O.P.S. Maker 2023-24 - Scuola secondaria di secondo grado - primo biennio
Liceo Scientifico delle Scienze Applicate "C. Cafiero" - Barletta (BT)

S.O.L.A.C.E.

System of Localization and Classroom Environment

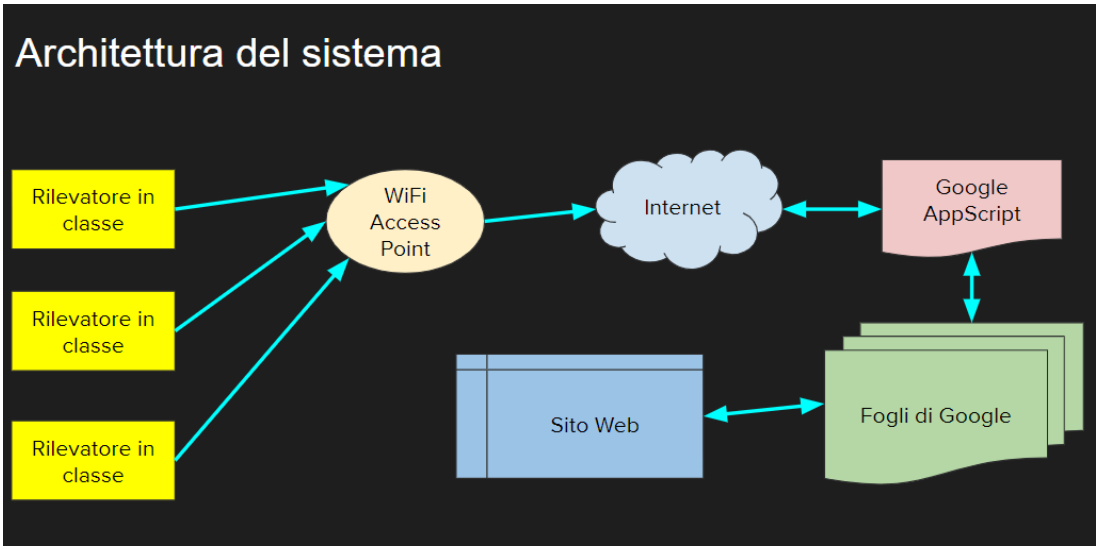
Indice

Introduzione

L'Internet of Things (IoT), o Internet delle Cose, si riferisce a una rete di dispositivi fisici, veicoli, elettrodomestici e altri oggetti che sono dotati di sensori, software e connettività di rete, consentendo loro di raccogliere e condividere dati. Questi dispositivi noti anche come "oggetti intelligenti", possono variare dai semplici dispositivi dai "smart home" agli indossabili come smartwatch, fino ad arrivare a complessi macchinari industriali e sistemi di trasporto. L'IoT consente a questi dispositivi intelligenti di comunicare tra loro e con altri dispositivi abilitati a Internet, creando una vasta rete di dispositivi interconnessi in grado di scambiare dati e svolgere varie attività autonomamente, proprio per questo man mano che il numero di dispositivi connessi a Internet continua a crescere, l'IoT con il passare del tempo avrà un ruolo sempre più importante nel plasmare il nostro mondo.

Problema posto e sviluppo di un'idea di realizzazione

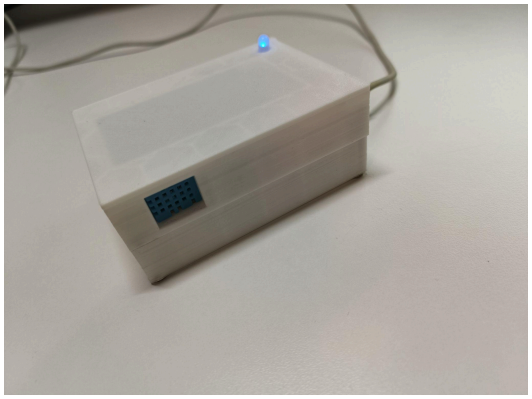
Il nostro istituto ha recentemente attivato una serie di ambienti comuni (laboratori) il cui utilizzo è condiviso da tutte le classi ed è proprio il numero delle classi ad essere maggiore rispetto a quello delle aule e pertanto è necessario che in ogni ora un certo numero di classi cambi aula. Da questa soluzione però nascono due problemi: il primo riguarda la posizione di una classe che deve essere sempre nota e localizzata in modo semplice e veloce, il secondo riguarda invece il rilevamento dei parametri ambientali dell'aula al fine di migliorare il comfort degli ambienti intervenendo sul sistema di riscaldamento e/o di ventilazione.
Per risolvere questi problemi l'ideale sarebbe costruire un dispositivo capace di rilevare la classe che si trova all'interno dell'aula dove è posizionato, azione svolta grazie ad un lettore RFID, e rilevare la temperatura e l'umidità di ogni aula, compito svolto da un sensore DHT11.



In ogni classe scelta per la sperimentazione viene posto un rilevatore che rileva i parametri ambientali (temperatura e umidità) e la classe presente tramite l'avvicinamento di una carta RFID all'ingresso e all'uscita dall'aula. Il rilevatore trasmette i dati via Internet periodicamente (ogni 30 secondi) ad un'applicazione web (realizzata con AppScript di Google) la quale li registra in un foglio di Google in una cartella di lavoro che contiene anche un foglio generale di tutti i dispositivi di monitoraggio. Infine un sito web, appositamente progettato, permette di visualizzare i dati pubblici fruibili dagli studenti e dai docenti.

