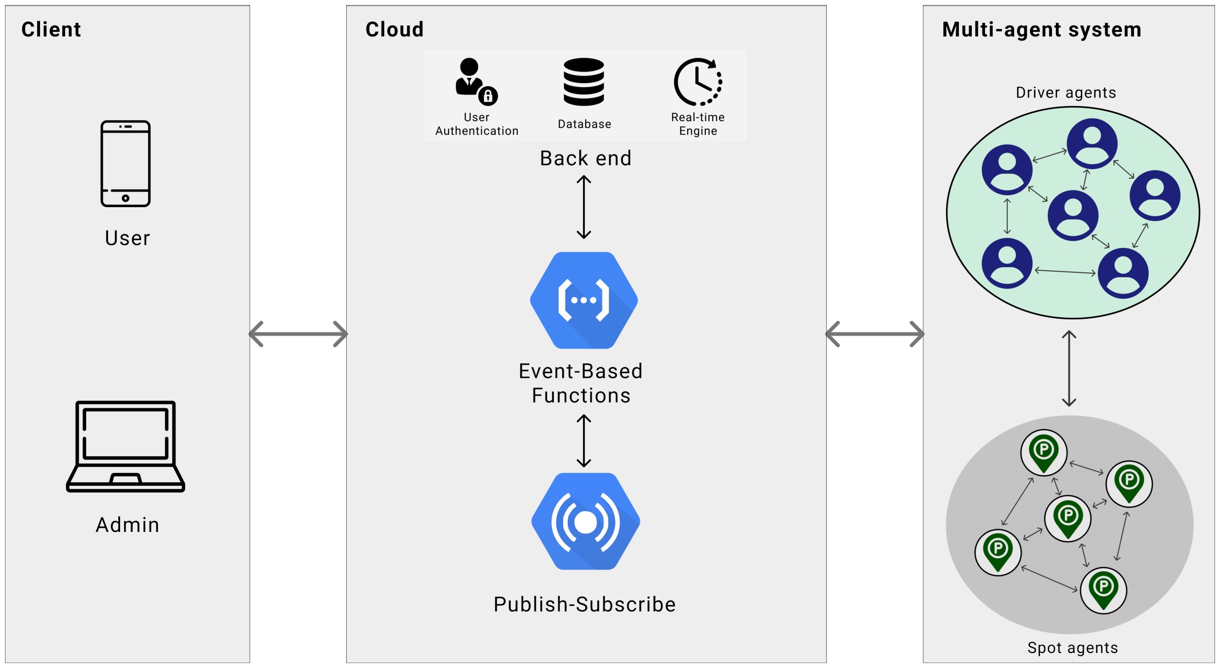


Figure 3.2: Architecture representation

parallel access to the database.

It also have a limited number of event-based functions, however, in a properly scalable architecture, this number should not be reached. These functions must not exceed a certain maximum size, neither last more than 540 seconds. It is important to note that, in the proposed architecture, implemented functions run on average from 1 to 2 seconds.

The architecture resolves the communication between users and their equivalent driver agents, establishing a secure connection through modules of the Cloud platform and preserving performance on the client side, as can be represented by the figure 3.2.

In this architecture, the main modules are user authentication, database layer, real-time engine, communication by Publish-Subscribe and event-based functions in the Cloud.

3.2 Back-end Platform

As a back-end platform, this module is responsible for providing the direct connection between the client side, responsible for providing the interaction of users (i.e, drivers and admins) with the system, and the Cloud service. It has fully scalable functionalities for user authentication, as well as storage in NoSQL databases and high performance data management.

3.2.1 User Authentication

The main purpose of an authentication module is to verify the user's identity, involving data validation methods to confirm its authenticity.

It is important to have a flexible authentication module accessible at all times for users to have access to the application. Besides identity validation, one of its objectives is to identify user roles such that they can only access the services allowed for their role. For example, common user roles may not allow access to other user's data, unlike administrative roles, which cover a wider range of permissions.

In currently applications, authentication using email and password, phone numbers, and even providers like Google, Facebook and Twitter are commonly supported. When using one of these methods, the system can recognize which user is performing a certain action, being able to control access to the use of system's resources depending on the role assigned.

3.2.2 NoSQL Database

There has been a massive rise in data storage in recent years, largely due to the huge social networks such as Facebook, Twitter, search engines such as Google and other websites that store huge amounts of data. Therefore, there are several challenges such as storage, capture, analysis and transfer of this data, and this is where NoSQL databases (e.g., MongoDB, Firestore, DynamoDB) come in, known for operating well with big data as well as with real-time applications.

Concepts

The main advantage in using this type of database is its extremely flexible model. When comparing with the development of relational databases such as MySQL or Postgres, it is noticeable the importance of modeling their whole structure in a predefined scheme, including tables, their fields, their constraints, and also the type of each field. In NoSQL databases, also known as non-relational databases, there is no need to create schemas, which means that there are no database level restrictions on pre-defined tables or what type of data to place in certain fields, being able to iterate very easily in the database design, adding or changing fields as needed without the high risk of corrupting current processes. However, some planning is important so that the structure will not be disorganized over time.

NoSQL databases are, in many cases, faster and cheaper to manage because of the scalability aspect. Scaling is highly important in today's industry, which is increasingly growing, where websites and apps tend to draw more customers and thus more data.

Relational databases tend to use vertical scaling methods, meaning that additional components such as Random-Access Memory (RAM), Central Processing Unit (CPU) power, network ports and other components that are normally expensive need to be attached to a machine in order to be scaled. There is some drawbacks to the vertical scaling process, such as the expensive cost of new materials, or even the need for new infrastructure. From this point of view, the hardware and the centralisation of data on a single machine have many disadvantages.

In the other hand, horizontal scaling approach allows new machines to be added to the resource cluster as nodes and facilitates the maintenance and addition of new ones in non-relational databases. This approach entails separating a sequential portion of the logic into smaller parts so that they can be performed on several machines in parallel, increasing the efficiency of large-scale readings and writings as well.

There are different types of NoSQL databases, like key-value stores, which is the simplest of them, where only keys and their respective value are present, being ideal for large

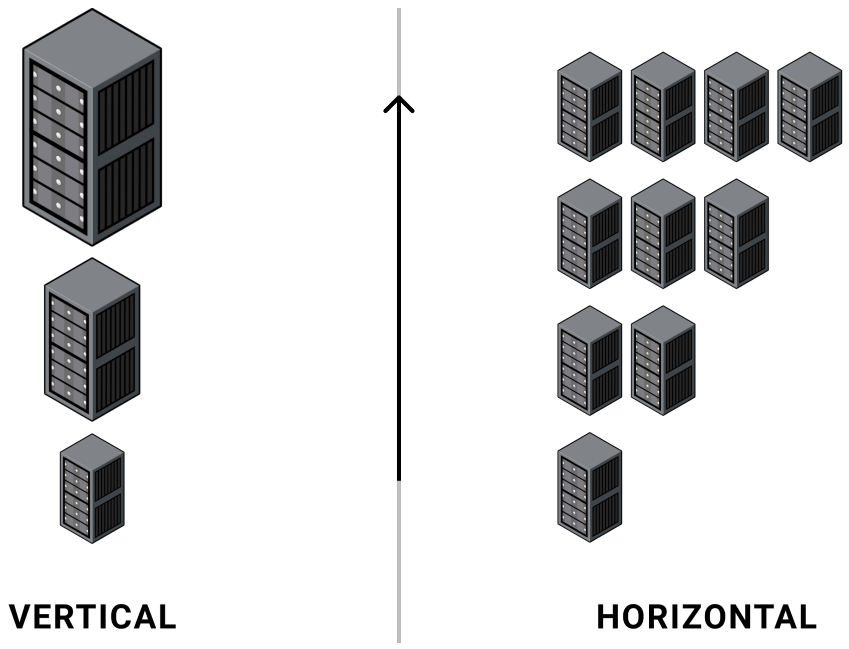


Figure 3.3: Vertical scaling vs Horizontal scaling

datasets with very basic details. It's extremely fast, customizable and not very complicated, close to a hash or an associative array. Additionally, there are Graph databases, which aren't that common, where everything is treated as a node, which can have relationships with other nodes through what is called an edge.

Finally, document databases that store data similar to JavaScript Object Notation (JSON), where the data is wrapped between curly braces and have key-value pairs. These values can hold strings, numbers, arrays, embedded objects, and so on. On the database level, it is completely schemaless, so the application can be very dynamic. For example, if the application is intended to have users that are able to add custom fields to their profile, a document database is certainly the solution to be adopted.

Organization model

In the proposed architecture, it is necessary to dismantle the database organization model with regard to the condition of smart parking. Based on the data structuring model related to parking provided by Fiware [61], a platform that provides standardized data

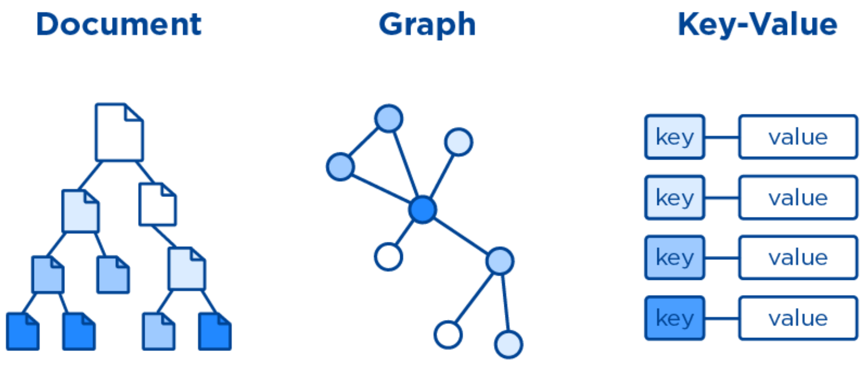


Figure 3.4: NoSQL database types representation

models, it was possible to adapt to a document-based database with NoSQL structure, using essential attributes according to the model studied, as well as adding new attributes that fit the project.

The figure 3.5 represents the database structure when it comes to storing information about registered smart parking places, and it is possible to verify that the abstraction forms a tree, originating from the most generalized elements (i.e., parking areas) to the most specific ones (i.e., spots).

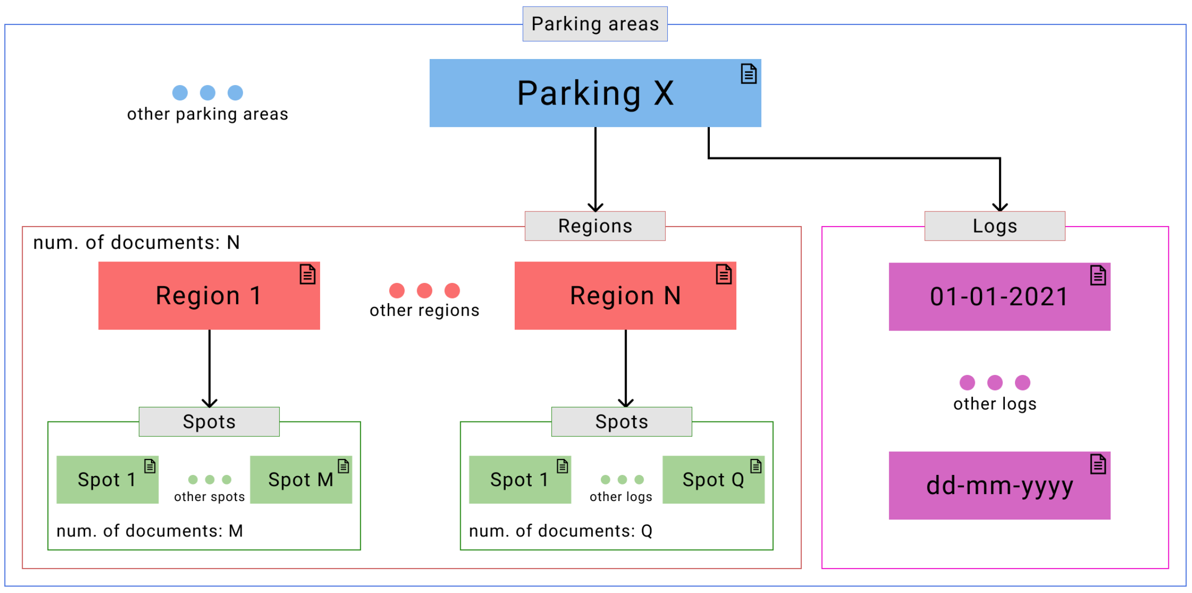


Figure 3.5: Parking database structure

The structuring of data have root in a collection of documents representing each parking area, which have the required attributes mentioned below.

- Name: full name of smart parking
- Description: parking description (e.g. shopping center, university)
- Max. parking duration (hours): maximum duration of each reservation
- Total spots: total number of spots
- Vehicles allowed: types of vehicles that can be parked
- Coordinates: latitude and longitude
- Address: complete address
- Image URL: URL of the parking logo

Such parking documents have a subcollection containing the individual regions, of which, like the previous documents, have attributes related to their basic identification and location information, which are:

- Name: region name (e.g. sector A, second floor)
- Description: region description
- Coordinates: latitude and longitude

Finally, each region has its subcollection of individual spots, which can be used to park bicycles, motorcycles or cars. Therefore, the attributes are basically:

- ID: spot identifier
- Spot type: bycicle/car/motorcycle
- Coordinates: latitude and longitude

The data for each reservation will be added as documents in a subcollection belonging to the user's document, which contains personal information, as well as another subcollection related to their registered vehicles. The figure 3.6 represents the users collection organization model.

