

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	2
2	Hardware Used	2
3	Software Used	3
3.1	Microcontroller	4
3.2	Server	4
3.3	Ubidots	4
4	Steps to Complete the Project	4
4.1	Step 1: Pre-installation of Necessary Software	4
4.2	Step 2: Circuit Assembly	6
4.3	Step 3: Project Execution	7
4.4	Step 4: Mobile Application Configuration	8
5	Project Programming	9
5.1	Microcontroller	9
5.1.1	main.py	9
5.1.2	sensor.py	11
5.1.3	data_sending_api.py	11
5.2	Server	13
5.2.1	api.py	13
5.2.2	initdb.py	15
5.2.3	build.bat, Dockerfile, and start.sh	17
5.2.4	ubidots_conf.py	18
6	Issues Encountered	18
6.1	Problem 1: Static Server Address	18
6.2	Problem 2: Docker Volume	19
6.3	Problem 3: Sending Data to Ubidots	20
7	Results Obtained	20
	References	21

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]. The demo video is available on YouTube [demo RFID-RC522 Key Reading](#).

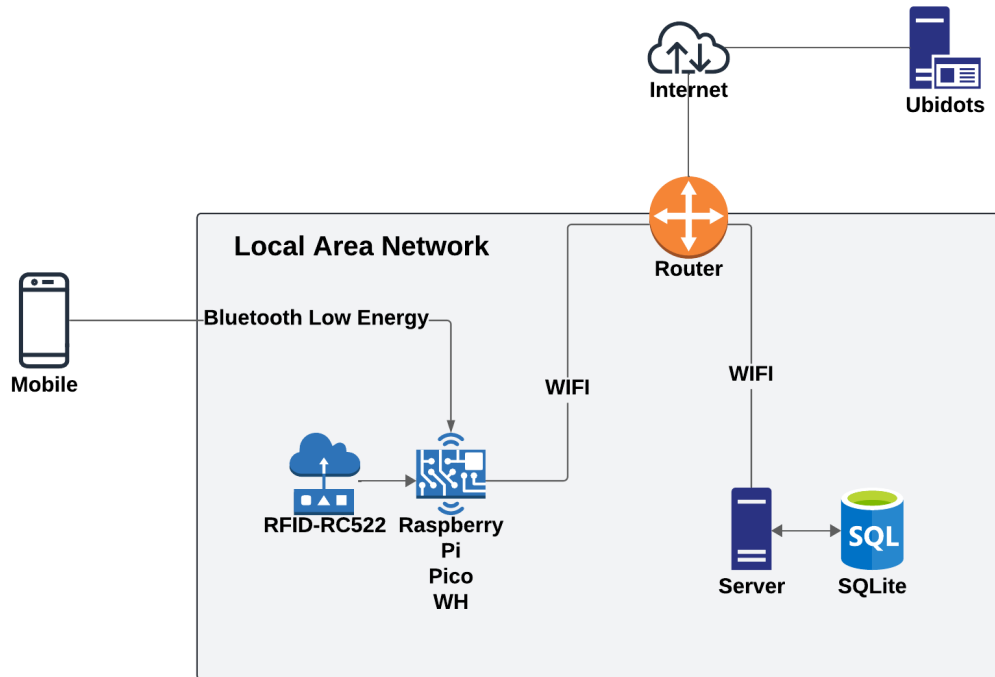
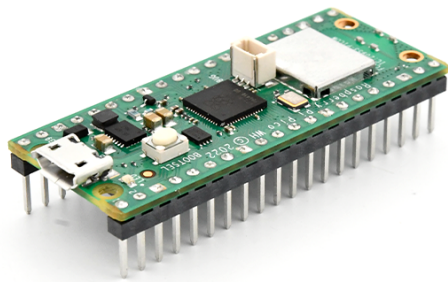


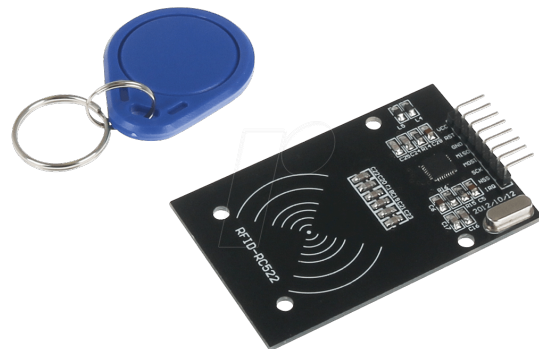
Figure 1: Project diagram.

