Two-Factor Identification Electronic Lock

Github: RaspberryPiHotelLockSystem

Quentin Beauchet, Yaacoub Yaacoub, Francesco Diana, Pietro Ventrella, Yann Forner

I. Introduction

The purpose of the project is to design an electronic lock system with 2FA (Two Factor Authentication), working with facial recognition and RFID badge. The system is designed to work in an hypothetical scenario of a hotel or an office where there will be the need of different authorization levels for the rooms. Suppose we deploy our solution in an environment made of multiple floors or buildings, the system will be composed of a central server that will give to the admin some special permissions in case of emergency and several building or floor servers.
Then, there will be a server on each floor that will have its own SQL database containing information about all the authorized users on that floor. Every door on a floor will have an electronic lock composed of one Raspberry Pi and it's camera and RFID tag reader. When a user intends to authenticate via one of the locks, a request will be sent through a local WiFi network from the Raspberry Pi of the door to the floor server that will check if the user is present in the database and if he's authorized. The floor server will answer back with the response, allowing or denying the user to access the room. Suppose that the Database contains the needed records, to unlock the door, the following steps have to be respected:

1) An employee / customer scans his RFID badge.
2) The Raspberry connected to the lock will send the id to the floor server.
3) The local server will check the presence of the id in the database; in case of a positive response it will send back to the Raspberry the associated encoded image in binary format, otherwise it will send a "Not found" message which will stop the process.
4) Then, in case of a positive response, the Raspberry will take a picture of the user and will verify if it matches the image provided by the database. If the response is negative, the process will be stopped, otherwise it will send a signal to the lock to open it.

Further details on our solution and its functionalities will be discussed more in detail in the following sections.
The rest of the paper is divided in the following sections: a proposed solution section where each component of the system will be analyzed in details, with a discussion on the choices made and why we did that; a state of the practice part where we evaluated possible alternatives and competitor to our product, listing possible advantages and disadvantages; a critical analysis of the product, evaluating its weaknesses and strength; an implementation and result section where it is highlighted why our solution is good or bad and finally a conclusion section.

II. Proposed solution

As described in the introduction, the idea is to design and provide a system to manage the room access in a scenario that could be a hotel or an office, where there could be several rooms with different levels of authorization.
Each user needs to get access to rooms using a RFID badge and then an identification and validation process with a face scan will possibly unlock the door.

A. Application Architecture

From an application architecture standpoint (Fig. 6), our product provides an hierarchical architecture where, as previously stated, there is a central server connected to several floor servers and then for each floor every lock is connected to the correspondent floor server.

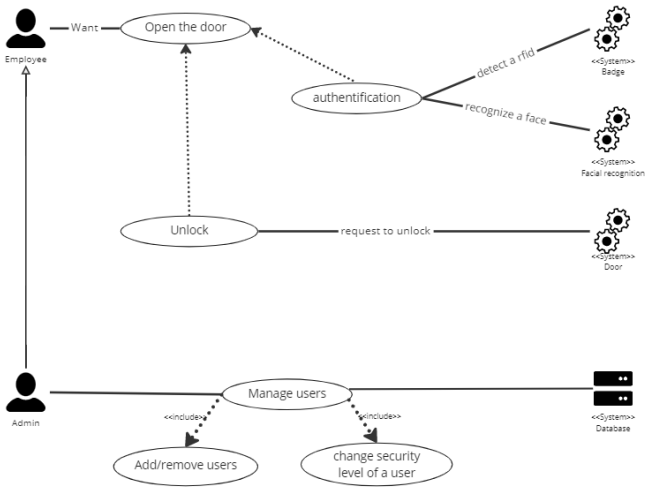


Fig. 1: Use case diagram.

In an ideal working scenario (Fig.1), the user will try to get access to a room scanning its badge. When this happens, the id of the badge will be read by the RFID reader connected to the Raspberry Pi which manages the lock. Then, the Raspberry will send, using a WiFi connection, the user id to the floor server with an HTTP GET request message. An alternative possibility could have been to store the database directly on the Raspberry, but, considering the working scenario,the managing process of the database would have been more difficult because each lock would have to be updated every time a new record is added or removed in the database. Even considering to update in an asynchronous way each Raspberry, managing only a central server, there could have been more risks to have inconsistency between locks

