

Universidad Politécnica de Valencia

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

LECTURA DE LLAVES RFID-RC522

Proyecto Internet de las Cosas

Abel Haro Armero

Junio 2024

Índice

1. Introducción	2
2. Hardware utilizado	2
3. Software utilizado	3
3.1. Microcontrolador	4
3.2. Servidor	4
3.3. Ubidots	4
4. Pasos para realizar el proyecto	4
4.1. Paso 1: Preinstalación de software necesario	4
4.2. Paso 2: Montaje circuito	6
4.3. Paso 3: Ejecución del proyecto	7
5. Programación del proyecto	7
5.1. Microcontrolador	8
5.1.1. main.py	8
5.1.2. sensor.py	9
5.1.3. data_sending_api.py	10
5.2. Servidor	11
5.2.1. api.py	11
5.2.2. initdb.py	13
5.2.3. build.bat, Dockerfile y start.sh	15
5.2.4. ubidots_conf.py	16
6. Problemas encontrados	16
6.1. Problema 1: Dirección estática del servidor	16
6.2. Problema 2: Volumen Docker	17
6.3. Problema 3: Envío de datos a Ubidots	18
7. Resultados obtenidos	18
Referencias	18

1. Introducción

Este proyecto consiste en desarrollar un sistema de control de acceso utilizando tecnología RFID (lector RFID-RC522). El sistema permitirá registrar usuarios y controlar su acceso mediante llaves y tarjetas RFID. Para el cambio de modo del lector, entre registro o acceso, se utilizará comunicación Bluetooth. Los usuarios registrados y los accesos se mantendrán en una base de datos accesible mediante una API REST dentro de un contenedor. Para la visualización se utilizará Ubidots. Todo el proyecto se encuentra en un repositorio de GitHub [Lectura de llaves RFID-RC522 \[1\]](#).

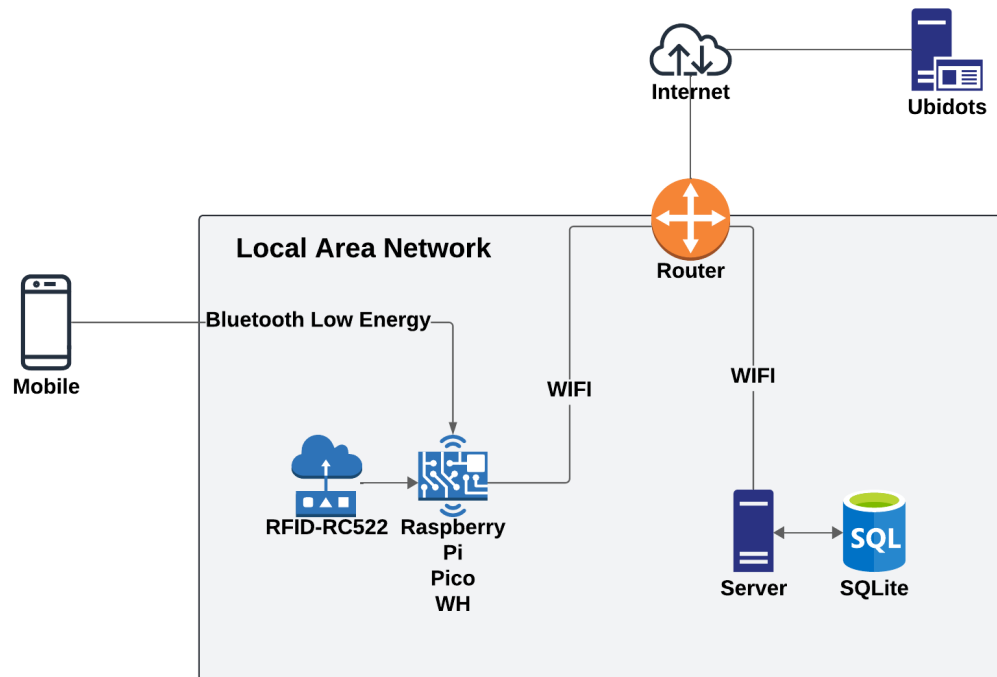
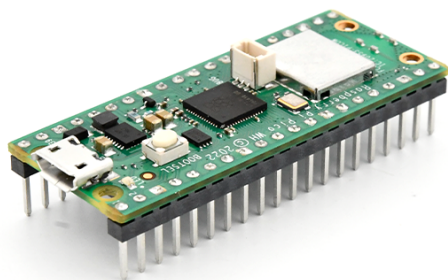