	Data ultimo aggiornamento	aula	classe	temp	umid	piano	capienza
WD1-083A8DF5B2E7	11/03/2024 13.28.05	2	--	20	46	terra	20
WD1-D8BFC00E1F62	11/03/2024 13.01.17	19	--	23	46	2	27
WD1-08F9E05D653E	11/03/2024 12.28.20	21	--	21	43	2	27
WD1-C8C9A31AEDE3	11/03/2024 13.28.29	26b	--	23	51	3	
WD1-08F9E05D4A87	11/03/2024 13.56.05	30	4P	23	54	3	26
WD1-08F9E05D4A80	11/03/2024 12.54.12	42	5P	22	49	4	23
WD1-08F9E05D5ECD	11/03/2024 13.00.16	46	--	24	44	4	

Rilevatore di classe



Il rilevatore di classe è un dispositivo IoT che ha il compito di:

- Registrare l'entrata e l'uscita di una classe da una stanza;
- Rilevare temperatura ed umidità relativa di una stanza;

- Registrare la classe e i parametri ambientali in un foglio Google

La registrazione dei dati avviene via Internet utilizzando uno Script di Google che funziona come applicazione web che riceve i dati da un rilevatore e li memorizza nel corrispondente foglio Google.
Inoltre i dati memorizzati nel foglio Google sono accessibili ad un sito web che li presenta in un formato pubblicamente fruibile come:

- conoscere l'aula in cui si trova correntemente una classe;
- conoscere i parametri ambientali in tempo reale di ogni ambiente della scuola;
- ottenere informazioni sull'andamento dei parametri ambientali nel tempo.

Funzionamento del rilevatore di classe

Il rilevatore di classe ha le seguenti funzioni:

- Registra l'entrata e l'uscita di una classe da una stanza;
- Rileva temperatura ed umidità relativa di una stanza;
- Registra la classe e i parametri ambientali in un foglio Google
- Legge e visualizza il contenuto della memoria di una carta RFID;
- Registra i dati di setup WiFi in una carta SETUP;
- Registra la classe di appartenenza in una carta CLASSE.

Il rilevatore di classe rileva l'ingresso e l'uscita di una classe tramite il passaggio di una carta RFID davanti al rilevatore il quale indicherà l'esito della lettura con il seguente colore (LED RGB):

- **ROSSO**: errore di lettura della carta o carta non riconosciuta;
- **VERDE**: classe rilevata correttamente in entrata;
- **GIALLO**: classe rilevata correttamente in uscita.

Il colore **BLU** viene invece utilizzato per indicare se il dispositivo è connesso o meno alla rete Internet.

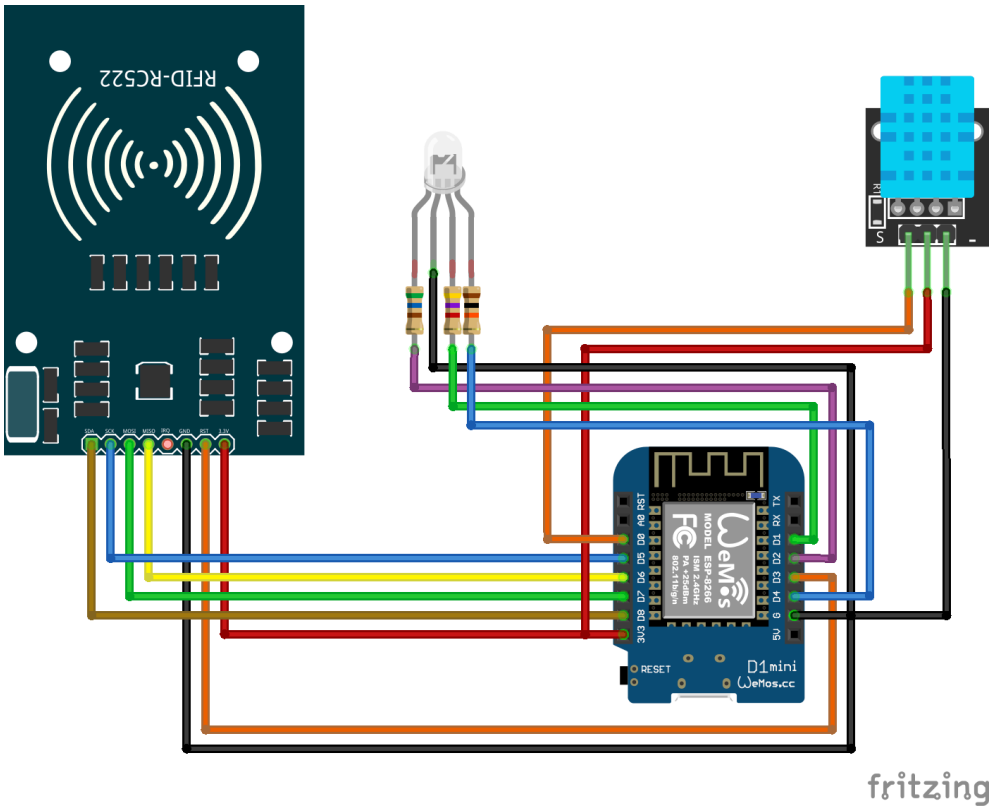
Funzionamento di una carta RFID

Il sistema SOLACE utilizza alcuni blocchi nelle carte RFID per memorizzare alcuni dati:

- Carta CLASSE:
  - blocco 1: contiene il nome della classe;
  - blocco 8: contiene il tipo di carta (classe o setup);
- Carta SETUP:
  - blocco 4: contiene il SSID della rete WiFi a cui è possibile collegarsi;
  - blocco 5: contiene la password di accesso alla rete WiFi a cui collegarsi.

