California State Polytechnic University Pomona
Department of Electrical and Computer Engineering
ECE 4303-01

RFID Attendance Project

Group 5
Presented By
Chia Yuan Wu
Ethan Sivasubramanian
Alexander Ov
Luke Yu
August 1, 2023

Table of Contents:

Abstract:

This paper presents an Internet of Things (IoT) attendance tracker project that uses a Raspberry Pi, Arduino UNO, and RFID trackers. The RFID trackers are used to authenticate the identity of students or employees, and the data is then transmitted through internet protocols to a database and a web app. This project provides an alternative way to take attendance that is more efficient and secure than traditional methods.

The project begins by setting up the hardware components. The Raspberry Pi is used as the central controller, and the Arduino UNO is used to read the RFID tags. The RFID tags are programmed with a unique identification number for each student or employee.

The next step is to develop the software. The software is responsible for reading the RFID tags, transmitting the data to the database, and updating the web app. The software is written in Python, and it uses serial communication to transmit the data.

The final step is to deploy the project. The database is hosted on a cloud server, and the web app is hosted on a web hosting service. RFID trackers are deployed in classrooms or workplaces.

The project has been tested successfully, and it has been shown to be a more efficient and secure way to take attendance. The project is also scalable, so it can be easily adapted to larger organizations.

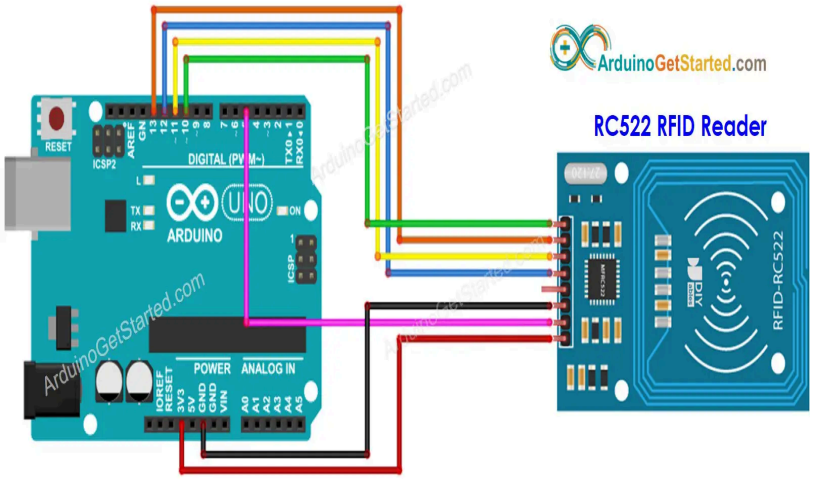

Figure 1: RFID Sensor using a Microcontroller

Introduction:

This paper presents a comprehensive analysis of an integrated system that uses RFID technology to create a seamless data storage and management solution. At the core of this system is the utilization of an Arduino Uno microcontroller, a Raspberry Pi single-board computer, Flask - a lightweight web framework, and the MySQL database. By interconnecting these components, the project aims to showcase the potential solution to RFID data handling, from tag reading to database storage and web-based visualization.

The integration of the RFID reader, Arduino Uno, Raspberry Pi, Flask server, and MySQL database serves as the backbone of a small system designed to streamline the process of capturing and storing RFID tag data. The RFID reader, acting as the gateway to the digital world, interacts with the physical RFID tags, extracting unique identification information from them. The Arduino Uno acts as the intermediary, processing the tag data and forwarding it to the Raspberry Pi for further processing.

The Raspberry Pi, a versatile and compact computing platform, serves as the bridge between the Arduino and the backend Flask server. It enables seamless data transmission and communication, ensuring that the RFID tag information is relayed to the Flask server. The Flask server, with its simplicity and extensibility, plays a pivotal role in data logging and real-time monitoring. It establishes a web-based interface through which users can access and visualize the collected RFID data.

The MySQL database provides the foundation for persistent and organized data storage. It not only receives and stores the RFID tag data but also ensures data integrity and availability. The collaborative operation of these components demonstrates the potential of a comprehensive RFID data management system that can be applied across diverse domains, such as inventory management, access control systems, and attendance tracking.

Throughout this paper, we detail the intricacies of each component's configuration, show the data flow from RFID tag detection to database storage, and present the experimental results that validate the system's efficacy and accuracy.