2 Hardware Used

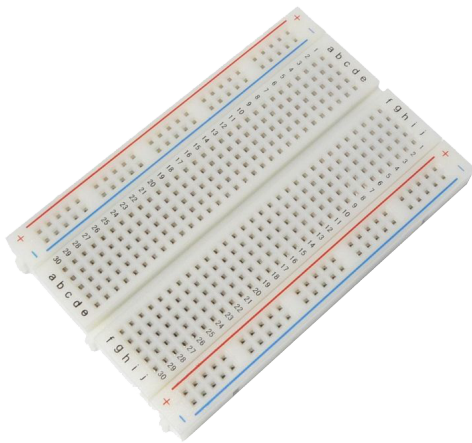
The following hardware was used for the project:



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



500 Ω 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 communication of the mobile device with the microcontroller, 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:

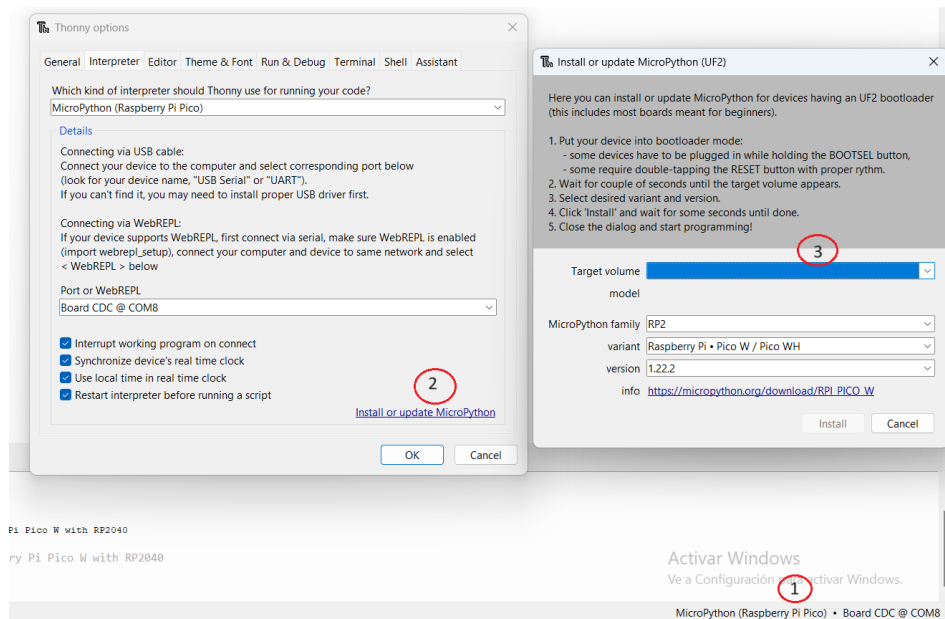


Figure 5: Steps to install the firmware.

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.

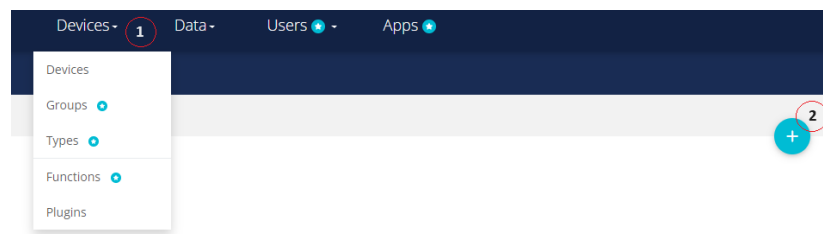


Figure 6: Creating a new device on Ubidots.

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’’.