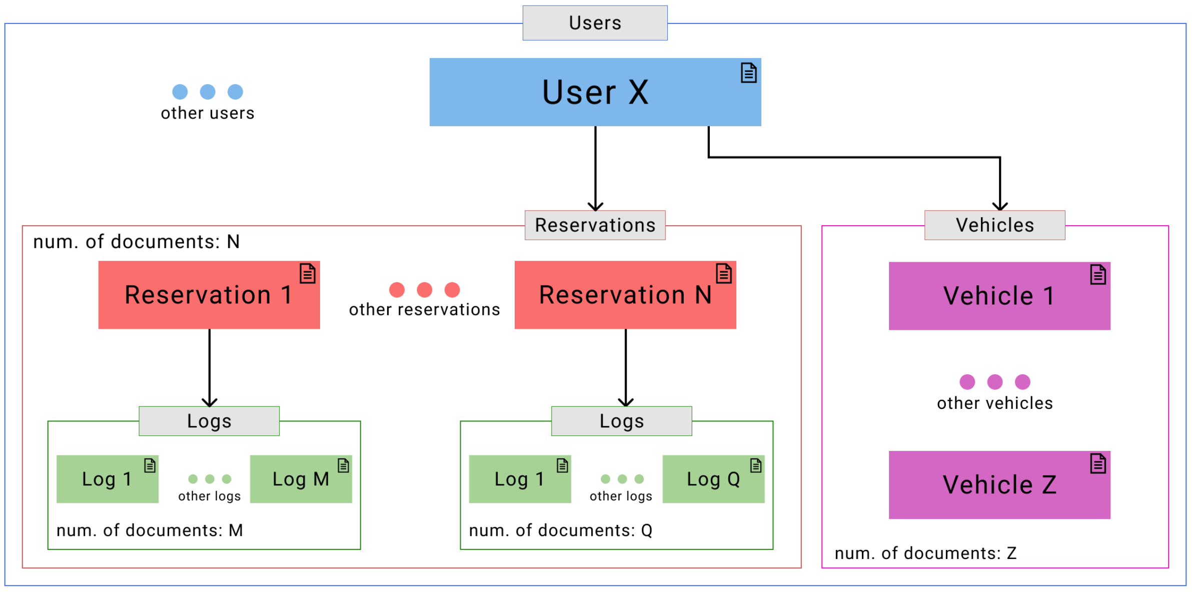


Figure 3.6: User database structure

3.2.3 Real-time Engine

The real-time engine uses a NoSQL database to store, synchronize and make data available to users in real time. This makes it easier for them to access data from any device, web or smartphone.

Instead of typical HTTP requests (2.3.1), real-time engines uses data synchronization through WebSocket (2.3.2), that is, whenever some data is updated in the database, it is stored in the Cloud and simultaneously notifies all devices interested in milliseconds. In such cases, for receiving data and detecting changes in real time, methods for observing these events must be used on the client side.

This approach can be similar to the MQTT broker (2.3.3), which reacts when a message is sent from a Publisher and redirect it to all Subscribers. However, the difference is the addition of the data persistence, which is the database itself.

To remain responsive to those devices that go offline, the database uses a local cache to serve and store changes. This ensures that the local data is automatically synchronized with the current state of the server when connectivity is restored.

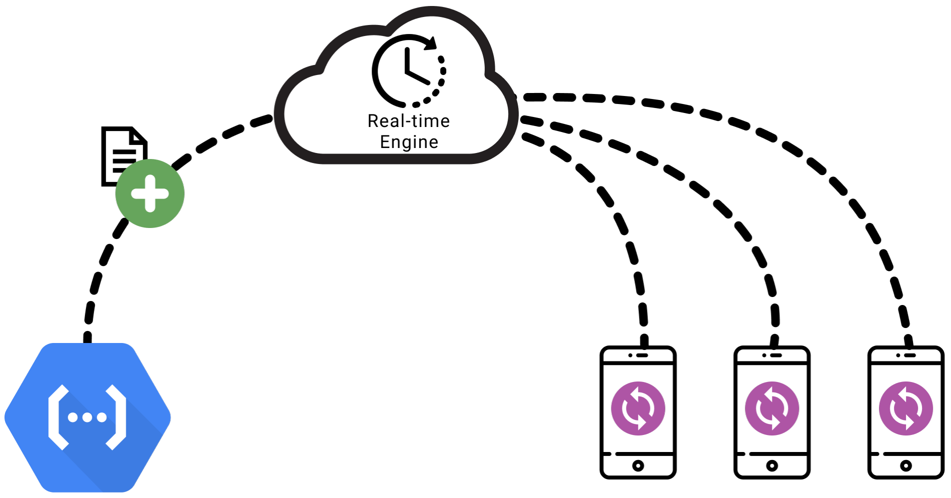


Figure 3.7: Real-time engine representation

In this architecture, this module is used to notify users in real time about the responses of requests made to the MAS. These notifications may include availability status of parking spots or the result of a reservation request.

3.3 Publish-Subscribe

Systems are decoupled into smaller and separate blocks in modern Cloud architecture, which become easier to maintain. The interaction between agents is continuous in a smart parking environment and requires asynchronism when exchanging messages. The Publish-Subscribe model therefore provides the broadcast of real-time message for these multiple devices, using methods similar to MQTT.

3.3.1 Concepts

The Publish-Subscribe standard emerged from the need for communication between components with dynamic scaling in applications, without the need for direct source-to-destination communication. To better understand the context, it is important to know the key concepts that encompass this model. First, there are the topics, which will be the central elements of the entire data flow. Each topic must be related to a particular category of information and will have the role of endpoints for connecting the individual elements that wish to engage in sending and receiving messages.

To send a message to other agents in the system, it is necessary for Publisher to simply publish a payload containing the message in a topic. Such payload can contain optional and customizable attributes that characterize can be used as a filtering mechanism by Subscribers. As soon as the message is published, the Cloud platform will act as a message broker, transferring this payload individually to all devices interested in the topic in question through a subscription, unless a filtering mechanism has been configured in it.

In case any device goes offline, published messages are temporarily stored until all Subscribers acknowledge receipt. After receiving the message, each Subscriber can use the message data according to their objectives. Therefore, the sender of the message does not need to know who the recipients will be or what they will do, just as the Subscribers do not need to know who published the message.

3.3.2 Topics

In a smart parking MAS context, topics should cover scenarios in which the target destination is either spot agents or users. Therefore, for each topic involving request messages from one side, another topic must correspond to a response with an inverted communication flow. Messages destined to end users are intercepted by event-based functions, which use the real-time engine to forward this data and inform the response.

Since it is an architecture with an emphasis on scalability, it is not possible to create topics with the same purpose for multiple parking areas. For this reason, it is used a filtering method capable of receiving unique messages that contain an specific attribute in its payload.

For example, a subscription can be configured to receive only messages where the *parkingId* attribute stands for the smart parking of which the spot belongs. As a consequence, the number of topics remains constant when compared to the number of features, however each Subscriber will receive only messages related to their own smart parking.

Topics where the destination is intended for spot agents must attach to the messages an attribute that indicates the identification of the parking, so that only spot agents of the requested parking receive them. In certain instances, it is also important to append the identifier of a particular spot, so that the message is not transmitted to every other spot agent.

In this work, messages published in topics use the JSON format, capable of storing data in the format of attribute and value pairs.

Search available spots

On some occasions, the user may request the smart parking to inform which spots are currently available, so that he can make his decision based on that information. Therefore, messages published in the *SearchRequest* topic are intended for spot agents, having an attribute referencing the unique identifier of the user who made the request.

In response to the previous topic, each available spot agent will publish a status message to the *SearchResponse* topic, containing the specific location as coordinates along with the identifier of the user who requested such data.

Request spot

The *ReservationRequest* topic is responsible for initiating communication between a user and the MAS. When a reservation is requested, a message is published to this topic containing the request parameters chosen by the user.

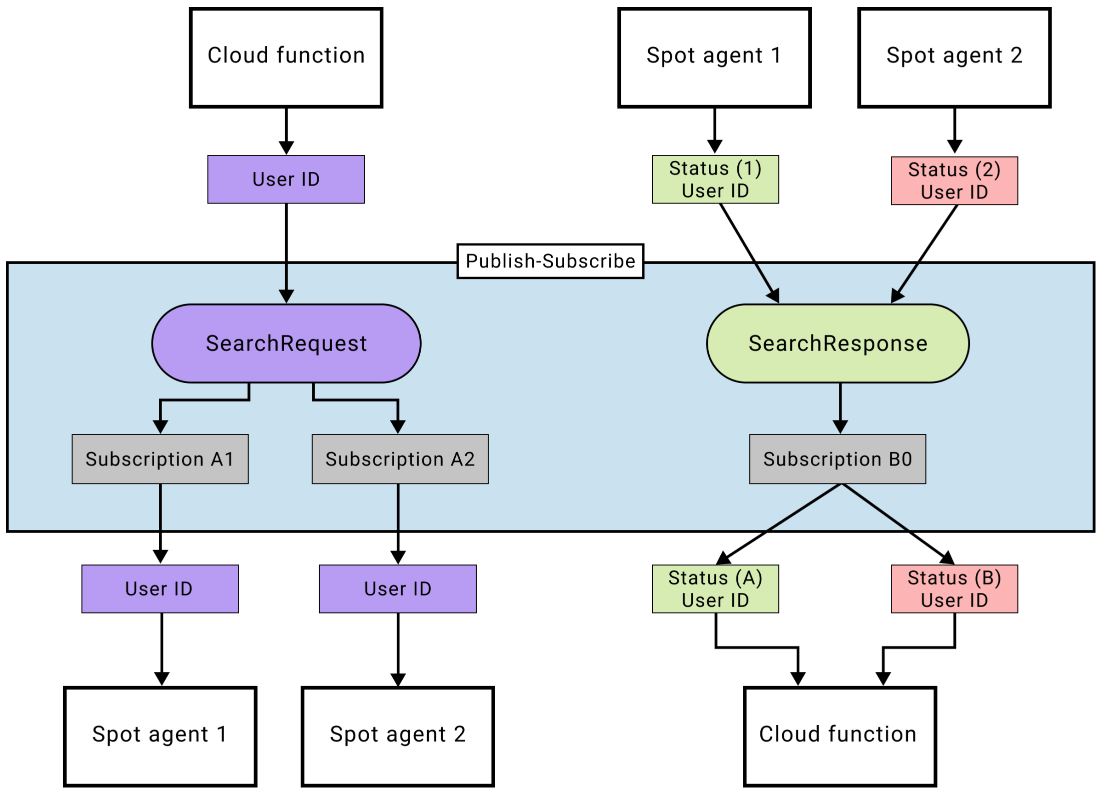


Figure 3.8: Search available spots representation (Adapted from [62])

Therefore, all spot agents of the requested parking that have a subscription in this topic will receive the message payload with the requisition, and then negotiate among themselves which is the most suitable spot for the user in the current situation.

As soon as the multi-agent system concludes its negotiation, the spot agent responsible for the reservation must inform the final details to the user. In order to transmit the message, this spot publishes it to the *ReservationResponse* topic, attaching the user's identifier so that the message is delivered to him.

Arriving and Departuring

Upon confirmation of the reservation and if it is within the period of occurrence, the user can request the entry or exit of the spot. When requesting one of these, a message is published in either *Arriving* or *Departuring* topics, corresponding respectively to the entry and exit actions, being sent only to the spot agent responsible for the reservation.

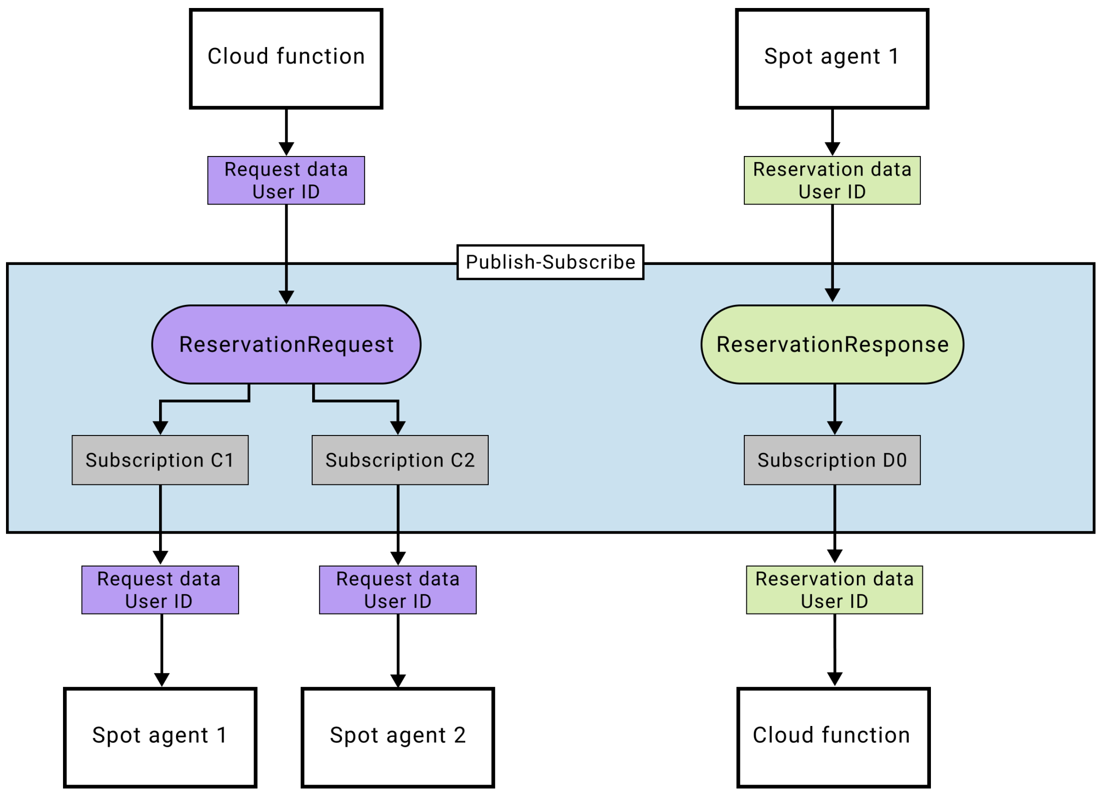


Figure 3.9: Request spot representation (Adapted from [62])

Therefore, this spot agent carries out its individual procedure regarding its type. The procedure can be to unlock upon arriving or lock when exiting a parking spot for bicycles, or even to open and close a barrier for cars.

Finish reservation

As soon as a spot's reservation period is over, a message is published to this topic in order to remove user's access permission. The message body will have attributes representing the unique identifiers of both the user and the reservation.