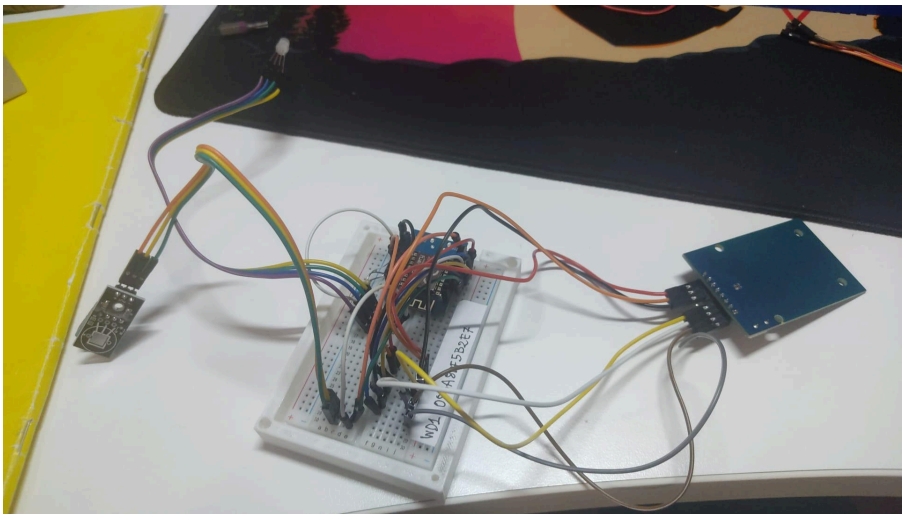
Una carta RFID MIFARE contiene una memoria di 1K byte organizzata in 16 settori contenenti 4 blocchi ciascuno. Ogni blocco contiene 16 byte. Il blocco 0 è riservato alle operazioni di sistema e il quarto blocco di ogni settore contiene le chiavi di accesso sicuro ai dati dei primi 3 blocchi.

Circuito elettrico e componenti del rilevatore di classe



Il rilevatore di classe è costituito da:

- un microcontrollore WeMos D1 mini compatibile che si occupa delle funzioni di collegamento WiFi, rilevamento della temperatura ed elaborazioni delle informazioni contenute in una carta RFID;
- un sensore di carte RFID - RC522 (carte MIFARE) che legge il contenuto della memoria di una carta RFID compatibile;
- un sensore DHT11 per la rilevazione della temperatura e dell'umidità;
- un LED RGB per la visualizzazione dell'esito della lettura di una carta RFID.



Software di controllo del rilevatore di classe

Il codice del rilevatore di classe è stato suddiviso in molteplici funzioni per migliorare la leggibilità e la manutenzione.

Abbiamo inserito 7 librerie per riuscire a sviluppare il codice in C++; le librerie utilizzate sono:

`<SPI.h>`	protocollo di comunicazione seriale
`<MFRC522.h>`	libreria per la gestione del lettore delle carte RFID
`<ESP8266WiFi.h>`	libreria per la gestione della connessione WiFi
`<ESP_EEPROM.h>`	libreria per leggere e scrivere dalla memoria flash di ESP8266
`<DHT11.h>`	libreria per leggere temperatura ed umidità dal sensore DHT11
`<ESP8266HTTPClient.h>`	libreria per gestire un client HTTP
`<WiFiClientSecureBearSSL.h>`	libreria per gestire un client sicuro HTTPS

Le funzioni principali del codice sono due e si trovano nel LOOP, esse sono checkcard e readDHT.
La prima controlla ripetutamente se il lettore RFID rileva una carta; la seconda, legge i valori raccolti dal sensore DHT11 e li invia al foglio Google.
Quando il lettore RFID attiva la procedura checkcard e riceve un errore nella lettura, quest'ultimo farà accendere il led di colore rosso.
Il funzionamento del codice può essere monitorato attraverso i messaggi nel monitor seriale dell'Arduino IDE per le attività di test e debug.

Rilevamento del tipo di carta (RFID): Inizialmente, il codice, stampa il tipo di carta RFID rilevata utilizzando la libreria MFRC522. Questa libreria è comunemente usata per interagire con i lettori RFID.
Controllo del tipo di carta: Viene verificato se il tipo di carta rilevata è compatibile con il lettore. Se non lo è, viene stampato un messaggio di errore e il programma termina.
Lettura dei dati dalla carta: Se la carta è compatibile, il programma procede con la lettura dei dati dalla carta RFID. In base al tipo di carta, vengono eseguite azioni diverse:

- Se la carta è di tipo "classe", vengono letti i dati dal blocco 1 e viene gestito l'ingresso o l'uscita della classe da una stanza. I dati vengono quindi registrati nella memoria flash.
- Se la carta è di tipo "setup WIFI", vengono letti i dati dal blocco 4 e 5, contenenti rispettivamente l'SSID e la password WiFi. Questi dati vengono quindi registrati nella memoria flash e utilizzati per connettersi a una rete WiFi.