Experimental Methodology:

This code is written in Arduino, Python, and MySQL programming languages and includes the use of an RFID sensor, a Raspberry Pi, and an Arduino UNO to sense RFID tags and store information on a web server. It was designed to be small, compact, and power efficient in order to be installed anywhere.

The Arduino code starts with including the Serial Peripheral Interface(SPI) and MFRC522 libraries, which enables the code to communicate with the RFID sensor. The code then defines some constants based on pin usage. It then initializes the RFID module and sets the constants to the associated pins. Then it sets up serial communication at a baud rate of 9600, initializes the SPI bus, and starts the MFRC522 library.

Experimental Methodology for the Integration of RFID Reader with Arduino Uno, Raspberry Pi, Flask Server, and MySQL Database

Hardware Setup:

The team connects the MFR 5200 RFID/NFC to Arduino Uno using the proper pin layouts given by the model's datasheet. They powered the Arduino Uno and RFID reader using a suitable power source as well as the necessary impedance. Set up the Raspberry Pi and connect it to the same network as the Flask server.

Arduino Programming:

Arduino code to interface with the RFID reader module.

- Utilize a suitable library (e.g., MFRC522 library) to communicate with the RFID reader.
- Configure the Arduino to read RFID tag data (UID) upon detection.

Raspberry Pi Configuration:

- Install necessary software packages on the Raspberry Pi, including Python and relevant libraries (e.g., RPi.GPIO).
- Establish a serial connection between the Raspberry Pi and Arduino Uno using GPIO pins.

Flask Server Setup:

- Install Flask on the Raspberry Pi using pip.
- Develop a Flask web application with routes for receiving RFID tag data from the Arduino.
- Configure routes for handling data storage and retrieval.

MySQL Database Configuration:

- Set up a MySQL database on a separate server or on the Raspberry Pi itself.
- Create a database schema with a table to store RFID tag data, including fields such as tag ID (UID), timestamp, and any additional relevant information.

Data Flow:

When an RFID tag is detected by the Arduino, the Arduino reads the tag's UID. The Arduino sends the UID to the Raspberry Pi via the established serial connection. On the Raspberry Pi, the Flask server receives the UID as an HTTP request. The Flask server parses the received data and inserts it into the MySQL database, along with a timestamp.

Data Logging and Web Interface:

- Develop Flask routes to retrieve data from the MySQL database for display on the web interface.
- Create HTML templates for the web interface to visualize the stored RFID data.
- Implement real-time updates or periodic data retrieval from the database to keep the web interface current.

Experimental Procedure:

1. Build a circuit that consists of an RFID reader, Arduino UNO, and Raspberry Pi.
2. Place various RFID tags within the detection range of the RFID reader.
3. Power on the Arduino Uno and Raspberry Pi.
4. Launch the Flask web application on the Raspberry Pi.
5. Monitor the Flask web interface to observe real-time RFID tag detection and data logging.

