Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería Informática

RFID-RC522 Key Reading

Internet of Things

Abel Haro Armero

June 2024

Contents

1 Introduction

This project involves developing an access control system using RFID technology (RFID-RC522 reader). The system will allow user registration and access control using RFID keys and cards. To switch the reader mode between registration and access, Bluetooth communication will be used. Registered users and accesses will be stored in a database accessible via a REST API within a container. For visualization, Ubidots will be used. The entire project is available in a GitHub repository RFID-RC522 Key Reading [1].

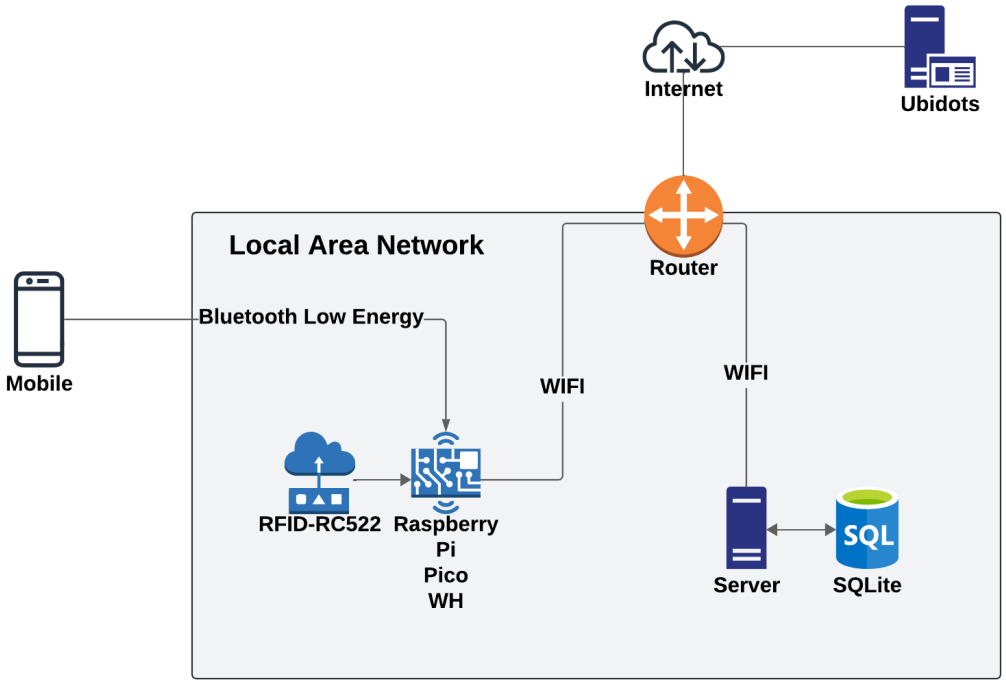
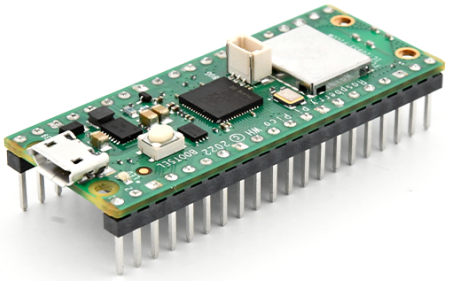


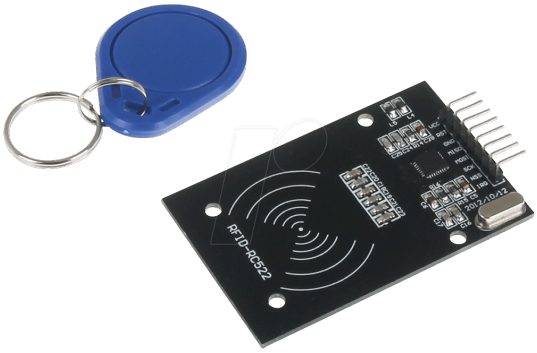
Figure 1: Project diagram.

2 Hardware Used

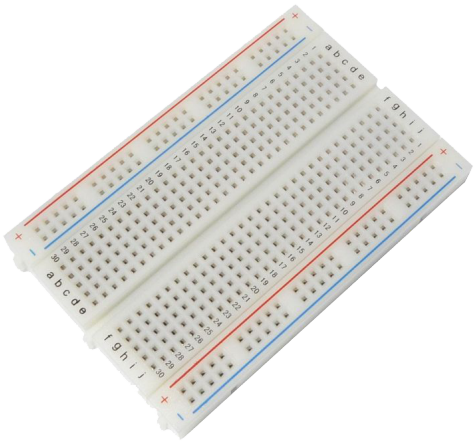
The following hardware was used for the project:



Raspberry Pi Pico WH Microcontroller



RFID-RC522 Radio Frequency Reader



Protoboard

Tri-color LED KY-016 SP00

Dupont Male-Male Cable x3

Dupont Female-Male Cable x5

500Ω Resistor x2

3 Software Used

The following software was used for the project.

3.1 Microcontroller

MicroPython was used as the programming language for the Raspberry Pi Pico WH microcontroller. To establish and manage Bluetooth communication on the microcontroller, the BLE (Bluetooth Low Energy) libraries [3] were utilized. For the mobile device, the Serial Bluetooth Terminal app [5] available on Play Store was used. For reading RFID keys and cards, the MFRC522 library [4] was employed. In communication with the server, a REST API was implemented to enable structured data exchange. Finally, the standard machine library was used to turn LEDs on and off.