and it would have been more difficult to manage the local network infrastructure, because, in case of more than one building for example, WiFi connection should consider many relays to connect all the Raspberries to a single server, or a wired connection system that is not always available. Instead, using floor servers, it is easier to manage the connection infrastructure, since the distance between servers and devices is smaller, and in addition it is possible to guarantee more data security. In fact, in the ideal scenario, a floor server should be positioned in rooms where only authorized people can have access while if the database is stored on each device, stealing a Raspberry could be a security and privacy issue, since pictures of the users would be stored locally.
Then, for what regards the choice of the networking technology, we might have chosen alternative technologies such as Bluetooth but for ease of implementation and interoperability with the rest of the architecture we decided to use WiFi. After that, considering the connection between the floor servers and the central server, we also decided to use WiFi. Each local server will be uniquely reachable through a socket and it will exchange HTTP messages with the central server. The idea is to use the central server as an endpoint for the admin to control, from a dashboard or an application, the possibility of opening the doors in case of need. In this way, it will be easier for him to control the locks without having to know which server to ask for which door. The only server that will be connected to the internet would be the central server which helps to protect the whole infrastructure. Alternatively if the user wants to he can still communicate directly with the floor servers using the HTTP API routes if he shares the same network.
Finally, supposing that a user is trying to get access and that each step has been executed successfully, the Raspberry will send the signal to the lock to open it using Bluetooth. The choice of using a Bluetooth connection is linked to the choice made to power the Raspberry: we decided to directly connect it to the electrical system of the building, therefore it would have been impossible to physically connect the Raspberry to the lock, for example embedding it in the lock structure. Additionally using Bluetooth means that it would be easier to implement some sort of security to prevent someone to send the open command to the lock than if we were to communicate with it using basic radio communication.

Then, concerning the application retrieval of data from the local server, we deploy for each floor server a MySQL database (Fig. 2) composed of the following tables:

- ***permissions***: this table contains the ***door id***, which works as the primary key and unique identifier of a door, and the permission requested to access the door, identified by the field ***permission id***. In this way it has been possible to set up a hierarchy of permissions for which only authorized people can gain access to specific areas of a building.
- ***doors***: this table contains the information related to the ***id***, which acts as the primary key of the door and its ***name***.
- ***user permission***: here are stored the ***user rfid***, used as primary key to uniquely identify a user, and a ***permission id*** which identifies the authorization level of the user.
- ***door permission***: here are stored the ***door id***, used as primary key to uniquely identify a door, and a ***permission id*** which identifies the authorization level of the user.
- ***users***: it contains all the information related to the users, including their ***RFID*** tag value, ***first name***, ***last name*** and of course the encoded ***picture***, saved as a blob type.

From the point of view of the admin, it will be possible to add, remove or modify users using a command line interface, directly from the floor servers. It is worth noting that each floor server will store only the information related to its floor, in order to limit a possible data breach in case someone manages to get access to the database. It also means that each floor server is independent so the others can still functions if one of them is not working.

B. Example of usage

In (Fig. 3) we described the 3 ways for an user to open a lock:

- Firstly, he can scan his RFID card and the Raspberry at the door will take a picture of him and compare it with the one stored in the database if his card is valid. This is the default way to open a door, the one we built the architecture upon.
- The second way to open a door is to use the RFID card owned by an admin. We called it a master key previously because it can open each lock on a floor without the need to communicate with the floor server. This was needed in case of an issue in the local network, a corrupted Database or anything that could prevent the Raspberry to fetch the user information from the Database. This adds a security flaw to the system but it could be countered by changing the master key each day which would prevent the use of it if it were to be stolen. There are a lot of countermeasures we could take to prevent a malicious use of this key but we did not implement any due to time constraints and the fact that this is a demo.
- Lastly, the doors can be opened remotely using a dashboard of an application, for this demo we do it using the central server API routes directly. Because the state of the door is closed by default and it returns to this state after 5 seconds there is no point in asking a door to close so you can't send this command. A user can only ask for a door to be opened and it will be when the Raspberry fetches the request from the floor-server. Complexity wise it was easier to avoid implementing a server on the Raspberry so we made it fetch the information periodically. This means that you can't send the request to the Raspberry directly but you have to do it from either the corresponding floor server or the central server.