3.4 Event-based Cloud functions

There are occasions where a specific piece of code must be run if a certain event occurs in the Cloud environment, such as adding a document to the database, or publishing a message on a topic, and to prevent unwanted replication, functions triggered by those events are used. These functions work as a serverless module for developing event-based microservices.

Due to the execution based on the occurrence of events, the use of this methodology reduces the use of computational processing of the server, as well as the need to periodically execute functions. In a smart parking system, where multiple requests are made by multiple users simultaneously, latency related to process queue is not a issue. This is due to the fact that functions present in the Cloud are executed through multiple instances in parallel, allowing scalability to occur horizontally in relation to the number of functions currently running.

It's essentially customizable and scalable, working as a Functions-as-a-Service (FaaS) to execute the code with zero server management. In this work, these functions have a fundamental role in the communication between users and the MAS.

To further demonstrate in depth the development and purpose of each function, it is important to better understand which event triggers are fundamentally used. One common use is through the manipulation of the database, triggering a function once any document is created, updated or deleted within a specific collection. It is useful for situations like adding extra information at document creation, validating information when updating or even publish its data to a Publish-Subscribe topic. Publishing a message in a specific topic and direct HTTP requests can also be used to trigger a function. These methods are represented by the figure 3.10.

3.4.1 HTTP Request

There are only two functions that are activated by this method, one related to the administration, and the other to the use of the application by drivers.

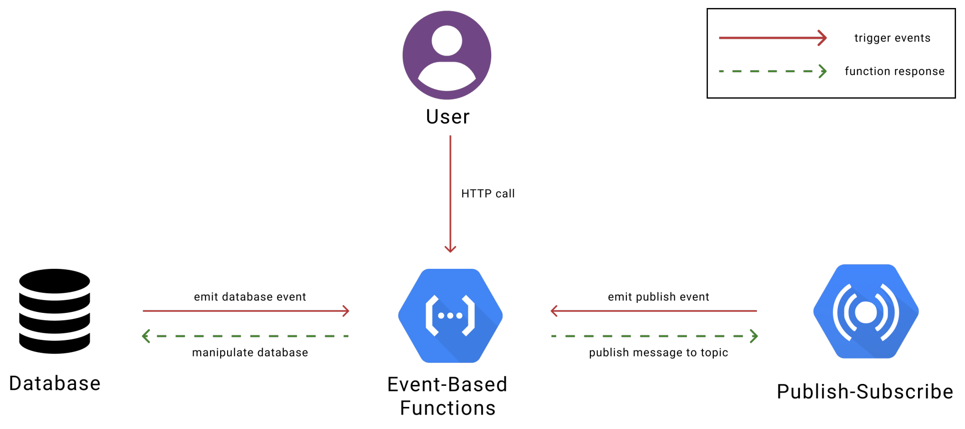


Figure 3.10: Event-based functions representation

addAdmin

The request body must contain the email address of a user who will receive the position of administrator. When activated, this function should check the user's admin active credentials and ensure that he is a system administrator, otherwise, it should return an error stating permission denied.

The methods of this function are to provide the administrator position for the user related to the informed email. This permission can be related to an attribute in the user's document present in the database, or the assignment of an authentication-related token.

checkAvailableSpots

This request must have the parking identifier, for which information about the available parking spots is requested. The function must be activated by an application that has a authenticated user.

Therefore, the function's objective is only to publish a message in the topic *SearchRequest*, containing identifiers of the parking area and the user. All agents subscribed to this topic receive the message that corresponds to a request about the current status of spots.

3.4.2 Database manipulation

onCreate Request

This function is triggered immediately after creating a document in the user's subcollection *Requests*. The creation of this document represents a request for a reservation by the user.

Therefore, the execution of this function is intended to forward this request to the MAS. For this, the function extracts the requested parking identifier present in the document, and publishes a message in the topic *ReservationRequest*. Finally, all agents registered in this topic and who are from the identified parking receive the request.

onUpdate Request

When a document related to a request is updated, it can be related to different situations of a reservation. However, sometimes the *userStatus* attribute can be changed, representing the user's intention to enter or leave the spot.

In this case, the function publishes a message to the topic related to the action (i.e., *Arriving* and *Departuring*), containing identifiers for the parking, spot and user for purposes of filtering by the subscription. In addition, the published message must have the timestamps referring to the beginning and end of the reservation, so that the agent responsible for the reservation has such information.

3.4.3 Publish to topic

SearchResponse and ReservationResponse

Each message published to this topic triggers a function, whose execution redirects the response of the spot agent to the user indicated in the message, using the real-time engine to perform this notification.

FinishReservation

The end of each reservation is indicated by messages published to this topic. Therefore, user and reservation identifiers present in the message are extracted by the function. With this information, the *status* attribute of the document related to the reservation is changed to *Finished*. When this change occurs, the function also increases the amount and total duration of reservations, present in the parking's document indicating the record for the day in the subcollection *Logs*, allowing administrators to have access to such data without the need for searching and counting of all reservations' documents.

3.5 Client-side

Contrarily to server-side processes, where most of the business logic is found, there is also the client-side, which applies to anything in an application that is shown or takes place on the user device. This covers visual components, such as text, photos, along with any feedback or behavior that the application perform. An interface that facilitates interaction with Cloud resources is essential for the user to be able to interact with the system.

In the modern industry, the use of smartphone technology has risen, from social media to systems management, making it interesting to use these applications as part of the architecture. The use of these facilitates access to the different functionalities of a system, allowing different portable devices to have a pleasant and fast experience when using the application's tools.

The use of mobile applications proves to be a good choice in a smart parking context, due to the possibility of drivers using their resources easily. Therefore, this application must allow users to register in the system, request reservation allocations from the MAS, as well as check the availability of parking spots in real time. For better understanding, the diagram 3.11 explains the use cases for each user.

In this section, some functional requirements and main processes that must be present in the mobile application for drivers will be addressed. These functionalities provide the

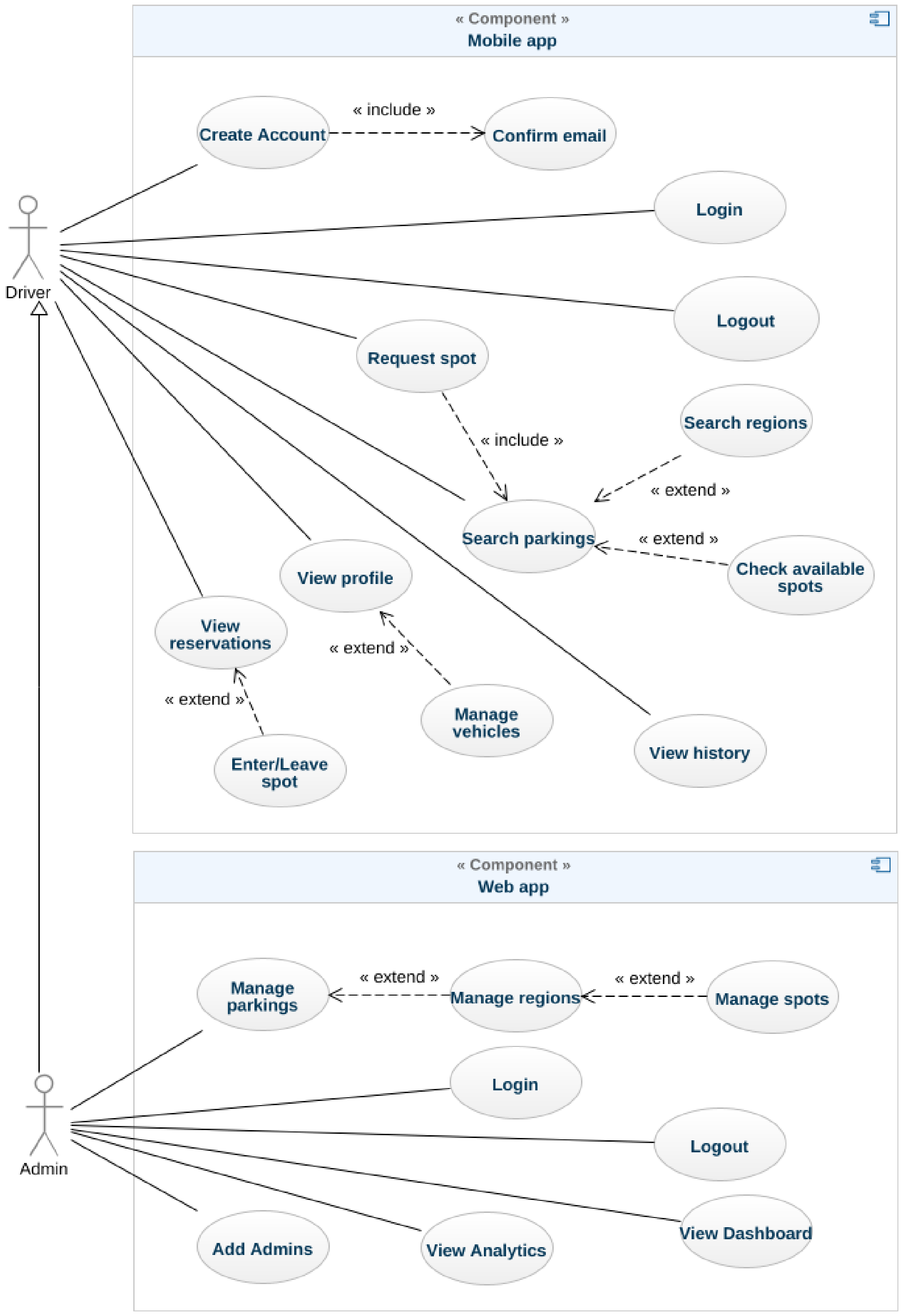


Figure 3.11: Applications' use case diagram

interaction between user and MAS, including authentication, search for available spots, request for allocation of reservations, and finally, entry and exit.

3.5.1 Register and Login

To use the application, the system must provide authentication methods for the user. Registration can be done through identity providers, phone numbers or simply by email and password.

The system assign a unique identifier to the user, which will be attached to every request. In addition, a document is created with user's personal information in the database, where all the actions performed by the same are added, such as requesting reservations or entering and leaving spots.

After this process is completed, the application must allow the user to access the system using his credentials. The registration and login process can be seen in the figure 3.12.

3.5.2 Search available spots

The application must offer the option to search for available spots in a given parking area. The search is made to the Cloud through a HTTP request. All spot agents receive this message, but only available spots will respond informing their status, containing the following essential information:

- driverId: identifier of the user who made the request.
- spotId: identifier of the available spot.
- coordinates: precise location of the spot by latitude and longitude.

Therefore, after multiple spot agents respond about their availability, the real-time database document will have several children containing information about different spots. The application receive spots' data in real-time in a distributed manner, displaying the

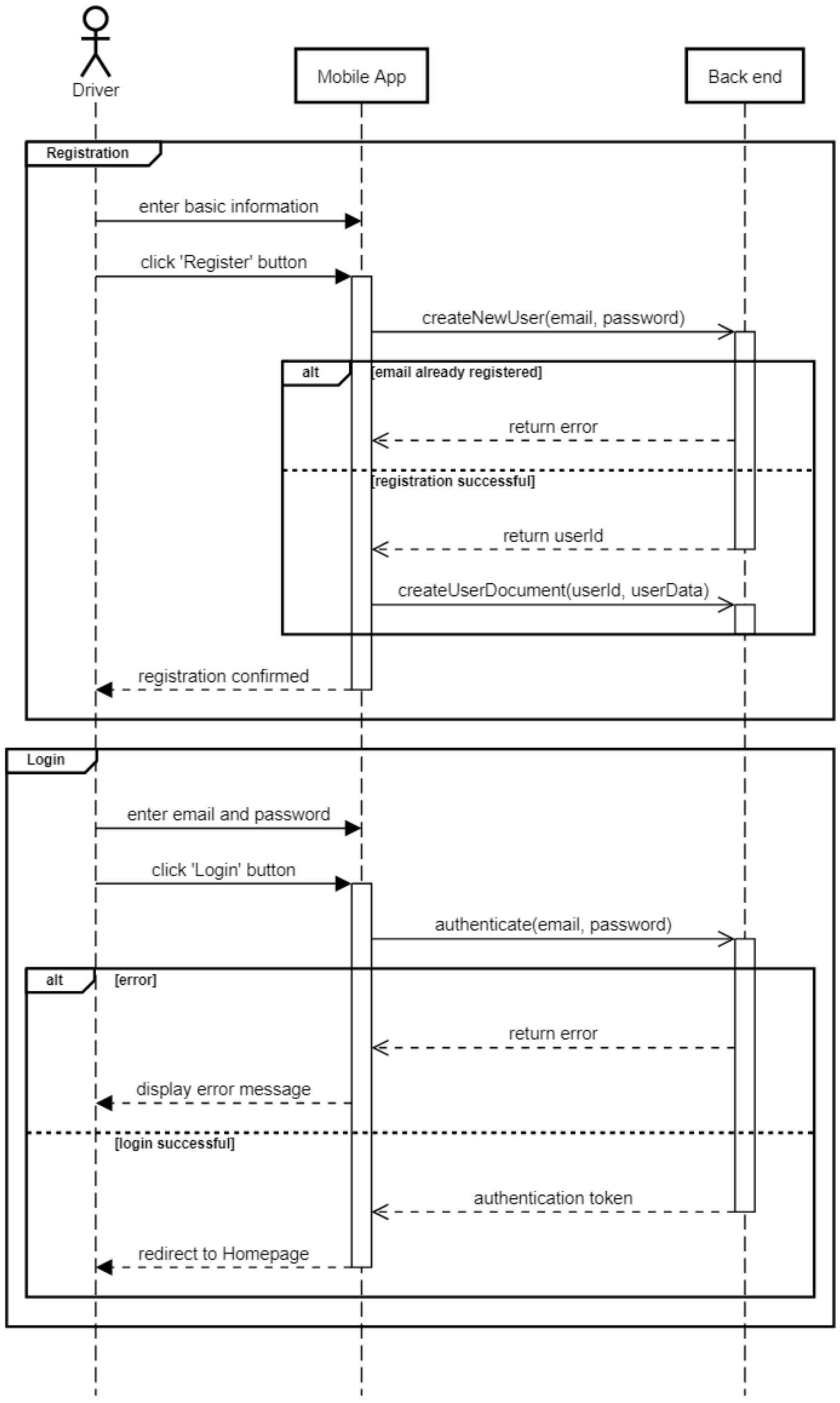


Figure 3.12: Authentication sequence diagram

identifier and coordinates of each spot available. Upon completion of this process, such information in the database must be deleted to avoid unnecessary storage of data that may not be up to date.

