










# Smart Campus Parking – Parking Made Easy

Amanda Vieira<sup>1</sup> , Iolanda Rosa<sup>1</sup> , Ivo Santos<sup>1,2</sup> ,  
Tiago Paulo<sup>1</sup> , Nuno Costa<sup>1,2</sup> , Marisa Maximiano<sup>1,2</sup> ,  
and Catarina I. Reis<sup>1</sup> 

<sup>1</sup> School of Technology and Management, Polytechnic of Leiria, Leiria, Portugal  
{2160832, 2161477, 2160837, 2161352}@my.ipleiria.pt,  
{nuno.costa, marisa.maximiano,  
catarina.reis}@ipleiria.pt

<sup>2</sup> Computer Science and Communication Research Centre,  
Polytechnic Institute of Leiria, Leiria, Portugal

**Abstract.** The number of users of the parking lots from the campus of the Polytechnic of Leiria, a higher education institution in Portugal, has been increasing each year. It is becoming a major concern to the organization to address the high demand for a free parking spot on campus. In order to ease this problem, this paper proposes the design of a smart parking system that can help users to easily find a free parking spot, using an integrated system that includes sensors and a mobile application.

The system is based on the information about the occupation status of parking lots generated by parking sensors. This information is available in the mobile application that consumes a REST webservice and is presented to end-users, thus contributing to the decrease of time wasted on the quest of finding a free spot. The software architecture consists on a set of decoupled modules that compute and share the information generated by sensors. This architectural approach is noteworthy because it maximizes the system scalability and responsiveness to change. It allows the system to expand with the integration of new applications and perform updates on the existing ones, without an overall impact on the operations of the other system modules.

**Keywords:** IoT · Sensors · Smart parking · Android · API · REST

## 1 Introduction

### 1.1 Context

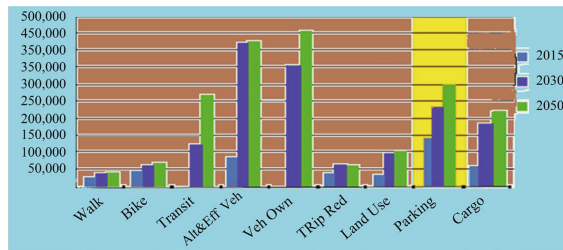
The development of the economy leads to a continuous growth in the number of available motor vehicles. Nowadays, the demand for a free parking spot versus the number of available spots is a daily concern [1]. Therefore, drivers that need to park their vehicles have the, sometimes troublesome, task to find a free parking slot. Usually drivers waste their time and end up parking in an available space on the streets that they find through a struck of luck or, in a worst case scenario, they fail to find a free parking slot [2]. Parking issues also show a close relation with traffic congestion, accident, and environment pollution [3].

This reality is starting to become a challenge even for higher educational institutions that want to provide the best parking conditions to their users, and the Polytechnic of Leiria is not an exception. The increasing number of enrollments (shown in Table 1) and the high percentage of vehicle ownership, lead to a decrease of supply of free parking spots, which results in triggering blockage of vehicles, traffic congestion, wastage of time and fuel [4] and to high levels of pollution.

**Table 1.** Evolution of the total number of enrolled students and employees of the Polytechnic of Leiria for the academic years 2015–2018 [5].

Academic year	2015/2016	2016/2017	2017/2018
Total students	10417	10472	11026
Total employees	1128	1162	1245

Therefore, the development of a smart parking system can play an important role to help with this scenario. According to a report, smart parking can result in 220 000 gallons of fuels saving till 2030 and approx. 300 000 gallons of fuels saved by 2050, when implemented successfully (shown in Fig. 1) [6].



**Fig. 1.** Assumption of saving fuel consumption.

In addition to the reasons described above, there are other advantages resulting from the usage of smart parking systems that also establish a motivation for this work:

- Efficiently manage the use of parking lots decreasing the traffic flow. Since drivers do not need to circle around to find an available parking spot, this leads to an improvement of the time that a driver takes to find the best spot, while minimizing the occurrence of incidents;
- Improve the environment by reducing pollutants' emission resulting from fuel wasted on the quest of searching for a parking spot;
- Allow users to access real-time information about the parking lots occupancy rate (information also available for any time of the day);
- Allow the integration with other functionalities such as obtaining user's vehicle location when it's parked, see the most popular parking spot, the last reported incidents on a parking spot, among others;

- Conduct various statistical analysis. For instance, explore the periods with most traffic congestion and see if there's a possible relation with the occurrence of incidents.