C. Hardware architecture and placement specification

Considering our system from a physical point of view, the solution is composed of a set of Raspberry Pi, each one connected to a lock. Then, suppose you work in a hotel made

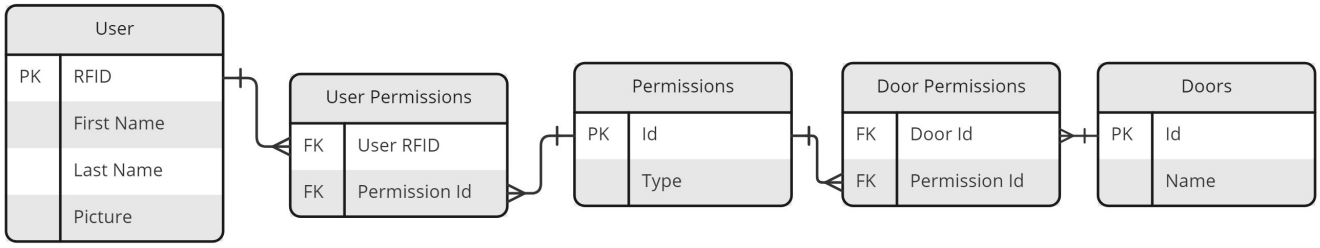


Fig. 2: Database

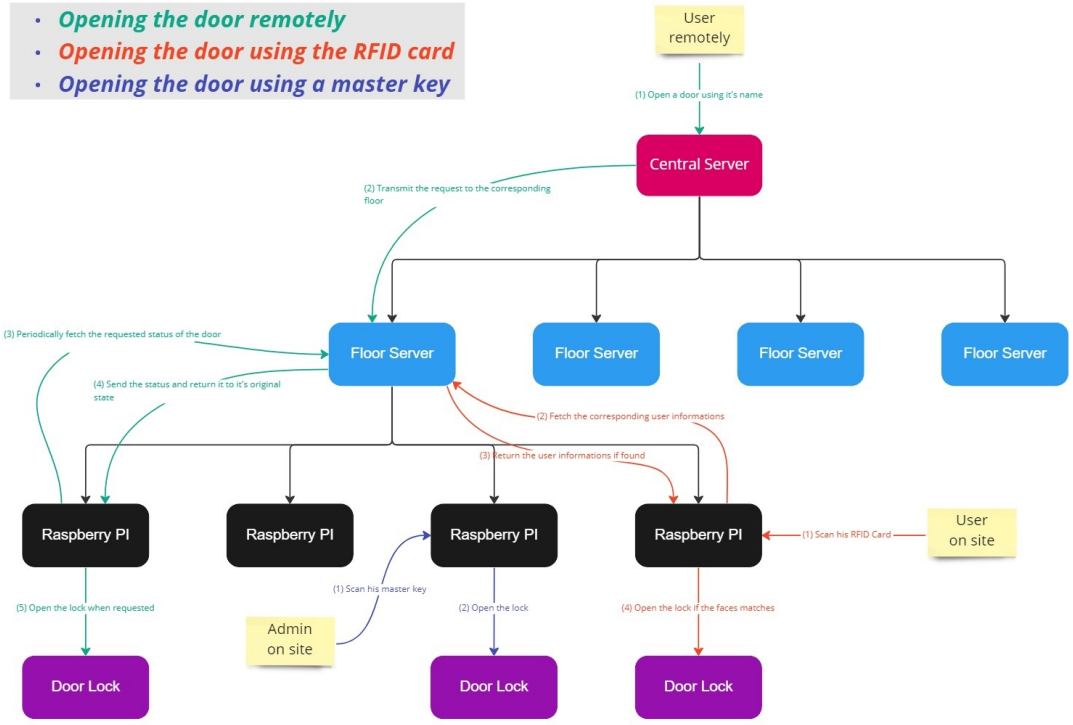


Fig. 3: Lock opening diagram

of more than one floor, our idea is to ensure to have a local server for each floor that stores the database and sends its data to the Raspberries connected to locks. Therefore, there will be a Raspberry Pi with a 64 GB storage that acts as a floor server. This choice was motivated by the possibility of having a period of power outage and also because a Raspberry Pi is powerful enough to manage the requests but also cheaper than a classical server station or PC. For what regards the power outage, our idea is to supply each Raspberry with a battery able to ensure the functionality of the system for 8h. The duration was selected considering Italian law and French data regarding power outage obtained from Statista (Fig. 4). Here, the biggest constraint is given by Italian law [1], which states that in case of cities with more than 50000 people, an electricity company must fix the problem in less than 8 hours, otherwise it will have to refund its customers. Given our setup and evaluating the consumption of a lock, we have a power consumption of:

