Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Sistemas de Computação Móvel e Ubíqua

2023/2024

park IT

Final Report

Authors:
60011 Bruno David
60310 Gonçalo Silva
68882 Gonçalo Cerveira

Professors:
Carmen Morgado
Vítor Duarte

June 2024

Conteúdo

1. Introduction

This report seeks the purpose of discussing both the approach and implementation for the SCMU course, edition of 2023-24.

For the project, it was decided to implement a parking lot management system for companies, designed to enhance the parking experience and ensure safety in the lot for all users-

The implementation of this system is both interesting and useful because it addresses common urban challenges, significantly improving user convenience. By integrating all the components, users can easily find and reserve parking spaces, reducing time and stress. Additionally, the system ensures a sense of security by monitoring and managing environmental levels.

This project incorporates pervasive computing elements and includes a mobile application that provides to users real-time data and control over the parking system. The role of the SuperUser emphasizes the importance of the system's user-centered design, allowing direct interaction between the app and the sensors and other functionalities to manage the parking lot.

Furthermore, the high scalability of this project illustrates a key principle of ubiquitous computing, demonstrating the capability to connect additional subsidiary microcontrollers to a main server, thereby accommodating more parking spaces as needed.

2. General Overview

The system incorporates RFID technology, allowing each car to use a designated card for accessing the parking lot. A proximity sensor at the gate activates upon detecting an approaching vehicle, prompting the user to authenticate with their card. Access is granted based on space availability and user permissions, with all authentication processes relying on prior registration using the company's email and details.

Once inside, drivers can easily identify available parking spots through the LEDs installed above each space: green indicates a free spot, red signals an occupied spot, and blue denotes a reserved spot. For testing purposes, we implemented this system with two parking spots, but it can be easily scaled to accommodate a larger number of spots. When a user parks in a free spot, an ultrasonic sensor verifies if the vehicle remains in the chosen spot, updating the spot's status to occupied after three seconds.

Additionally, the system supports reservations via app, with reserved spots clearly indicated by an LCD on-site.

Given the parking lot is indoors, monitoring air quality is crucial. The app provides real-time updates on noxious gas levels detected by the gas sensors, ensuring that users are informed about environmental conditions. Furthermore, it also displays temperature and humidity levels, thanks to the DHT sensor.

At last a superuser role has been implemented within the system. This role allows them to adjust sensor thresholds, test equipment functionality, such as alarm levels, and automatically open the parking gate when necessary. The superuser also manages employee access and their reservations. Furthermore, the superuser can review the

complete history of the parking lot, including records of entering and exiting and the current parked users.

3. System Architecture

The system's architecture relies on a mobile application that sends and receives information from/to two servers (cloud and local).

On site, there are two microcontrollers, each one with different purposes. The main one works as a base station for the parking lot, controlling entrances, environment, and other information. The second microcontroller, to ensure the system is well designed, implemented and is scalable, manages the sensors and actuators for a given number of parking spaces, which ultimately shares information with the main microcontroller.

To implement the system, the following sensors/actuators are needed:

- *1x RFID* – access and exit control (for proof of concept, unrealistic considering the real world since each entry/exit point would have its own RFID anchor). Each RFID tag is encoded with a unique code. This code includes a user identifier and a system identifier to ensure that the tag belongs to this park. This encoding allows the system to count the number of users in the park, which users are allowed to enter and list an access history on the mobile application. A superuser can, therefore, enable or block access to a given user, with such information being sent to this component. Component is connected to the main microcontroller.

- *1x Motion Sensor* – placed at every entrance and exit point, once there is motion detected, such information is sent through the main microcontroller to turn on the RFID system for the next 30 seconds (time interval valid for proof of concept, unrealistic considering the real world). This component is mainly used for energy-saving purposes.

- *4x LED* - two are used to simulate the behavior of the entry and exit gates, according to the RFID system's response, with such information being received by the main microcontroller. Another indicates if the park is full, with such information being managed by the main microcontroller. The fourth LED indicates an alarm, from main microcontroller information.

- *2x Ultrasonic Sensor* – each one of these sensors is placed above each parking space to determine if it is empty or not (valid for proof of concept, unrealistic considering the real world since at least two must be used for higher accuracy). Such information is sent to the subsidiary microcontroller, so the number of available parking spaces is maintained in real-time and corresponding LED's color is updated.