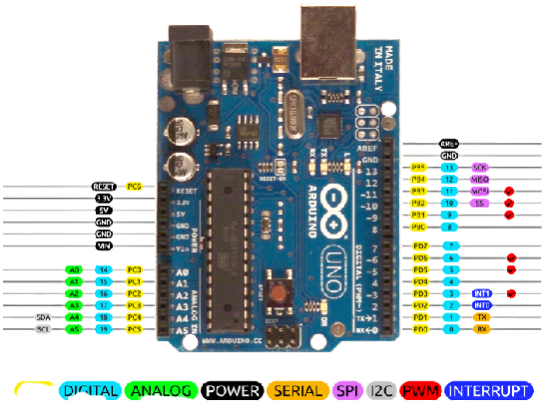

Figure 2: Diagram of Arduino Development Board

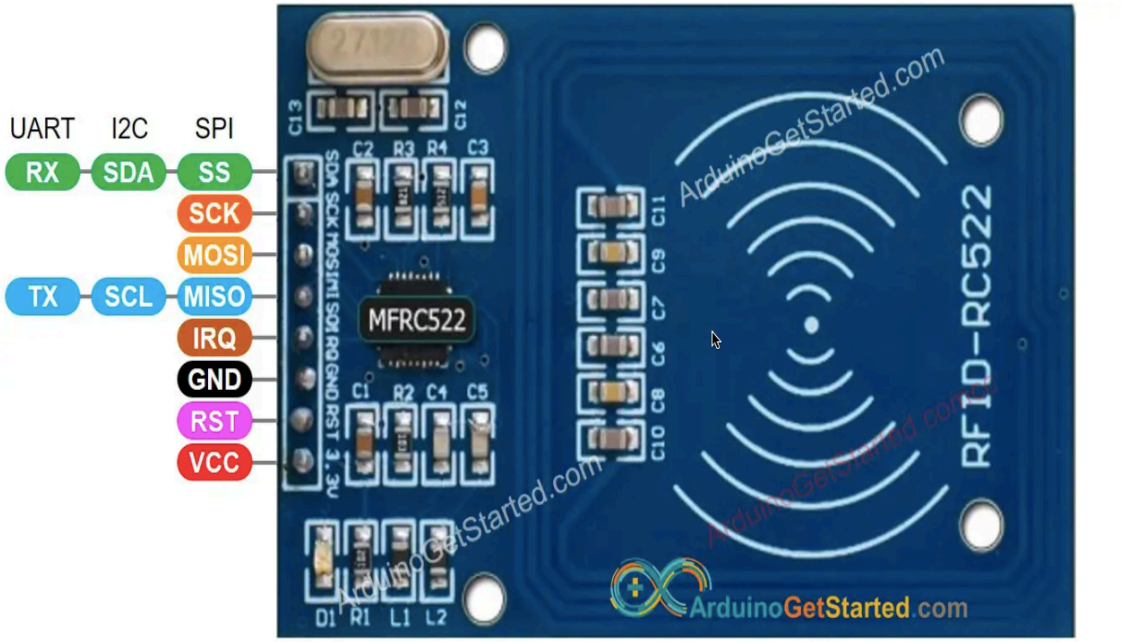

Figure 3: MFR 5200 RFID/NFC Sensor

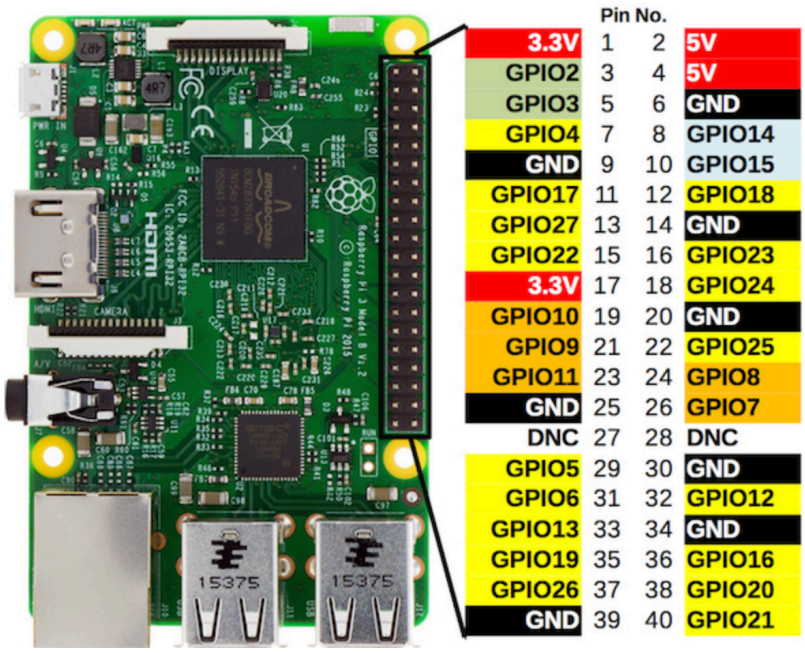

Figure 4: Diagram of Raspberry Pi 3

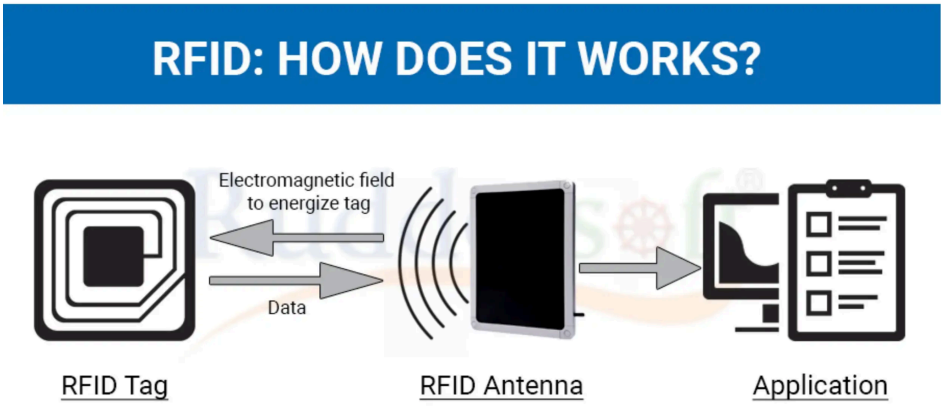

Figure 5: Diagram of RFID communication

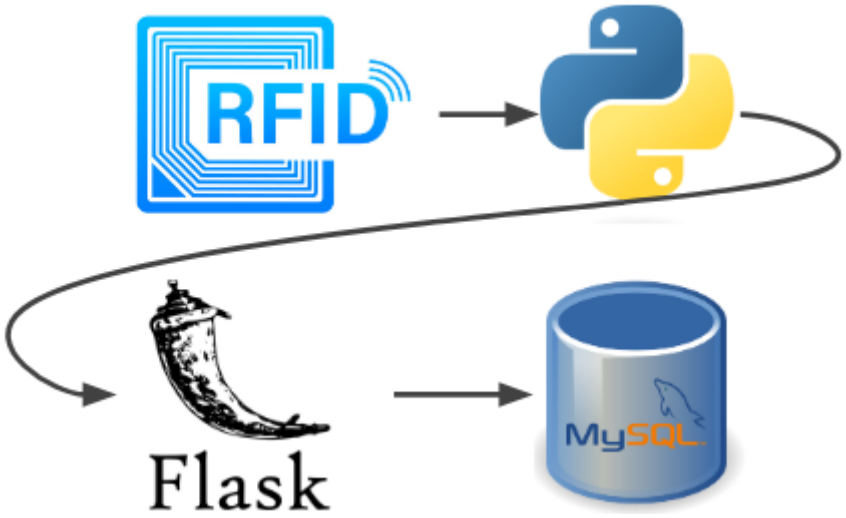

Figure 6: Diagram of our Software Flow

Experimental Results:

This study presents the experimental results of a project that integrates Raspberry Pi, Arduino UNO, and RFID trackers, which creates a system that efficiently reads and stores RFID tags into an open-source database. The execution of transmission of data is operated by the Python and RFID codes which incorporate appropriate protocols at the software level.

Figure **8** shows the steps in order to ensure the readings for the RFID tags by the Arduino UNO. The RFID program enables the Arduino to read the unique identifier or serial number of each tag and display the data in the serial monitor. As each unique UIDs are shown, it is confirmed that the Arduino has correctly read and stored the data. After the tags of the RFID are successfully read by the Arduino, the Python script function is to transfer the UIDs from the Arduino UNO to the Rasberry Pi. This is transmission is due to the fact that the Rasberry Pi functions as the central controller that incorporates the connection of the Flask server and mySQL while the Arduino UNO's function is to simply read the serial inputs. Figure **9** shows the outputs of the Python script which are for intaking serial data from Arduino UNO. The serial tags are the same as the ones read by the Arduino UNO which can then verify a successful transmission.