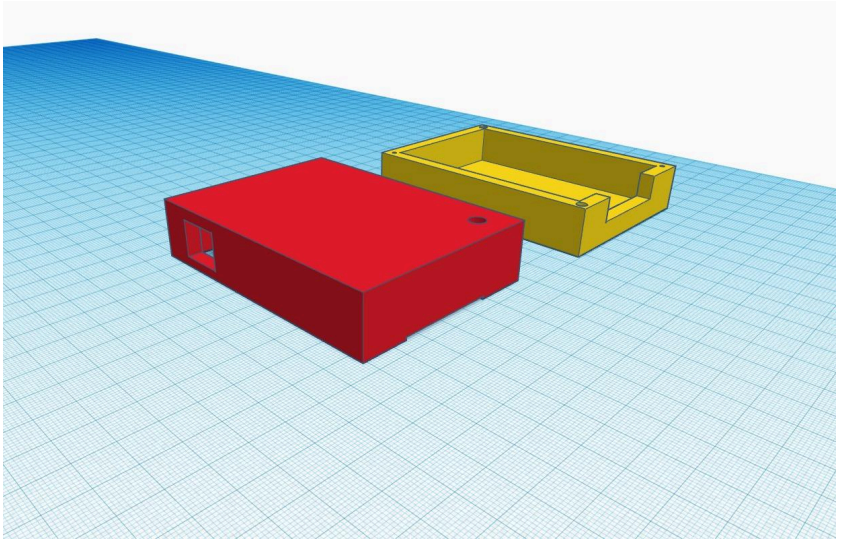
Gestione della lettura della carta: Alla fine della lettura dei dati, viene chiamata la funzione stopReadingCard() per fermare la lettura della carta.
Altre funzioni: Il codice contiene anche altre funzioni come readDataFromBlock() per leggere i dati dai blocchi della carta, displayRGB() per gestire la visualizzazione del LED RGB, e altre funzioni per gestire eventi WiFi e leggere i dati da un sensore DHT.

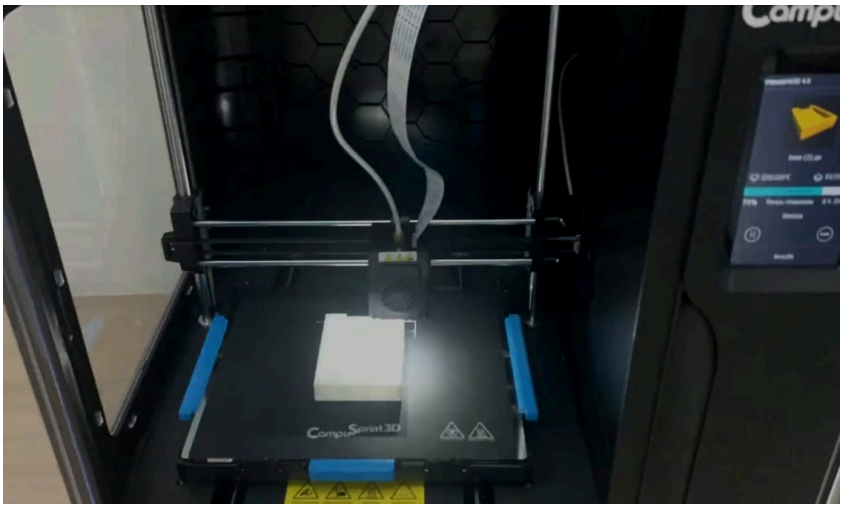
Inoltre, ci sono alcune funzioni di supporto come writeFlash() e readFlash() che gestiscono la scrittura e la lettura dei dati nella memoria flash.

Progettazione e realizzazione del contenitore del rilevatore di classe

Il progetto 3D è stato realizzato con:

- la piattaforma TinkerCad
- stampanti 3D a base di PLA





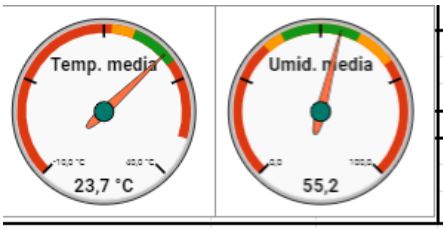
Applicazione web per la registrazione dei dati

L'applicazione web è stata sviluppata nell'ambiente Apps Script di Google che consente di automatizzare l'ambiente Workspace di Google con procedure scritte in javascript.
Lo script gestisce le richieste GET provenienti dai dispositivi di rilevazione di classe.
I fogli di Google di raccolta dati contengono:

- un foglio generale che riporta ogni dispositivo registrato con gli ultimi dati rilevati: temperatura, umidità e classe;
- un foglio per ogni dispositivo registrato che riporta lo storico dei dati registrati con alcuni indicatori di temperatura e umidità media e un andamento della temperatura

rispetto al tempo.

- Foglio dati generale: SOLACE_Dati - Fogli Google
- Foglio singolo dispositivo: SOLACE_Dati - Singolo dispositivo



Dispositivo	Stanza	Classe	Temp. media	Umid. media
WD1-C8C9A31AEDE3	26b	--	23,7 °C	55,2
Ultima lettura	**Temperatura**	**Umidità**		
11/03/2024 13.28.29	23,0 °C	51,0		

Lo script è stato sviluppato nell'ambiente di sviluppo Apps Script di Google che consente di automatizzare l'ambiente Workspace di Google con procedure scritte in javascript.

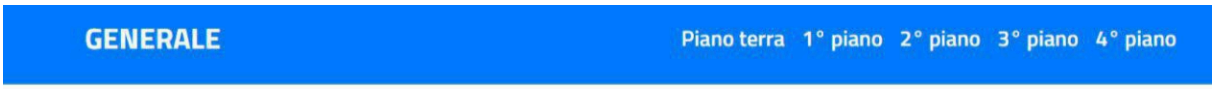
Il foglio dei dati è gestito da uno script in modalità Applicazione Web organizzato in due principali funzioni che gestiscono:

- le richieste GET provenienti dai dispositivi di rilevazione per la registrazione dei dati nel foglio elettronico;
- le richieste GET provenienti dal sito web per la lettura dei dati registrati al fine di visualizzarli nelle pagine HTML; i dati vengono inviati in formato JSON.