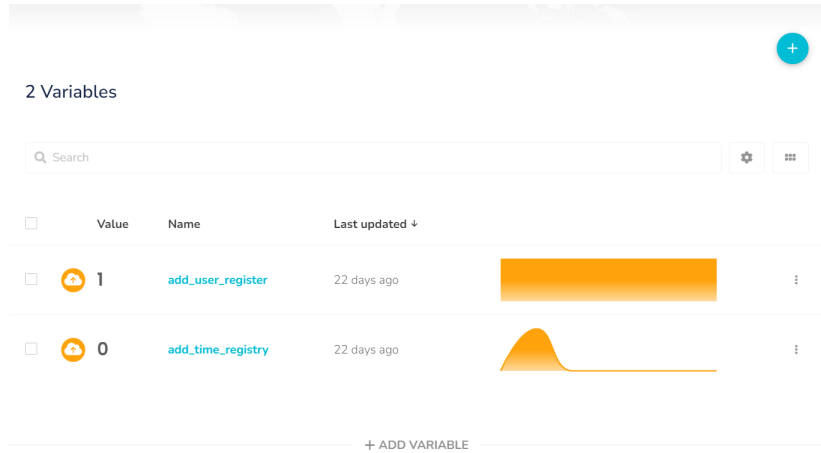


Figure 7: Creating a variable on Ubidots.

5. Obtain the API token.

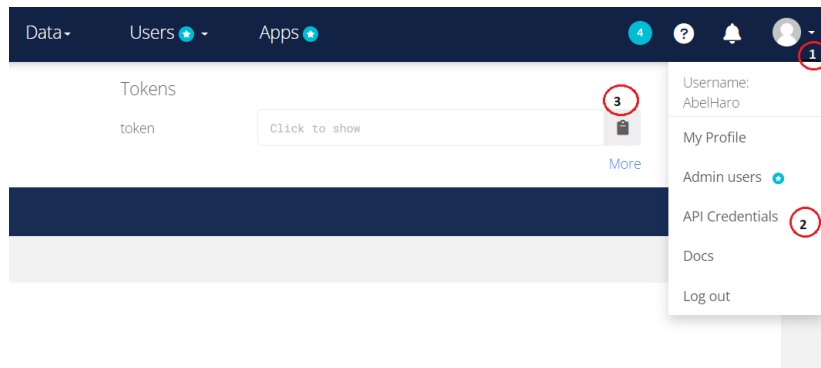


Figure 8: Obtaining the API token on Ubidots.

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:

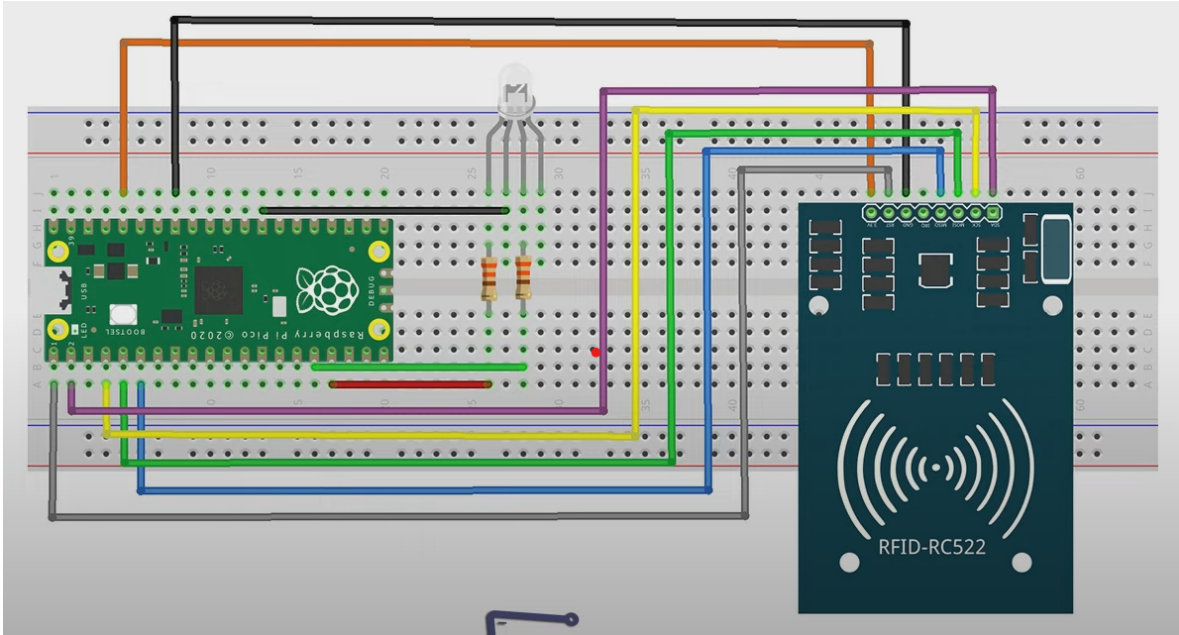


Figure 9: Project circuit diagram.