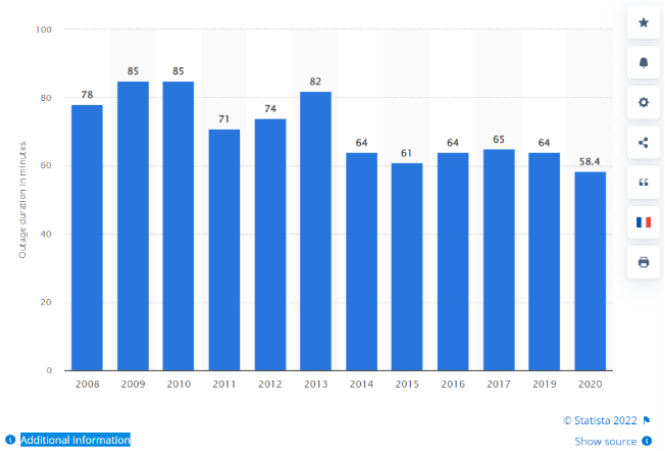


Fig. 4: Average power outage duration in France. [2]

- 900 mA for a Raspberry Pi 3 B+ in working mode.
- 250 mA for the Raspberry Pi Camera Module 2.

- 100 mA for the RFID Reader SCL3711.

Therefore, to power each lock for 8h it is needed, considering a tension of 5 V and an average current of 1250 mA , a power of 50W is needed which corresponds to a battery pack of capacity at least 10,000 mAh .
Lastly, concerning the central server, it could be a PC or a Raspberry Pi, since also in this case the supposed computation will not be enough to overload the Raspberry but, if we consider to implement a dashboard or a web interface, it should probably be better to use a PC.
Now, for the positioning of servers and locks, in the ideal scenario each indoor door should be provided with a lock connected to a Raspberry, and then all of the Pi of a floor should be connected to a floor server. This solution has been planned based on the idea of avoiding using a wired infrastructure for the network, in fact, considering that WiFi range of a Raspberry Pi 3 B+ is, in case of clear line of sight, around 100 m, it would have been difficult to setup only a central server to control one or more buildings, but using multiple floor server this problem is avoided. In addition, using a set of floor servers reduces the possibility of attacks from the Internet since everything works in a local network.
Moving on to an estimate of costs, considering the Raspberry connected to the lock, there are the following estimated prices:

- Raspberry Pi 3 B+ €35,00
- Pi Camera 2 €25,00
- RFID Reader SCL371 €30,00
- Micro SD 64 GB €10,00
- Power bank/battery €20,00

Then, there is also to take into account the cost of the lock, that could vary based on the type of lock chosen. Instead, for what regards the servers, the estimated cost will be given by the Raspberry and the Micro SD card, so it will be around €45,00.

III. State of the practice

In this section there will be an overview of some existing solutions on the market and their advantages and disadvantages, to compare possible competitors with our proposed solution.
One of the most relevant and popular solutions is the Nuki Smart Lock 3.0, which is one of the solutions for keyless access in many scenarios, including smart homes and smart offices. The strengths of this device include its simplicity of installation, as it only requires a quick check, of about three minutes, to see if it can be installed on a door, and as it does not require the presence of an IT specialist to be set up. However, it doesn't have a 2-factor authentication mechanism, requires a periodic change of batteries as its autonomy depends only on them, and is also not scalable.
Another existing solution is Orbitatech's E4041 LCD smart electronic hotel lock. It is a device from a leading company that, with an attractive design given by the presence of an LCD screen that allows a lot of useful information to be displayed, can be integrated in many areas, such as hotels, apartments, dormitories, guesthouses, etc. One of its advantages, apart from its design, is its scalability, as each device can be integrated with a network of devices managed by the establishment and is easily controlled either by using a card (an RFID tag) or an app on the smartphone. Its 4 AA batteries allow it to have a battery life of approximately 12-18 months. However, it too doesn't have a 2-factor authentication mechanism like the solution we propose, and can be purchased at a considerable cost.
The last solution on the market considered is the Video Reader Pro from Openpath, which is a security device that acts as both a door reader and as a video access monitoring device. It can be used via a high-resolution camera to visually monitor entrances and associate video footage with access events and notifications that are shown on the smartphone of the admin. It certainly has the advantage that it can be easily integrated with third-party video management systems and that it is easy to configure, allowing video parameters to be adjusted. However, the camera only operates as a device to monitor access and not as an additional layer of user authentication; moreover, the cost is certainly significant, also due to the multitude of features it offers.