The complete process can be seen in the figure 3.13.

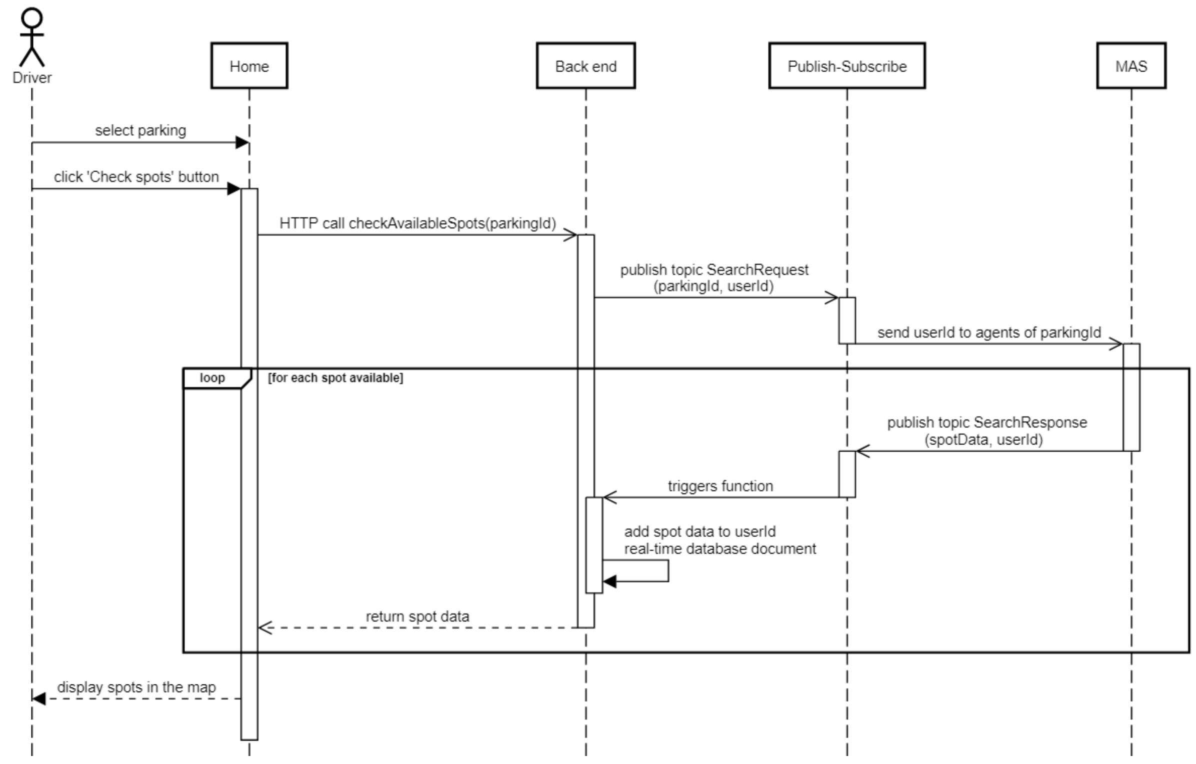

Driver
Home
Back end
Publish-Subscribe
MAS
select parking
click 'Check spots' button
HTTP call checkAvailableSpots(parkingId)
publish topic SearchRequest (parkingId, userId)
send userId to agents of parkingId
loop
[for each spot available]
publish topic SearchResponse (spotData, userId)
triggers function
add spot data to userId real-time database document
return spot data
display spots in the map

Figure 3.13: Check spots diagram

3.5.3 Request spot

It is the most important functionality of the application, in which the user can make a request for a reservation, specifying parameters that have an effect on the decision of the MAS. These parameters include the smart parking and preferred region, which can vary depending on geography, and can be related to sectors of a university or to floors in a shopping center. Inform the vehicle to be used is also important, so that the MAS assign a spot for the type of vehicle chosen. Then, the user also decide the date of the reservation, as well as the initial and end time.

Finally, the user only needs to inform which would be the ideal spot, the maximum acceptable distance from this spot's location and the maximum price that he is willing to pay. For the MAS to be able to negotiate the best spot, it is necessary that the user informs how much the chosen location and price must weight in the agents' decision. Weights are mutually exclusive on a 100 percent scale, that is, when choosing 40% weight for one of them, the other will necessarily correspond to 60%. Therefore, the request payload has the following attributes:

- driverId: identifier of the user who made the request.
- initialDate: initial timestamp of the reservation.
- endDate: end timestamp of the reservation.
- region: parking region that the user wants to park.
- spotWanted: ideal spot chosen by the user.
- distanceRange: maximum range in meters of the spot the user has chosen.
- locationWeight: weight that determines the importance of the location.
- maxPrice: maximum price that the user is willing to pay.
- priceWeight: weight that determines the importance of the price.

As soon as the application makes the request, such information is stored in the database as a document in the *Requests* subcollection, causing a function to be activated by this database writing event (3.4.2).

With the decision of the agents on which spot is assigned to the user, it is up to the assigned spot to send a message on the topic *Reservation*, informing the final reservation details.

The final details of the request must be obtained by the application through the real-time engine, leaving the user to accept or reject the proposal offered by the MAS. The received message has the following data:

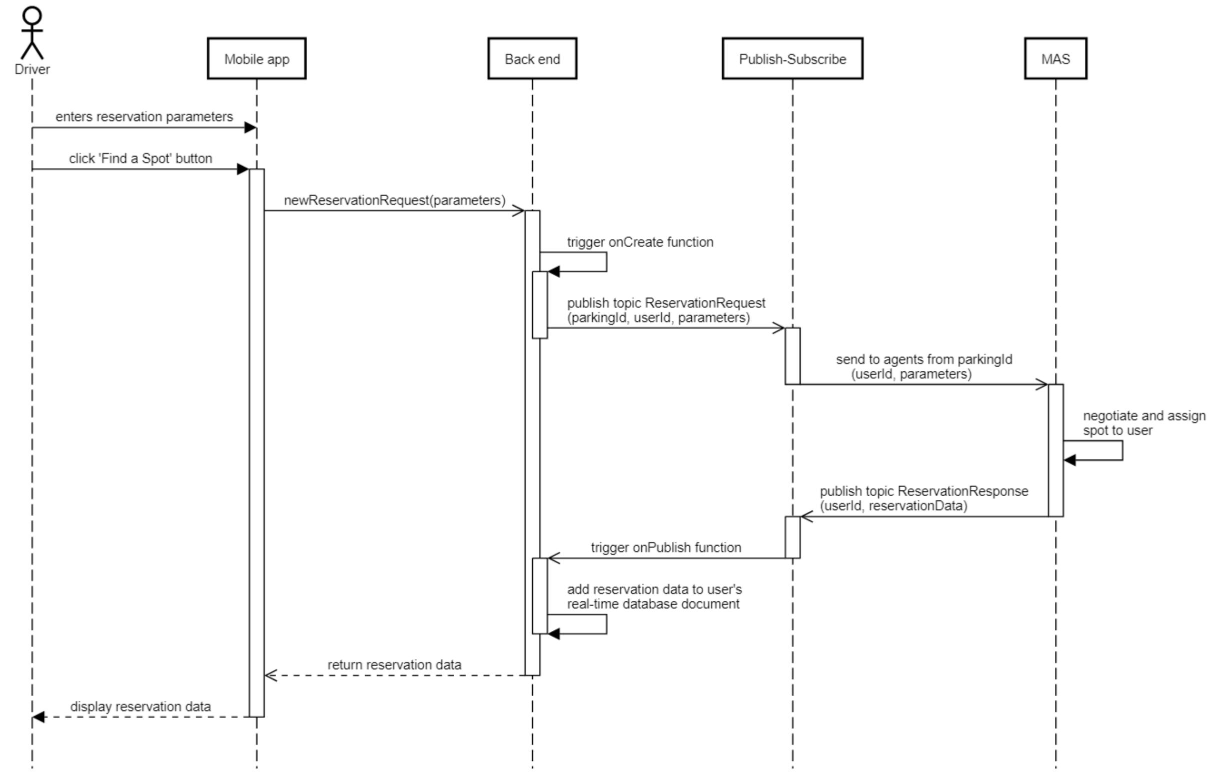

Driver
Mobile app
Back end
Publish-Subscribe
MAS
enters reservation parameters
click 'Find a Spot' button
newReservationRequest(parameters)
trigger onCreate function
publish topic ReservationRequest (parkingId, userId, parameters)
send to agents from parkingId (userId, parameters)
negotiate and assign spot to user
publish topic ReservationResponse (userId, reservationData)
trigger onPublish function
add reservation data to user's real-time database document
return reservation data
display reservation data

Figure 3.14: Request spot diagram

- driverId: identifier of the user who made the request.
- reservation: represents the success of the reservation (true/false).
- spotId: identifier of the assigned spot.
- price: final price that the user will pay.

The complete reservation request and allocation process is represented by the figure 3.14.

3.5.4 Enter and leave spot

This is the last feature related to MAS, in which the application must include the possibility for the user to request the entry or exit of his spot, at the moment of the reservation.

Therefore, the application must change the *userStatus* attribute in the reservation's document to its correspondent value, causing a function to be triggered, informing the spot agent about the desired action.

Once the reservation's duration reaches the end, the application must remove user's access to this functionality, so that he can no longer interact with it.

Chapter 4

Development and Testing

Constant integration between different modules of a smart parking requires validations in terms of project scalability, allowing new functionalities to be implemented as the application grows gradually. On a practical level, it is important that these validations occur within a collective use of technologies, which have different individual objectives but share a common environment and data among themselves.

Based on this, in order to validate the proposed architecture, it was necessary to create applications that would allow drivers and administrators to act on a real smart parking system.

Therefore, this section demonstrates technical and architectural implementation issues, such as detailing the Publish-Subscribe topics used, activation and execution of functions based on events, as well as the development of a mobile application for drivers and a web application intended for administrators.

The proposed architecture was implemented using the Google Cloud Platform (GCP) services, however, it is independent and can be implemented in any other Cloud platform.

4.1 Web system

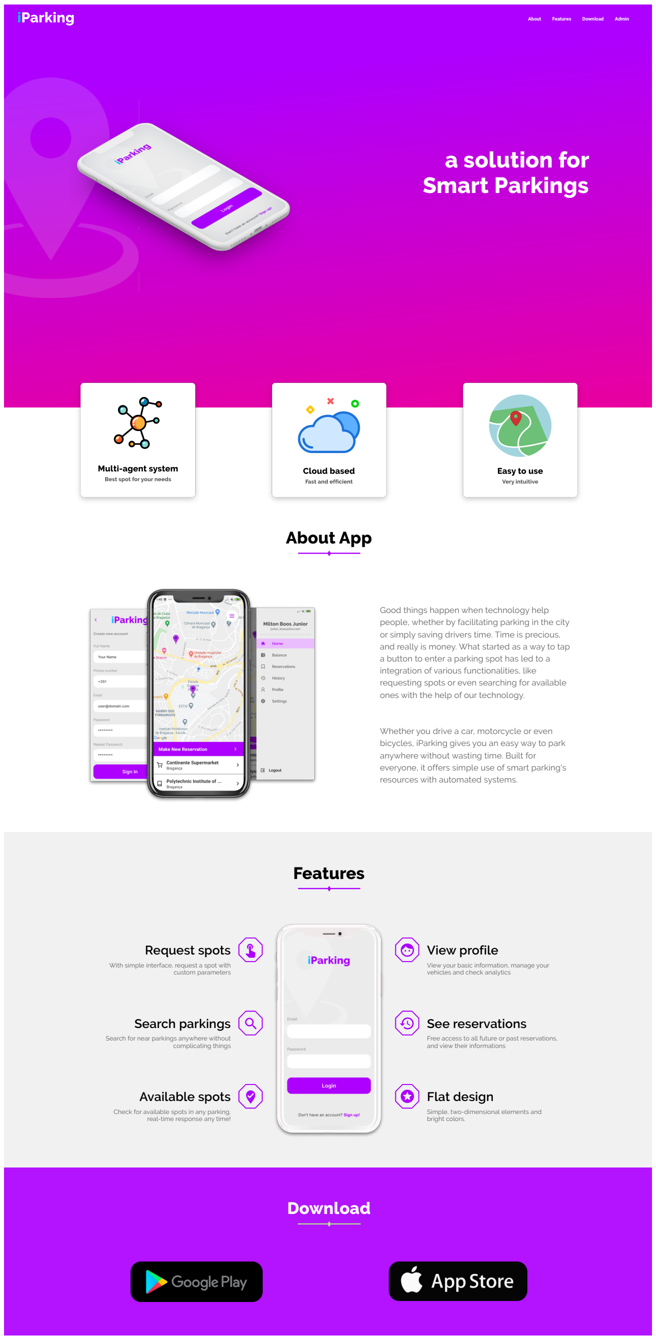
To develop a platform for managing and maintaining smart parking systems, the React.js library was used, which is capable of dynamically building single-page applications. Created by Jordan Walke and currently maintained by Facebook, it has become one of the most famous Javascript libraries for developing web applications.

Aiming to obtain, change, add or delete HTML elements, web pages use Document Object Model (DOM), an object-oriented representation of the web page model that can be modified with a scripting language such as JavaScript. Since DOM manipulation is slow, React.js makes use of a virtual DOM technique, having the same properties as a real DOM object. Every time a component is updated, it causes to modify the virtual DOM, and will result in a comparison of differences so that only the updated objects are changed in the real DOM, allowing a performance improvement when comparing to static web pages, because it reduces the unnecessary reloading of certain components.