Script per il foglio dati: Solace_script - Editor progetto - Apps Script (google.com)

Sito web di presentazione dei dati

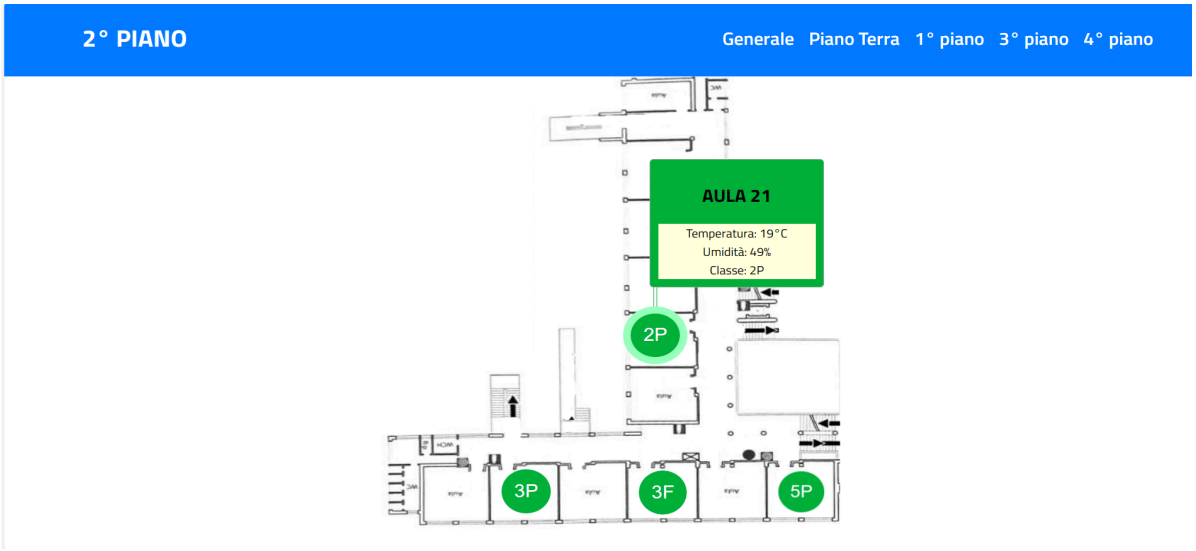
Abbiamo creato il sito web per comunicare in maniera semplice e veloce i dati raccolti dal dispositivo e stampati su fogli google.
Sulla pagina "GENERALE" vi è una tabella che mostra tutte le classi che hanno effettuato l'entrata in un'aula, oltre i dati ambientali.



Classe	Aula	Temperatura	Umidità
3F	2	20	46
3P	19	23	46
2P	21	21	43
3I	26b	23	51
4P	30	22	47
5P	42	22	49
2I	46	24	44

Il codice per l'aggiornamento del contenuto delle pagine è scritto in Javascript.
La funzione getJSON utilizza un oggetto XMLHttpRequest per richiedere i dati in formato JSON dall'applicazione Google Apps Script. Successivamente itera fra tutti gli elementi di

esso e, per ogni classe trovata, aggiunge alla tabella presente nel div "table-container" una riga contenente i valori: nome della classe, numero dell'aula, temperatura e umidità.
La tabella viene aggiornata ogni 15 secondi in modo asincrono senza dover ricaricare la pagina.



Le pagine "PIANO", prelevano in maniera asincrona i dati attraverso la funzione getDatiAule, la quale effettua una richiesta all'applicazione Google Apps Script e, in base alla risposta ricevuta, visualizza i dati specifici di un'aula.
I dati ricevuti vengono passati alla funzione visualizzaDati, che converte la stringa JSON in un oggetto Javascript.
In seguito, se non sono presenti errori, la funzione itera ogni elemento dell'oggetto Javascript e, se trova l'aula indicata nel codice html come "data-aula", visualizza sulla piantina un pulsante (button) circolare avente come testo, la classe presente nell'aula (o – in caso di aula vuota). Tale pulsante sarà:

- verde, se il valore della temperatura all'interno della classe è accettabile.
- arancione, se il valore della temperatura all'interno della classe è tollerabile.
- rosso, se il valore della temperatura all'interno della classe è estremamente freddo o estremamente caldo.

Quando il cursore viene posto al disopra del pulsante apparirà un box con all'interno i dati dell'aula; se si fa clic sul pulsante si ottiene l'accesso direttamente al foglio Google in cui è presente lo storico dei dati del singolo dispositivo installato nell'aula in oggetto.

Il sito di presentazione dei dati è disponibile all'indirizzo:
http://solace.arduinolabliceocafiero.it