3.2 Server

For the server implementation, a Docker container with the base image of Ubuntu was used. Python was installed on the image along with the Flask package to manage server logic through HTTP requests. For data persistence, a Docker volume was used along with an SQLite database. For server development, the code from the Lab 6 - REST [2] was modified.

3.3 Ubidots

The Ubidots platform was used for visualization through a STEM account. Ubidots allows real-time data visualization with a request rate of 1 req/s.

4 Steps to Complete the Project

The following steps should be followed to complete the project.

4.1 Step 1: Pre-installation of Necessary Software

Server Application Installations:

1. Install Thonny.
2. Install Visual Studio Code.
3. Install Docker.
4. Install Python interpreter.

File Installation on Raspberry Pi Pico WH:

1. Install MicroPython firmware:
   (a) Insert the USB into the computer while pressing the BOOTSEL button.
   (b) Open Thonny and follow these steps:

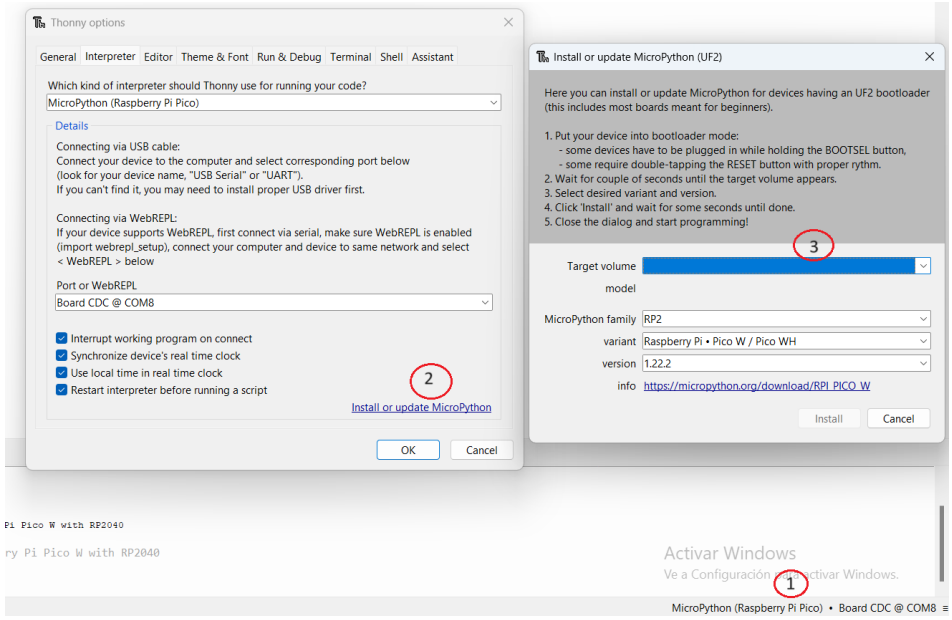


Figure 5: Steps to install the firmware.

2. Copy the contents of the ‘‘`microcontroller`’’ folder from the project to the Raspberry Pi Pico WH.

3. Configure the ‘‘`ssid`’’ and ‘‘`password`’’ variables inside the ‘‘`microcontroller/wifi_connect.py`’’ file with your network's `ssid` and `password`.

Ubidots Registration and Configuration:

1. Create an account on Ubidots Stem.

2. Create a new device.

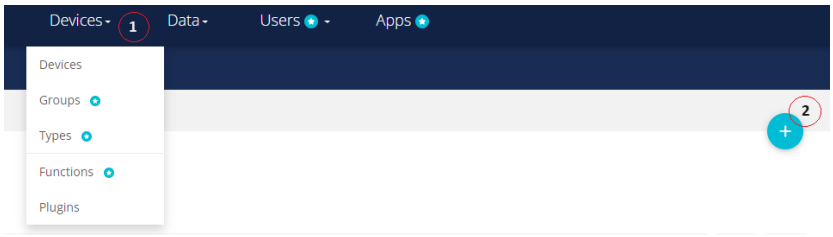


Figure 6: Creating a new device on Ubidots.

3. Add the device name to the ‘‘`DISPOSITIVE_NAME`’’ variable inside the ‘‘`api/ubidots_conf.py`’’ file.

4. Within the device, create two ‘‘`raw variable`’’, one for user registration ‘‘`add_user_register`’’ and another for user access registration ‘‘`add_time_registry`’’.

Figure 7: Creating a variable on Ubidots.

5. Obtain the API token.

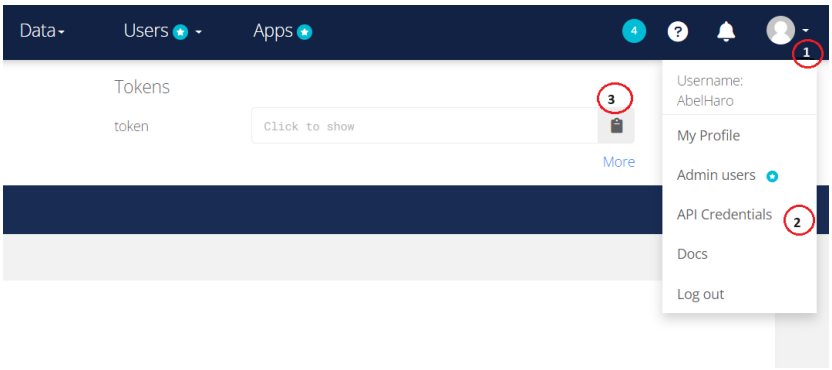


Figure 8: Obtaining the API token on Ubidots.