4.3 Step 3: 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:

The screenshot shows the Ubidots 'New Dashboard' interface. It contains two data tables. The first table, 'Registered Users', has columns for DEVICE NAME, VARIABLE NAME, DATE, and UID FROM REGISTERED USER. The second table, 'Time Registry', has columns for DEVICE NAME, VARIABLE NAME, DATE, UID FROM USER TIME REGISTERED, and ENTERED.

Registered Users			
DEVICE NAME	VARIABLE NAME	DATE	UID FROM REGISTERED USER
Controlador de acceso	add_user_register	06/04/2024 20:24	688232451

Time Registry				
DEVICE NAME	VARIABLE NAME	DATE	UID FROM USER TIME REGISTERED	ENTERED
Controlador de acceso	add_time_registry	06/04/2024 20:23	239600531	1.00

Figure 10: Ubidots data visualization.

In the variable ‘‘add_user_register’’ shows the user UID and the date of the registration. In the variable ‘‘add_time_registry’’ shows the user UID, the date of the access, and a column ‘‘entered’’, which is 1 if the user was registered and 0 if not.

4.4 Step 4: Mobile Application Configuration

To configure the mobile application, follow these steps:

1. Install the [Serial Bluetooth Terminal](#) app on your mobile device.
2. Open the Serial Bluetooth Terminal app and connect to the Raspberry Pi Pico WH.
3. Send the message ‘‘ADD_USER_REGISTER’’ to the Raspberry Pi Pico WH to change the mode to user registration.
4. 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:

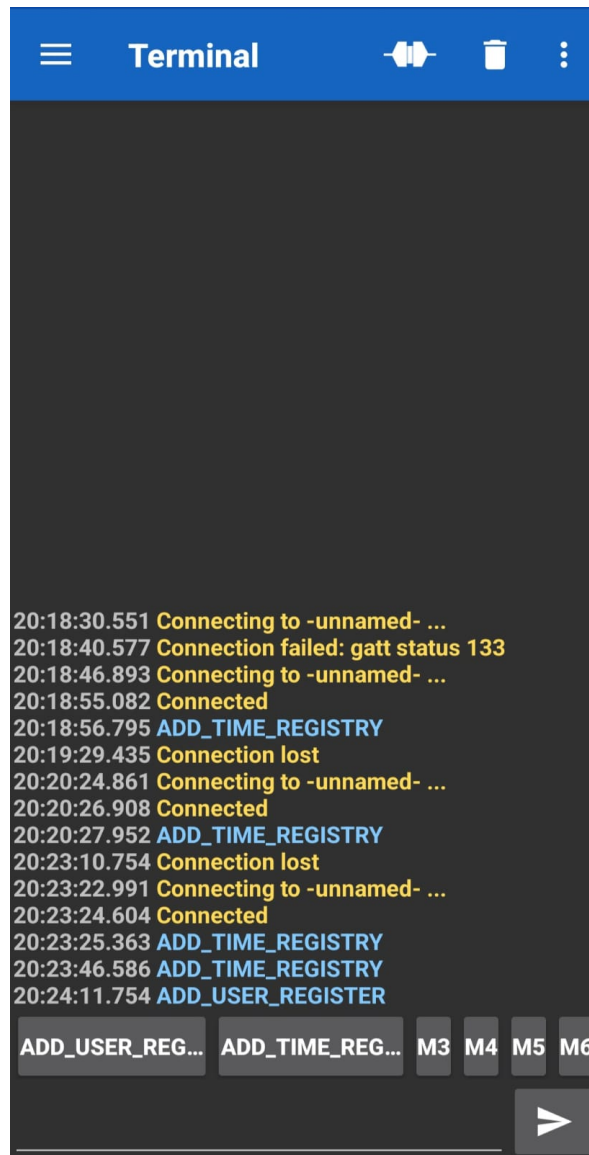


Figure 11: Serial Bluetooth Terminal app.