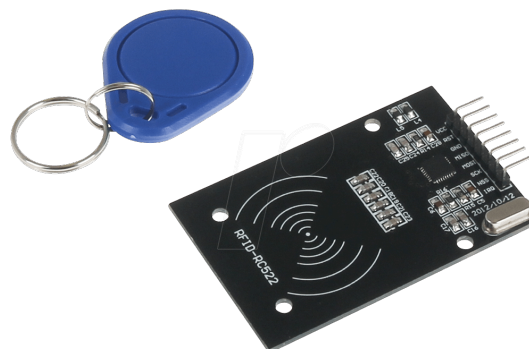
Figura 1: Esquema del proyecto.

2. Hardware utilizado

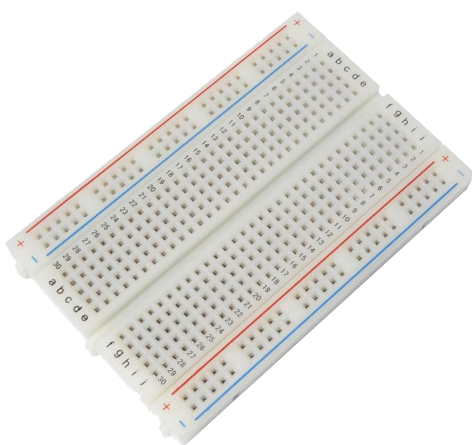
Para el proyecto se ha utilizado el siguiente hardware:



Microcontrolador Raspberry Pi Pico WH



Lector de radiofrecuencia RFID-RC522



Protoboard



Led tricolor KY-016 SP00



Cable Dupont Macho-Macho x3



Cable Dupont Hembra-Macho x5



Resistencia de 500Ω x2

3. Software utilizado

Para la realización del proyecto se ha utilizado el siguiente software.

3.1. Microcontrolador

Como lenguaje de programación para el microcontrolador Raspberry Pi Pico WH se ha utilizado Micropython. Para establecer y gestionar la comunicación Bluetooth en el microcontrolador se hizo uso de las [bibliotecas BLE](#) (Bluetooth Low Energy)[3]. Para el dispositivo móvil se utilizó la aplicación [Serial Bluetooth Terminal](#)[5] disponible en Play Store. Para la lectura de llaves y tarjetas basadas en radiofrecuencia se utilizó la [biblioteca MFRC522](#) [4]. En la comunicación con el servidor se implementó una API REST que permite el envío y recepción de datos de manera estructurada. Por último, se hizo uso de la librería estándar machine para encender y apagar LEDs.

3.2. Servidor

En la implementación del servidor se hace uso de un contenedor Docker con la imagen base de Ubuntu. A la imagen se le instala Python junto con el paquete Flask para gestionar la lógica del servidor mediante solicitudes HTTP. Para la persistencia de datos se emplea un volumen de Docker junto con una base de datos SQLite. Para el desarrollo del servidor se modifica el código de la [práctica 6 - REST](#) [2].

3.3. Ubidots

Para la plataforma de visualización se ha utilizado [Ubidots](#) mediante una cuenta STEM. Ubidots permite la visualización de datos en tiempo real y un envío de 1 req/s.

4. Pasos para realizar el proyecto

Para la realización del proyecto se deben seguir los siguientes pasos.

4.1. Paso 1: Preinstalación de software necesario

Instalaciones de aplicaciones en el servidor:

1. Instalar [Thonny](#).
2. Instalar [Visual Studio Code](#).
3. Instalar [Docker](#).
4. Instalar [intérprete de Python](#).