In addition, the use of React.js promotes other benefits, one of which is to increase productivity, facilitating the reuse of components in different interfaces or even between projects, allowing the customization of these through properties. It also has an active community of millions of developers, making it easy to access various pre-made components and features.

The initial interface of the system is responsible for digital marketing of the mobile application. A broad description about the application's purpose and its various functionalities is displayed, as well as ways to redirect the user to download. Furthermore, the web system modules are divided into Dashboard, Management and Admin.

This application is focused on the administrator user, who must be able to manage different parts of the smart parking areas fast and efficiently. It is necessary for users to be authenticated in order to access the system. When requesting access, a verification taken to ensure that the user is registered in the system as shown in section 4.2.1 and, if so, he must have the administrator role as assigned in section 4.1.3.



a solution for Smart Parkings

About App

Good things happen when technology help people, whether by facilitating parking in the city or simply saving drivers time. Time is precious, and really is money. What started as a way to tap a button to enter a parking spot has led to a integration of various functionalities, like requesting spots or even searching for available ones with the help of our technology.

Whether you drive a car, motorcycle or even bicycles, iParking gives you an easy way to park anywhere without wasting time. Built for everyone, it offers simple use of smart parking's resources with automated systems.

Features

Request spots
With simple interface, request a spot with custom parameters

Search parkings
Search for near parkings anywhere without complicating things

Available spots
Check for available spots in any parking, real-time response any time!

View profile
View your basic information, manage your vehicles and check analytics

See reservations
Free access to all future or past reservations, and view their informations

Flat design
Simple, two-dimensional elements and bright colors.

Download

Figure 4.1: Landing page

4.1.1 Dashboard

In the context of a management system, a dashboard is a visual panel which presents, in a simple manner, a collection of key information regarding the state of the system and its respective current values. This information can be both indicators of the physical area and the analytical part in a smart parking environment.

Consequently, this feature helps in decision making. Dashboards can present the health of the complete system on a single screen, so that the administrator can then check in detail these different modules on other specific interfaces. Through this page, a global view of the main processes is displayed, where it can also be seen, dynamically and objectively, data related to individual processes.

With a focus on technical issues, and in the case of an infrastructure, a dashboard is used to analyze meteorological factors, and additionally to check the availability of devices that represent parking spots. As a form of business management, it is also important to provide an overview of overall performance with profit indicators and system usage. By displaying this organized data and through efficient monitoring of operations, it becomes easier to plan, implement process improvements and correct failures.

Using dashboards is an interesting way to dynamize the management of a system that integrates physical and digital elements such as smart parking. It allows access to clearer and more concise information about specific aspects, that is, data that really matters at most moments, all in a single interface. All of this updated in real-time and automatically, without the need to use professionals for manual updating.

Based on this, in the Dashboard interface of this system demonstrated by the figure 4.2, it is possible to navigate between all the parking areas in operation to check basic information related to current functioning. Each smart parking will have some data presented in the form of cards, displaying information such as the number of reservations made throughout the day, regardless of the date on which they will occur.

This information is obtained through a document in the database representing the daily record. This document also stores the total time spent in vacancies, together with

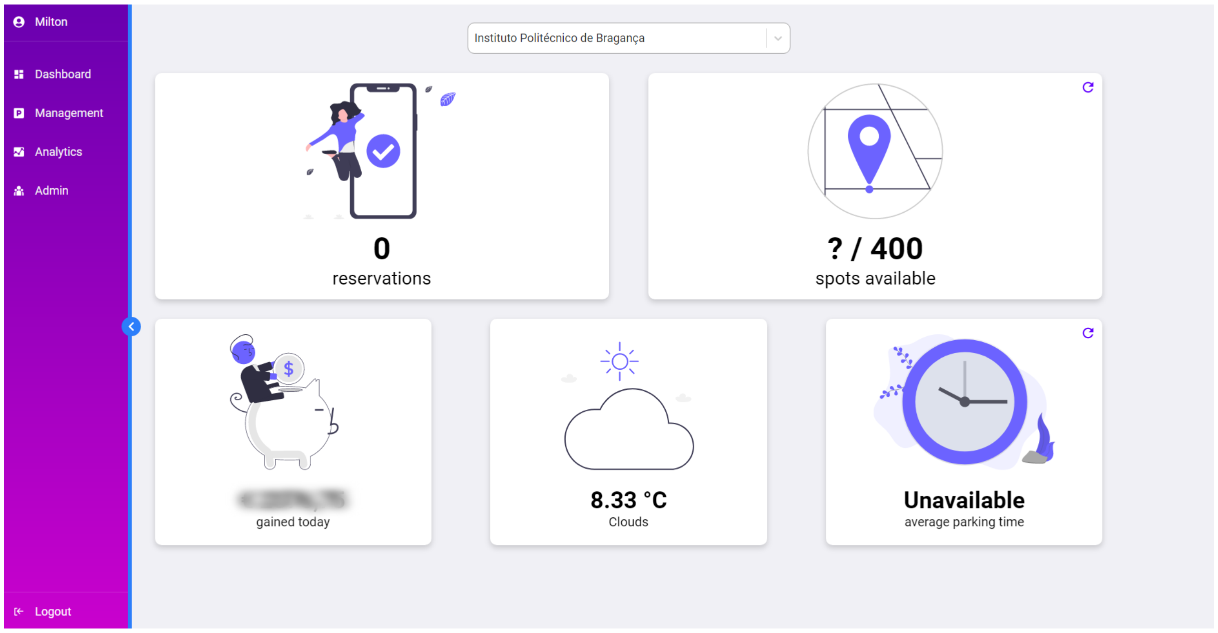


Figure 4.2: Dashboard page

the number of completed reservations, and through a HTTP request, a function in the cloud is capable of calculate the average time of these reservations and return the result for display.

Using the location, more specifically the smart parking coordinates, and through the OpenWeatherMap Application Programming Interface (API) [63], a card is displayed showing the main weather information. This data is important because it can affect the use of smart parking in specific cases such as rain and snow, where many users can choose not to use bicycles, consequently causing an increase in the number of cars parked.

A feature for checking the availability of spots in this dashboard is also present for users of the mobile application 3.5.2. However, in this system only the number of available spots is displayed, not taking into account their data and exact locations.

Finally, in a system that has a cash flow, it is also important to show the gains for the day so that a future analysis is possible along with other data regarding the operation of the smart parking. Therefore, a card with this objective is displayed to the user as part of a possible integration with a financial module.

4.1.2 Management

For any system, a module is required to manage the data that will be used by the application as a whole, so that the correct handling of the data is simple and intuitive. With this module, it is possible to perform Create, Read, Update and Delete (CRUD) operations in the context of smart parking, having an immediate impact on the use of the application as a whole.

Effective data management is the leading step in creating a scalable architecture, because without a structure that allows access to data in an organized way, it is not possible to obtain an adequate management experience as the database grows. Not to mention that companies that properly manage organizational data have witnessed a 12% increase in revenue, meaning a strong relationship between data management and financial return [64].

On this page, the administrator will have the ability to view information about smart parking places and manage them, using functionalities to create, update or remove new parking areas, including their regions and spots.

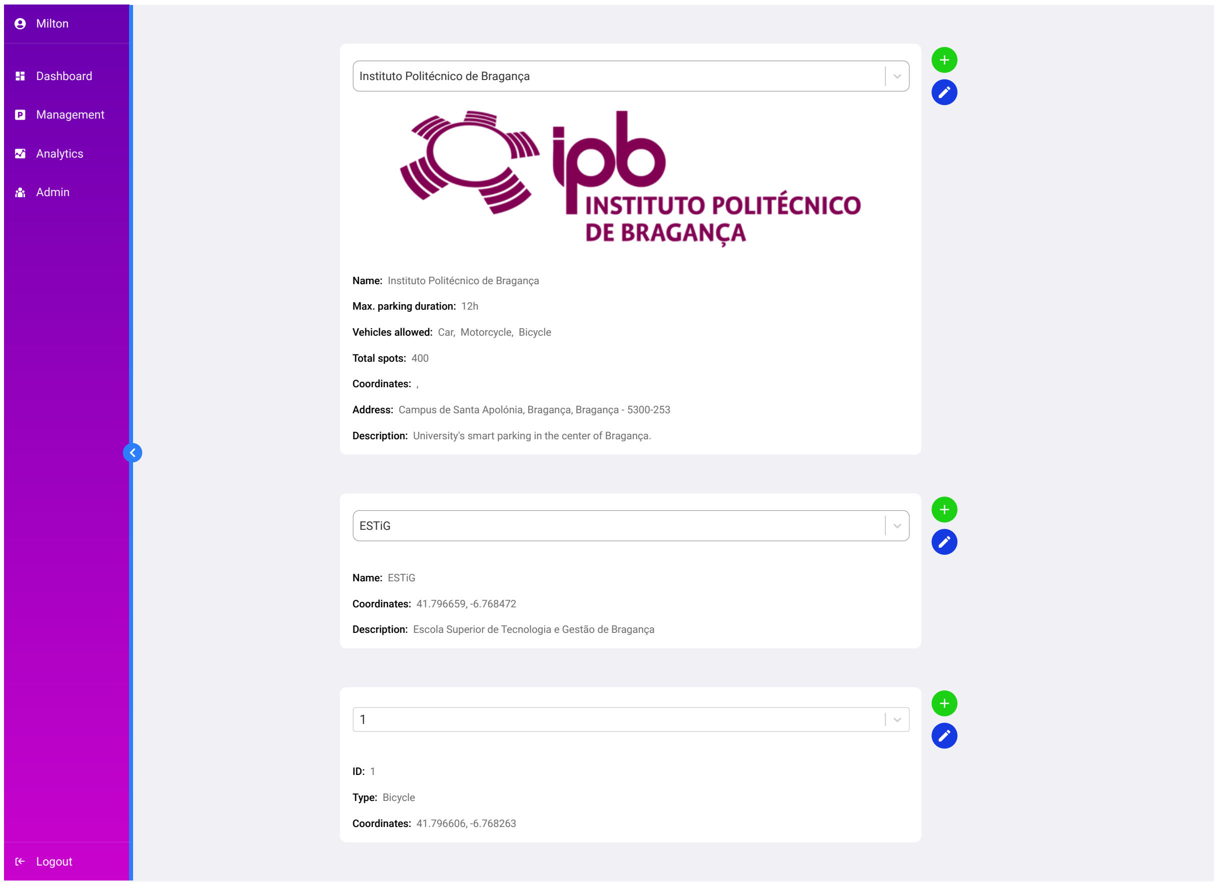
As a result, it is possible to view this data as shown in the figure 4.3.

4.1.3 Admin

To receive the admin role, it is necessary that the user is already registered in the system. So, upon the entry of the desired user's email, the function responsible for assigning this position will be activated through a HTTP request. In case of success, the user will receive a custom claim with the administrator token, making it possible to access the management system.

4.2 Mobile app

There has been a growing preference recently for frameworks that allow applications to be built for multiple platforms. These frameworks reduce the need for programming the


Milton
Dashboard
Management
Analytics
Admin
Logout
Instituto Politécnico de Bragança
ipb INSTITUTO POLITÉCNICO DE BRAGANÇA
ESTiG
1

Name: Instituto Politécnico de Bragança
Max. parking duration: 12h
Vehicles allowed: Car, Motorcycle, Bicycle
Total spots: 400
Coordinates: ,
Address: Campus de Santa Apolónia, Bragança, Bragança - 5300-253
Description: University's smart parking in the center of Bragança.

Name: ESTiG
Coordinates: 41.796659, -6.768472
Description: Escola Superior de Tecnologia e Gestão de Bragança

ID: 1
Type: Bicycle
Coordinates: 41.796606, -6.768263

Figure 4.3: Management page

same features in different languages, such as Java for Android and Swift for iOS, using web languages instead, such as Javascript.

Following the release of React.js and with the growing increase in the use and development of mobile technologies, in 2015 Facebook decided to launch the React Native framework, capable of creating mobile applications using the React declarative User Interface (UI) as a base. With this framework, it is possible to have access to device's native functionalities, such as location and camera, as well as offering a faster, more reliable and responsive experience to users when compared to web applications.

In 2018, React Native had the second highest number of contributors for any repository in GitHub [65], and today, is supported by developers and companies around the world. One of this companies is Expo, which essentially provides a set of tools and services built around React Native and native platforms that help to develop, build and deploy

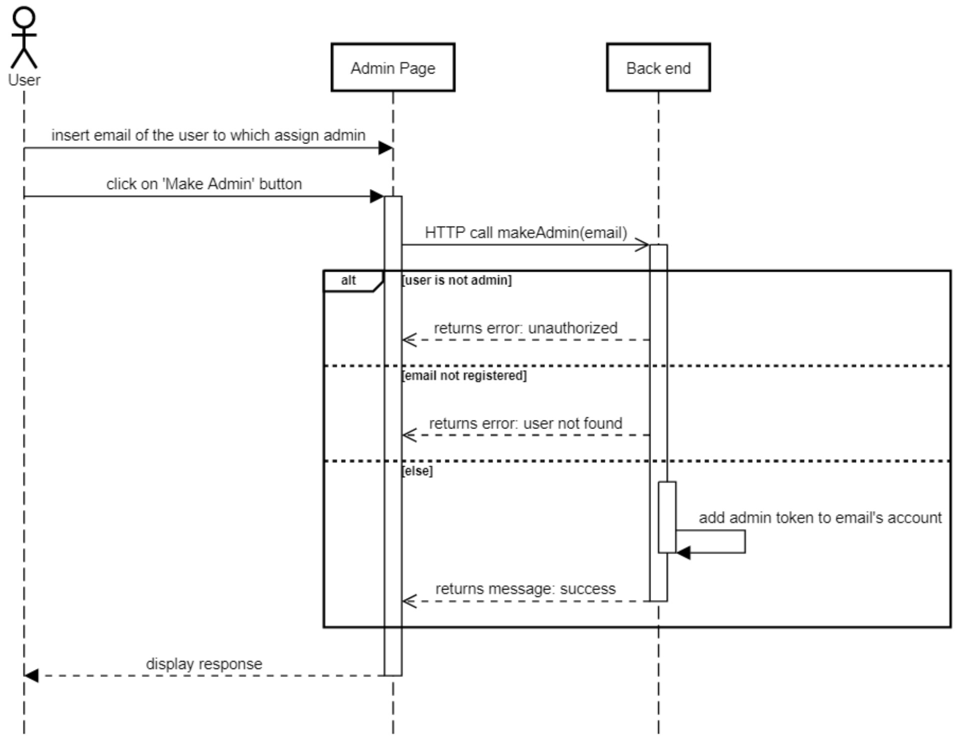

User
Admin Page
Back end
insert email of the user to which assign admin
click on 'Make Admin' button
HTTP call makeAdmin(email)
alt
[user is not admin]
returns error: unauthorized
[email not registered]
returns error: user not found
[else]
add admin token to email's account
returns message: success
display response

Figure 4.4: Admin attribution sequence diagram

applications.