6. Copy the API token to the ‘‘`TOKEN_UBIDOTS`’’ variable inside the ‘‘`api/ubidots_conf.py`’’ file.

4.2 Step 2: Circuit Assembly

For circuit assembly, the following video was used as a reference ‘RFID RC522 with Raspberry Pi Pico and MicroPython Codes for Simple Access Control’ [6].
Connect the RFID-RC522 reader to the Raspberry Pi Pico WH following this table:

RFID-RC522 Reader	Raspberry Pi Pico WH
VCC	3.3V
RST	GP0
GND	GND
IRQ	Not connected
MISO	GP4
MOSI	GP3
SCK	GP2
SDA	GP1

Table 1: Connections between the RFID-RC522 reader and the Raspberry Pi Pico WH.

Connect the KY-016 SP00 tri-color LED to the Raspberry Pi Pico WH following this table:

KY-016 SP00	Raspberry Pi Pico WH
R	GP13
G	GP12
B	Not connected
-	GND

Table 2: Connections between the KY-016 SP00 tri-color LED and the Raspberry Pi Pico WH.

The circuit assembly should look like the following figure:

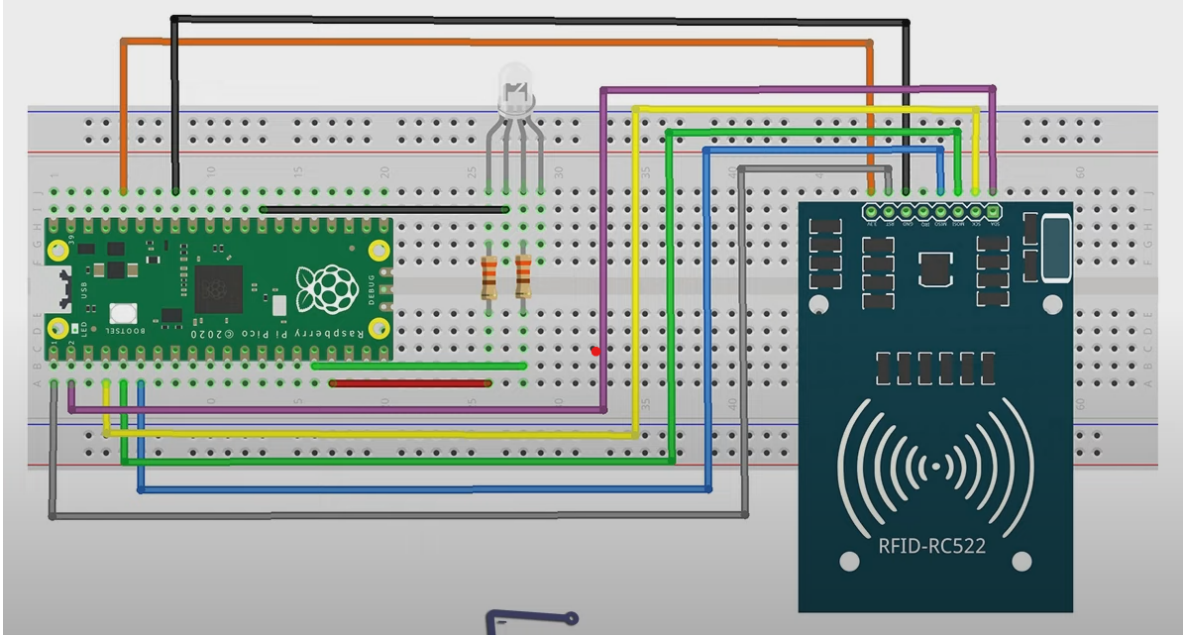


Figure 9: Project circuit diagram.

4.3 Step 3: Mobile Application Configuration

To configure the mobile application, follow these steps:

1. Install the Serial Bluetooth Terminal app on your mobile device.
2. Pair the Raspberry Pi Pico WH with the mobile device via Bluetooth.
3. Open the Serial Bluetooth Terminal app and connect to the Raspberry Pi Pico WH.
4. Send the message ‘‘`ADD_USER_REGISTER`’’ to the Raspberry Pi Pico WH to change the mode to user registration.
5. Send the message ‘‘`ADD_TIME_REGISTRY`’’ to the Raspberry Pi Pico WH to change the mode to access registration.

This is how it’s shown in the Serial Bluetooth Terminal app:

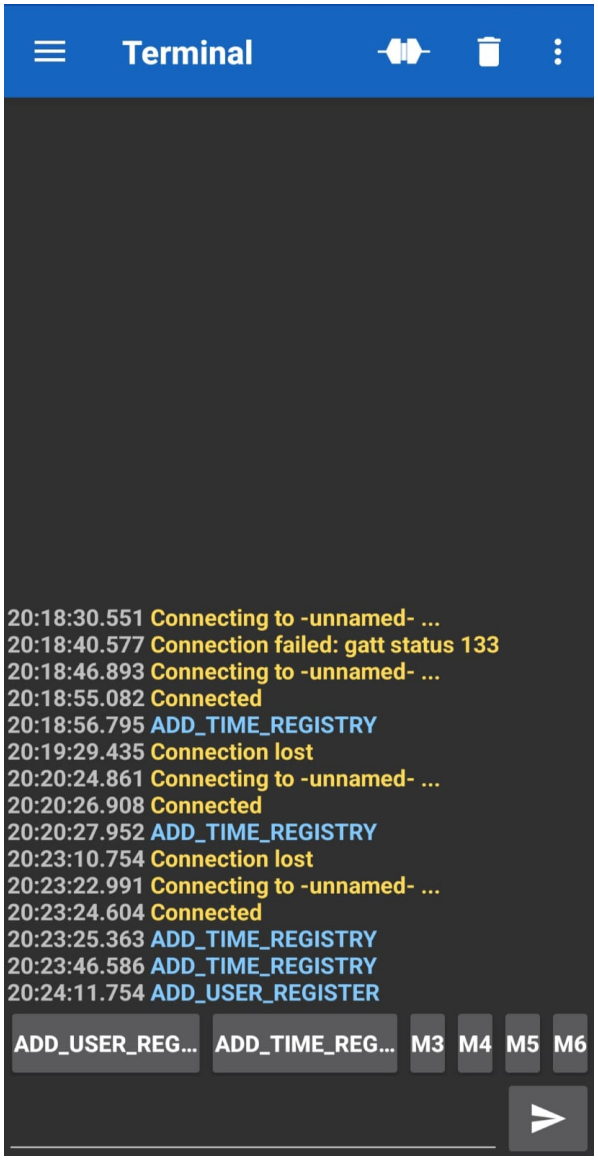


Figure 10: Serial Bluetooth Terminal app.

4.4 Step 4: Project Execution

To execute the project, follow these steps:

1. Connect the Raspberry Pi Pico WH to the computer.
2. Open Thonny and run the ‘‘`microcontroller/main.py`’’ file on the Raspberry Pi Pico WH.
3. Start the Docker daemon.
4. Run the ‘‘`build.bat`’’ file on the server.
5. Open Ubidots and visualize the data.
6. Perform registration and access tests.

This is how it’s shown in Ubidots:

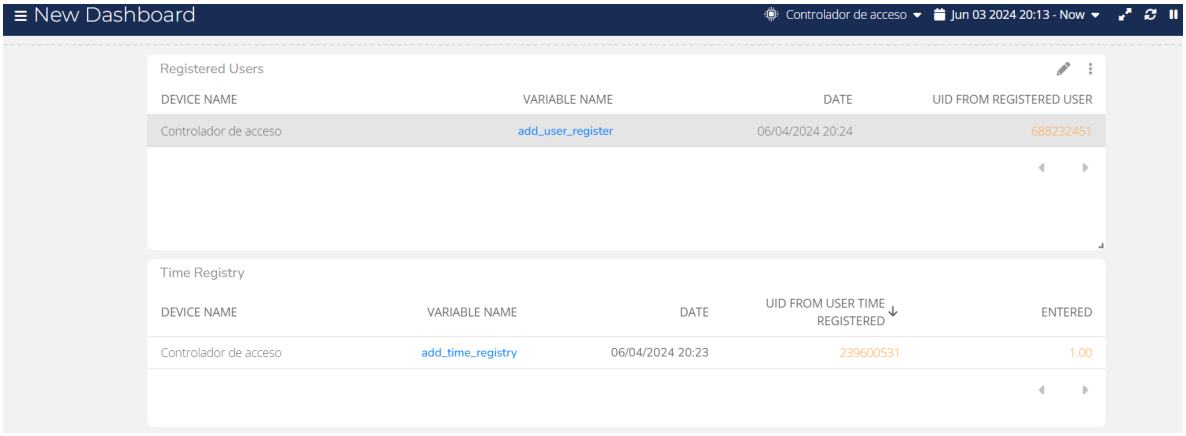


Figure 11: Ubidots data visualization.

5 Project Programming

In this section, the code of the main project files is detailed.

5.1 Microcontroller

The microcontroller code is divided into several files:

5.1.1 main.py

This file is the main script for the microcontroller. It contains the configuration for Bluetooth communication and the function ‘‘`on_rx(data)`’’ where the Bluetooth message is received to change the reading mode. In the ‘‘`main()`’’ function, ‘‘`wifi.connect()`’’ is called to connect to the WiFi network, and the main loop of the program is started. In the loop, the ‘‘`sensor.read_sensor()`’’ function is called to get the UID of the RFID key or card. If the mode is registration, the ‘‘`sender.add_user_register(uid)`’’ function is called, and if the mode is entry registration, the ‘‘`sender.add_time_registry(uid)`’’ function is called. Finally, the ‘‘`led.blink_led(response['api_status'])`’’ function is called to turn on the tricolor LED based on the API response.

```
import bluetooth  # Bluetooth module
from ble.ble_simple_peripheral import BLESimplePeripheral  # BLE module
import time
import wifi_connect as wifi
import data_sending_api as sender
import sensor
import led_control as led
import ubidots

# Initialize Bluetooth Low Energy (BLE) interface and Simple Peripheral
ble = bluetooth.BLE()
sp = BLESimplePeripheral(ble, name="Pico WH")

# Default mode for RFID sensor operation
MODE = 'ADD_USER_REGISTER'  # Default mode is to add user registration

def on_rx(data):
    """
    Callback function for receiving data from BLE.

    Parameters:
        data (bytes): Received data as bytes.