To finally implement the RFID project, the Python scripts that connects the Flask server, MySQL, and for intaking serial data and posting it to the web server were implemented. The script also included the time and date with the serial numbers that were read by the Arduino UNO as shown in the Python script in **Figure 10**. The program first retrieves the UID's from the function data.get() and then checks for possible errors by boolean expressions. Finally, the data and time is retrieved and then is logged into the MySQL database. In Figure 10, the Logs of the MySQL database accurately stored and output the correct data read by the Arduino.

The experimental results of this project showcase the successful integration of an Arduino-based RFID reader, Raspberry Pi, Flask and MySQL, resulting in a reliable and efficient system for RFID data collection, storage, and real-time monitoring. The system's accurate reading performance, minimal latency, robust database management, and user-friendly web interface highlight its suitability for a range of practical applications. Further development and customization can extend its capabilities to address specific use cases and industry needs.

Figure 7: Visual Assembly of the project

RFID_NFC_Code.ino

```
11
12  #define SS_PIN 10
13  #define RST_PIN 5
14
15  MFRC522 rfid(SS_PIN, RST_PIN);
16
17  void setup() {
18    Serial.begin(9600);
19    SPI.begin(); // init SPI bus
20    rfid.PCD_Init(); // init MFRC522
21
22    Serial.println("Tap RFID/NFC Tag on reader");
23  }
24
25  void loop() {
26    if (rfid.PICC_IsNewCardPresent()) { // new tag is available
27      if (rfid.PICC_ReadCardSerial()) { // NUID has been readed
28        Serial.println("Test");
29        MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
30        //Serial.print("RFID/NFC Tag Type: ");
31        //Serial.println(rfid.PICC_GetTypeName(piccType));
32
33        // print NUID in Serial Monitor in the hex format
34        Serial.print("UID:");
35        for (int i = 0; i < rfid.uid.size; i++) {
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno' on 'COM3')