## 1.2 Motivation

This paper aims to present the design and development of a smart parking system for the efficient management of parking lots in the Campus 2 of the Polytechnic of Leiria. The main goal is to provide guidance to drivers about the availability of parking spots at the campus, delivering them with up-to-date information obtained from sensors in the parking lots.

The work relies on the integration of several technologies and applications that aim to collect, compute and persist data generated by parking sensors in real-time. End-users can access this data using an Android application that communicates with a REST webservice in order to obtain a response to the user's requests.

The operation of the system, shown in Fig. 2, can be described as a group of sensors detecting the occupancy status of the parking spots and sending this data to a set of integrated applications that unify and persist the information on a centralized database. On the other hand, there is a REST webservice that will expose this data to answer to the received requests from end-user applications. View parking information, use and find the best parking spots, report incidents, view statistical information among other administrative tasks, are some of the features provided by the web service.

One of the main concerns while developing the system was its scalability considering the premise that this platform should easily allow the future integration of new applications that will also receive and share information. Besides, the system must be responsive to change, and thus, it should be easy to expand in order to integrate new parking lots, new services and perform updates in a part of the system without affecting the other modules. In order to meet this goal, the approach consists of an interconnected modules' architecture with maximum decoupling between them.

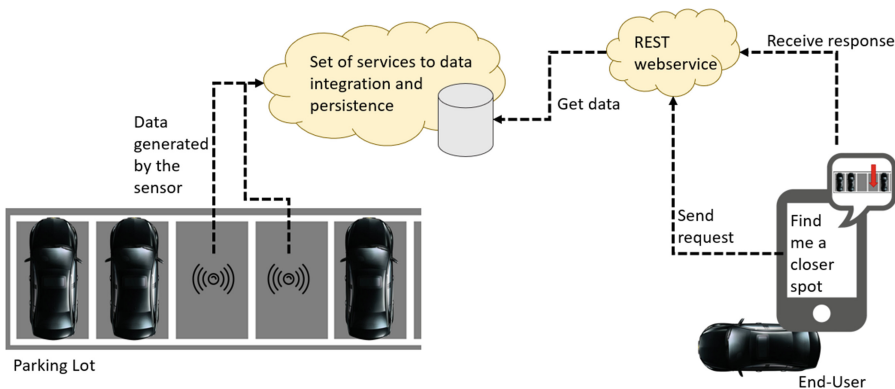
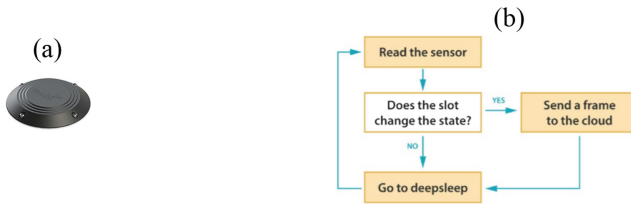


Fig. 2. General outline of system operation.

There are some important features that should be considered when choosing real sensor devices to use in the system:

- Low installation and replacing time to reduce the costs;
- Resistance to atmospheric conditions specially in parking lots located outdoors;
- Compatibility with wireless networks;
- Optimized for low-power consumption;
- Simple for developers to work;
- Allow remote management of several node parameters at the same time.

A good example of a sensor device with all these features is the one of Libelium shown in Fig. 3(a).



**Fig. 3.** (a) Libelium smart parking sensor device and (b) the basic working loop diagram [7].

This sensor can be configured remotely using wireless access and supports bi-directional communication. It works in a very efficient way as shown in Fig. 3(b). Reads the sensor data and sends a frame when the parking spot changes its status. Then, it sleeps a desired time before starting the loop again [7].

Currently, and regarding the work here described, the Polytechnic of Leiria does not have sensors on its parking spots due to the initial economic investment that is necessary to deploy such a system. Thus, to mitigate this limitation, two applications that mimic the data generated by real sensors were created (providers).