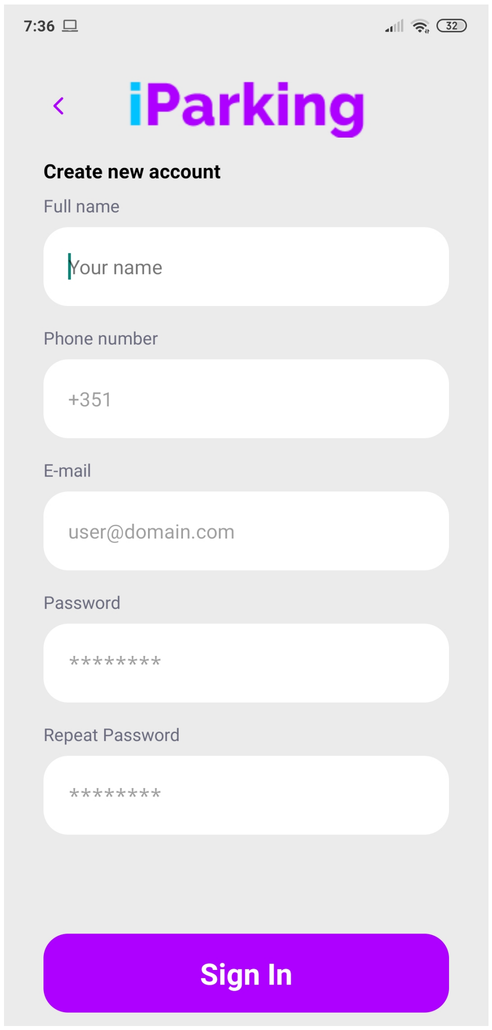
Therefore, the React Native framework was used with the help of Expo tools to build a mobile application called iParking, allowing it to be natively compiled on both Android and iOS platforms. All development took place without the use of pre-produced templates or stylizations, and was based on prototyped interfaces using the Figma tool [66].

This application is focused on the driver user, who must be able to use the functionalities provided by smart parking in an intuitive and effective way.

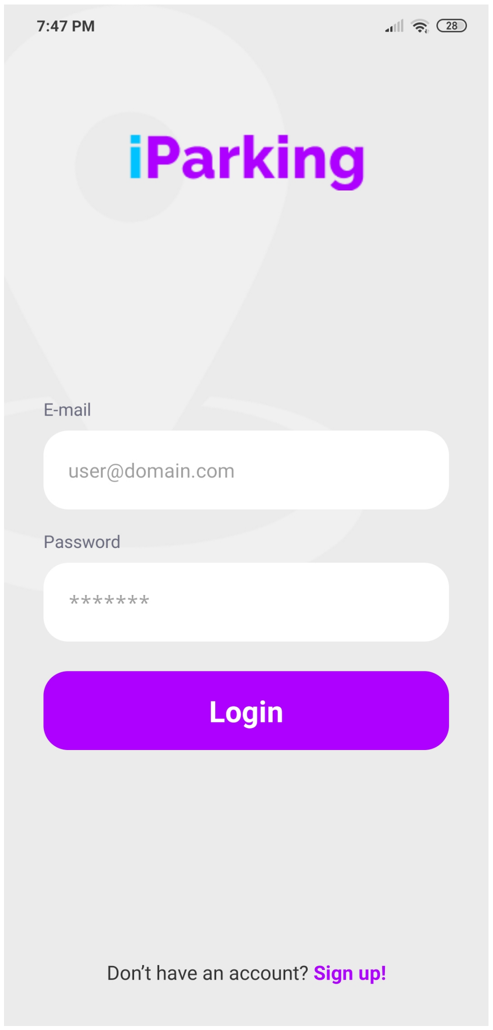
4.2.1 Register and Login

First, to use the program, it is necessary to register. As shown in the figure 4.5a, it is necessary for the user to enter their personal data such as full name, phone number, and also email and password to use as access credentials. After this step, it is necessary for the user to confirm the registration in his email. The user is able to access the application using previously registered email and password after such a procedure has been completed

4.5b.



(a) Register



(b) Login

Figure 4.5: Authentication page

In this application the Firebase authentication module was used, but other alternatives include the use of the AWS Cognito module, Backendless, Auth0 for pure authentication management, or even its own implementation using methodologies with JSON Web Token (JWT).

4.2.2 Home page

After authentication, for reasons of performance and scalability, the application makes a request to obtain the parking areas that are at a distance of 10 kilometers from the user. Therefore, selective loading of only data close to the locations occurs, keeping applications light, responsive and without making too much use of the user's bandwidth, even with a large number of smart parking places registered in the system. For this purpose, the Geofire library was used, capable of performing queries in the database based on a specific location.

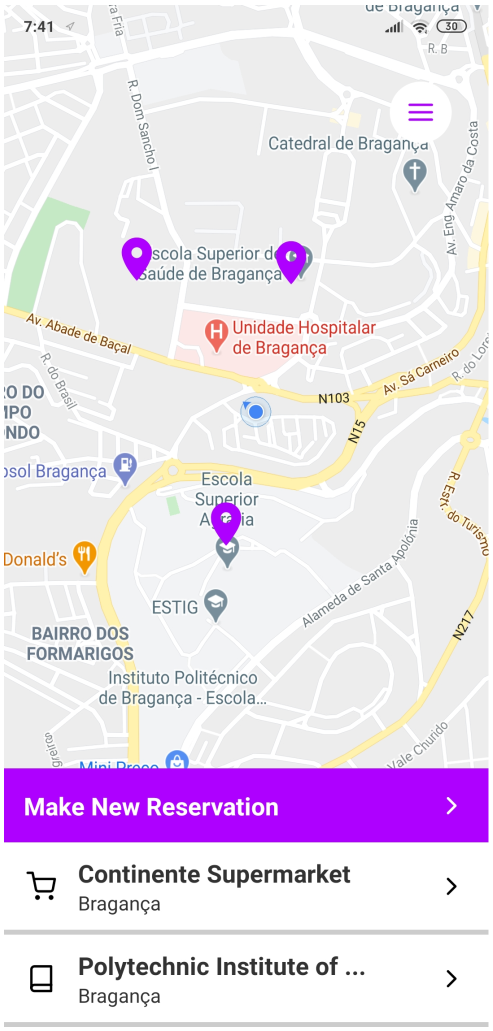
With this information, it is possible to view a map of the region where the user is located along with markers for each smart parking nearby 4.6a. Thus, the user can choose where to park, as well as obtain more information about each parking by clicking on the markers.

As shown in the figure 4.6b, on all main pages there is also a side menu containing basic user information, links allowing navigation between different pages of the application, as well as the option to log out of the system.

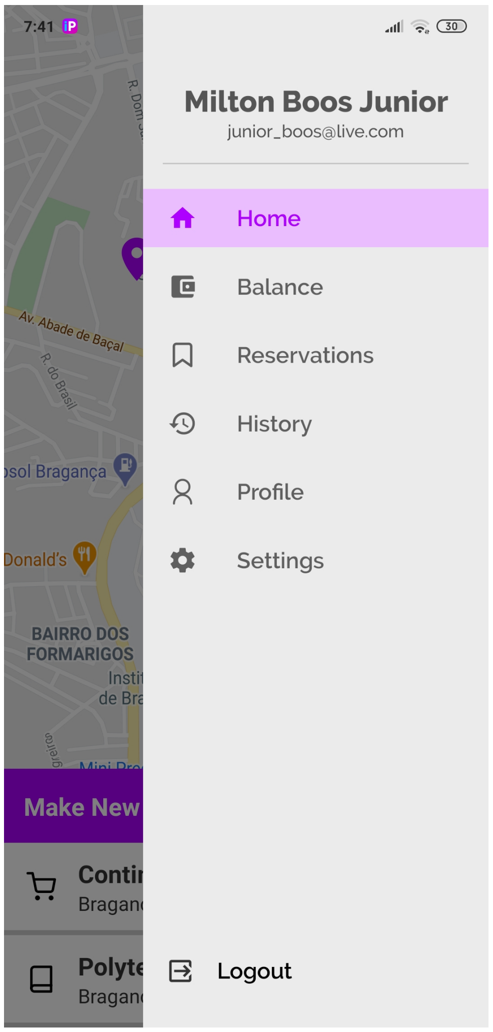
When clicking on the markers at the home page, a card is shown with the name and an image representing the parking 4.7a. The *Check spots* button aims to request by HTTP and show to the user which spots are currently available 3.5.2.

4.2.3 Profile

On the profile page, the user finds information on the use of smart parking services, such as the number of reservations made and the total time spent using the spots. Vehicle registration is necessary to request reservations, so that the system knows what type of parking spot it must be assigned to the user. A section for vehicle management is present on this page, along with basic user information.


7:41
Catedral de Bragança
Escola Superior de Saúde de Bragança
Unidade Hospitalar de Bragança
Escola Superior Agrária
ESTIG
BAIRRO DOS FORMARIGOS
Instituto Politécnico de Bragança - Escola...
Make New Reservation
Continente Supermarket
Bragança
Polytechnic Institute of ...
Bragança

(a) Home



(b) Sidemenu

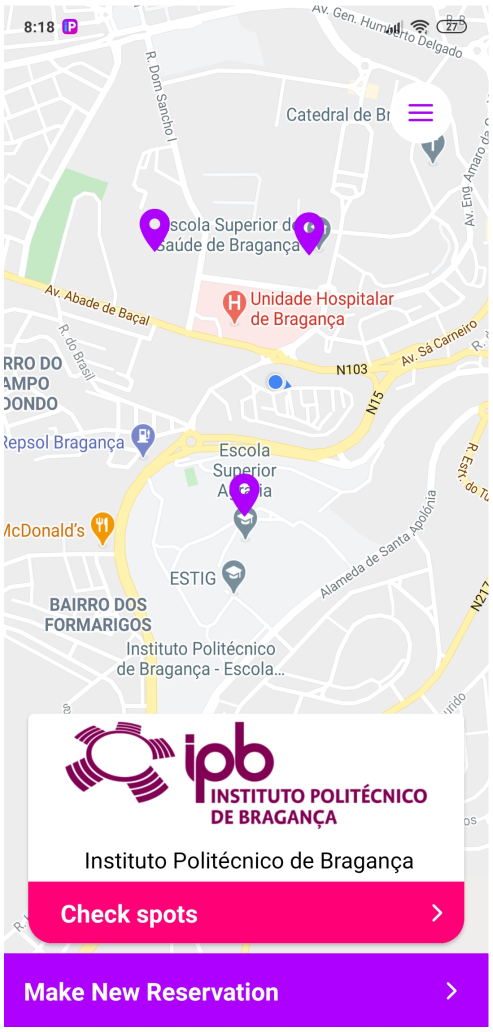
Figure 4.6: Home page

4.2.4 Request spot

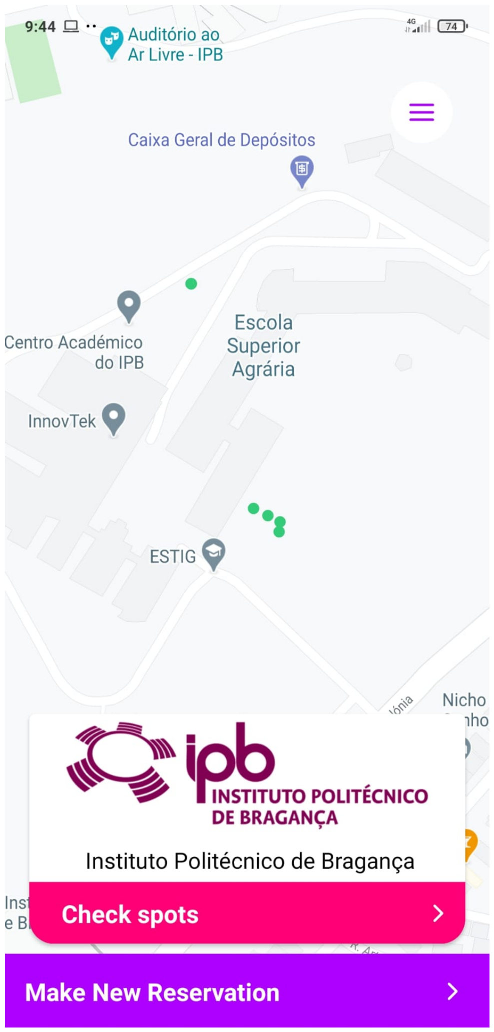
To make a reservation, the user must navigate to this page. A form is presented to request the allocation of a reservation according to the indicated parameters. The reservation is sent to MAS, where the negotiation and decision between agents takes place.

This page makes requests for regions to the database each time the smart parking is changed by the user. Upon entering the page, a request is also made to load the vehicles and their specific types.