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.

```

1 import bluetooth # Bluetooth module
2 from ble.ble_simple_peripheral import BLESimplePeripheral # BLE module
3 import time
4 import wifi_connect as wifi
5 import data_sending_api as sender
6 import sensor
7 import led_control as led
8 import ubidots
9
10 # Initialize Bluetooth Low Energy (BLE) interface and Simple Peripheral
11 ble = bluetooth.BLE()
12 sp = BLESimplePeripheral(ble, name="Pico WH")
13
14 # Default mode for RFID sensor operation
15 MODE = 'ADD_USER_REGISTER' # Default mode is to add user registration
16
17 def on_rx(data):
18     """
19     Callback function for receiving data from BLE.
20
21     Parameters:
22         data (bytes): Received data as bytes.
23
24     Global Variables Modified:
25         MODE (str): Updated mode based on received data.
26     """
27     global MODE # Access global variable MODE within the function
28     print("Data received:", data)
29
30     # Update mode based on received data
31     if data == b'ADD_USER_REGISTER\r\n':
32         MODE = 'ADD_USER_REGISTER'
33     elif data == b'ADD_TIME_REGISTRY\r\n':
34         MODE = 'ADD_TIME_REGISTRY'
35
36
37 if __name__ == '__main__':
38     # Connect to WiFi
39     wifi.connect()
40
41     try:
42         while True:
43             if sp.is_connected():
44                 sp.on_write(on_rx) # Register callback for BLE data reception
45                 # Read UID from sensor
46                 uid = sensor.read_sensor()
47
48                 # Determine mode and call appropriate API
49                 if MODE == 'ADD_USER_REGISTER':
50                     response = sender.add_user_register(uid) # Call API to add
51                     user registration
52                 elif MODE == 'ADD_TIME_REGISTRY':
53                     response = sender.add_time_registry(uid) # Call API to add
54                     time registry
55                 else:
56                     print('Error: Invalid mode')
57                     raise Exception('Invalid mode detected') # Raise an exception
58                     for invalid mode

```

```

57         # Blink LED based on API response status
58         led.blink_led(response['api_status'])
59         time.sleep(1)
60     except KeyboardInterrupt:
61         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.

```

1 from lib.mfrc522.mfrc522 import MFRC522 # RFID reader module
2 import time # Time-related functions
3
4 def read_sensor() -> str:
5     """
6     Function to read RFID sensor and perform actions based on the received
7     data.
8
9     Returns:
10         str: UID from card or key readed.
11     """
12     # Initialize the MFRC522 RFID reader
13     reader = MFRC522(spi_id=0, sck=2, miso=4, mosi=3, cs=1, rst=0)
14
15     print("RFID sensor active...\n")
16
17     try:
18         while True:
19             reader.init() # Initialize the RFID reader
20             (stat, tag_type) = reader.request(reader.REQIDL) # Request tag
21                             detection
22
23             if stat == reader.OK:
24                 (stat, uid) = reader.SelectTagSN() # Select detected tag
25                 if stat == reader.OK:
26                     # Convert UID bytes to integer for identification
27                     identifier = int.from_bytes(bytes(uid), "little", False)
28                     print("UID: " + str(identifier)) # Print detected UID
29
30                     return str(identifier) # Convert UID to string
31
32             time.sleep(1) # Sleep for 1 second between iterations
33
34     except KeyboardInterrupt:
35         print("Program terminated with CTRL+C from sensor.py") # Handle
36             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.

```

1 import time # Standard Python time module
2 import ujson # Module for handling JSON data
3 import requests as requests # Module for making HTTP requests (alias for
   requests)
4
5 URL = 'http://192.168.1.2:8888/api/'
6 URL_add_user_register = URL + 'user_register/add'
7 URL_add_time_registry = URL + 'time_registry/add'
8 URL_get_user_registered_by_uid = URL + 'user_register'
9
10 def get_local_time() -> str:
11     """
12     Returns the timestamp formatted.
13     """
14     local_time = time.localtime()
15     formatted_time = "{:04d}-{:02d}-{:02d} {:02d}:{:02d}:{:02d}".format(
16     local_time[0], # year
17     local_time[1], # month
18     local_time[2], # day
19     local_time[3], # hour
20     local_time[4], # minute
21     local_time[5] # second
22     )
23     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     """
55     data_sending = {
56         "UID": uid,
57         "user_registry_tstamp": get_local_time()
58     }

```

```

58     response = requests.post(URL_add_time_registry, headers={'Content-Type': 'application/json'}, data=ujson.dumps(data_sending))
59     return ujson.loads(response.content)
60
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     """
73     response = requests.get(URL_get_user_registered_by_uid + '/%s'.format(
        uid))
    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.