### 1.3 Related Work

Related systems already exist. We started by identifying related mobile applications available in the Apple Store [8] and Play Store [9]. Their main functionalities include obtaining the location of an available parking slot in an area, reserve it and pay it via app.

Next, we present some examples of applications with the referred functionalities:

- **JustPark Parking:** Android application that includes around 20 000 parking locations that are available for booking [10].
- **ParkMobile Parking:** Android application that allows the users to have favorite parking zones, update personal details, change payment method, download their invoice, view parking history, etc. [11].

- **ParkNow Parking:** Android application that works on-street and off-street and the payment due is charged monthly. It also allows a user to choose a parking spot based on distance and price [12].
- **PayByPhone:** Smart parking application available on iPhone, Android, Blackberry and Windows. Allows the user to find parking lots, pay the parking fee through the application and even extend the time the user wishes to be parked for [13].
- **BestParking:** iPhone application that allows the user to find an available spot at the best price [14].
- **SmoothParking:** iPhone application that notifies the user when the legal parking time is up. It's available in English and Spanish [15].
- **Parking Hero:** iPhone application that notifies the user about promotions, and shows the locations of special spots like taxis, pickup and delivery points, tourist buses or truck parking on the highway [16].

## 2 Smart Campus Parking – System Architecture

Figure 4 presents the general architecture of the system. The parking sensors on our system are represented by the providers.

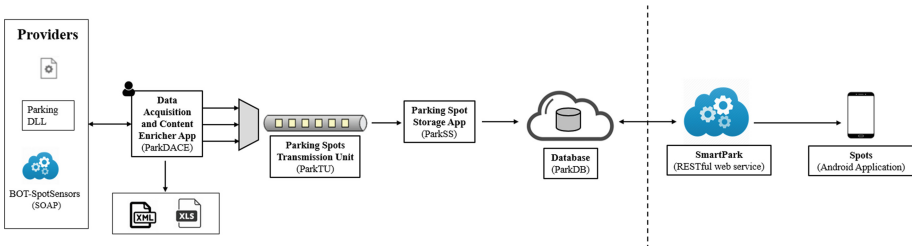


Fig. 4. General service and Spots app communication layer architectural schema.

### 2.1 Application Programming Interface – Tailored Services

The Spots app (see Sect. 3) will obtain the parking lots information using a REST webservice named SmartPark. The REST architectural style [17] was adopted in the development of the SmartPark API because it is stateless, which is a fundamental aspect for the scalability of the system allowing many clients to connect to the API without degradation of the response time. Thus, the SmartPark role is to provide endpoints and supply data to client applications such as:

- Parking lots general information: name, operating hours, total of spots, etc.;
- Information about occupation status of parking spots and sensor's battery status;
- Instant occupation rate value and perform historical queries about parking occupation at a specific time or during a time period, among others.

This service acquires information from a database where data is placed by the ParkSS application which, on its side, is a part of a system formed by a layer of modules whose purpose is to receive and persist the information generated by the sensors in a unified format.

## 2.2 Sensors Data Collector and Transformation Modules

As previously mentioned, there are no sensors installed on the parking lots of the Polytechnic of Leiria, so in order to develop the system two providers that mimic the behavior of real sensors were made. The ParkingSensorNodeDLL is a library that generates and pushes data directly to the receiver application in a preset time interval. The BOT-SpotSensors, on other way, is a SOAP webservice [18] that generates data in xml and data contract formats. BOT-SpotSensors follows a “pull” approach meaning that the applications who want to obtain the sensors’ data are responsible to request this information to the service. Both providers generate data about occupation status and sensor battery, in a random way. This approach is not the ideal, but it’s the simplest and fastest way to implement and, accurately, test the system.

The data generated by these providers needs to be integrated with other information (geolocation and parking lots data) into a unified format and this is done by the ParkDACE application. Therefore, the ParkDACE goal is to receive and collect the data from different sources, validate and integrate it in order to send this information to other application that is responsible for the data persistence.

The sharing of the data is made using a messaging system working with the MQTT [19] connectivity protocol (see Sect. 2.3). Under this protocol, ParkDACE plays the role of a publisher that sends messages with the unified information to the broker. The broker is responsible for distributing the data to all of the applications that subscribed the publisher topics. This system allows multiple subscriptions from various applications to the same topic, without service degradation.