```

```
    Global Variables Modified:
        MODE (str): Updated mode based on received data.
    """
    global MODE  # Access global variable MODE within the function
    print("Data received:", data)

    # Update mode based on received data
    if data == b'ADD_USER_REGISTER\r\n':
        MODE = 'ADD_USER_REGISTER'
    elif data == b'ADD_TIME_REGISTRY\r\n':
        MODE = 'ADD_TIME_REGISTRY'


if __name__ == '__main__':
    # Connect to WiFi
    wifi.connect()

    try:
        while True:
            if sp.is_connected():
                sp.on_write(on_rx)  # Register callback for BLE data reception
            # Read UID from sensor
            uid = sensor.read_sensor()

            # Determine mode and call appropriate API
            if MODE == 'ADD_USER_REGISTER':
                response = sender.add_user_register(uid)  # Call API to add
                    user registration
            elif MODE == 'ADD_TIME_REGISTRY':
                response = sender.add_time_registry(uid)  # Call API to add
                    time registry
            else:
                print('Error: Invalid mode')
                raise Exception('Invalid mode detected')  # Raise an exception
                     for invalid mode

            # Blink LED based on API response status
            led.blink_led(response['api_status'])
            time.sleep(1)
    except KeyboardInterrupt:
        print('Programa abortado con CTRL+C desde main.py')  # Handle keyboard
             interrupt
```

Figure 12: Code for the ‘‘microcontroller/main.py’’ file of the microcontroller.

5.1.2 sensor.py

This file contains the ‘‘read_sensor()’’ function which reads the RFID sensor and returns the UID of the read RFID key or card.

```
from lib.mfrc522.mfrc522 import MFRC522  # RFID reader module
import time  # Time-related functions
import data_sending_api as sender  # Custom API module for data sending

def read_sensor() -> str:
    """
    Function to read RFID sensor and perform actions based on the received
        data.

    Returns:
```

```
        str: UID from card or key readed.
    """
    # Initialize the MFRC522 RFID reader
    reader = MFRC522(spi_id=0, sck=2, miso=4, mosi=3, cs=1, rst=0)

    print("RFID sensor active...\n")

    try:
        while True:
            reader.init()  # Initialize the RFID reader
            (stat, tag_type) = reader.request(reader.REQIDL)  # Request tag
                detection

            if stat == reader.OK:
                (stat, uid) = reader.SelectTagSN()  # Select detected tag
                if stat == reader.OK:
                    # Convert UID bytes to integer for identification
                    identifier = int.from_bytes(bytes(uid), "little", False)
                    print("UID: " + str(identifier))  # Print detected UID

                    return str(identifier)  # Convert UID to string

            time.sleep(1)  # Sleep for 1 second between iterations

    except KeyboardInterrupt:
        print("Program terminated with CTRL+C from sensor.py")  # Handle
            keyboard interrupt
```

Figure 13: Code of the file ‘‘microcontroller/sensor.py’’ of the microcontroller.

5.1.3 data_sending_api.py

This file contains the functions to send data using the server's REST API. It includes the functions ‘‘add_user_register(uid)’’ and ‘‘add_time_registry(uid)’’ to send user registration and access data respectively.

```
    import time  # Standard Python time module
    import ujson  # Module for handling JSON data
    import urequests as requests  # Module for making HTTP requests (alias for
        urequests)

    URL = 'http://192.168.1.2:8888/api/'
    URL_add_user_register = URL + 'user_register/add'
    URL_add_time_registry = URL + 'time_registry/add'
    URL_get_user_registered_by_uid = URL + 'user_register'

    def get_local_time() -> str:
        """
        Returns the timestamp formatted.
        """
        local_time = time.localtime()
        formatted_time = "{:04d}-{:02d}-{:02d} {:02d}:{:02d}:{:02d}".format(
        local_time[0],  # year
        local_time[1],  # month
        local_time[2],  # day
        local_time[3],  # hour
        local_time[4],  # minute
        local_time[5]   # second
        )
        return str(formatted_time)
```

```
24
25      def add_user_register(uid):
26          """
27          Adds the user to the database using a POST request.
28
29          Parameters:
30              uid (str): The UID (Unique ID) of the user to be registered.
31
32          Returns:
33              dict: A dictionary containing the response message from the server
                    .
34          """
35
36          data_sending = {
37              "UID": uid,
38              "user_creation_tstamp": get_local_time()
39          }
40
41          response = requests.post(URL_add_user_register, headers={'Content-Type
                ': 'application/json'}, data=ujson.dumps(data_sending))
42          return ujson.loads(response.content)
43
44      def add_time_registry(uid):
45          """
46          Adds the time registry to the database using a POST request.
47
48          Parameters:
49              uid (str): The UID (Unique ID) of the user to be registered.
50
51          Returns:
52              dict: A dictionary containing the response message from the server
                    .
53          """
54          data_sending = {
55              "UID": uid,
56              "user_registry_tstamp": get_local_time()
57          }
58
59          response = requests.post(URL_add_time_registry, headers={'Content-Type
                ': 'application/json'}, data=ujson.dumps(data_sending))
60          return ujson.loads(response.content)
61
62      def get_user_registered_by_uid(uid):
63          """
64          Retrieves user registration details from the server based on UID using
                a GET request.
65
66          Parameters:
67              uid (str): The UID (Unique ID) of the user to be registered.
68
69          Returns:
70              dict: A dictionary containing the response message from the server
                    .
71          """
72          response = requests.get(URL_get_user_registered_by_uid + '/%s'.format(
                uid))
73          return ujson.loads(response.content)
```

Figure 14: Code of the file ‘‘microcontroller/data_sending_api.py’’ of the microcontroller.