Therefore, after a response from the MAS about the reservation, a card is displayed



(a) Parking card

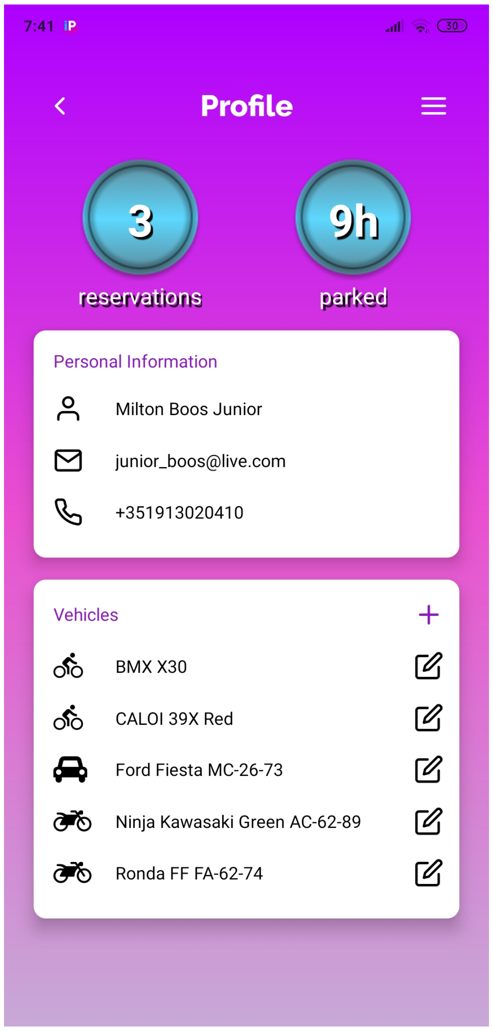


(b) Spots available in green

Figure 4.7: Check available spots page

informing the proposal, the user being able to accept or refuse it. This response is converted by the application into a change to the *status* attribute in the request document present in the database.

In order for the user to be alerted about the reservation, a local notification system has been implemented. At the end of this process, the application schedules notifications based on the 30 minutes prior to the start and end of the reservation. As these are local notifications, the user will be alerted even if the application do not have access to the internet.


7:41

Profile

3
reservations

9h
parked

Personal Information

Milton Boos Junior

junior_boos@live.com

+351913020410

Vehicles

BMX X30

CALOI 39X Red

Ford Fiesta MC-26-73

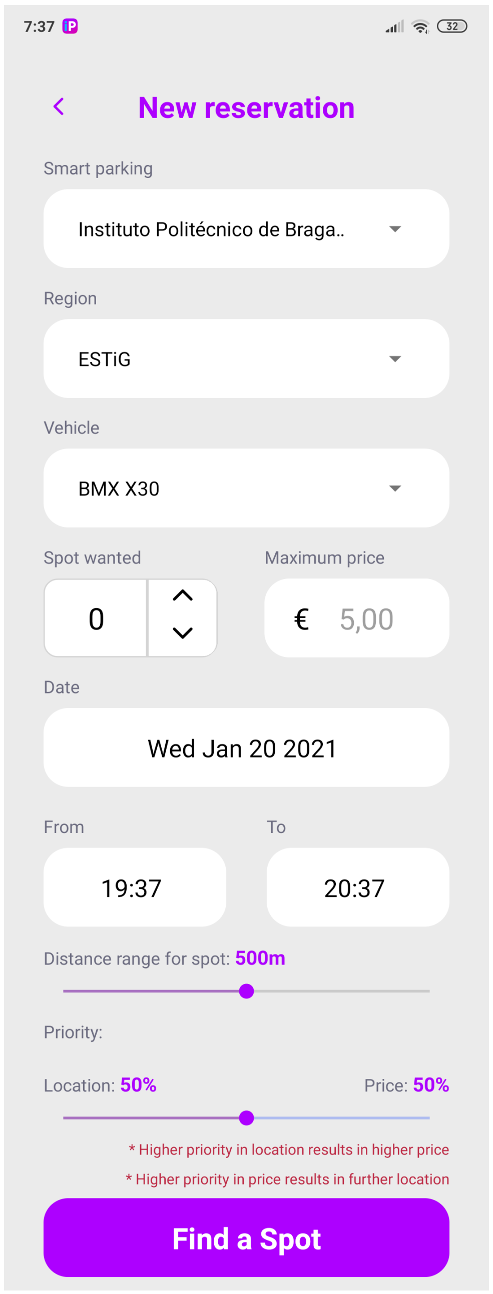
Ninja Kawasaki Green AC-62-89

Ronda FF FA-62-74

Figure 4.8: Profile page

4.2.5 Reservations and History

If the user accepts the reservation by agreeing on final details, its *status* attribute change to *Accepted* in the database, and can be checked on the Reservations page. As shown in the figure 4.10a, it is possible to observe more information about the reservation, as well as open Global Positioning System (GPS) applications with routes directly to the spot's location. When the current time is between the beginning and end of the reservation, the action of entering the spot is made available 4.10b.


7:37

New reservation

Smart parking

Instituto Politécnico de Braga..

Region

ESTiG

Vehicle

BMX X30

Spot wanted

0

Maximum price

€ 5,00

Date

Wed Jan 20 2021

From

19:37

To

20:37

Distance range for spot: 500m

Priority:

Location: 50%

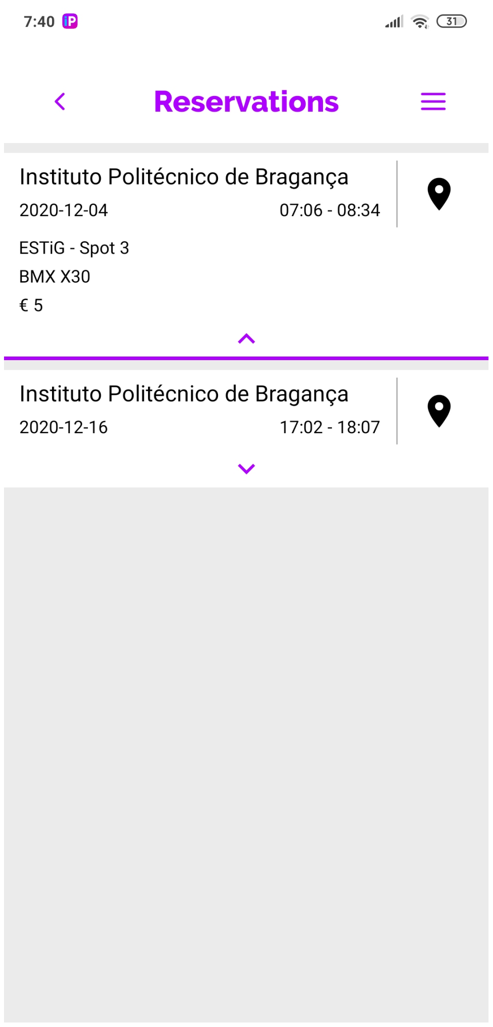
Price: 50%

* Higher priority in location results in higher price

* Higher priority in price results in further location

Find a Spot

Figure 4.9: Request reservation form

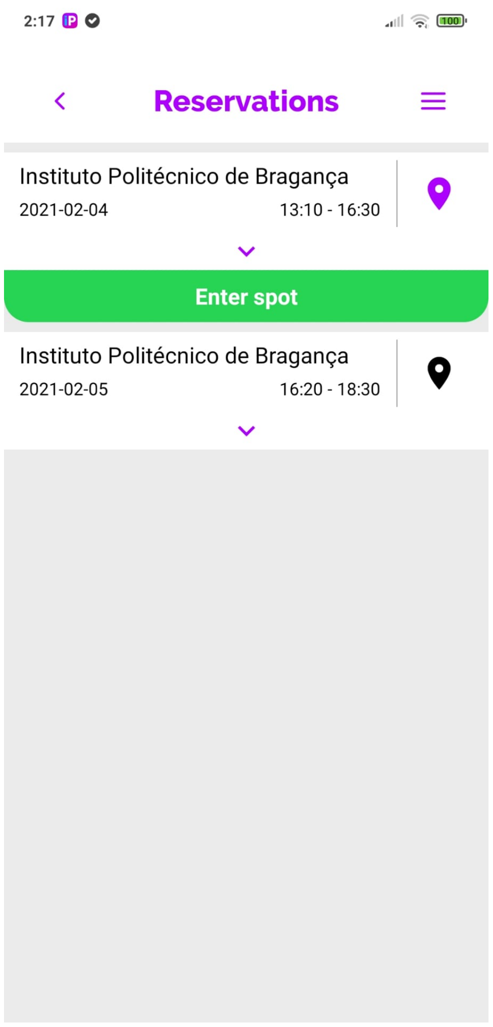

7:40

Reservations

Instituto Politécnico de Bragança
2020-12-04 07:06 - 08:34
ESTiG - Spot 3
BMX X30
€ 5

Instituto Politécnico de Bragança
2020-12-16 17:02 - 18:07

(a) Future reservations


2:17

Reservations

Instituto Politécnico de Bragança
2021-02-04 13:10 - 16:30

Instituto Politécnico de Bragança
2021-02-05 16:20 - 18:30

(b) Current reservation

Figure 4.10: Reservations page

Chapter 5

Tests and Results

This chapter presents tests in a case study, involving the integration between different components of the system. The tests contained address end-to-end communication, from client applications, through Cloud modules, and finally with the MAS.

Tests were carried out using the IPB smart parking as a reference, calculating response time and communication effectiveness. In this case, the three functionalities referring to MAS have been tested multiple times, i.e., the search for available spots, the request for reservations, and finally, entry or exit of spots.

For analysis, 70 sequential tests were carried out in different cases to check the consistency of communication between the parties. There was no data loss in any of the tests, and in all of them did the information from the mobile application arrive in the multi-agent system, and returned the expected result. This is due to the QoS guaranteed by the Publish-Subscribe service, which uses delivery recognition methods for each subscriber.

The tables 5.1, 5.2 and 5.3 show the total execution time of these functions, as well as the final average time, with the number of available spots being 10, 100 and 200 respectively. It is important to consider that the time may vary according to the latency of the mobile device connection and also according to the response time of the MAS.

Note that the average time for the user to receive the final information is relatively low, considering the entire request process by the application and response from the MAS. According to the 5.4 table, the process of entering and leaving the spot is shown to be

Test	Search available spots	Request reservation
1	3872 ms	3513 ms
2	5464 ms	4425 ms
3	1258 ms	3450 ms
4	1200 ms	2755 ms
5	1101 ms	3659 ms
6	2788 ms	3207 ms
7	1887 ms	4028 ms
8	3261 ms	3864 ms
9	5152 ms	3779 ms
10	1041 ms	4046 ms
Average	2702 ms	3672 ms

Table 5.1: Execution time with 10 spots available

Test	Search available spots	Request reservation
1	9428 ms	3107 ms
2	4358 ms	3561 ms
3	4954 ms	3256 ms
4	5047 ms	2970 ms
5	8400 ms	3355 ms
6	6959 ms	3048 ms
7	6866 ms	3298 ms
8	5505 ms	2402 ms
9	4737 ms	1933 ms
10	5194 ms	3006 ms
Average	6144 ms	3595 ms

Table 5.2: Execution time with 100 spots available

Test	Search available spots	Request reservation
1	9835 ms	8785 ms
2	7029 ms	4218 ms
3	5938 ms	3907 ms
4	7358 ms	2444 ms
5	12713 ms	2076 ms
6	6804 ms	2151 ms
7	16032 ms	2937 ms
8	5005 ms	4561 ms
9	5019 ms	3067 ms
10	5281 ms	2241 ms
Average	8715 ms	3638 ms

Table 5.3: Execution time with 200 spots available

Test	Enter / Leave spot
1	316 ms
2	312 ms
3	44 ms
4	58 ms
5	120 ms
6	204 ms
7	64 ms
8	50 ms
9	126 ms
10	44 ms
Average	133 ms

Table 5.4: Execution time to enter and exit the spot

the fastest, as it is a one-way communication, and above all, as its execution time it is not related to the number of spots available.

The request for available spots was the second fastest in the case of 10 spots, but it increased according to the number of available spots, due to the large amount of information being sent by the real-time engine. Finally, the request for a reservation was shown to be constant, having a common average time between the 3 cases.

In general, the application's communication with the backend module was fast, proving efficiency in cases with local and mobile network connection, and flexible in case of connection loss. The platform used proved to be a great option for developing applications that require authentication, a database and a real-time engine.

In this case [60], the database supports up to 10,000 recordings per second, while the real-time engine can maintain 200,000 simultaneous connections. Event-based functions can be invoked 3,000 times simultaneously, and shown to be sufficient according to the execution times presented.

For comparison, other tests could be done integrating different multi-agent systems. Based on these tests and the limits imposed by Cloud providers, it is possible to verify the scalability and flexibility of the architecture.

Chapter 6

Conclusion and Future Work

To develop the architecture, several approaches with regard to the integration of different modules were analyzed, focusing on scalability, security and decentralization of systems. Therefore, the present work proposed and developed an architecture for smart parking systems using Cloud computing technologies. Methodologies for dynamically storing information in the database were approached, enabling the addition of new features, as well as facilitating adaptation to different smart parking systems.

The communication between MAS and the Cloud was carried out through a Publish-Subscribe service, which provided a simple integration between system logic and physical components. Event-based functions were used in order to provide asynchronous and stable communication between various components of the Cloud. Finally, web and mobile applications were developed to validate the user's interactivity with the system, with access to the different functionalities available.

The mobile application for drivers was tested in different case studies, using execution times to compare results. These were shown to be promising in the issues addressed, mainly in the successful communication between user and MAS, as well as in the scalability of the project. The architecture in general proved to be flexible, capable of incorporating new smart parking modules, as well as integrating different MAS.

This work shows that the use of platforms acting as PaaS provide a complete environment for implementing IoT systems. These platforms deliver computational services, abstracting technical security issues, guaranteeing end-to-end communication, QoS, facilitating long-term application scalability, and in addition, provide a pay-per-use methodology that reduces unnecessary expenses.

Future work may address different methodologies in the communication flow between the user and the Cloud, changing event-based functions for direct activation through HTTP requests. In this case, it is important that the system has ways of controlling the user's actions to avoid overloading in cases of multiple requests with duplicate data. Furthermore, as it is an architecture with easy integration, a financial module for payments using credit or debit cards becomes interesting, enabling a practical implementation of the smart parking. Regarding the integration with the MAS, it is interesting to explore the addition of features that allow agent management through a web interface.

Bibliography

[1] *World urbanization prospects: The 2018 revision (st/esa/ser.a/420)*, eng, New York: United Nations, 2019.

