**Arduino code breakdown**

#include <MFRC522.h>

#include <MFRC522Extended.h>

#include <deprecated.h>

#include <require\_cpp11.h>

- This code is an Arduino sketch that deals with RFID (Radio-Frequency Identification) using the MFRC522 module. The sketch includes two libraries for the MFRC522 module, and it enforces that the code should be compiled using C++11 or a higher standard. The libraries help read and write data to RFID cards or tags. The mention of "deprecated.h" suggests that some parts of the code may be outdated and should be used with caution.

#include <SPI.h>

#include <MFRC522.h>

this code sets up the groundwork to interact with RFID cards, but the actual functionality for reading or writing data would require additional code beyond what's shown.

#include <ESP8266WiFi.h>

#include <ESP8266HTTPClient.h>

this code sets up the ESP8266 to connect to a Wi-Fi network and then use the **ESP8266HTTPClient** library to interact with web servers over the internet. It could be used, for example, to send sensor data to a web server, receive data from an API, or perform other web-related tasks. However, the actual functionality and usage will depend on the rest of the code that follows these library inclusions.

#define SS\_PIN D8

#define RST\_PIN D3

#define RedLed D1

#define GreenLed D2

The provided code uses **#define** statements to create shorthand names for specific GPIO (General Purpose Input/Output) pins on an ESP8266 microcontroller. These names (**SS\_PIN**, **RST\_PIN**, **RedLed**, and **GreenLed**) provide a more convenient way to refer to the specific GPIO pins in the rest of the code, making it easier to understand and maintain the program.

MFRC522 mfrc522(SS\_PIN, RST\_PIN);

it creates a tool named **mfrc522** that can interact with an RFID module connected to the microcontroller. It uses the **SS\_PIN** and **RST\_PIN** numbers to communicate with the RFID module and reset it when needed. With this **mfrc522** tool, you can perform tasks like reading information from RFID tags or cards, writing data to them, or doing other things related to RFID technology.

const char \*ssid = "your ssid";        /

const char \*password = "password";

It only contains the password and the SSID of the current Wi-Fi you're connected

const char\* device\_tokens[] = {

   "22d5f5da0cd25f3b", // Replace with the actual device tokens

  ""

  // Add more tokens here if needed

};

this code provides a convenient way to store multiple device tokens in an array, making it easier to manage and iterate through them when needed in the program. Remember to replace the placeholder **"your token"** with actual device token values for your application.

const int num\_tokens = sizeof(device\_tokens) / sizeof(device\_tokens[0]);

const int ledPin = D4;

1. **const int num\_tokens = sizeof(device\_tokens) / sizeof(device\_tokens[0]);**

This line counts how many device tokens are in the **device\_tokens** array and stores the count in the **num\_tokens** variable.

1. **const int ledPin = D4;**

This line sets the **ledPin** variable to a specific pin number (likely on a microcontroller or development board) that will be used to control an LED or some other component in the rest of the code.

String URL = "http://192.168.1.6/WEB1/dataService.php"; // Replace with your server IP or domain

String dataService, Link;

String OldCardID = "";

unsigned long previousMillis = 0;

this code sets up variables and configurations needed to interact with a web server through HTTP. The actual interaction with the server and data retrieval or posting is likely handled in the rest of the program. The code might be part of a project that communicates with the server to fetch or update data, such as reading sensor data, controlling actuators, or handling user authentication.

void setup() {

  pinMode(ledPin, OUTPUT); // Initialize the LED pin as an OUTPUT.

  delay(1000);

  Serial.begin(115200);

  SPI.begin();  // Init SPI bus

  mfrc522.PCD\_Init(); // Init MFRC522 card

  //---------------------------------------------

  connectToWiFi();

  pinMode(RedLed,OUTPUT);

  pinMode(GreenLed,OUTPUT);

}

* Configures the LED pin as an output and sets up a delay to allow the board to stabilize.
* Initializes serial communication to communicate with a computer or another device.
* Initializes the SPI bus for communication with other SPI devices.
* Initializes the RFID card reader module (MFRC522).
* Connects to a Wi-Fi network (using the **connectToWiFi()** function, which is not shown).
* Configures two pins as output pins to control LEDs.

After the setup function is executed, the Arduino/ESP8266/ESP32 board is ready to perform its main tasks, which are typically defined in the **loop()** function.

void loop() {

  // check if there's a connection to Wi-Fi or not

  if (!WiFi.isConnected()) {

    connectToWiFi();    // Retry to connect to Wi-Fi

  }

  //---------------------------------------------

  if (millis() - previousMillis >= 15000) {

    previousMillis = millis();

    OldCardID = "";

  }

  delay(50);

  //---------------------------------------------

  // look for a new card

  if (!mfrc522.PICC\_IsNewCardPresent()) {

    // Serial.println("No card present."); // Debug print

    return; // Go to the start of the loop if there is no card present

  }

  // Select one of the cards

  if (!mfrc522.PICC\_ReadCardSerial()) {

    Serial.println("Card read failed."); // Debug print

    return; // If read card serial(0) returns 1, the uid struct contains the ID of the read card.