The ParkSS application is a subscriber that will receive the information sent by ParkDACE and will persist the data into a relational database, named ParkDB.

All the information described above can be seen in Fig. 4.

## 2.3 Broker – Sensors Integration

To make parks and sensors’ information available to interested applications it was used a message broker system. We used the broker Eclipse Mosquitto, which is an open source broker, [21] that implements the MQTT protocol, an open protocol that uses the publish-subscribe model, being the application ParkDACE responsible for publishing information about the parking lots and sensors.

The MQTT protocol is lightweight and, ultimately, a good choice for sensors that have limited resources, since it allows an efficient transmission of information.

The ParkTU application plays the essential role of the broker since it is the mediator that controls the connection and message transmission between applications.

Regarding the Quality of Service (QoS) we used a Level 2 – Exactly Once – since it ensures that each message is only received once. It is considered the most reliable level in terms of guarantees for the application because it delivers all the messages. If we

have used a Level 0 – At Most Once – the analysis of the state of the parks and sensors, such as future statistical analyzes would not be reliable. Similarly, on Level 1 – At Least Once – the duplication of messages could occur, being then necessary an additional processing by the applications that would receive the information to manipulate the potential duplicated messages. Nevertheless, this level could be considerable advantageous if the performance of the transmission was indeed a crucial factor for the application that receives the data.

In the message broker, two topics were created for the transmission of information:

- Parks – information of parking lots;
- Spots – information about the sensors’ status.

When using the message broker with the publish-subscribe model, the application becomes mores scalable. Thus, any application that wants to receive information provided by the ParkDACE can simply subscribe to the desired topics, according to the information that it wants to receive.

### 3 Spots

The Spots mobile application was developed for the Android operating system and allows users to obtain information about the occupation state of the available parks.

The mobile application consumes data from a Firebase instance and from the RESTful API described in Sect. 2. It also uses the Google API (including Maps), specific georeferentiation libraries from the mobile device (GPS and Gallery) and a specific library to create charts. The architecture can be visualized in Fig. 5 and, additionally, more details can be found on Appendix A.

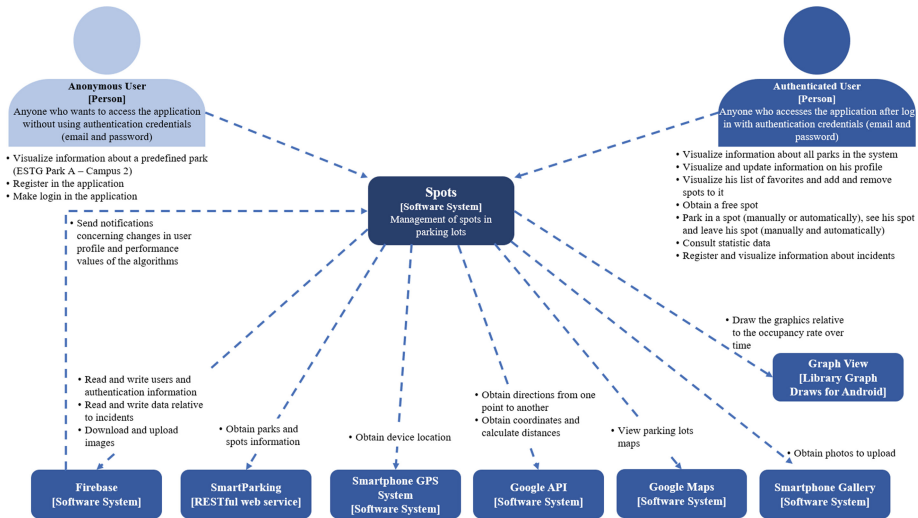
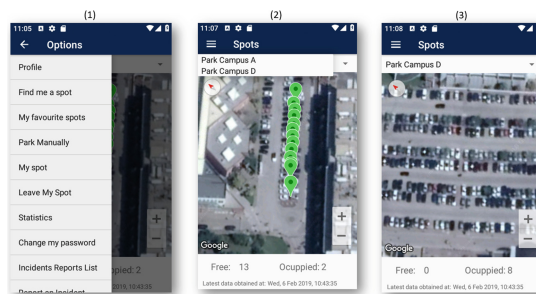


Fig. 5. Spots application architecture.