```

1  import sqlite3
2  from flask import Flask, request, jsonify
3  import requests
4  from api.ubidots_conf import URL_UBIDOTS, TOKEN_UBIDOTS
5
6  ...
7
8  def insert_user_register(user):
9      """
10     Inserts a new user registration record into the 'user_register' table
        and send it to Ubidots.
11
12     Parameters:
13         user (dict): Dictionary containing user information with keys:
14             - 'UID': Unique ID of the user.
15             - 'user_creation_timestamp': Timestamp of user creation.
16
17     Returns:
18         dict: Dictionary containing the status of the operation and error
        message (if any).
19         Keys:
20             - 'api_status': Boolean indicating the success of the
        operation.
21             - 'error': Error message if an error occurred during the
        operation.
22             - 'ubidots_status': HTTP status code of the request to
        Ubidots.

```

```

23         - 'UID': Unique ID of the user.
24         - 'user_creation_tstamp': Timestamp of user creation.
25     """
26     inserted_user = {'api_status': False, 'error': None, 'ubidots_status':
27                       False, 'UID': None, 'user_creation_tstamp': None}
28     try:
29         conn = connect_to_db()
30         conn.row_factory = sqlite3.Row
31         cur = conn.cursor()
32         cur.execute("SELECT * FROM user_register WHERE UID = ?", (user['
33                       UID'],))
34         rows = cur.fetchall()
35         if len(rows) > 0:
36             inserted_user['error'] = "User already exists"
37             return inserted_user
38
39         cur.execute("INSERT INTO user_register (UID, user_creation_tstamp)
40                       VALUES (?, ?)",
41                     (user['UID'], user['user_creation_tstamp']))
42         conn.commit()
43         inserted_user.update(get_user_register_by_uid(user['UID']))
44         ubidots_status = add_user_ubidots(user['UID'])
45         if ubidots_status == 200:
46             inserted_user['api_status'] = True
47         else :
48             inserted_user['error'] = "Error adding user to Ubidots"
49
50         inserted_user['ubidots_status'] = ubidots_status
51     except:
52         conn.rollback()
53     finally:
54         conn.close()
55     return inserted_user
56
57 def get_user_register_by_uid(uid):
58     """
59     Retrieves a user record from the 'user_register' table based on the
60     provided UID.
61
62     Parameters:
63         uid (str): The UID (Unique ID) of the user to retrieve.
64
65     Returns:
66         dict: A dictionary representing the user record if found,
67               otherwise an empty dictionary.
68               The dictionary contains keys 'UID' and 'user_creation_tstamp'
69               with corresponding values.
70     """
71     user = {}
72     try:
73         conn = connect_to_db()
74         conn.row_factory = sqlite3.Row
75         cur = conn.cursor()
76         cur.execute("SELECT * FROM user_register WHERE UID = ?", (uid,))
77         rows = cur.fetchall()
78
79         # convert row objects to dictionary
80         for i in rows:
81             user["UID"] = i["UID"]
82             user["user_creation_tstamp"] = i["user_creation_tstamp"]
83     except:

```

```

79         user = {}
80     return user
81
82     def add_user_ubidots(uid):
83         """
84         Sends an HTTP POST request to Ubidots API to add a user registration.
85
86         Parameters:
87             uid (str): The UID (User ID) of the user to register.
88
89         Returns:
90             int: HTTP status code of the request.
91         """
92         data = {
93             'add_user_register': {
94                 'value': 1,
95                 'context': {
96                     'UID': uid
97                 }
98             }
99         }
100         request = requests.post(
101             URL_UBIDOTS,
102             headers={'X-Auth-Token': TOKEN_UBIDOTS, 'Content-Type': 'application/json'},
103             json=data
104         )
105         return request.status_code
106
107     if __name__ == "__main__":
108
109         app = Flask(__name__)
110
111         ...
112
113         @app.route('/api/user_register/<uid>', methods=['GET'])
114         def api_get_user_register_by_id(uid):
115             return jsonify(get_user_register_by_uid(uid))
116
117         @app.route('/api/user_register/add', methods=['POST'])
118         def api_add_user_register():
119             user = request.get_json()
120             return jsonify(insert_user_register(user))
121
122         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.