Instalación de archivos en la Raspberry Pi Pico WH:

1. Instalar firmware de MicroPython:
 - a) Introducir el USB en el ordenador mientras se aprieta el botón BOOTSEL.
 - b) Abrir Thonny y realizar los siguientes pasos:

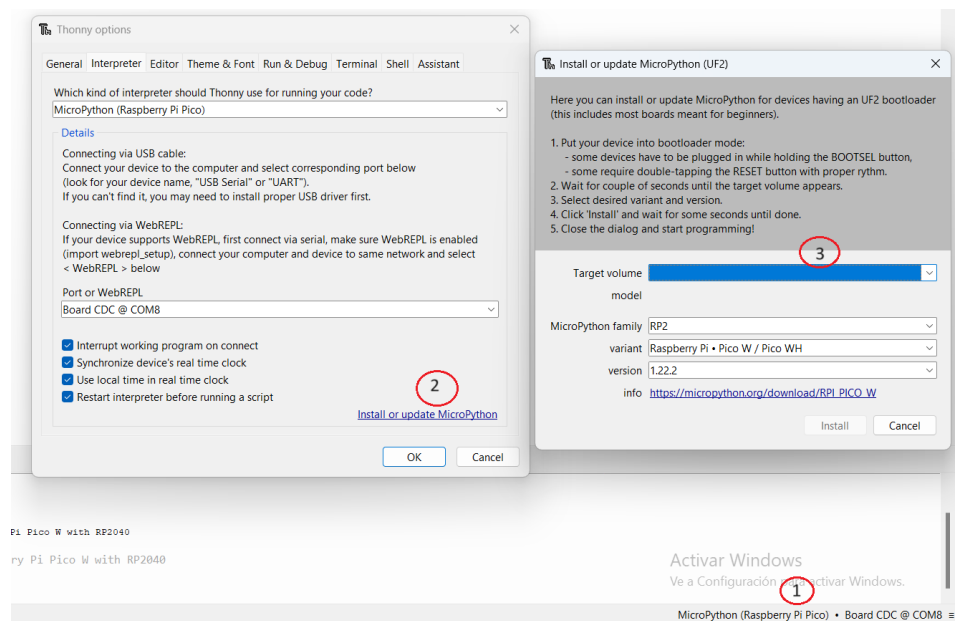


Figura 5: Pasos para la instalación del firmware.

2. Copiar el contenido de la carpeta ‘microcontrolador’ del proyecto en la Raspberry Pi Pico WH.
3. Configurar las variables ‘ssid’ y ‘password’ dentro del archivo ‘microcontrolador/wifi_connect.py’ con el ssid y password de tu red.

Registro y configuración de Ubidots:

1. Crear una cuenta en [Ubidots Stem](#).
2. Crear un nuevo dispositivo.

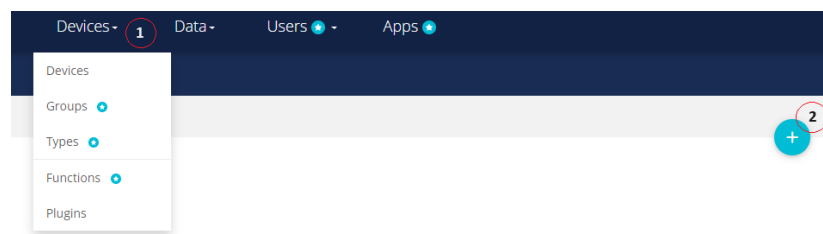


Figura 6: Creación de un nuevo dispositivo en Ubidots.

3. Añadir el nombre del dispositivo a la variable ‘DISPOSITIVE_NAME’ dentro del archivo ‘api/ubidots_conf.py’.
4. Dentro del dispositivo crear dos ‘raw variable’, una para el registro de usuarios ‘add_user_register’ y otra para el registro de accesos de los usuarios ‘add_time_registry’.