IV. Critical analysis

Like every possible product, there is not a perfect solution that can fit each use case and scenario, therefore in this section we will analyze the strength and weaknesses of our proposed system, considering also possible improvements.
First of all, we decided to not have a physical key to open the door in case of an emergency. Although this choice can be a strength from a security point of view, because there will be no possibility to bypass the 2 factor authentication for someone who is not the admin, there are also downsides, for example, in case of power outage only the admin will be able to manually control each door, deciding to lock them or not, and, in case of low battery, there will be no way to control locks that will stay locked or unlocked until power is restored.
Another weakness of our product, compared to other solutions in the market, is the necessity to have someone who is dedicated to the management of the users: even though it could be possible to modify the database from a CLI, there will still need someone who is in charge to execute this process.
Furthermore, the set up of all the infrastructure has a cost and even if we will be the ones who take care of this problem, there are solutions on the market that could result to be more suitable and competitive than ours due to the ease of use and set up. Furthermore, in case of errors or of a broken Raspberry lock, there will be the need of an IT specialist (internal of the customer company or one of us) that should step in to repair the fault and this could result not only in a waste of time and a disservice for our customers, but also in a waste of money in case of specific sensitive room break.
Finally, considering alternative to the implemented behavior of the lock system we could have shuttled down the camera to save power consumption but this would have had an impact on performances while it would not have had a big impact on the costs; given the cost of electricity in Italy of €0.50 kW/h

on average, it would have resulted in a saving of only €0,45 per month for each lock, which in our opinion was not worth it.
In the next section we will highlight all the potential weaknesses and countermeasures that we decide to adopt in our product.

A. Critical failures and countermeasures

1) Security concerns: In Fig.5 is reported a sequence diagram which describes the authentication process of an user and the door unlock process:

We analyze what are possible weaknesses and vulnerabilities and how we decided to handle them:

(i) RFID security could be bypassed, for example stealing a badge or copying the RFID identifier, but, since it is an intrinsic physical weakness of RFID technology, the only way to counter it is to rely on the 2FA. Even though someone manages to steal a badge, if it is not recognized by the facial scan he will not be able to gain access to the room. However the master key of an admin can open a door without the need to check the face of an user and could be stolen or replicated which would be an issue.
(ii) In a few cases, by putting an image in front of the camera we can validate the facial recognition. Here the proposed countermeasure is to improve our facial recognition algorithm in eventual next releases since, due to lack of time, we had to rely on the one that is currently implemented.

Also we can note that all the requests between two actors can be "potentially" attacked during an exchange of data, for example with a man in the middle attack.
For HTTP data, the most common countermeasure would have been to use HTTPS instead of HTTP which ensures that in case of network sniffing data will be secure. And each API could have an API key and some tokens that would ensure that the user if verified each time he use them.
For the Bluetooth connection between the lock and its relative Raspberry, the best way to secure it would be to set up an encryption system.

2) Architecture weaknesses: Then, there are also other possible weaknesses to explore looking at the architectural diagram in Fig.6:

(i) HTTP API endpoint could be not working, or the connection could be down, not ensuring the communication between the central server and each floor server. In this case, all the systems will continue to correctly work because the database will be stored on the floor servers. The only weakness here is that, in case of power outage the admin will not be able to manually control the locks from afar. This link is only there for helping to manage the doors by someone that does not know how to use API routes and using the dashboard or application add on that we could provide.
(ii) The local connection between floor servers and locks could be down, this means that the Raspberry won't be able to send the authentication data to the server and to receive an answer. Here, we decided to have a master key that is stored as an admin in the database and it will bypass face recognition meaning it will work even if the connection between the Raspberry and the floor server is closed. This master key is fetched each time the Raspberry starts so it can be changed when you restart it and the information in the database has changed. It can be an issue if the Raspberry decides to restart while it can't access the floor server but, since it is supplied by an additional battery, it should never happen unless the user prompts it. This solution is better than storing the master key inside a Raspberry because it could lead to it being stolen potentially giving access to each door in the floor.
(iii) An Electrical outage could happen, in this case, as previously described, the Raspberries and the floor servers will continue to work correctly for additional 8 hours thanks to an emergency battery. If the power is not restored after this period, the admin will manually decide if to close the doors or let them be open. Anyway, the locks considered for the system will be openable from the inside, avoiding the case of someone stuck in a room.
(iv) This issue could happen for 3 different reasons:
- Hardware issue: if this happens, the only possible solution is to change and repair the Raspberry or the lock.
- Bluetooth pairing issues: in this case the admin will just pair the devices again.
- Bluetooth jamming attack: this will interrupt the connection between locks and doors and we did not have a solution for this scenario. In this case, not only Bluetooth connections but also WiFi connection will be affected.