5.2 Server

5.2.1 api.py

This file contains the REST API of the server. It includes functions to connect to the database, add users and access records, retrieve users and access records based on UID, and send user and access records to Ubidots. Only the functions for user registration ‘‘insert_user_register()’’, ‘‘get_user_register_by_uid(uid)’’, and ‘‘add_user_ubidots(uid)’’ are shown in the following figure.

```
import sqlite3
from flask import Flask, request, jsonify
import requests
from api.ubidots_conf import URL_UBIDOTS, TOKEN_UBIDOTS

...

def insert_user_register(user):
    """
    Inserts a new user registration record into the 'user_register' table
        and send it to Ubidots.

    Parameters:
        user (dict): Dictionary containing user information with keys:
                     - 'UID': Unique ID of the user.
                     - 'user_creation_tstamp': Timestamp of user creation.

    Returns:
        dict: Dictionary containing the status of the operation and error
            message (if any).
               Keys:
               - 'api_status': Boolean indicating the success of the
                   operation.
               - 'error': Error message if an error occurred during the
                   operation.
               - 'ubidots_status': HTTP status code of the request to
                   Ubidots.
               - 'UID': Unique ID of the user.
               - 'user_creation_tstamp': Timestamp of user creation.
    """
    inserted_user = {'api_status': False, 'error': None, 'ubidots_status':
         False, 'UID': None, 'user_creation_tstamp': None}
    try:
        conn = connect_to_db()
        conn.row_factory = sqlite3.Row
        cur = conn.cursor()
        cur.execute("SELECT * FROM user_register WHERE UID = ?", (user['
            UID'],))
        rows = cur.fetchall()
        if len(rows) > 0:
            inserted_user['error'] = "User already exists"
            return inserted_user

        cur.execute("INSERT INTO user_register (UID, user_creation_tstamp)
             VALUES (?, ?)",
                    (user['UID'], user['user_creation_tstamp']) )
        conn.commit()
        inserted_user.update(get_user_register_by_uid(user['UID']))
        ubidots_status = add_user_ubidots(user['UID'])
        if ubidots_status == 200:
            inserted_user['api_status'] = True
        else :
```

```
            inserted_user['error'] = "Error adding user to Ubidots"

        inserted_user['ubidots_status'] = ubidots_status
    except:
        conn.rollback()
    finally:
        conn.close()
    return inserted_user

    def get_user_register_by_uid(uid):
    """
    Retrieves a user record from the 'user_register' table based on the
        provided UID.

    Parameters:
        uid (str): The UID (Unique ID) of the user to retrieve.

    Returns:
        dict: A dictionary representing the user record if found,
            otherwise an empty dictionary.
               The dictionary contains keys 'UID' and 'user_creation_tstamp
                   ' with corresponding values.
    """
    user = {}
    try:
        conn = connect_to_db()
        conn.row_factory = sqlite3.Row
        cur = conn.cursor()
        cur.execute("SELECT * FROM user_register WHERE UID = ?", (uid,))
        rows = cur.fetchall()

        # convert row objects to dictionary
        for i in rows:
            user["UID"]  = i["UID"]
            user["user_creation_tstamp"]  = i["user_creation_tstamp"]

    except:
        user = {}
    return user

    def add_user_ubidots(uid):
    """
    Sends an HTTP POST request to Ubidots API to add a user registration.

    Parameters:
        uid (str): The UID (User ID) of the user to register.

    Returns:
        int: HTTP status code of the request.
    """
    data = {
        'add_user_register': {
            'value': 1,
            'context': {
                'UID': uid
            }
        }
    }
    request = requests.post(
        URL_UBIDOTS,
        headers={'X-Auth-Token': TOKEN_UBIDOTS, 'Content-Type': '
            application/json'},
```

```
            json=data
        )
        return request.status_code

        @app.route('/api/user_register/<uid>', methods=['GET'])
        def api_get_user_register_by_id(uid):
            return jsonify(get_user_register_by_uid(uid))

        @app.route('/api/user_register/add', methods=['POST'])
        def api_add_user_register():
            user = request.get_json()
            return jsonify(insert_user_register(user))

        app.run(host="0.0.0.0")
```

Figure 15: Code of the file ‘‘`api/api.py`’’ of the server.

5.2.2 initdb.py

This file contains the initialization of the database. It creates the ‘‘`database`’’ directory if it does not exist and the ‘‘`database.db`’’ file if it does not exist. It creates the ‘‘`user_register`’’ and ‘‘`time_registry`’’ tables if they do not exist.

Column Name	Data Type	Constraints
UID	TEXT	PRIMARY KEY, NOT NULL
user_creation_tstamp	TEXT	NOT NULL

Table 3: Table `user_register` schema.

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_registry_tstamp	TEXT	NOT NULL
UID	TEXT	NOT NULL, REFERENCES user_register(UID)

Table 4: Table `time_registry` schema.

‘‘‘

```
import sqlite3
import os

if __name__ == '__main__':
    try:
        # Create the directory if it doesn't exist
        os.makedirs('database', exist_ok=True)
        print("Database directory created successfully.")

        # Create the database file if it doesn't exist
        open('database/database.db', 'a').close()
        print("Database file created successfully.")

        # Establish connection to the database
        conn = sqlite3.connect('database/database.db')
        print("Connection to the database established successfully.")

        # Create user_register table
        conn.execute('''
```

```
                    CREATE TABLE IF NOT EXISTS user_register (
                        UID    TEXT PRIMARY KEY NOT NULL,
                        user_creation_tstamp TEXT NOT NULL
                    )
        ''')
        print("Table 'user_register' created successfully.")

        # Create time_registry table
        conn.execute('''
                    CREATE TABLE IF NOT EXISTS time_registry (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        user_registry_tstamp TEXT NOT NULL,
                        UID TEXT NOT NULL REFERENCES user_register(UID)
                    )
         ''')

        print("Table 'time_registry' created successfully.")

        # Commit changes
        conn.commit()
        print("Changes committed successfully.")
    except Exception as e:
        print(e)
        print("Table creation failed")
    finally:
        conn.close()
```

Figure 16: Code of the file ‘‘`api/initdb.py`’’ of the server.