```
23:42:58.643 -> UID: E3 27 68 A5
23:42:59.729 -> Test
23:42:59.772 -> UID: E3 27 68 A5
23:43:02.567 -> Test
23:43:02.567 -> UID: E3 27 68 A5
23:43:03.345 -> Test
23:43:03.345 -> UID: E3 27 68 A5
23:43:03.984 -> Test
23:43:03.984 -> UID: E3 27 68 A5
23:43:04.459 -> Test
23:43:04.459 -> UID: E3 27 68 A5
23:43:06.978 -> Test
23:43:06.978 -> UID: 93 49 FE 1C
23:43:07.625 -> Test
23:43:07.625 -> UID: 93 49 FE 1C
23:43:08.145 -> Test
23:43:08.185 -> UID: 93 49 FE 1C
```

Figure 8: Outputs of the UID's of each tag read by the Arduino UNO

```
Starting Capture
{'rfid_uid': 'E32768A5'}
{'rfid_uid': '9349FE1C'}
{'rfid_uid': 'HEC45793'}
PS C:\Users\alexo\Desktop\RFID_NFC_Code>
```

Figure 9: Outputs of the Python script for intaking serial data from Arduino UNO

```python
@app.route('/api/rfid_entry', methods=['POST'])
def rfid_entry():
    data = request.get_json()
    rfid_uid = data.get('rfid_uid')
    if not rfid_uid:
        return jsonify({'error': 'Invalid data'}), 400

    # Check if the UID is already registered in the database
    name = "Unknown"  # Default name for unregistered UIDs

    # Query the database for the name associated with the UID
    query = "SELECT name FROM uid_entries WHERE uid = %s"
    db_cursor.execute(query, (rfid_uid,))
    result = db_cursor.fetchone()
    if result:
        name = result[0]
    else:
        # If the UID is not registered, add it to the database
        insert_query = "INSERT INTO uid_entries (uid, name, login_time) VALUES (%s, %s, %s)"
        current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        db_cursor.execute(insert_query, (rfid_uid, name, current_time))
        db_connection.commit()

    # Log the login/entry time
    login_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    return jsonify({'name': name, 'login_time': login_time}), 200