(v) This link is the easiest to ignore as it's only purpose is to help manage the doors for someone who does not know how to use API endpoints manually. Either the central server is down and we can still use the floor-server or the application has an issue and the system will still work while we try to fix it.

V. Implementation

The implementation of the previously described multifactor lock system will be divided in two parts, the server part and the lock control part.

A. Lock Control Part

For this part, we used a Raspberry Pi 3 B+ to do the actual computation that will be used to unlock the door. This choice was motivated by the fact that doing the computation on the server implies that the Raspberry paired with the lock would have to take a picture of the user and then send it to the server. Testing this process, it results to be too slow, taking more than 8s, while, doing the computation on the Pi, it becomes possible to send only an encoded version of the image from the server, resulting in it being faster. Moreover, if the image encoding was done on the Pi, it would still have taken more time than doing the computation.
Considering now the process, first, the user will present his

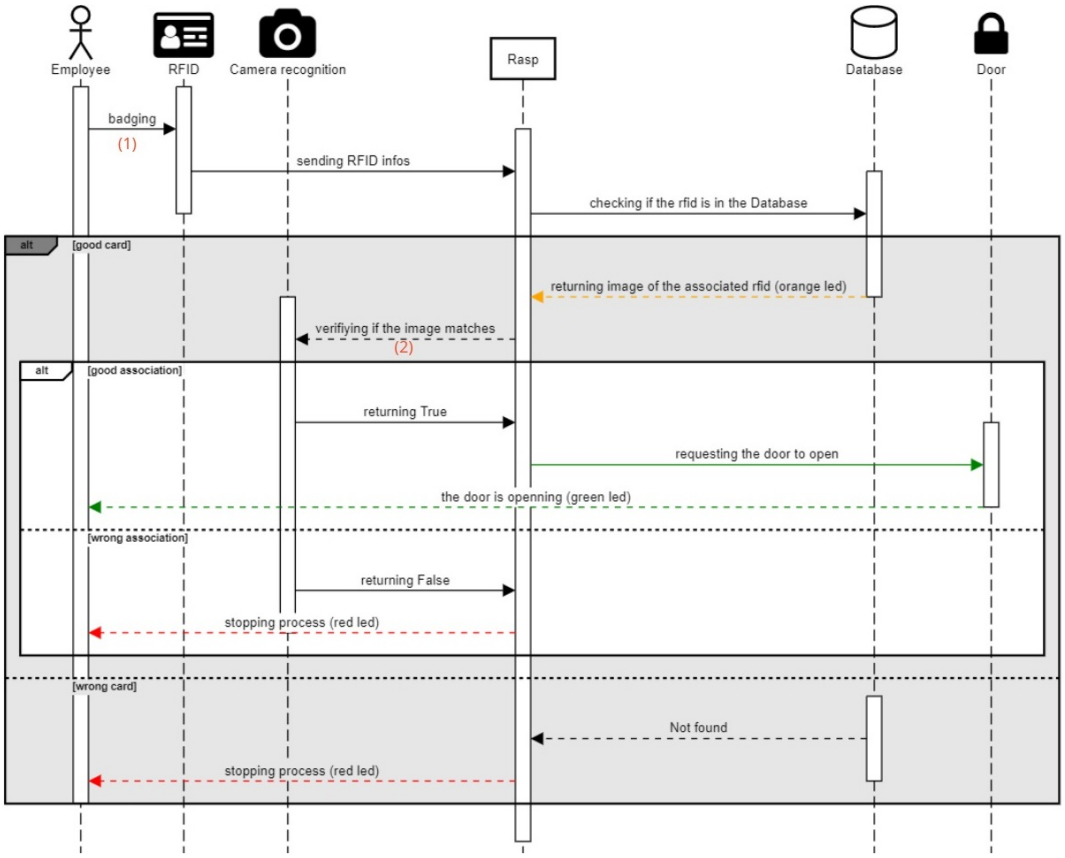


Fig. 5: Sequence diagram

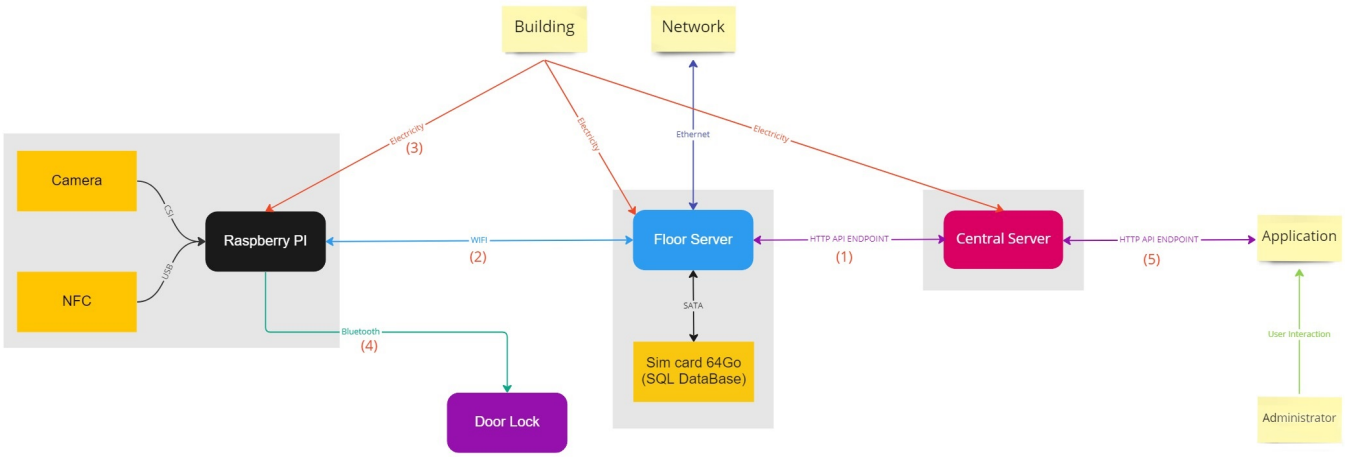
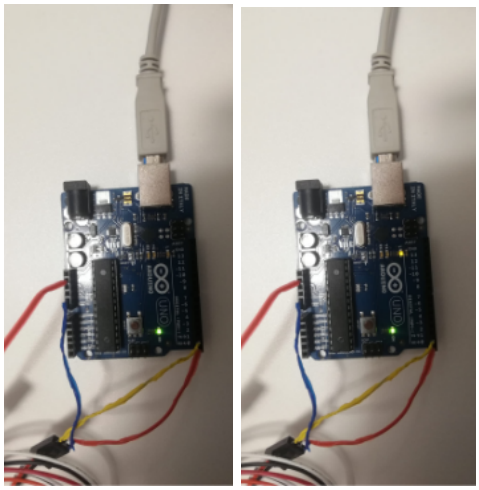


Fig. 6: Architecture diagram

card to the RFID reader connected to the Raspberry Pi via usb. To actually read this card, we make use of the NFC library in Python where we configure it to define an object to read the data sent from the usb port connected to the RFID reader. Checking first if a card is presented by using an infinite loop always checking if data is present on the RFID port.
In case data is available, which means a card is presented, the card ID along with the door ID will be sent to the server to fetch an encoded image of the person that corresponds to the presented tag. In case a photo is found, the Raspberry Pi will get it in the response and then the camera will be activated to take a picture of the person and compare it with the fetched image from the database. To do this comparison first we will use the OpenCV library to handle both fetched and captured images. Then the actual comparison will be made using a Python library called face-recognition which is built and optimized exactly for this purpose. Using this library we will encode both pictures and feed them to the face_recognition.compare_faces() function that will actually do the comparison. In case a match is found the Raspberry Pi will send a signal to the lock to be unlocked. Testing the process, it took 7s on average to open the door after the tag was presented.
In our project, the lock will be simulated by an Arduino Uno and a LED. When the door is unlocked, the LED will be ON (Fig. 7b), else it will be OFF (Fig 7a).