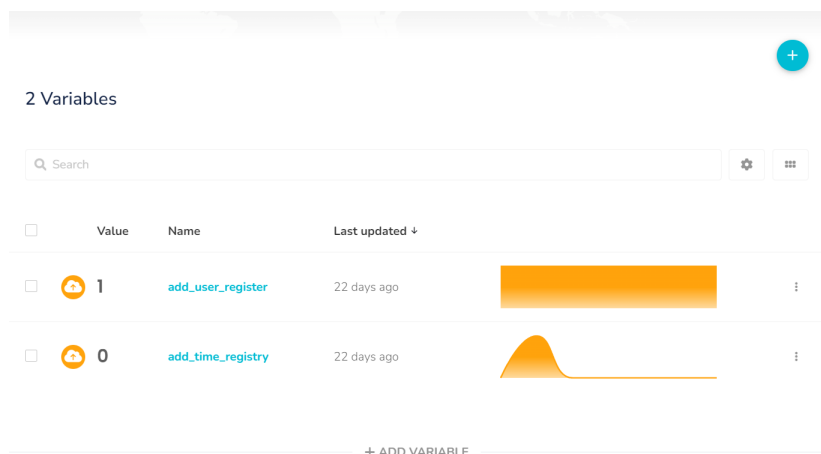


Figura 7: Creación de una variable en Ubidots.

5. Obteber el token de la API.

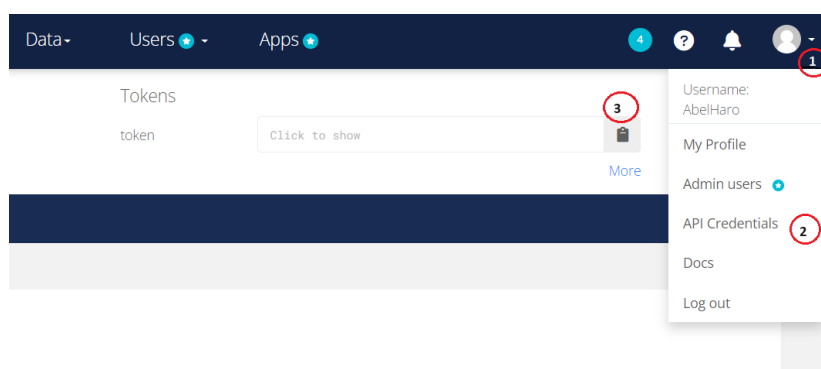


Figura 8: Obtención del token de la API en Ubidots.

6. Copiar el token de la API a la variable ‘‘TOKEN_UBIDOTS’’ dentro del archivo ‘‘api/ubidots.conf.py’’.

4.2. Paso 2: Montaje circuito

Para el montaje del circuito se ha seguido como referencia el siguiente vídeo ‘[RFID RC522 con Raspberry Pi Pico y Códigos en MicroPython para simple Control de Acceso](#)’ [6].

Conectar el lector RFID-RC522 a la Raspberry Pi Pico WH siguiendo la siguiente tabla:

Lector RFID-RC522	Raspberry Pi Pico WH
VCC	3.3V
RST	GP0
GND	GND
IRQ	No conectado
MISO	GP4
MOSI	GP3
SCK	GP2
SDA	GP1

Cuadro 1: Conexiones entre el lector RFID-RC522 y la Raspberry Pi Pico WH.

Conectar el LED tricolor KY-016 SP00 a la Raspberry Pi Pico WH siguiendo la siguiente tabla:

KY-016 SP00	Raspberry Pi Pico WH
R	GP13
G	GP12
B	No conectado
-	GND

Cuadro 2: Conexiones entre el LED tricolor KY-016 SP00 y la Raspberry Pi Pico WH.

El montaje del circuito debe quedar como se muestra en la siguiente figura:

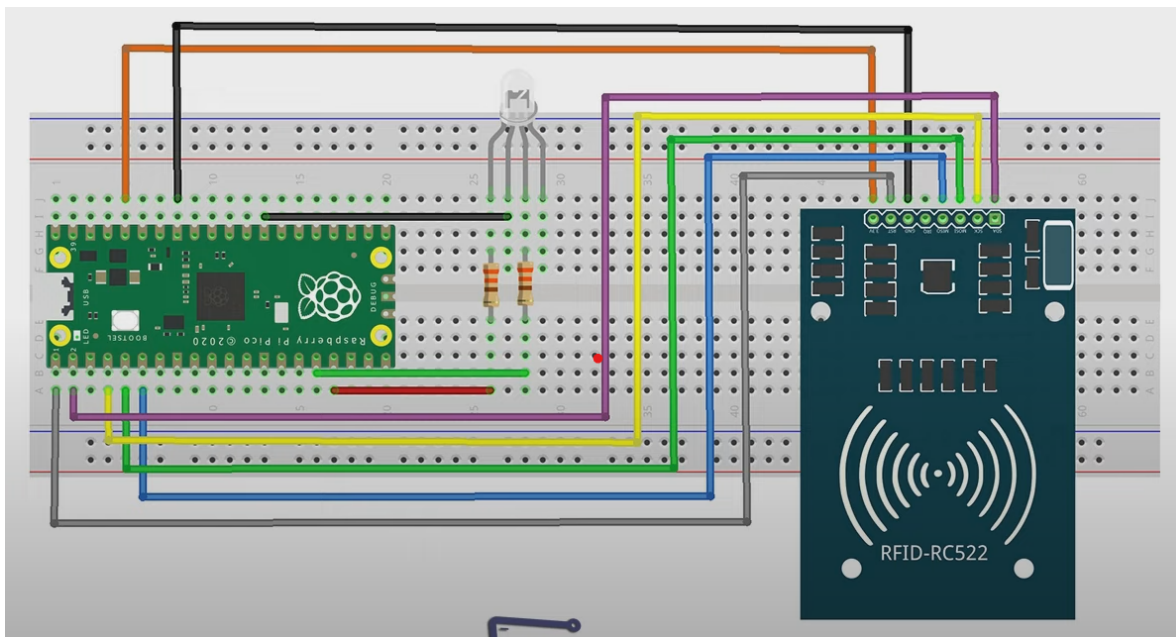


Figura 9: Esquema de conexionado del proyecto.

4.3. Paso 3: Ejecución del proyecto

Para ejecutar el proyecto se deben seguir los siguientes pasos:

1. Conectar la Raspberry Pi Pico WH al ordenador.
2. Abrir Thonny y ejecutar el archivo ‘‘microcontrolador/main.py’’ en la Raspberry Pi Pico WH.
3. Arrancar el daemon de Docker.
4. Ejecutar el archivo ‘‘build.bat’’ en el servidor.
5. Abrir Ubidots y visualizar los datos.
6. Realizar pruebas de registro y acceso.

5. Programación del proyecto

En esta sección se muestra en detalle el código de los archivos de los archivos principales del proyecto.

5.1. Microcontrolador

5.1.1. main.py

Este archivo es el principal del microcontrolador. Contiene la configuración de la comunicación Bluetooth y la función ‘‘on_rx(data)’’ donde se recibe el mensaje bluetooth para el cambio de modo de lectura. En la función ‘‘main()’’ se llama a la función ‘‘wifi.connect()’’ para conectar a la red WiFi y se inicia el bucle principal del programa. En el bucle se llama a la función ‘‘sensor.read_sensor()’’ y recibe el uid de la llave o tarjeta RFID. Si el modo es de registro se llama a la función ‘‘sender.add_user_register(uid)’’ y si es el modo de registro de entrada se llama a la función ‘‘sensor.add_time_registry(uid)’’. Por último se llama a la función ‘‘led.blink_led(response[‘api_status’])’’ para encender el led tricolor en función de la respuesta de la API.

```
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         if sp.is_connected():
43             sp.on_write(on_rx) # Register callback for BLE data reception
44
45         while True:
46             # Read UID from sensor
47             uid = sensor.read_sensor()
48
```

```

49     # Determine mode and call appropriate API
50     if MODE == 'ADD_USER_REGISTER':
51         response = sender.add_user_register(uid) # Call API to add user
52         registration
53     elif MODE == 'ADD_TIME_REGISTRY':
54         response = sender.add_time_registry(uid) # Call API to add time
55         registry
56     else:
57         print('Error: Invalid mode')
58         raise Exception('Invalid mode detected') # Raise an exception for
59         invalid mode
60
61     # Blink LED based on API response status
62     led.blink_led(response['api_status'])
63     time.sleep(1)
64 except KeyboardInterrupt:
65     print('Programa abortado con CTRL+C desde main.py') # Handle keyboard
66     interrupt

```

Figura 10: Código del archivo ‘‘microcontrolador/main.py’’ del microcontrolador.