““

```

1 import sqlite3
2 import os
3
4 if __name__ == '__main__':
5     try:
6         # Create the directory if it doesn't exist
7         os.makedirs('database', exist_ok=True)
8         print("Database directory created successfully.")
9
10        # Create the database file if it doesn't exist
11        open('database/database.db', 'a').close()
12        print("Database file created successfully.")
13
14        # Establish connection to the database
15        conn = sqlite3.connect('database/database.db')
16        print("Connection to the database established successfully.")
17
18        # Create user_register table
19        conn.execute('''
20            CREATE TABLE IF NOT EXISTS user_register (
21                UID TEXT PRIMARY KEY NOT NULL,
22                user_creation_tstamp TEXT NOT NULL
23            )
24        ''')
25        print("Table 'user_register' created successfully.")
26
27        # Create time_registry table
28        conn.execute('''
29            CREATE TABLE IF NOT EXISTS time_registry (
30                id INTEGER PRIMARY KEY AUTOINCREMENT,
31                user_registry_tstamp TEXT NOT NULL,
32                UID TEXT NOT NULL REFERENCES user_register(UID)
33            )
34        ''')
35
36        print("Table 'time_registry' created successfully.")
37
38        # Commit changes
39        conn.commit()
40        print("Changes committed successfully.")
41    except Exception as e:
42        print(e)
43        print("Table creation failed")
44    finally:
45        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.

```
1  REM Change directory to the location of the Dockerfile
2  cd ./api
3
4  REM Build Docker image
5  docker build -t server_rfid .
6
7  REM Run Docker container
8  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.

```
1  FROM ubuntu
2
3  # Instalar Python 3 y Flask
4  RUN apt update
5  RUN apt install python3 python3-pip -y
6  RUN apt install python3-flask -y
7  RUN apt install python3-requests -y
8
9  # Establecer el directorio de trabajo y copiar los archivos
10 WORKDIR /home/
11 COPY initdb.py .
12 COPY api.py .
13 COPY start.sh .
14 COPY ubidots.py .
15
16 # Dar permisos para ejecutar el script
17 RUN chmod +x start.sh
18
19 # Exponer el puerto
20 EXPOSE 5000
21
22 # Ejecutar los scripts
23 CMD ["/start.sh"]
```

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

```
1  #!/bin/sh
2
3  # Check if the database directory exists, if not, create it (Only for the
   first run)
4  if [ ! -d "database" ]; then
5      mkdir database
6  fi
7
8  # Initialize the database
9  python3 initdb.py
10
11 # Start the API
12 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.

```
1 DISPOSITIVE_NAME = '' # Device name in Ubidots
2 URL_UBIDOTS = f'http://industrial.api.ubidots.com/api/v1.6/devices/{
    DISPOSITIVE_NAME}/'
3 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.

192.168.1.1/index.htm

Livebox 6

admin: [cerrar sesión](#)

Español

[mi red local](#) [Wi-Fi](#) [mis archivos](#) [mi teléfono](#) [información y diagnóstico](#) [configuración avanzada](#) **1**

servidor de impresión

La información de direccionamiento IPv6 se mostrará en la página DNS.

configuración de DHCP

servidor DHCP IPv4 ☒ activar ☐ desactivar

dirección IP del Router en la LAN 192 . 168 . 1 . 1

máscara de subred LAN 255.255.255.0

dirección IP inicial 192 . 168 . 1 . 10 **2**

dirección IP final 192 . 168 . 1 . 150

[cancelar](#) [guardar](#)

Puedes ver las direcciones IP dinámicas asignadas por el servidor DHCP

nombre	dirección IP	dirección MAC
name unavailable	192.168.1.10	94:01:12:3A:23:7C
A14-de-Franisco	192.168.1.113	A2:EF:FC:AE:97:57
name unavailable	192.168.1.111	00:F3:01:80:73:89
S23-de-Abel	192.168.1.115	0E:8D:00:48:3E:8F
PORTILLO-DE-ABE	192.168.1.2	94:08:53:90:96:1B

Puedes reservar una dirección IP estática para cada dispositivo de tu red local. El dispositivo siempre tendrá la misma dirección IP

dirección IP estática

nombre	dirección IP	dirección MAC	
name unavailable	192.168.1.10	84:01:12:3A:23:7C	añadir
name unavailable	192.168.1.2	94:08:53:90:96:1B	borrar
PicoW	192.168.1.3	08:3A:DD:74:17:22	borrar

3

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.