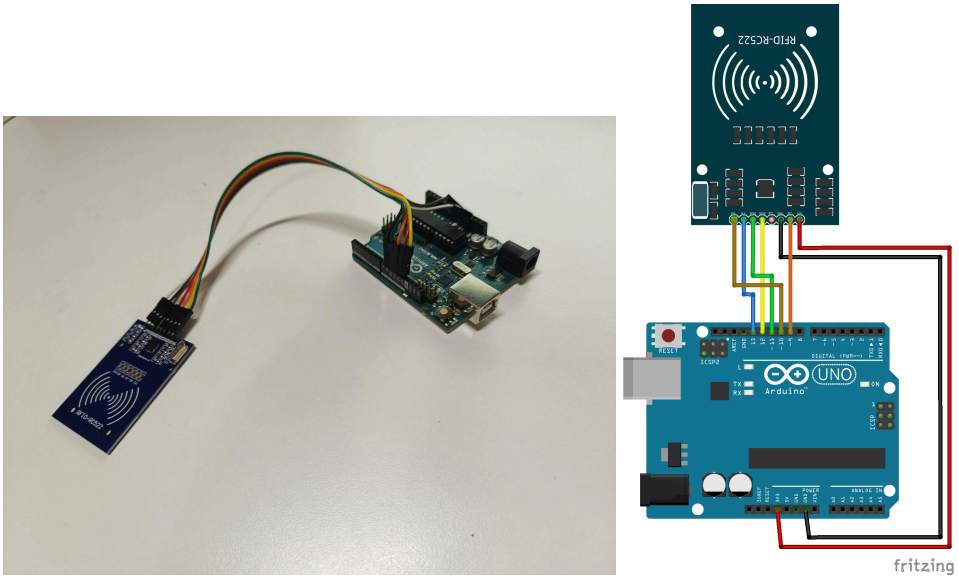
Programmatore di schede di setup

Contemporaneamente al rilevatore di classe abbiamo quindi sviluppato anche un programmatore di carte RFID, collegando un lettore di carte RFID ad un Arduino uno.
Il codice in C++ da noi sviluppato permette di instaurare una connessione, tramite cavo, con un monitor seriale come quello della piattaforma IDE. Appoggiando una carta RFID al sensore vengono stampati sul monitor seriale i valori contenuti proprio nella carta. In questo momento l'utente può scegliere se scrivere nuovi dati o meno.

Se non si vuole modificare nulla si uscirà dalla procedura, nel caso contrario verrà richiesto il tipo di carta da programmare:

1. carta classe
2. carta setup

Programmando una carta setup il dispositivo chiederà di digitare la classe sul monitor seriale che poi verrà memorizzata in un determinato blocco della scheda stessa.
Se invece si vuole programmare una scheda setup verranno richieste due informazioni ovvero l'SSID del wifi al quale si vuole connettere il rilevatore di classe e la sua relativa password. Una volta scritte queste informazioni il programmatore di schede ri-stamperà sul monitor seriale i dati aggiornati della scheda.
Questo programmatore di classe ci ha permesso di non modificare il codice ogni volta che abbiamo avuto bisogno di connettere il rilevatore di classe ad un nuovo access point.



Il codice utilizzato è stato realizzato in linguaggio Arduino. Questo programma è progettato per interagire con una scheda RFID, che è un tipo di tecnologia usata per identificare e tracciare gli oggetti attraverso segnali radio.
Il programma inizia con l'inizializzazione della comunicazione seriale, SPI e il modulo RFID. Questo serve a preparare il microcontrollore per comunicare con il lettore RFID.
In seguito imposta una chiave di lettura predefinita per leggere le informazioni memorizzate sulla carta RFID. Questa chiave viene utilizzata per decodificare i dati crittografati sulla carta.
Entra poi in un ciclo continuo (loop) in cui controlla se una carta RFID è stata rilevata. Se viene rilevata una carta, il programma legge il suo tipo e l'identificatore univoco (UID). Se la carta è del tipo corretto, il programma può anche leggere alcuni dati specifici memorizzati sulla carta stessa.
Visualizzazione delle informazioni: Le informazioni lette dalla carta, come il tipo e l'UID, vengono mostrate tramite la comunicazione seriale. Questo consente all'utente di monitorare e registrare le informazioni lette dalla carta RFID

Sperimentazione dei prototipi

Dopo aver passato la fase di costruzione dei vari dispositivi SOLACE, abbiamo iniziato la fase di testing, durata **3** settimane, dove abbiamo verificato il corretto funzionamento dei dispositivi con l'entrata e l'uscita delle classi e con le misure dell'umidità e della temperatura.

Per questa fase di sperimentazione abbiamo installato 7 dispositivi diversi in 7 classi dando il compito ad un alunno per classe di registrare l'entrata la mattina, eventuali uscite durante le ore scolastiche e l'uscita al termine della giornata. I dati raccolti vengono mandati ai fogli per la raccolta dati i quali sono inseriti all'interno di un sito accessibile a tutti gli studenti, docenti e personale ATA.

Realizzazione del progetto

Il progetto SOLACE è stato diviso in diversi moduli come:

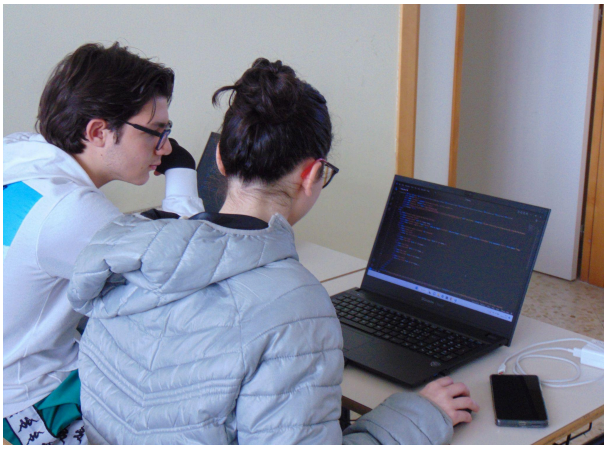
- **Contenitore**: progettazione e realizzazione di un contenitore per il rilevatore di classe e il programmatore base delle carte RFID;
- **Documentazione**: realizzazione del documento contenente le informazioni del progetto e delle fasi di realizzazione;
- **Video**: realizzazione del video di max 5 min di presentazione del progetto;
- **Montaggio**: montaggio di ulteriori rilevatori da inserire nelle prove e nelle presentazione del sistema;
- **Cartellone**: realizzazione del cartellone di presentazione del progetto;
- **Fogli e sito web:** studio e miglioramento delle caratteristiche dei fogli Google dei dati rilevati con eventuale sviluppo di un sito web di utilizzo del sistema;
- **Sviluppo**: studio e miglioramento del funzionamento delle unità di rilevamento e della base di programmazione.