5.2.3 build.bat, Dockerfile, and start.sh

These files are necessary for creating the Docker container for the server. The file ‘‘`build.bat`’’ contains the commands for building the image and starting the container. The file ‘‘`api/Dockerfile`’’ contains the instructions for building the container image. The file ‘‘`api/start.sh`’’ contains the instructions for initializing the database and running the API.

```
REM Change directory to the location of the Dockerfile
cd ./api

REM Build Docker image
docker build -t server_rfid .

REM Run Docker container
docker run --rm -it -v database:/home/database -p 8888:5000 --name
    server_rfid server_rfid
```

Figure 17: Code of the file ‘‘`build.bat`’’ of the server.

```
FROM ubuntu

# Instalar Python 3 y Flask
RUN apt update
RUN apt install python3 python3-pip -y
RUN apt install python3-flask -y
RUN apt install python3-requests -y

# Establecer el directorio de trabajo y copiar los archivos
```

```
WORKDIR /home/
COPY initdb.py .
COPY api.py .
COPY start.sh .
COPY ubidots.py .

# Dar permisos para ejecutar el script
RUN chmod +x start.sh

# Exponer el puerto
EXPOSE 5000

# Ejecutar los scripts
CMD ["./start.sh"]
```

Figure 18: Code of the file ‘‘api/Dockerfile’’ of the server.

```
  #!/bin/sh

# Check if the database directory exists, if not, create it (Only for the
    first run)
if [ ! -d "database" ]; then
    mkdir database
fi

# Initialize the database
python3 initdb.py

# Start the API
python3 api.py
```

Figure 19: Code of the file ‘‘api/start.sh’’ of the server.

5.2.4 ubidots_conf.py

This file contains the configuration of Ubidots. It includes the Ubidots URL, device, and API token.

```
DISPOSITIVE_NAME = '' # Device name in Ubidots
URL_UBIDOTS = f'http://industrial.api.ubidots.com/api/v1.6/devices/{
    DISPOSITIVE_NAME}/'
TOKEN_UBIDOTS = '' # Token of the API
```

Figure 20: Code from the file ‘‘api/ubidots_conf.py’’ on the server.

6 Issues Encountered

The following sections describe the issues encountered during the project and the proposed solutions:

6.1 Problem 1: Static Server Address

In order to access the server from the Raspberry Pi Pico WH consistently without having to modify the code, a static IP address must be set for the server. This can be achieved by configuring the router's DHCP settings.

Access the router's configuration page using a web browser. Typically, this can be done by entering the router's IP address, in my case 192.168.1.1, in the address bar and logging in with the username ‘‘`admin`’’ and the password. Once logged in, navigate to the advanced configuration or DHCP settings section to assign a static IP address to the server. In my case, as shown in Figure 21, the router dynamically assigns IP addresses to devices connected in the range of 192.168.1.10 - 192.168.1.150. Therefore, the IP address 192.168.1.2 was assigned to the server to ensure it always has the same IP address and does not collide with other devices.