This application can be used by both authenticated and unauthenticated users. Unauthenticated users can access to the initial screens where they: see information about the occupation state of a specific park, and where they can register or authenticate in the application.

Figure 6(2) presents the parking layout where the green spots correspond to the geolocation of available slots, the total number of free and occupied slots on the park and the last update date for the information presented. It's also possible to unauthenticated users to register or authenticate themselves on the application. To register and authenticate on the Spots app, an email address and password are required.

For authenticated users, the Spots app provides a set of functionalities displayed on an always-present slide-drawer menu. Authenticated users can see the occupation state for all the available parking lots of the system (Fig. 6).



**Fig. 6.** Initial menu functionalities available for authenticated users (1), initial screen displaying Parking A occupation status on Campus 2 and the selection of menu with all parks available (2), Parking D with all slots occupied (3).

The most relevant features are:

- **Profile:** Users can see the information of their profile and update it. They have the possibility to manage their favorite parking spots' list;
- **My Spot:** When the user is parked the application shows the vehicle location on the parking lot (the spot);
- **Find a me a spot:** The aim of this functionality is to provide the location of an available spot on the user's selected parking according to his preference. There are three search algorithms that can be applied in order to find a free place: select the spot with closest location according to user's GPS location; select the best rated spot from all the spots in a parking lot; and, select the best rated place from the current user's favorites list.

The user can preset a preference for the "Find me a spot" option in the update profile screen and the application will choose the best spot according to user's preference. Otherwise, if the user does not have a preset preference, the application will ask him what option to use. After that, the application will present an available spot; will use the Google Maps service where the geolocation spot is shown, and the user could get the directions to reach that location;



- Park/Leave My Spot:** The Spots application has an automatic detection mechanism. When there is a change in the occupancy state of a parking, it will be checked if there is any logged user parking or leaving the spot on that sensor location (Fig. 7). However, it also has options in the menu to allow a user to manually specify where he intends to park or when he's leaving a spot. Whenever the user leaves a parking spot, he can rate it and add it to his favorites list if that spot isn't already part of that list.

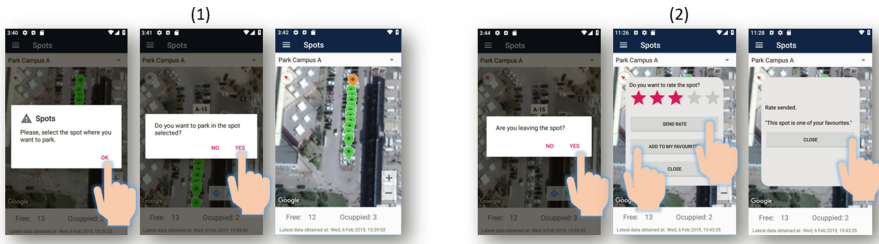


Fig. 7. Park Manually operation (1) and leave my spot manually (2).

- Occupation rate/Statistics/Algorithm Performance:** The occupation rate screen shows the evolution of the average occupation rate during a time period in a graphical representation. There is also some additional information regarding users and application usage statistics, as shown in Fig. 8. The algorithm performance screen shows the average computation time in milliseconds that each algorithm takes to give a response to users. All this information is updated in real-time.

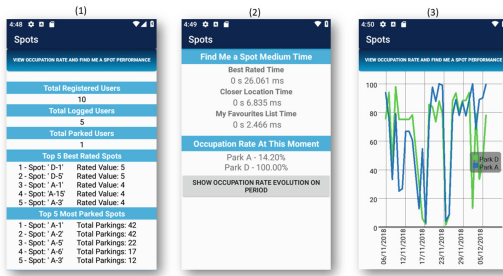
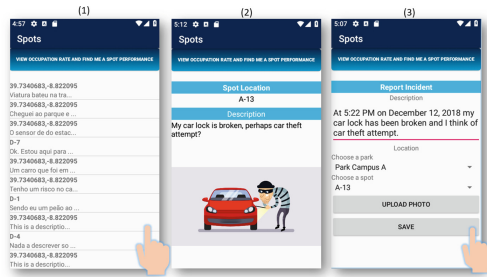


Fig. 8. Statistics (1), algorithm performance (2) and occupation rate during time (3) charts/graphical representation.