I vari moduli sono stati affidati a differenti **gruppi di lavoro**. Ogni gruppo ha un tutor responsabile e si occupa di uno dei seguenti aspetti:

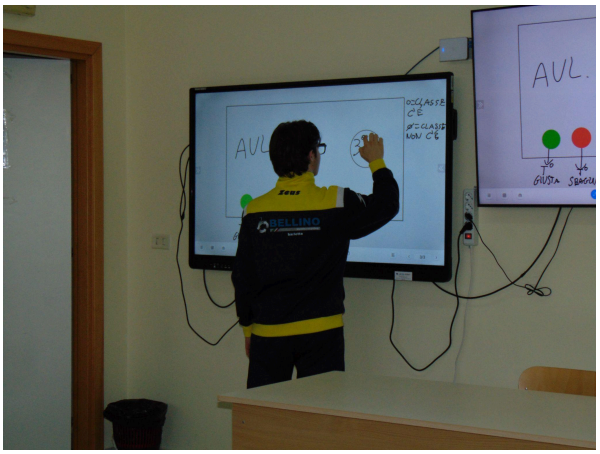
- Progettazione e realizzazione della struttura in 3D del contenitore di S.O.L.A.C.E;
- Produzione della documentazione con video ed immagini dello sviluppo del progetto;
- Progettazione e sviluppo del software in linguaggio C++ per la piattaforma WeMOs;
- Progettazione e sviluppo del sito internet che raccoglie i dati;
- Progettazione e montaggio dell'hardware;
- Stesura di questo documento.
- raccolta analisi dati

Produzione della documentazione e del video

Durante tutte le fasi di realizzazione del progetto alcuni studenti si sono presi l'incarico di fare foto e riprese da destinare alla realizzazione di questo documento e del video di presentazione. Sono state effettuate anche riprese con la tecnica del chroma key per i narratori del video.




PRIMO PIANO


AUL
AUL

Problemi aperti e attività di miglioramento del prototipo

Alcune questioni però non sono completamente risolte:

- instabilità del dispositivo WeMos D1 mini che spesso si riavvia;
- bassa sensibilità del WeMos D1 mini della rete WiFi: necessario che l'access point sia nella vicinanze dell'AP;
- difficoltà del WeMos D1 a ottenere l'indirizzo IP dal DHCP server se l'access point è lontano;
- migliorare il timing del codice: a volte la carta non viene rilevata o viene visualizzato un errore dovuto al processore occupato nelle operazioni WiFi;
- migliorare l'architettura del programmare per il WeMos D1 utilizzando codice asincrono;
- utilizzare il sistema per raccogliere dati ambientali come la presenza di CO_2 come indicatore della qualità dell'aria all'interno delle aule.

Conclusioni

Questo progetto ci ha permesso di migliorare le nostre conoscenze, competenze e capacità di coordinamento e più specificatamente:

- abbiamo risolto un problema dividendolo in parti più semplici, coordinando le loro soluzioni verso il prodotto finale;
- abbiamo incrementato le capacità di problem solving in un contesto di condivisione e collaborazione;
- tutti noi ci siamo avvicinati alla piattaforma WeMos che ha contribuito nel processo di apprendimento e miglioramento delle nostre capacità nell'ambito della programmazione;
- siamo stati introdotti a differenti linguaggi di programmazione e abbiamo compreso l'importanza della specificità di applicazione di ogni linguaggio;
- abbiamo sviluppato il senso del lavoro di equipe e di responsabilità verso gli altri;

Infine ringraziamo i 5 studenti del terzo anno che hanno fatto attività di tutoraggio ed assistenza durante le fasi di realizzazione del progetto, mettendo a disposizione le loro conoscenze e le loro esperienze pregresse.

Gruppo di lavoro

Gruppo di lavoro:

- Francesco Carpagnano-2I
- Massimo Giuseppe Curci-2I
- Leonardo Rana-2I
- Michele Filannino-2I
- Vittorio Zenobj-2P

Tutor:

- Domenico Capuano-3I
- Nicoleta Doinita Maria Floarea-3F
- Alessandro Lamacchia-3P
- Enza Sciusco-3P
- Maria Ilenia Lacavalla-3P

Referente:

- prof. Francesco Paolo Vino

Dirigente scolastico:

- prof.ssa Rosanna Diviccaro

Scuola:

- Liceo Scientifico opzione Scienze Applicate “C.Cafiero” - Barletta(BT)