- *2x RGB LEDs* – each one is placed next to the corresponding ultrasonic sensor, to make the space's availability visible, changing color based on the information received from the subsidiary microcontroller.

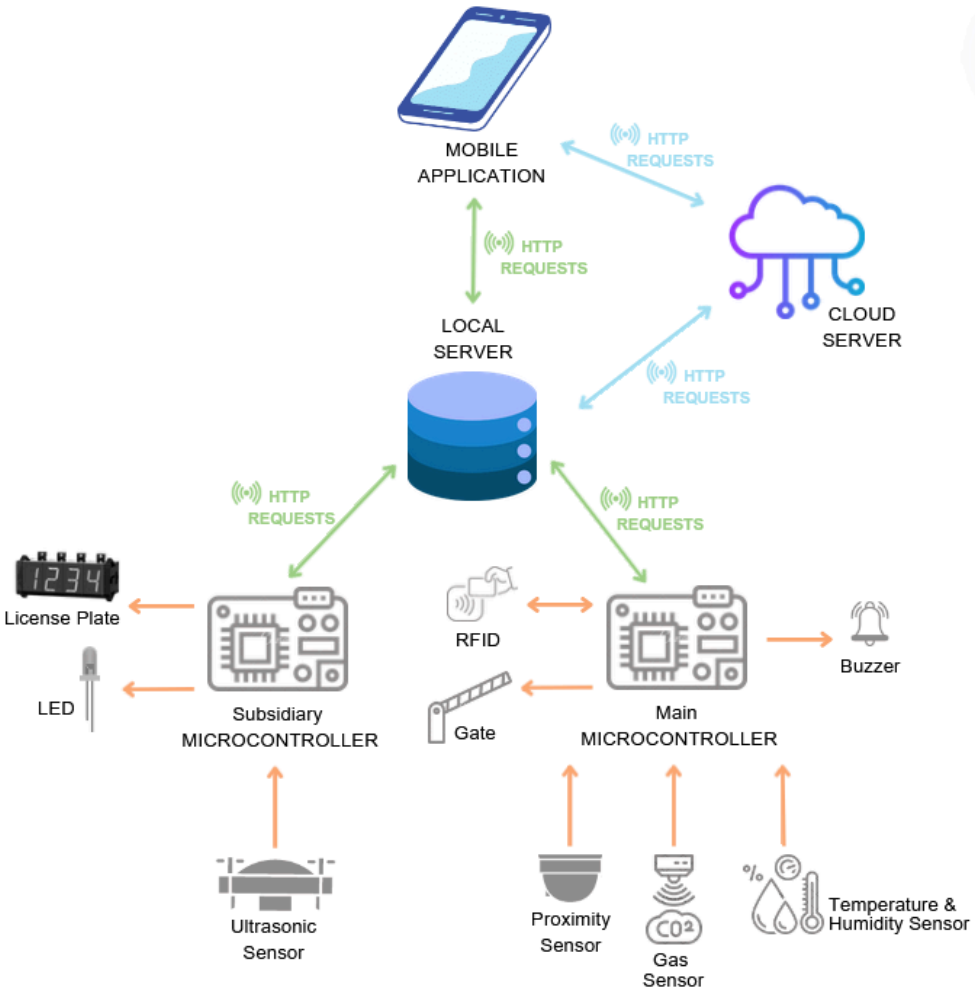
- *1x LCD Display* – placed next to the ultrasonic sensor, displaying the license plate of the user who reserved that parking space. This component is connected to the subsidiary microcontroller and such information is received in this microcontroller from the cloud server, which allows updates on the display.

- *1x Gas Sensor* – This component is connected to the main microcontroller, and, at a time interval of 5 seconds, the cloud server receives information to display in real-time on the mobile application (for proof of concept, unrealistic considering the real world, since time interval must be smaller for higher accuracy). If levels become dangerous, the microcontroller immediately initiates a sequence of alarms on site, through the buzzer, and such alerts are also displayed on the mobile application.

- *1x Temperature & Humidity Sensor* - This component is connected to the main microcontroller, and, at a time interval of 5 seconds, the cloud server receives information to display in real-time on the mobile application. If levels become dangerous, such alert is displayed on the mobile application.

- *1x Buzzer* – The component is connected to the main microcontroller and is activated by received information.

A scheme for the system's architecture is presented in the next page.