- Incidents:** Allows the user to report incidents, see the list of all reported incidents and their details (see Fig. 9).



**Fig. 9.** Incidents report list (1), incident details (2) and report an incident (3) screens.

The development of the Spots application rests in some general principles of agile methodologies like Scrum [22], XP [23] and Kanban [24]. In this app the features were developed in an incremental and iterative way which allows developers to get constant feedback and improvement of the application.

For automated testing, the Cucumber tool [25] was used and it's based on the XP principle of Test First Programming [23], which leads the team to have a clearly understanding of the requirements before their actual implementation. The acceptance tests were written using the Gherkin language [26] - a "natural" language that contributes to part of the documentation. For the specific implementation of the "steps" we used the Espresso framework [27], the standard for automated testing for Android.

## 4 Conclusion and Future Work

This paper aims to present the design of a platform for an efficient, cost-effective smart parking system to the Polytechnic of Leiria and thereby enhance the parking quality to the students and employees working on this institution. This system manages data received from sensors with focus on providing real-time information regarding the availability of parking spots to end-users. The main goal is achieving the maximum decoupling between the several modules that compose the system, in order to maximize scalability and minimize the failure of the entire system just because a single module becomes unavailable.

From the analysis of related work, we conclude that a smart parking system has many advantages: decreases traffic congestion; minimizes fuel consumption; reduces the number of incidents on parking lots, and optimizes the time wasted by the drivers to search for a parking spot. Although there are still no sensors implemented on the Polytechnic of Leiria parking lots, in the future, when it becomes a reality, the timely planning of the platform presented on this paper will speed up the process of implementation of the smart park system.

There is future work that could be done in order to improve this system like the upgrade of the authentication mechanism from the client application allowing the users to have other options besides the email to authenticate themselves, like authentication by username or with their personal institutional identification number.

Also, it should be considered the development of an administration platform for the parking lots and other client applications, alike the proposed on this paper, and that work on distinct operating systems. Besides, the option to incorporate activity detection and even environmental factors would also allow us to predict, with some accuracy, the occupancy rate and the other significant statistics of the platform.

In the future, it would also be interesting to allow remote users to previously book an available spot for a short amount of time, before they get there, and park. This mechanism is more difficult to implement as many changes would be required since nowadays the Polytechnic of Leiria offers free parking. The solution would require, at least, an access control mechanism to the parks and the count of available spots excluding the ones that have been booked. In addition to this, it should also be considered the implementation of a security mechanism that ensures the use of the parking lots only by students and employees from the institution.

**Acknowledgements.** This work is financed by national funds through the FCT - Foundation for Science and Technology, I.P., under project UID/CEC/04524/2016.

## Appendix A

See Figs. 10, 11, 12, 13, 14 and 15.

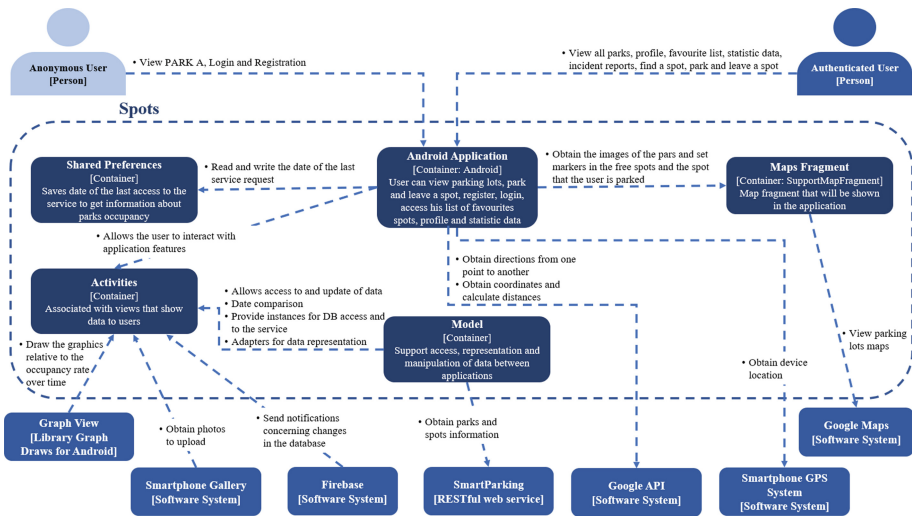


Fig. 10. Spots application containers details architectural scheme.