  }

  String CardID = "";

  for (byte i = 0; i < mfrc522.uid.size; i++) {

    CardID += mfrc522.uid.uidByte[i];

  }

  //---------------------------------------------

  if (CardID == OldCardID) {

    Serial.println("Same Card as before."); // Debug print

    return;

  } else {

    OldCardID = CardID;

  }

  //---------------------------------------------

  Serial.println("New Card detected."); // Debug print

  SendCardID(CardID);

  delay(1000);

}

1. The code continuously runs in a loop.
2. It checks if the device is connected to Wi-Fi. If not, it tries to connect.
3. Every 15 seconds, it clears the stored RFID card ID (**OldCardID**).
4. It checks if there is an RFID card present near the reader.
5. If a new card is detected, it reads the card's unique ID.
6. If it's a new card (not the same as the previous one), it prints "New Card detected."
7. It sends the detected card ID to an external system (not shown in the code).
8. There's a 1-second delay between card detections to prevent rapid re-detection.

Overall, this code continuously looks for RFID cards and sends their IDs to an external system when new cards are detected. It's likely part of a larger project that interacts with RFID cards and communicates with a server or cloud service using Wi-Fi.

void SendCardID(String Card\_uid) {

  Serial.println("Sending the Card ID");

  if (WiFi.isConnected()) {

    HTTPClient http;

    for (int i = 0; i < num\_tokens; i++) {

      dataService = "?card\_uid=" + String(Card\_uid) + "&device\_token=" + String(device\_tokens[i]);

      Link = URL + dataService;

      http.begin(Link);

      int httpCode = http.GET();

      String payload = http.getString();

      Serial.println(Link);

      Serial.println(httpCode);

      Serial.println(Card\_uid);

      Serial.println(payload);

      if (httpCode == 200) {

        if (payload.substring(0, 5) == "login") {

          // Logic for "login" response

          digitalWrite(GreenLed, HIGH);

          digitalWrite(RedLed, LOW);

        } else if (payload.substring(0, 6) == "logout") {

          // Logic for "logout" response

          digitalWrite(RedLed, HIGH);

          digitalWrite(GreenLed, LOW);

        } else if (payload == "succesful" || payload == "available") {

          // Logic for "succesful" or "available" response

          digitalWrite(RedLed, HIGH);

          digitalWrite(GreenLed, HIGH);

        } else {

          // Default behavior if response doesn't match any of the above conditions

          digitalWrite(RedLed, LOW);

          digitalWrite(GreenLed, LOW);

        }

        delay(100);

      } else {

        // Handling other HTTP response codes here

        Serial.println("Error: HTTP Code - " + String(httpCode));

        for (int i = 0; i < 2; i++) {

          digitalWrite(RedLed, HIGH);

          delay(200);

          digitalWrite(RedLed, LOW);

          delay(200);

        }

      }

      http.end();

    }

  }

}

1. This code defines a function named **SendCardID** that sends an RFID card ID to a web service using HTTP GET requests.
2. The function checks if the device is connected to Wi-Fi. If not, it skips the process.
3. For each device token in the **device\_tokens** array, the function constructs a URL with the RFID card ID and device token as query parameters.
4. It sends an HTTP GET request to the constructed URL.
5. Based on the server's response, it controls two LEDs (**RedLed** and **GreenLed**) to show different statuses: login, logout, successful, available, or an error.
6. The function waits for a short time before processing the next card ID.

In simple terms, this function sends RFID card data to a server and uses LEDs to show the result of the communication with the server.

void connectToWiFi() {

  WiFi.mode(WIFI\_OFF);        // Prevents reconnection issue (taking too long to connect)

  delay(1000);

  WiFi.mode(WIFI\_STA);

  Serial.print("Connecting to ");

  Serial.println(ssid);

  digitalWrite(ledPin, HIGH); // Turn on the LED while connecting.

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(500);

    Serial.print(".");

  }

  digitalWrite(ledPin, LOW); // Turn off the LED when connected.

  Serial.println("");

  Serial.println("Connected");

  Serial.print("IP address: ");

  Serial.println(WiFi.localIP());  // IP address assigned to your ESP

  delay(1000);

}

This code defines a function named **connectToWiFi()** that connects the ESP8266 or ESP32 board to a Wi-Fi network. Here's what it does in simple terms:

1. The function turns off Wi-Fi temporarily to prevent reconnection issues.
2. It sets the Wi-Fi mode to "station mode" to connect to an existing Wi-Fi network.
3. The function prints the name of the Wi-Fi network it's connecting to.
4. It turns on an LED to indicate that it's attempting to connect to Wi-Fi.
5. The board tries to connect to the specified Wi-Fi network using the provided credentials (SSID and password).
6. It waits until the Wi-Fi connection is established, printing dots during the process.
7. When the connection is successful, it turns off the LED and prints "Connected" to the serial monitor.
8. The function then displays the IP address assigned to the board on the Wi-Fi network.
9. After a short delay, it returns from the function.