5.1.2. sensor.py

Este archivo contiene la función ‘‘read_sensor()’’ que lee el sensor RFID y devuelve el uid de la llave o tarjeta RFID leída.

```

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

```

```

32
33 except KeyboardInterrupt:
34     print("Program terminated with CTRL+C from sensor.py") # Handle
        keyboard interrupt

```

Figura 11: Código del archivo ‘microcontrolador/sensor.py’ del microcontrolador.

5.1.3. data_sending_api.py

Este archivo contiene las funciones para enviar datos mediante la API REST del servidor. Se tienen las funciones ‘add_user_register(uid)’ y ‘add_time_registry(uid)’ para enviar los datos de registro de usuario y de acceso respectivamente.

```

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

```

Figura 12: Código del archivo ‘microcontrolador/data_sending_api.py’ del microcontrolador.

5.2. Servidor

5.2.1. api.py

Este archivo contiene la API REST del servidor. Contiene funciones para conectarse a la base de datos, añadir usuarios y registros de acceso, recuperar usuarios y registros de acceso mediante el UID y enviar el registro de usuarios y accesos a Ubidots. Para ello se muestran solo las funciones para el registro de usuarios ‘insert_user_register()’, ‘get_user_register_by_uid(uid)’ y ‘add_user_ubidots(uid)’ en la siguiente figura.

```

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
11     send it to Ubidots.
12
13     Parameters:
14         user (dict): Dictionary containing user information with keys:
15             - 'UID': Unique ID of the user.

```

```

15         - 'user_creation_tstamp': Timestamp of user creation.
16
17 Returns:
18     dict: Dictionary containing the status of the operation and error
19           message (if any).
20           Keys:
21             - 'api_status': Boolean indicating the success of the operation.
22             - 'error': Error message if an error occurred during the
23               operation.
24             - 'ubidots_status': HTTP status code of the request to Ubidots.
25             - 'UID': Unique ID of the user.
26             - 'user_creation_tstamp': Timestamp of user creation.
27
28 """
29 inserted_user = {'api_status': False, 'error': None, 'ubidots_status':
30                 False, 'UID': None, 'user_creation_tstamp': None}
31
32 try:
33     conn = connect_to_db()
34     conn.row_factory = sqlite3.Row
35     cur = conn.cursor()
36     cur.execute("SELECT * FROM user_register WHERE UID = ?", (user['UID']
37                                                             ],))
38     rows = cur.fetchall()
39     if len(rows) > 0:
40         inserted_user['error'] = "User already exists"
41         return inserted_user
42
43     cur.execute("INSERT INTO user_register (UID, user_creation_tstamp)
44                 VALUES (?, ?)",
45                 (user['UID'], user['user_creation_tstamp']) )
46     conn.commit()
47     inserted_user.update(get_user_register_by_uid(user['UID']))
48     ubidots_status = add_user_ubidots(user['UID'])
49     if ubidots_status == 200:
50         inserted_user['api_status'] = True
51     else :
52         inserted_user['error'] = "Error adding user to Ubidots"
53
54     inserted_user['ubidots_status'] = ubidots_status
55 except:
56     conn.rollback()
57 finally:
58     conn.close()
59 return inserted_user
60
61 def get_user_register_by_uid(uid):
62     """
63     Retrieves a user record from the 'user_register' table based on the
64     provided UID.
65
66     Parameters:
67         uid (str): The UID (Unique ID) of the user to retrieve.
68
69     Returns:
70         dict: A dictionary representing the user record if found, otherwise an
71               empty dictionary.
72               The dictionary contains keys 'UID' and 'user_creation_tstamp'
73               with corresponding values.
74
75     """
76     user = {}
77     try:
78         conn = connect_to_db()
79         conn.row_factory = sqlite3.Row

```

```

69         cur = conn.cursor()
70         cur.execute("SELECT * FROM user_register WHERE UID = ?", (uid,))
71         rows = cur.fetchall()
72
73         # convert row objects to dictionary
74         for i in rows:
75             user["UID"] = i["UID"]
76             user["user_creation_tstamp"] = i["user_creation_tstamp"]
77
78     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/
103             json'},
104         json=data
105     )
106     return request.status_code
107
108 @app.route('/api/user_register/<uid>', methods=['GET'])
109 def api_get_user_register_by_id(uid):
110     return jsonify(get_user_register_by_uid(uid))
111
112 @app.route('/api/user_register/add', methods=['POST'])
113 def api_add_user_register():
114     user = request.get_json()
115     return jsonify(insert_user_register(user))
116
117 app.run(host="0.0.0.0")

```

Figura 13: Código del archivo ‘‘api/api.py’’ del servidor.