MOBILE
APPLICATION
HTTP
REQUESTS
HTTP
REQUESTS
LOCAL
SERVER
CLOUD
SERVER
HTTP
REQUESTS
HTTP
REQUESTS
HTTP
REQUESTS
License Plate
LED
Subsidiary
MICROCONTROLLER
RFID
Gate
Main
MICROCONTROLLER
Buzzer
Ultrasonic
Sensor
Proximity
Sensor
Gas
Sensor
Temperature &
Humidity Sensor

4. Communication Protocols

For the purpose of this project, we decided to use Firebase as a cloud server, due to previous experience and easiness, from a programming point of view. The use of a cloud server allows to maintain information in a database for both the information of the system on site and mobile application. From microcontrollers to cloud servers, and vice-versa, communication done is through HTTP requests, controlled by a local Web Server.

The local Web Server is used to “understand” which information is sent to which microcontroller, serving as a router. Therefore, exchange of information between microcontrollers is done through HTTP requests to this local server. Both microcontrollers and the Web Server are in the same local network and can only be accessed by devices in the same local network. This allows the system to be secure from a communication point of view. For proof of concept, a hotspot of mobile data is this private network. Connecting the mobile device to this private network allows a superuser, and only the superuser, to engage with the system in a direct interaction with sensors/actuators.

When internet connection is not available, the sensing processes carry on, not affecting the system functionality on site, and the microcontrollers keep the information on device until a connection is reestablished. After reestablishing it, the system returns to its normal state, with the HTTP requests becoming available again.

In the case of a microcontroller turning off, when turned on again, it requests essential information to the cloud server via some HTTP requests. In the case of the local Web Server shutting down, when turned on again, it sends essential information from the cloud servers to both the microcontrollers via HTTP requests. This is done to improve fault tolerance.

5. Mobile Application

For the purpose of this project, it was decided to use Flutter, due to previous experience and easiness in integrating Firebase.

To ensure the application is connected to the right Firebase project and instance, several configurations were conducted, such as target platform (in this case, Android), API key, app ID, project ID and database URL, to protect the server from being used by a disallowed application, while also protecting sensitive data from trivial attacks.

Screenshots and implementation details are presented next.

- *Login* - to use the mobile app, the user needs to be logged in. No registration option is provided, considering the context in which the application is meant to be used, having each account created beforehand in Firebase Authentication system. Login method uses this system to provide a token that is used to perform operations over the server data and access the application's functionalities. The resulting authentication instance is persisted locally to allow the current session to be renewed without having to log in again. This token will be deleted the moment the user hits the *logout* button.

On each application start/boot, in the presence of a session token:

  - User data is fetched from the cloud database to keep *name*, *email*, *phone number*, *license plate*, *RFID* tag, *reserved* parking space (if so), *latest entry* and system *permissions*.
  - Streams for the sensors' reads are opened, so the application keeps track of those values in real-time to keep the user informed as on-time as possible.
  - Current thresholds are fetched from the cloud database to understand if the sensors' reads are within the acceptable range that is safe for users.
  - System state is retrieved, i.e., park is open/close, gates are open/close, x sensor is on/off to display such information to users, so the application becomes more useful from the managing point of view.

If there is no session token stored on the device, then there won't be any server calls to the cloud server, since that means the user is not logged in and won't be able to see any information.

Based on these streams and values, the user is either presented with the regular homepage or the alert one.

- *Homepage* - user is welcomed by their name and whether or not is parked, followed by the date and hour of the latest entry in the park. Sensors' latest reads are displayed with the according color when compared to their threshold (red color if read exceeds thresholds).
- *Alarm Screen* - Sensors' latest reads are still displayed with the according color. However, since the purpose of this screen is to alert the user, red is predominant

to catch their attention. If the current user is the superuser and is on site, meaning that, is connected to the same network as the ESPs and the device's location corresponds to the ESPs location, it is possible to turn off the buzzer of the parking lot. (Permission to access the device's location is required and mandatory).

Home and Alarm screens are intervaled according to the *alarm*'s stream value, which means that, in almost real-time, the user will know whether or not it is safe to be in the park.

- *Reserve* - (this option is only available if the user has permission to reserve a parking space.) If there is an available parking space to reserve, ACABAR

(For demonstration purposes, due to the fact that the LCD is only connected to the parking space number 1, reservations can only be made for that parking space. However, this is easily scalable by assigning a random number among the non-reserved and available parking spaces).