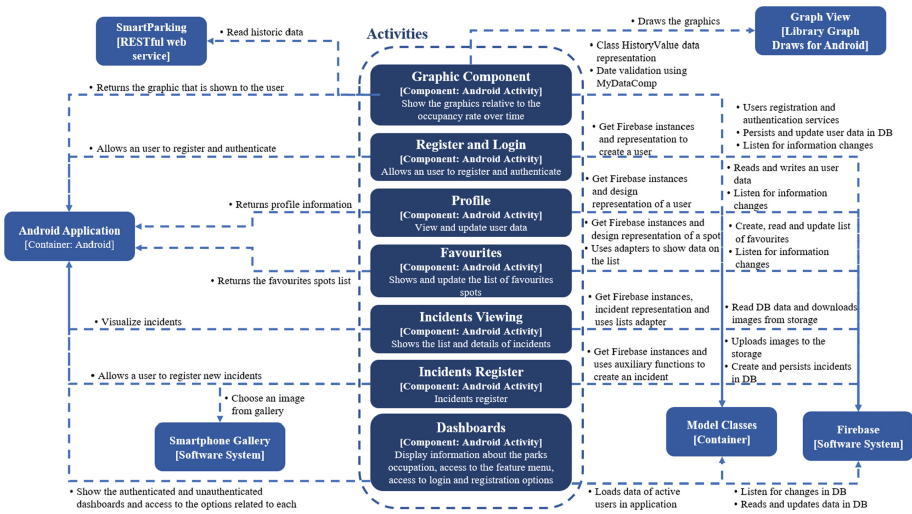


Fig. 11. Components of activities container of Spots application.

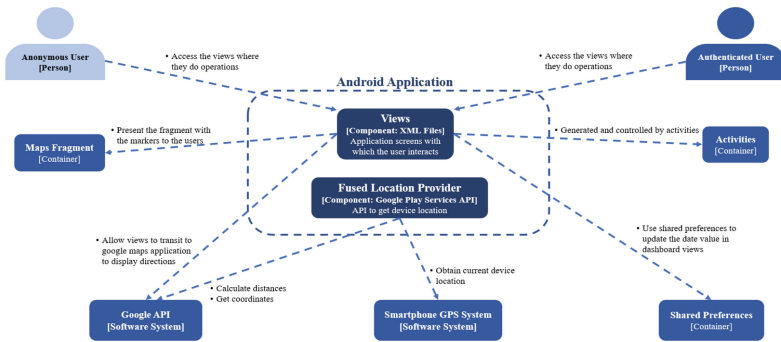


Fig. 12. Components of android application container of Spots application.

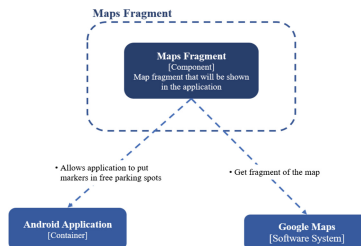


Fig. 13. Components of maps fragment container of Spots application.

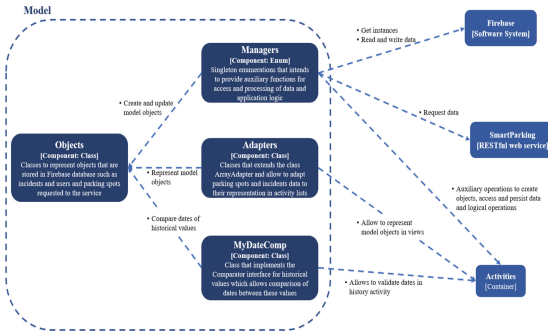


Fig. 14. Components of model container of Spots application.