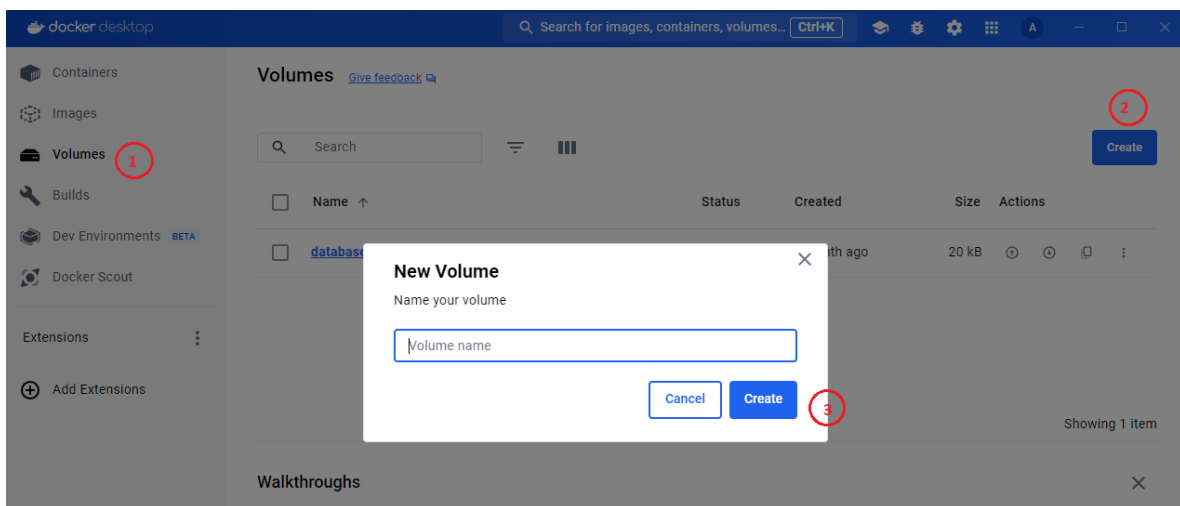


Figure 22: Creating a Docker volume.

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.

```
1 def add_user_ubidots(uid):
2     ...
3     request = requests.post(
4         URL_UBIDOTS,
5         headers={'X-Auth-Token': TOKEN_UBIDOTS, 'Content-Type': 'application/
6             json'},
7         json=data
8     )
9     ...
```

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.

List of Figures

1	Project diagram.	2
5	Steps to install the firmware.	5
6	Creating a new device on Ubidots.	5
7	Creating a variable on Ubidots.	6
8	Obtaining the API token on Ubidots.	6
9	Project circuit diagram.	7
10	Ubidots data visualization.	8
11	Serial Bluetooth Terminal app.	9
12	Code for the ‘‘microcontroller/main.py’’ file of the microcontroller.	11
13	Code of the file ‘‘microcontroller/sensor.py’’ of the microcontroller.	11
14	Code of the file ‘‘microcontroller/data_sending_api.py’’ of the microcontroller. . .	13
15	Code of the file ‘‘api/api.py’’ of the server.	15
16	Code of the file ‘‘api/initdb.py’’ of the server.	16
17	Code of the file ‘‘build.bat’’ of the server.	17
18	Code of the file ‘‘api/Dockerfile’’ of the server.	17
19	Code of the file ‘‘api/start.sh’’ of the server.	18
20	Code from the file ‘‘api/ubidots.conf.py’’ on the server.	18
21	Router network configuration.	19
22	Creating a Docker volume.	19
23	Code for sending data to Ubidots from the file ‘‘api/api.py’’.	20

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.