if __name__ == '__main__':
    app.run(debug=True)
```

```
1 •  select * from rfid_db.uid_logs;
2
3
```

Result Grid | Filter Rows: | Edit: | Export/

id	uid	name	login_time
1	E32768A5	TEST	2023-07-27 02:42:54
2	E32768A5	TEST	2023-07-27 02:43:10
3	HEC45793	Unknown	2023-07-27 04:19:58
NULL	NULL	NULL	NULL

Figure 10: Logs of the MySQL database

Summary of Challenges:

After the team built a circuit incorporating the Raspberry Pi, RFID reader and LCD screen, the LCD screen was not displaying anything on the screen. The hardware was first inspected for incorrect pining. However, after rebuilding the circuit from scratch again, the LCD screen still gave major problems, and the team decided that it must be a programming error. This led to the team rewriting a simple Python script to read the RFID tags and then display the UIDs in the LCD as well as the terminal. The script ran with no errors and was able to display the UID in the terminal but was not appearing the LCD screen. We concluded that the data was successfully read and stored but the signals were not successfully transmitted to the LCD display in the hardware. This was possibly the LCD display being damaged due to the soldering the pins. The team decided to move on from the issue due to it being low priority because the mySQL database will also show all the outputs needed to ensure correct RFID reading and transmission. The next issue was attempting to incorporate the function of the RFID MFRC522 to work with the raspberry pi 3 model. The team attempted to locate the issue by first starting to examine the hardware setup. Just like the previous issue with the LCD screen, we attempted several times to rewire the circuit on the breadboard and see if that was causing an issue, followed by reviewing the software requirements of this concept including the drivers, libraries, and ay other dependencies necessary. We even tried upgrading and downgrading certain dependency versions to see if the compatibility had changed over updates and when that did not work the main chunk of troubleshooting went towards verifying the serial communication between the pi and the RFID module. The only workaround to this was to switch out the PI for an Arduino development board which was able to replicate the serial communication and allow us to make a working product. Finally, we ran into a similar issue with the LCD screen and using the Arduino instead

of the PI. We repeated the same troubleshooting steps for the Arduino except had a couple more bases to cover. Along with checking the previous items such as wiring and software, we also needed to verify the initializations for the Arduino, check the power supply for the board, and make sure the power signal is a valid level for the Arduino to function. Attempting to use the default Serial communication pins was another option that was tried

Conclusion:

The RFID Attendance project was a successful application of creating a small portable web server that could keep track of people's whereabouts. The project accomplished the main goal of creating an attendance tracker that can keep track of who has been coming and going through a single tap of their RFID tag.

Through testing, the project was successfully able to log unique tags and users and keep track of the exact time and date it was tapped. The information was transmitted from the sensor to a web server using Python and stored in a MySQL database.

While the project's scope was only intended for keeping track of attendance, the technology could be further expanded. For example, by pairing it with facial recognition and weight sensors, Amazon Go has been able to create a fully autonomous store that keeps track of what you grab and charges you appropriately. Furthermore, RFID/NFC technology has also been used in things such as Apple Pay and Google Pay, allowing for quick encrypted ways of transferring data.

With the rise of RFID and NFC technology, we are on the cusp of a world where people can instantly transmit data with a single tap. This has the potential to revolutionize the way we interact with the world around us. For example, RFID tags could be used to track the location of assets in a warehouse or to provide access control to a building. NFC tags could be used to pay for goods and services or to share contact information with other people.

The possibilities are endless, and the RFID Attendance project is just a small glimpse of what is to come.

References:

[1] “Arduino - RFID/NFC: Arduino tutorial,” Arduino Getting Started, https://arduinogetstarted.com/tutorials/arduino-rfid-nfc#content_arduino_rfid_nfc_code (accessed Aug. 3, 2023).

[2] B. A. Forouzan, TCP/IP Protocol Suite. Boston: McGraw-Hill/Higher education, 2006.

[3] “MFRC522,” MFRC522 - Arduino Reference, https://www.arduino.cc/reference/en/libraries/mfrc522/ (accessed Aug. 3, 2023).

[4] T. Malik, “RC522 RFID reader module,” Microcontrollers Lab, https://microcontrollerslab.com/rc522-rfid-reader-pinout-arduino-interfacing-examples-features/ (accessed Aug. 3, 2023).

Code Addendum:

RFID_NFC_Code.ino

```ino
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(9600);
  SPI.begin(); // init SPI bus
  rfid.PCD_Init(); // init MFRC522
  Serial.println("Tap RFID/NFC Tag on reader");
}

void loop() {
  if (rfid.PICC_IsNewCardPresent()) { // new tag is available
    if (rfid.PICC_ReadCardSerial()) { // NUID has been readed
      MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);

      // print NUID in Serial Monitor in the hex format
      Serial.print("UID:");
      for (int i = 0; i < rfid.uid.size; i++) {
        Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(rfid.uid.uidByte[i], HEX);
      }

      Serial.println();
      rfid.PICC_HaltA(); // halt PICC
      rfid.PCD_StopCrypto1(); // stop encryption on PCD
    }
  }
}
```

```python
from flask import Flask, request, jsonify
import datetime
import mysql.connector

app = Flask(__name__)

# Database connection configuration
db_config = {
    'host': '127.0.0.1',
    'user': 'test',
    'password': 'c!@#$',
    'database': 'rfid_db',
}

# Establish connection
db_connection = mysql.connector.connect(**db_config)
db_cursor = db_connection.cursor()
```

```python
@app.route('/api/rfid_entry', methods=['POST'])
def rfid_entry():
    data = request.get_json()
    rfid_uid = data.get('rfid_uid')
    if not rfid_uid:
        return jsonify({'error': 'Invalid data'}), 400

    # Check if the UID is already registered in the database
    name = "Unknown"  # Default name for unregistered UIDs

    # Query the database for the name associated with the UID
    query = "SELECT name FROM uid_entries WHERE uid = %s"
    db_cursor.execute(query, (rfid_uid,))
    result = db_cursor.fetchone()
    if result:
        name = result[0]
    else:
        # If the UID is not registered, add it to the database
        insert_query = "INSERT INTO uid_entries (uid, name, login_time) VALUES (%s, %s, %s)"
        current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        db_cursor.execute(insert_query, (rfid_uid, name, current_time))
        db_connection.commit()

    # Log the login/entry time
    login_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    return jsonify({'name': name, 'login_time': login_time}), 200


if __name__ == '__main__':
    app.run(debug=True)
```