5.2.2. initdb.py

Este archivo contiene la inicialización de la base de datos. Para ello crea el directorio ‘‘database’’ si no existe y el archivo ‘‘database.db’’ si no existe. Crea las tablas ‘‘user_register’’ y ‘‘time_registry’’ si no existen.

Nombre columna	Tipo de dato	Restricciones
UID	TEXT	PRIMARY KEY, NOT NULL
user_creation_tstamp	TEXT	NOT NULL

Cuadro 3: Tabla user_register esquema.

Nombre columna	Tipo de dato	Restricciones
id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_registry_tstamp	TEXT	NOT NULL
UID	TEXT	NOT NULL, REFERENCES user_register(UID)

Cuadro 4: Tabla time_registry esquema.

```

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()

```

Figura 14: Código del archivo ‘‘api/initdb.py’’ del servidor.

5.2.3. build.bat, Dockerfile y start.sh

Estos archivos son necesarios para la creación del contenedor Docker del servidor. El archivo ‘‘build.bat’’ contiene los comandos para construir la imagen e iniciar el contenedor. El archivo ‘‘api/Dockerfile’’ contiene las instrucciones para construir la imagen del contenedor. El archivo ‘‘api/start.sh’’ contiene las instrucciones para inicializar la base de datos y ejecutar la 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
```

Figura 15: Código del archivo ‘‘build.bat’’ del servidor.

```
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"]
```

Figura 16: Código del archivo ‘‘api/Dockerfile’’ del servidor.

```
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
```

Figura 17: Código del archivo ‘‘api/start.sh’’ del servidor.

5.2.4. ubidots_conf.py

Este archivo contiene la configuración de Ubidots. Contiene la URL de Ubidots, el dispositivo y el token de la API.