- *Manage Screen* - (this option is only available if the user is a superuser). By navigating to this screen, the superuser can manage the system by viewing:
    - accesses history - for each registered user, all timestamps of entry and exit (if so exists) are presented. If the user has entered and not exited, *CURRENTLY PARKED* in the user's card is also displayed, to make this list more useful to the administrators division in the company.
    - reservations - both active and inactive reservations are displayed to keep track of reservations history. For each active reservation it is possible to disable it, which is done by, in the cloud server, delete it from *reserveUsersList/active* and *reserveUsersRFID.* The corresponding parking space is also updated with a *free* state so the app, when looking for a parking space to reserve, knows that this given space is now available to be reserved. For each user, their inactive reservations are grouped displaying the date and hour of start and the assigned space (History of inactive reservations is not currently implemented due to time constraints to deliver the project. All the presented ones were hard coded for demonstration purposes).

    - users - *name*, id/*nickname*, *phone number* and *license plate* are listed for each registered user in the system. Furthermore, for each user, except superuser, it is possible to:
        - disable the chance of reserving a space - this affects the user's experience by removing the functionality from the app and also deleting any on-going reservations.
        - disable access to park - this deletes the user's RFID tag information from *usersRFID,* in the cloud server, prohibiting their entrance. It also deletes any on-going reservations.

It's important to note that to allow the chance of reserving a space, the superuser must first enable access to the park, if it is not enabled yet.

- *System Control Screen* - (this option appears in the manage menu only if connected to the same network as the ESPs and the device's location corresponds to the ESPs location). In this screen, the superuser is able to define new thresholds for the sensors, choose which ones are kept active, and change any other settings, such as opening/closing park and leaving gates open/closed. Once the superuser is capable of navigating to this page, from the implementation point of view, it is known that a direct connection to the microcontrollers is possible and any updates in the system's settings are performed by an HTTP Post request to the local web server. These updates are also sent to the cloud server for an easier retrieval in the following application starts/boots.

- *Account Screen* - user information is displayed with the resulting data from the initial fetch. In this screen it is possible to update license plate information (for demonstration purposes only, since it would be possible to add new attributes in the update operation). In the case a user has reserved a parking space, *parkingSpots* in the cloud server is also updated to display on the LCD the new license plate, since the reservation needs to be maintained.

Since a lot of the application architecture relies on an established wireless connection, if WI-FI or mobile data is not turned on, the mobile application will request the user to do so. To not leave the user blind to what is happening while interacting with the application, dialogs were implemented in almost every screen and operation, to keep the

user informed at all times. Some don’t have them due to time constraints in delivering the project.

The architecture of the mobile application is scalable enough to include, in the future, more functionalities (one being the management of individual parking spaces making it available or not, which was meant to be implemented using a map, but, unfortunately, was not possible due to time constraints in delivering the project).

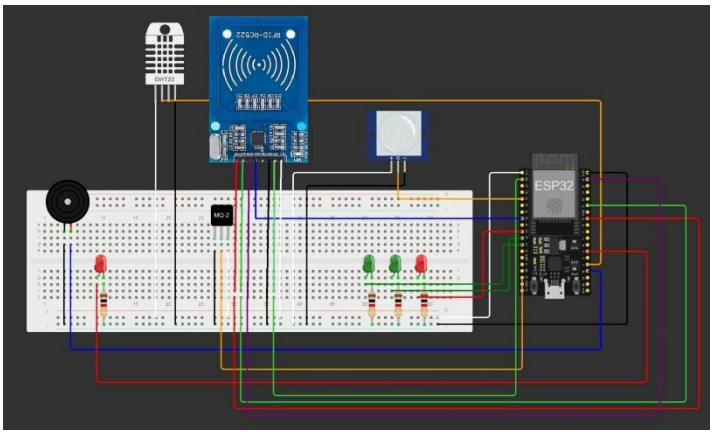
6. Sensing and Reacting

The electronic schema includes two ESP32 microcontrollers, each with distinct sensors and functionalities. The primary microcontroller oversees the overall parking lot, managing entries, monitoring environmental levels, and ensuring the general operation of the system. The secondary microcontroller is dedicated to two specific parking spaces, controlling their availability and reservation status.