(a) (b)

Fig. 7: Arduino Uno Door Lock simulation - (a) Locked state. (b) Unlocked state.

The communication between the Raspberry Pi and the Arduino Uno board will be made via Bluetooth (Fig. 8). For Raspberry Pi we will use the built in Bluetooth module, as for the Arduino Uno board, we will use the HC06 Bluetooth module the connection between the HC06 Bluetooth module and the Arduino Uno is given below.
So, the Raspberry Pi will send a Bluetooth signal to the Arduino and then the Arduino will open the door (LED will turn on) for 5 seconds. After 5 seconds, the lock will be closed (LED will turn OFF) and the Arduino board will send a signal

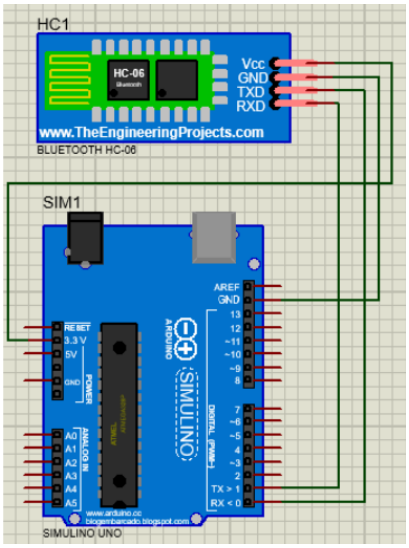


Fig. 8: Arduino with Bluetooth module

informing the Raspberry Pi that the door is closed and it can now continue waiting for the next person to present his badge.

B. Server part

Concerning the server interaction between the Raspberry locks and the servers, we decided to use JavaScript and implement a REST API with Node JS. In case of communication between a floor server and a lock Pi, the Pi will make a GET request to an address which follows this format: {API_URL}/user?rfid={tag}&door_id={DOOR_ID}, where API_URL is the address of the server, DOOR_ID is the id of the door which sends the request and tag is the RFID UID. Then, the floor server listening to the port declared in API_URL will answer the request fetching the UID from the MySQL database with a query and, in case of positive response it will send it back to the user with the corresponding binary encoded picture.
In addition, using a POST request to the server, it is possible to add a new user. To do so, the process will consist of first reading the new tag and then encode the picture of the new user then send these data to be saved in the database.
Finally, the mechanism of door opening could be triggered by the central server that, with a GET request which specifies the door id, could send a request to open a door in case of emergency.

VI. Future improvements and conclusions

Possible future implementations and improvements could foresee the implementation of a more user-friendly admin web dashboard, that could result in a better service for the customer that could avoid having a dedicated IT specialist. Another possible improvement could be the implementation of a spanning tree-like protocol that, in case of problem in the establishment of a connection between a floor server and a device, could avoid the impossibility of opening a door creating a spanning tree topology between the locks at the

same floor in order to retrieve the information. Nevertheless, this should not resolve the problem in case of failure of the floor server, which remains a Single Point of Failure. To avoid having a SPOF in the floor servers, an idea could be to replicate all the data in all the floors, but this could bring to a security potential threat: if someone manages to steal one of the floor servers, it will only obtain data of people saved in that floor while, saving data of every floor in every server would mean potentially give access to all the data.
Furthermore, from a performance point of view, an idea could be to implement a local cache in each lock, in order to speed up the identification process, even though there is always the downside related to possible data loss.
In conclusion, the main advantage of our architecture is the scalability and the capacity to adapt to different use cases, from hotels with tens of rooms, to ones with hundreds of rooms, but also to other domains such as company offices. Moreover, it allows to completely remove a physical key,but continuing to correctly work also in case of power outage, enhancing the security of a building with a lower price compared to other solutions that the market proposes, ensuring a 2FA. While, looking at the other side of the coin, our solution is less easy to deploy and use, maybe slower and with some possible weaknesses that even though were countered, could still bring to a disservice.

VII. Bibliography

[1] https://www.sorgenia.it/guida-energia/interruzione-elettricita-come-funziona-il-risarcimento
[2] https://www.statista.com/statistics/751170/enedis-france-power-cut-duration-electricity-grid/
[3] https://nuki.io/en/business-solutions/
[4] https://www.orbitatech.com/products/Hotel-Locks/975.html
[5] https://www.openpath.com/product/video-reader-pro