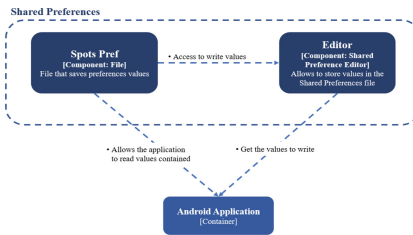


Fig. 15. Components of shared preferences container of spots application.

## References

1. Wang, Z., Lian, Z., Han, K.: Designing of intelligent parking lot based on MQTT. *Int. J. Adv. Netw. Monit. Control.* **2**(4), 317–322 (2017)
2. Vishwanath, Y., Aishwarya, D.K., Debarupa, R.: Smart parking system based on Internet of Things. *Int. J. Recent Trends Eng. Res.* **2**(3), 156–160 (2016)
3. Liu, Y., et al.: Metropolis parking problems and management planning solutions for traffic operation effectiveness. *Math. Probl. Eng.* **2012**, 6 (2012)
4. Lingam, C.: *University Car Parking Application*. Texas (2018)
5. Polytechnic of Leiria: Factos e Números. <https://www.ipleiria.pt/ipleiria/ipl-em-numeros/>. Accessed 07 Feb 2019
6. Ahad, A., Khan, Z.R., Ahmad, S.A.: Intelligent parking system. *World J. Eng. Technol.* **4**, 160–167 (2016)
7. Libelium: *Smart Parking* (2018)
8. Apple Inc.: *App Store* (2019). <https://www.apple.com/pt/ios/app-store/>. Accessed 13 Feb 2019
9. Google: *Google Play* (2019). <https://play.google.com/store/apps>. Accessed 13 Feb 2019
10. JustPark: *JustPark Parking – Apps on Google Play*. <https://play.google.com/store/apps/details?id=com.justpark.jp>. Accessed 29 Jan 2019
11. ParkMobile: *Parkmobile Parking – Apps on Google Play*. <https://play.google.com/store/apps/details?id=com.parkmobile>. Accessed 29 Jan 2019
12. ParkNow: *ParkNow Parking – Apps on Google Play*. <https://play.google.com/store/apps/details?id=com.parknow.app>. Accessed 29 Jan 2019

13. PayByPhone: Start new parking. PayByPhone. <https://m2.paybyphone.com/parking/start/location>. Accessed 29 Jan 2019
14. BestParking: BestParking: Find and Book Parking Anywhere. <https://www.bestparking.com/>. Accessed 29 Jan 2019
15. SmoothParking Inc.: SmoothParking. <https://itunes.apple.com/us/app/smoothparking/id615418863?mt=8>. Accessed 07 Feb 2019
16. Parking Hero: Smart Parking - Parking Hero. <https://itunes.apple.com/us/app/smart-parking-parking-hero/id1249834332?mt=8>. Accessed 06 Feb 2019
17. Lange, K.: The Little Book on REST Services. Copenhagen (2016)
18. Tidwell, D., Snell, J., Kulchenko, P.: Programming Web Services with SOAP, 1st edn. O'Reilly, pp. 21–36 (2001)
19. MQTT: MQTT.org (2014). <http://mqtt.org>. Accessed 13 Feb 2019
20. Fawcett, J., Quin, L.R.E., Ayers, D.: Beginning XML, 5th edn. (2012)
21. Eclipse Foundation: Eclipse Mosquitto. <https://mosquitto.org>. Accessed 13 Feb 2019
22. Schwaber, K., Sutherland, J.: The Scrum Guide: The Definitive the Rules of the Game. Scrum.org and Scrum Inc., p. 19 (2017)
23. Beck, K., Andres, C.: Extreme Programming Explained - Embrace Change, 2nd edn. John Wait, United States (2004)
24. Anderson, D.J., Carmichael, A.: Essential Kanban Condensed, 1st edn. Lean Kanban University Press, Seattle (2016)
25. Wynne, M., Hellesoy, A., Tooke, S.: The Cucumber Book - Behaviour-Driven Development for Testers and Developers, 2nd edn. Pragmatic Programmers, United States (2017)
26. Gherkin Reference. <https://docs.cucumber.io/gherkin/reference/>. Accessed 13 Feb 2019
27. Espresso (2019). <https://developer.android.com/training/testing/espresso/>. Accessed 13 Feb 2019