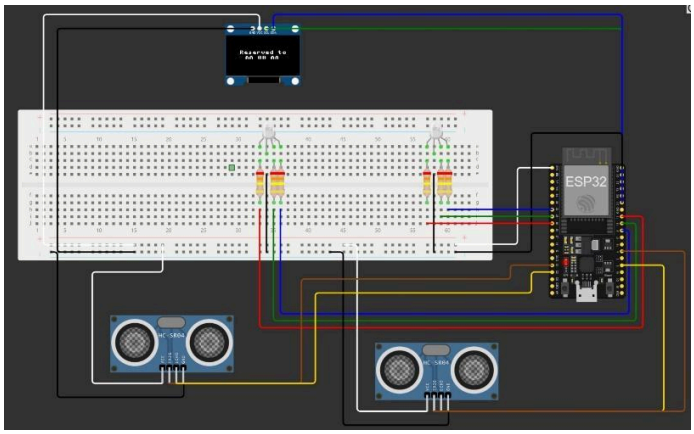
In the main microcontroller, instead of actuators, green LEDs were used to simulate the behavior of motors, signaling user entry and exit. The authentication process for entry is handled by the main microcontroller, which maintains a list of users with access to the parking lot. This list is a replica of the list saved on the cloud server, and whenever the server list is updated, a trigger automatically updates the list on the ESP32. Other sensors connected to the main microcontroller also communicate with the server through the microcontroller to provide real-time information to app users.

The subsidiary microcontroller connects to the server to manage the reservation status of parking spots. It enables or disables spots based on real-time reservation updates, ensuring the availability information remains accurate and up-to-date.

The system has four triggers in the local Web Server, in order to maintain updated data in both cloud server and the microcontrollers. The first trigger listens to changes in the Sparking spots info, i.e., if the name of its (the spot's) reservation changes. The second trigger listens to changes in the list of RFIDs of users with reservations. The third trigger listens to changes in the list of RFIDs of users that are allowed to enter the park, containing both users with and without parking spot reservation. The fourth trigger listens to changes in the values of thresholds, namely *gas_danger, humidity_danger* and *temperature_danger*.



Main Microcontroller Implementation



Subsidiary Microcontroller Implementation

The two microcontrollers' implementation is divided among two schemes, but only one breadboard will be used.

7. Considerations

Even though the current implementation strategy is already considered multiple factors related to bringing the system to the real world, some potential issues still exist and would need to be addressed.

Concerning the mobile application:

- Ideally user information would be cached on the device to reduce the number of requests to the cloud server, but current initial fetch is also able to reduce this number considerably.
- Ideally, the alarm's stream value would not depend only on retrieving it during the application boot, having also a push notification sent to all devices where a session token persists.
- In case of no internet connection is detected, currently only the information streams about the park would be able to restore their operation, which means that in a real-world scenario the application would require a more robust architecture to deal with such an issue.
- All list operations are performed every time the user navigates to the corresponding screen, i.e., a large number of calls to the cloud server are made to fetch information. Ideally, most fetches would be cached locally to reduce the number of requests to the server, while using the following possible solutions:
  - . Use a filtering query to only list what the user wants/expects;
  - . Use a query to paginate information using cursors;
- . Use a filtering operation to only update the list with new values or updated ones;

Concerning the system, the project has a point of failure - the local server. However, there are several strategies to mitigate this issue. Currently, the project implements a semi-alternative by storing local replicas of the user access lists. In the event of a server failure, these lists remain accessible, even though they won't be updated due to the server being down.

To ensure the continuity of operations, one approach is to implement a variety of backup servers located in different places. If the primary server fails or becomes unreachable, the microcontrollers can automatically switch to a backup server.

A more cost-efficient alternative is the use of transaction logs on the microcontrollers. These logs can store changes made during server downtime and upload the data to the server once the connection is reestablished. This ensures that any updates

made while the server was down are eventually synchronized, maintaining the system's integrity and continuity, without losing information.

The cloud server can also be considered a potential point of failure. However, since we chose Firebase as our cloud server, it benefits from robust failure-handling protocols due to its integration with Google's infrastructure. With that, Firebase provides high reliability and availability by replicating data across multiple servers and data centers, ensuring continuity even in the event of individual server failures.

8. Conclusions

Through this project, we learned how to create and implement a small circuit to address real-world problems. We also discovered new methods for integrating all aspects of ubiquitous computing, including the cloud server, the mobile app, and the sensors/actuators in the microcontroller. Additionally, we developed critical thinking skills for designing a system and organizing its architecture to handle potential failures and downtime.