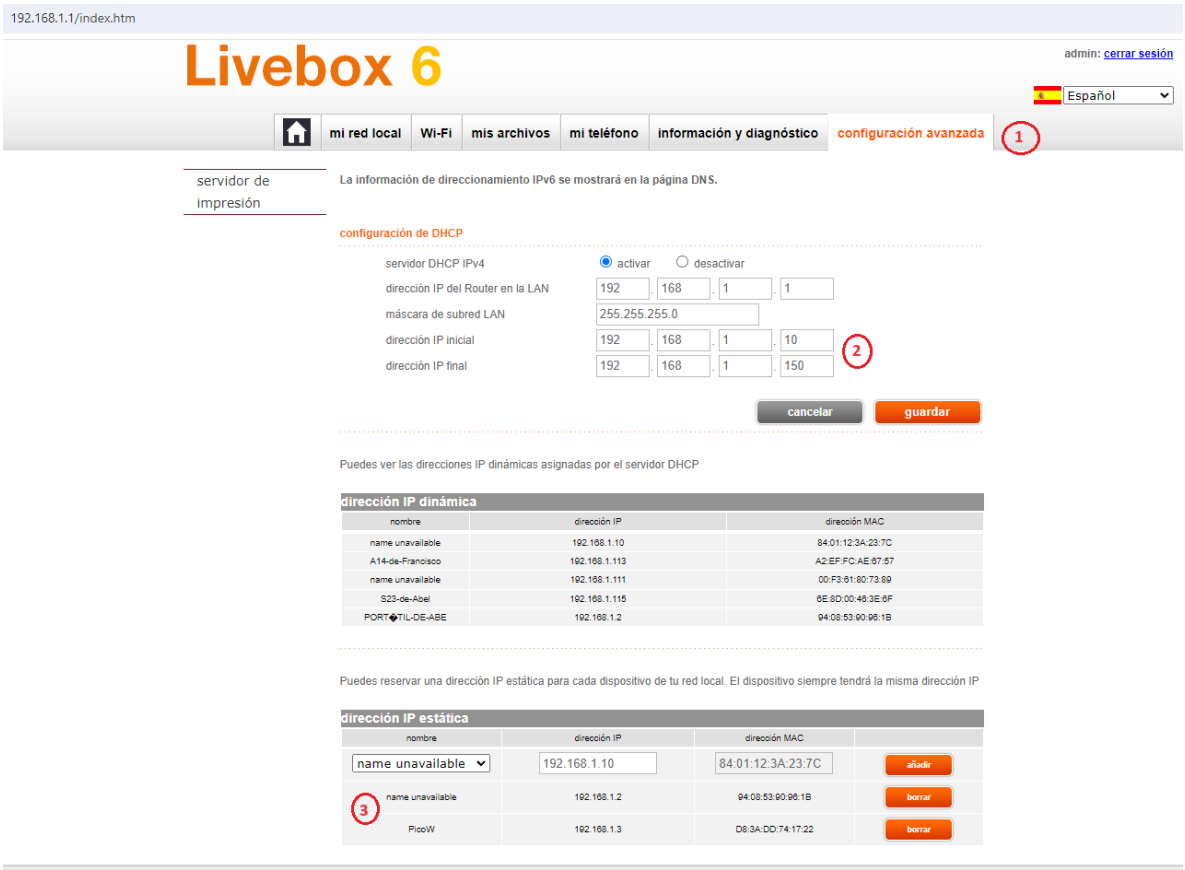


Figure 21: Router network configuration.

6.2 Problem 2: Docker Volume

To maintain database data persistently, a Docker volume must be created. In the Docker Desktop application, navigate to the ‘‘`Volumes`’’ option and create a volume named ‘‘`database`’’ as shown in Figure 22.

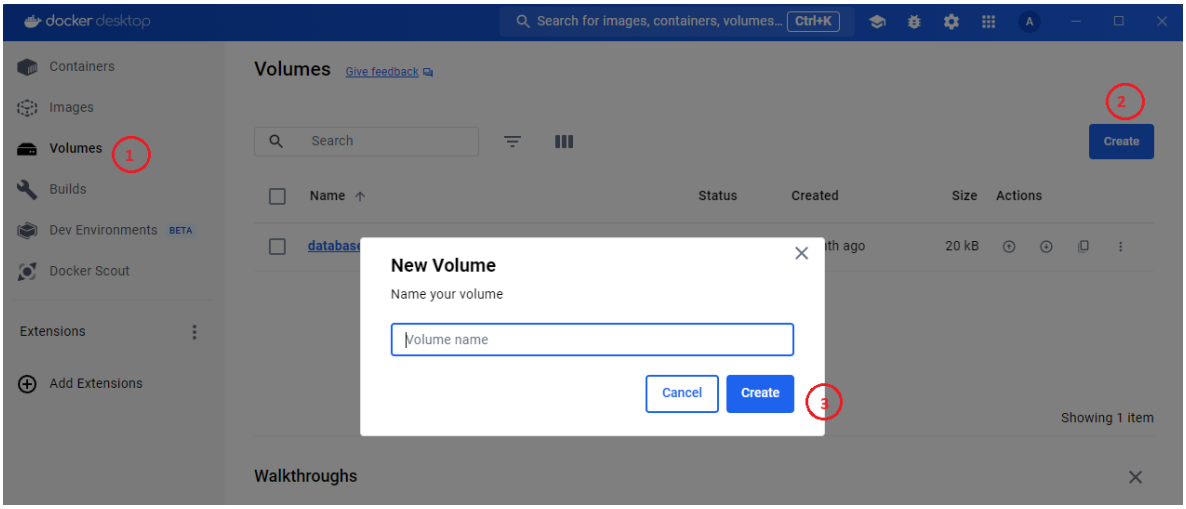


Figure 22: Creating a Docker volume.

In the ‘‘`build.bat`’’ file, the volume is linked to the container using the command ‘‘`-v database:/home/database`’’.

6.3 Problem 3: Sending Data to Ubidots

To send data to Ubidots, the device must be configured in Ubidots and the API token must be obtained. The API token must be sent in the HTTP request header. To do this, refer to the Ubidots documentation to find the request format and send the data in the correct format with the ‘‘`X-Auth-Token`’’ header.

```
def add_user_ubidots(uid):
    ...
    request = requests.post(
        URL_UBIDOTS,
        headers={'X-Auth-Token': TOKEN_UBIDOTS, 'Content-Type': 'application/
            json'},
        json=data
    )
    ...
```

Figure 23: Code for sending data to Ubidots from the file ‘‘`api/api.py`’’.

7 Results Obtained

A access control system using RFID keys and cards has been successfully implemented with a Raspberry Pi Pico WH microcontroller and a server with a SQLite database and a REST API.
User registration and access data has been successfully sent to Ubidots for real-time visualization. The reading mode of the RFID key or card has been successfully changed using Bluetooth Low Energy. A tricolor LED has been successfully turned on based on the API response.
The application has been successfully tested and user registration and access have been correctly recorded.
Future work includes improving the system's security by encrypting data and scalability of the system with multiple Raspberry Pi Pico WHs and RFID readers communicating with the server. The technology for reading RFID keys and cards could also be changed to NFC, a facial recognition system, or fingerprint recognition to enhance the system's security.

References

[1] A. Haro. Rfid-rc522 key reading, 2024. GitHub Repository.

[2] P. Manzoni. Lab 6: Rest, 2024. Notion.

[3] MicroPython. Micropython bluetooth examples, 2023. GitHub Repository.

[4] Daniel Perron. Micropython mfrc522 library, 2022. GitHub Repository.

[5] Sam. Cómo usar ble bluetooth low energy con micropython, November 2023. Video File.

[6] Computadoras y Sensores. Rfid rc522 con raspberry pi pico y códigos en micropython para simple control de acceso, September 2022. Video File.