[2] B. Johnson, "Cities, systems of innovation and economic development", *Innovation*, vol. 10, no. 2-3, pp. 146–155, 2008. DOI: 10.5172/impp.453.10.2-3.146. eprint: https://doi.org/10.5172/impp.453.10.2-3.146. [Online]. Available: https://doi.org/10.5172/impp.453.10.2-3.146.

[3] *Ibm global parking survey: Drivers share worldwide parking woes*, eng, https://www-03.ibm.com/press/us/en/pressrelease/35515.wss, Accessed: 20/11/2020, 2011.

[4] G. Cookson and B. Pishue, *The impact of parking pain in the us, uk and germany*, https://www2.inrix.com/research-parking-2017, Accessed: 04/01/2020, 2017.

[5] A. R. Goetz, "Transport challenges in rapidly growing cities: Is there a magic bullet?", *Transport Reviews*, vol. 39, no. 6, pp. 701–705, 2019. DOI: 10.1080/01441647.2019.1654201.

[6] L. Sakurada, J. Barbosa, P. Leitão, G. Alves, A. P. Borges, and P. Botelho, "Development of agent-based cps for smart parking systems", in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2019, pp. 2964–2969. DOI: 10.1109/IECON.2019.8926653.

[7] Faheem, S. Mahmud, G. Khan, M. Rahman, and H. Zafar, "A survey of intelligent car parking system", *Journal of Applied Research and Technology*, vol. 11,

no. 5, pp. 714–726, 2013, ISSN: 1665-6423. DOI: https://doi.org/10.1016/S1665-6423(13)71580-3. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1665642313715803.

[8] G. Yan, W. Yang, D. B. Rawat, and S. Olariu, "Smartparking: A secure and intelligent parking system", *IEEE Intelligent Transportation Systems Magazine*, vol. 3, no. 1, pp. 18–30, 2011. DOI: 10.1109/MITS.2011.940473.

[9] T. N. Pham, M. Tsai, D. B. Nguyen, C. Dow, and D. Deng, "A cloud-based smart-parking system based on internet-of-things technologies", *IEEE Access*, vol. 3, pp. 1581–1591, 2015. DOI: 10.1109/ACCESS.2015.2477299.

[10] Y. Geng and C. G. Cassandras, "A new "smart parking" system infrastructure and implementation", *Procedia - Social and Behavioral Sciences*, vol. 54, pp. 1278–1287, 2012, Proceedings of EWGT2012 - 15th Meeting of the EURO Working Group on Transportation, September 2012, Paris, ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2012.09.842. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042812043042.

[11] A. Khanna and R. Anand, "Iot based smart parking system", in *2016 International Conference on Internet of Things and Applications (IOTA)*, 2016, pp. 266–270. DOI: 10.1109/IOTA.2016.7562735.

[12] Hongwei Wang and Wenbo He, "A reservation-based smart parking system", in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011, pp. 690–695. DOI: 10.1109/INFCOMW.2011.5928901.

[13] Z. Pala and N. Inanc, "Smart parking applications using rfid technology", in *2007 1st Annual RFID Eurasia*, 2007, pp. 1–3. DOI: 10.1109/RFIDEURASIA.2007.4368108.

[14] S.-Y. Chou, S.-W. Lin, and C.-C. Li, "Dynamic parking negotiation and guidance using an agent-based platform", *Expert Systems with Applications*, vol. 35, no. 3, pp. 805–817, 2008, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.

2007.07.042. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740700293X.

[15] A. Muñoz and J. Botía, "Developing an intelligent parking management application based on multi-agent systems and semantic web technologies", vol. 6076, Jun. 2010, pp. 64–72. DOI: 10.1007/978-3-642-13769-3_8.

[16] H. Dia, "An agent-based approach to modelling driver route choice behaviour under the influence of real-time information", *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 5, pp. 331–349, 2002, ISSN: 0968-090X. DOI: https://doi.org/10.1016/S0968-090X(02)00025-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X02000256.

[17] M. Wooldridge, "An introduction to multiagent systems", in. Jun. 2002, pp. -348, ISBN: 047149691X.

[18] P. Leitão and S. Karnouskos, *Industrial Agents: Emerging Applications of Software Agents in Industry.* Mar. 2015, ISBN: 9780128003411.

[19] Chieh-Chang Li, Shuo-Yan Chou, and Shih-Wei Lin, "An agent-based platform for drivers and car parks negotiation", in *IEEE International Conference on Networking, Sensing and Control, 2004*, vol. 2, 2004, 1038–1043 Vol.2. DOI: 10.1109/ICNSC.2004.1297090.

[20] Wang Longfei, Chen Hong, and Li Yang, "Integrating mobile agent with multi-agent system for intelligent parking negotiation and guidance", in *2009 4th IEEE Conference on Industrial Electronics and Applications*, 2009, pp. 1704–1707. DOI: 10.1109/ICIEA.2009.5138486.

[21] L. Souza de Castro, G. Vaz Alves, and A. Borges, "Utilização de grau de confiança entre agentes para alocação de vagas em um smart parking", May 2016. DOI: 10.13140/RG.2.1.3452.1200.

[22] B. Hayes, "Cloud computing", *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008, ISSN: 0001-0782. DOI: `10.1145/1364782.1364786`. [Online]. Available: `https://doi.org/10.1145/1364782.1364786`.

[23] D. Bharti and R. Goudar, "Cloud computing–research issues, challenges, architecture, platforms and applications: A survey", Dec. 2012. DOI: `10.7763/IJFCC.2012.V1.95`.

[24] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, J. Tao, and C. Fu, "Cloud computing: A perspective study.", *New Gener. Comput.*, vol. 28, no. 2, pp. 137–146, 2010. [Online]. Available: `http://dblp.uni-trier.de/db/journals/ngc/ngc28.html#WangLYHKTF10`.

[25] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing", *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010, ISSN: 0001-0782. DOI: `10.1145/1721654.1721672`. [Online]. Available: `https://doi.org/10.1145/1721654.1721672`.

[26] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges", in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33. DOI: `10.1109/AINA.2010.187`.

[27] S. T. Siddiqui, S. Alam, Z. A. Khan, and A. Gupta, "Cloud-based e-learning: Using cloud computing platform for an effective e-learning", in *Smart Innovations in Communication and Computational Sciences*, S. Tiwari, M. C. Trivedi, K. K. Mishra, A. K. Misra, and K. K. Kumar, Eds., Singapore: Springer Singapore, 2019, pp. 335–346.

[28] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology", in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 631–636. DOI: `10.1109/CCGRID.2010.72`.

[29] P. D. Kaur and I. Chana, "Cloud based intelligent system for delivering health care as a service", *Computer Methods and Programs in Biomedicine*, vol. 113, no. 1, pp. 346–359, 2014, ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2013.09.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169260713003209.

[30] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles", in *2017 First IEEE International Conference on Robotic Computing (IRC)*, 2017, pp. 36–43. DOI: 10.1109/IRC.2017.77.

[31] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, "Security and privacy for cloud-based iot: Challenges", *IEEE Communications Magazine*, vol. 55, no. 1, pp. 26–33, 2017. DOI: 10.1109/MCOM.2017.1600363CM.

[32] K. Salah, J. M. Alcaraz Calero, S. Zeadally, S. Al-Mulla, and M. Alzaabi, "Using cloud computing to implement a security overlay network", *IEEE Security Privacy*, vol. 11, no. 1, pp. 44–53, 2013. DOI: 10.1109/MSP.2012.88.

[33] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey", *Journal of Network and Computer Applications*, vol. 79, pp. 88–115, 2017, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.11.027. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516302983.

[34] G. Fylaktopoulos, G. Goumas, M. Skolarikis, A. Sotiropoulos, and I. Maglogiannis, "An overview of platforms for cloud based development", *SpringerPlus*, vol. 5, Sep. 2015. DOI: 10.1186/s40064-016-1688-5.

[35] N. Khan, N. Ahmad, M. Mat Deris, and E. Ismail, "Cloud computing: Comparison of various features.", vol. 194, Jan. 2011, pp. 243–254, ISBN: 978-3-642-22602-1. DOI: 10.1007/978-3-642-22603-8_22.

[36] C. Kotas, T. Naughton, and N. Imam, "A comparison of amazon web services and microsoft azure cloud platforms for high performance computing", in *2018 IEEE*

International Conference on Consumer Electronics (ICCE), 2018, pp. 1–4. DOI: 10.1109/ICCE.2018.8326349.

[37] P. P. Ray, “A survey of iot cloud platforms”, *Future Computing and Informatics Journal*, vol. 1, no. 1, pp. 35–46, 2016, ISSN: 2314-7288. DOI: https://doi.org/10.1016/j.fcij.2017.02.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2314728816300149.

[38] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li, “Comparison of several cloud computing platforms”, in *2009 Second International Symposium on Information Science and Engineering*, 2009, pp. 23–27. DOI: 10.1109/ISISE.2009.94.

[39] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Jun. 1999. DOI: 10.17487/RFC2616. [Online]. Available: https://rfc-editor.org/rfc/rfc2616.txt.

[40] M. Belshe, R. Peon, and M. Thomson, *Hypertext transfer protocol version 2 (http/2)*, 2015.

[41] J. C. Mogul, “The case for persistent-connection http”, in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’95, Cambridge, Massachusetts, USA: Association for Computing Machinery, 1995, pp. 299–313, ISBN: 0897917111. DOI: 10.1145/217382.217465. [Online]. Available: https://doi.org/10.1145/217382.217465.

[42] *Http request methods*, Accessed: 18/01/2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[43] L. D. Catledge and J. E. Pitkow, “Characterizing browsing strategies in the world-wide web”, *Computer Networks and ISDN Systems*, vol. 27, no. 6, pp. 1065–1073, 1995, Proceedings of the Third International World-Wide Web Conference, ISSN:

0169-7552. DOI: https://doi.org/10.1016/0169-7552(95)00043-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169755295000437.

[44] N. Naik, P. Jenkins, P. Davies, and D. Newell, "Native web communication protocols and their effects on the performance of web services and systems", in *2016 IEEE International Conference on Computer and Information Technology (CIT)*, 2016, pp. 219–225. DOI: 10.1109/CIT.2016.100.

[45] I. Paterson, D. Smith, P. Saint-Andre, and J. Moffitt, "Bidirectional-streams over synchronous http (bosh)", 2009.

[46] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with websocket", *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012. DOI: 10.1109/MIC.2012.64.

[47] Kaazing, *Html5 websocket - a quantum leap in scalability for the web*, Accessed: 19/01/2021. [Online]. Available: https://www.websocket.org/quantum.html.

[48] I. Fette and A. Melnikov, "The websocket protocol", RFC Editor, RFC 6455, Dec. 2011, http://www.rfc-editor.org/rfc/rfc6455.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6455.txt.

[49] V. Wang, F. Salim, and P. Moskovits, *The Definitive Guide to HTML5 WebSocket*, 1st. USA: Apress, 2013, ISBN: 1430247401.

[50] D. Skvorc, M. Horvat, and S. Srbljic, "Performance evaluation of websocket protocol for implementation of full-duplex web streams", in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014, pp. 1003–1008. DOI: 10.1109/MIPRO.2014.6859715.

[51] *Websockets - a conceptual deep dive*, Accessed: 19/01/2020. [Online]. Available: https://www.ably.io/topic/websockets.

[52] N. Witthayawiroj and P. Nilaphruek, "The development of smart home system for controlling and monitoring energy consumption using WebSocket protocol", *IOP Conference Series: Materials Science and Engineering*, vol. 185, p. 012 019, Mar. 2017. DOI: 10.1088/1757-899x/185/1/012019. [Online]. Available: https://doi.org/10.1088/1757-899x/185/1/012019.

[53] B. Soewito, Christian, F. E. Gunawan, Diana, and I. G. P. Kusuma, "Websocket to support real time smart home applications", *Procedia Computer Science*, vol. 157, pp. 560–566, 2019, The 4th International Conference on Computer Science and Computational Intelligence (ICCSCI 2019) : Enabling Collaboration to Escalate Impact of Research Results for Society, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.09.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091931172X.

[54] 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. DOI: 10.1109/COMSWA.2008.4554519.

[55] D. Soni and A. Makwana, "A survey on mqtt: A protocol of internet of things (iot)", in *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, vol. 20, 2017.

[56] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, "Internet of things: Survey and open issues of mqtt protocol", in *2017 International Conference on Engineering MIS (ICEMIS)*, 2017, pp. 1–6. DOI: 10.1109/ICEMIS.2017.8273112.

[57] M. Singh, M. A. Rajan, V. L. Shivraj, and P. Balamuralidhar, "Secure mqtt for internet of things (iot)", in *2015 Fifth International Conference on Communication Systems and Network Technologies*, 2015, pp. 746–751. DOI: 10.1109/CSNT.2015.16.

[58] 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)*, IEEE, 2014, pp. 1–6.

[59] Google, *Google cloud platform*, Accessed: 13/02/2020. [Online]. Available: `https://cloud.google.com/`.

[60] ——, *Firebase*, Accessed: 13/02/2021. [Online]. Available: `https://firebase.google.com/`.

[61] *On street parking*, Accessed: 23/11/2020. [Online]. Available: `https://fiware-datamodels.readthedocs.io/en/latest/Parking/OnStreetParking/doc/spec/index.html`.

[62] K. Docs, *Pub/sub*, Accessed: 20/01/2021. [Online]. Available: `https://docs.kubemq.io/learn/message-patterns/pubsub`.

[63] OpenWeatherMap, *Current weather and forecast*, Accessed: 20/09/2020. [Online]. Available: `https://openweathermap.org`.

[64] E. Rizzon and U. Kayande, *The value of analytics in 2019*, Accessed: 18/01/2020. [Online]. Available: `https://www.kearney.com/analytics/article/?/a/the-value-of-analytics-in-2019`.

[65] Github, *Top open source projects*, Accessed: 01/02/2021. [Online]. Available: `https://octoverse.github.com/2018/projects#repositories`.

[66] Figma, *A collaborative interface design tool*, Accessed: 01/02/2021. [Online]. Available: `https://www.figma.com/`.