```
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
```

Figura 18: Código del archivo ‘‘api/ubidots_conf.py’’ del servidor.

6. Problemas encontrados

En los siguientes apartados se describen los problemas encontrados durante la realización del proyecto y las soluciones propuestas:

6.1. Problema 1: Dirección estática del servidor

Para poder acceder al servidor desde la Raspberry Pi Pico WH siempre de la misma manera sin tener que modificar el código se debe tener una dirección IP estática en el servidor. Para ello se debe configurar la dirección IP estática accediendo a la configuración del router. Se accede mediante un navegador web a la dirección IP del router en mi caso 192.168.1.1 y se introduce el usuario ‘‘admin’’ y la contraseña. Una vez dentro se busca la opción de configuración avanzada para acceder al servidor DHCP y se asigna una dirección IP estática al servidor. En mi caso como se muestra en la figura 19 el router asigna de manera dinámica direcciones IP a los dispositivos conectados en el rango de 192.168.1.10 - 192.168.1.150. Por lo que se asignó la dirección IP 192.168.1.2 al servidor para que siempre tenga la misma dirección IP y esta no pueda colisionar con otros dispositivos.

192.168.1.1/index.htm

Livebox 6

admin: [cerrar sesión](#)

España Español

mi red local Wi-Fi mis archivos mi teléfono información y diagnóstico **configuración avanzada**

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

dirección IP final 192 . 168 . 1 . 150

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:00:49:3E: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	94:01:12:3A:23:7C	<input type="button" value="añadir"/>
name unavailable	192.168.1.2	94:08:53:90:96:1B	<input type="button" value="borrar"/>
PicoW	192.168.1.3	08:3A:DD:74:17:22	<input type="button" value="borrar"/>

Figura 19: Configuración de red del router.

6.2. Problema 2: Volumen Docker

Para poder mantener los datos de la base de datos de manera persistente se debe crear un volumen de Docker. Para ello en la aplicación Docker Desktop se accede a la opción de ‘‘Volumes’’ y se crea un volumen con el nombre ‘‘database’’ como se muestra en la figura 20.

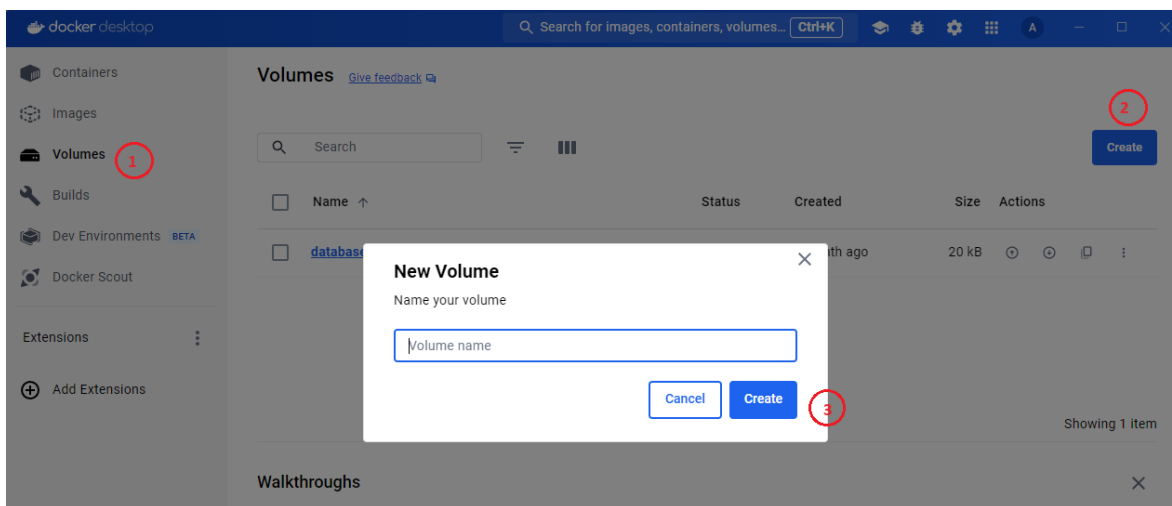


Figura 20: Creación de un volumen de Docker.

Y se observa como en el archivo ‘‘build.bat’’ se enlaza el volumen con el contenedor mediante el comando ‘‘-v database:/home/database’’.

6.3. Problema 3: Envío de datos a Ubidots

Para poder enviar los datos a Ubidots se debe configurar el dispositivo en Ubidots y obtener el token de la API. El token de la API se debe enviar en la cabecera de la petición HTTP. Para ello se debe buscar en la [documentación de Ubidots](#) el formato de la petición y enviar los datos en el formato correcto con la cabecera ‘X-Auth-Token’.

```
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     ...
```

7. Resultados obtenidos

Se ha logrado implementar un sistema de control de acceso mediante llaves y tarjetas RFID con un microcontrolador Raspberry Pi Pico WH y un servidor con una base de datos SQLite y una API REST. Se ha logrado enviar los datos de registro de usuarios y accesos a Ubidots para su visualización en tiempo real. Se ha logrado cambiar el modo de lectura de la llave o tarjeta RFID mediante Bluetooth Low Energy. Se ha logrado encender un LED tricolor en función de la respuesta de la API. La aplicación ha sido probada con éxito y se ha logrado registrar usuarios y accesos de manera correcta.

Como trabajo futuro se propone mejorar la seguridad del sistema mediante la encriptación de los datos y la escalabilidad del sistema con varias Raspberry Pi Pico WH y lectores RFID que se comuniquen con el servidor. También se podría cambiar la tecnología de lectura de llaves y tarjetas RFID por NFC, un sistema de reconocimiento facial o de huella dactilar para mejorar la seguridad del sistema.

Referencias

- [1] A. Haro. Lectura de llaves rfid-rc522, 2024.
- [2] P. Manzoni. Lab 6: Rest, 2024. Notion.
- [3] MicroPython. Micropython bluetooth examples, 2023. Repositorio GitHub.
- [4] Daniel Perron. Micropython mfrc522 library, 2022. Repositorio GitHub.
- [5] Sam. Cómo usar ble bluetooth low energy con micropython, November 2023. Archivo de video.
- [6] Computadoras y Sensores. Rfid rc522 con raspberry pi pico y códigos en micropython para simple control de acceso, September 2022. Archivo de video.