Appendici

Codice C++ del rilevatore di classe

```
#include <SPI.h>                      // protocollo di comunicazione seriale
#include <MFRC522.h>                  // libreria per la gestione del lettore delle carte RFID
#include <ESP8266WiFi.h>              // libreria per la gestione della connession WiFi
#include <ESP_EEPROM.h>               // libreria per leggere e scrivere dalla memoria flash di
ESP8266
#include <DHT11.h>                    // libreria per leggere temperatura ed umidità dal sensore
DHT11
#include <ESP8266HTTPClient.h>        // libreria per gestire un client HTTP
#include <WiFiClientSecureBearSSL.h>  // libreria per gestire un client sicuro HTTPS

// pin lettore carta RFID
#define SS_PIN D8
#define RST_PIN D3
/*
RST  - D3
MISO - D6
MOSI - D7
SCK  - D5
SDA  - D8
*/
// pin led RGB (usati solo il rosso e il verde)
#define LED_R D2
#define LED_G D1

MFRC522 rfid(SS_PIN, RST_PIN); // oggetto di classe MFRC522
MFRC522::MIFARE_Key key; // array per memorizzare le chiavi di lettura

// array per memorizzare l'UID dell'ultima carta letta
byte nuidPICC[4];
// array per il nome della classe
byte classe[18];
// array per il WiFi - SSID
byte SSID[18];
// array per il WiFi - Password
byte password[18];
// array per il numero della stanza
byte stanza[18];

// indirizzo dello script che fa da server e che raccoglie i dati del dispositivo
const String
urlScript="https://script.google.com/macros/s/AKfycbyb_DabvG-vxd73PmXLj_CieaQY8Dw3e73KYZROPJsaxAt
ek7dJH7Nxcv9mKDhcaEXc/exec";

// MAC Address della scheda Wifi
```

```
String macAddress;

// dati di stato della carta
// questi dati vengono memorizzati nella memoria flash
// per poter essere recuperati ad ogni riavvio
// vengono modificati da una carta SETUP o da una carta CLASSE
struct Stato{
  byte classe[16];    // ultima classe nella stanza - se contiene 2 spazi la stanza è vuota
  byte SSID[17];      // SSID WiFi
  byte password[17];  // password WiFi
};
Stato datiStato;

// led RGB utilizzato per la carta
// il led blu sulla scheda viene usato per lo stato della connessione WiFi
enum StatoLed {
  CARD_ERROR = 0,
  CARD_OK = 1,
  CARD_NOT_PRESENT = 2,
  CLASS_IN = 3,
  CLASS_OUT = 4
};

StatoLed currStateLed;
WiFiEvent_t statoWiFi;

long ledTimer; // per temporizzare l'accensione del led rgb
long ledInterval=3000;  // intervallo di tempo in cui rimane accesso il led rgb

// Gestione sensore DHT11 per temperatura e umidità
DHT11 dht11(D0); // dati sensore DHT11 su pin D0

long timerDht;
long DHTInterval=30000; // lettura parametri ambientali ogni 30 secondi
int DHTTemp;     // temperatura
int DHTHum;      // umidità

/*
 * Setup
 */
void setup() {
  // inizializza interfaccia seriale
  Serial.begin(74880);
  delay(5000);
  // inizializza interfaccia di collegamento con la base di lettura della carta
  SPI.begin(); // inizializza BUS SPI
  rfid.PCD_Init(); // inizializza circuito  MF-RC522
```

```
  // imposta la chiave di lettura della carta
  // i dati memorizzati nella carta sono crittografati con una chiave
  // la chiave iniziale è FFFFFFFFFFFF (hex) (chiave di default)
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }
  Serial.println();
  Serial.println();
  Serial.println(F("Inizializzato lettore carte MIFARE."));

  // led RGB
  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  currStateLed=CARD_NOT_PRESENT;
  displayRGB(currStateLed);

  // altre inizializzazioni
  macAddress=WiFi.macAddress();
  Serial.print(F("Identificativo scheda: "));
  Serial.println(macAddress);

  // imposta valori di default per datiStato
  datiStato.classe[0]=' ';
  datiStato.classe[1]=' ';
  datiStato.SSID[0]=' ';
  datiStato.password[0]=' ';

  // recupera dati dalla memoria flash
  readFlash();
  // WiFi
  pinMode(BUILTIN_LED,OUTPUT);
  digitalWrite(BUILTIN_LED,HIGH);
  WiFi.onEvent(wifiEvent);      // gestore eventi WiFi
  WiFi.mode(WIFI_STA);          // modalità client

  creaHostName();               // imposta l'hostname
  // connessione all'AP WiFi
  WiFi.begin((char*)datiStato.SSID,(char*)datiStato.password);
  Serial.print(F("Connessione a: "));
  Serial.print((char*)datiStato.SSID);
  Serial.print(" - ");
  Serial.println((char*)datiStato.password);

  // sensore DHT11 per temperatura e umidità
  timerDht=millis();

}
```

```
/*
 * Loop principale di gestione
 */
void loop() {
  checkCard();        // controlla la presenza di una carta e ne legge ed invia i dati
  readDHT();          // rileva ed invia periodicamente temperatura e umidità dell'ambiente


}

/*
 * Esegue il controllo della presenza di una carta e in caso affermativo
 * legge i dati e li memorizza nelle corrispondenti variabili globali
 * riporta
 * 0: nessuna carta letta
 * 1: carta CLASSE letta
 * 2: carta SETUP WIFI letta
 */
byte checkCard(){

  // Esce dalla procedura se non vi è nessuna carta rilevata
  if ( ! rfid.PICC_IsNewCardPresent()){
    displayRGB(CARD_NOT_PRESENT);
    return 0;
  }

  // Esce dalla procedura se la carta non è stata correttamente letta
  if ( ! rfid.PICC_ReadCardSerial()){
    displayRGB(CARD_ERROR);
    return 0;
  }

  // rileva il tipo di carta
  Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
  Serial.println(rfid.PICC_GetTypeName(piccType));

  // controlla se la carta è del tipo compatibile con il lettore
  if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
    piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
    piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
    Serial.println(F("La carta non è di tipo MIFARE classic"));
    return 0;
  }

  // carta rilevata di formato corretto
  Serial.println(F("Carta rilevata."));
```

```
  // visualizza l'UID in esadecimale
  Serial.print(F("NUID carta: "));
  printHex(rfid.uid.uidByte, rfid.uid.size);
  Serial.println();

  // memorizza per usi futuri l'UID letto
  for (byte i = 0; i < 4; i++) {
    nuidPICC[i] = rfid.uid.uidByte[i];
  }

  // legge alcuni dati dai blocchi in base al tipo di carta
  byte data[18]; // buffer dati da scrivere o leggere nella carta - max numero di caratteri per un blocco: 16

  // lettura dati blocco 8 - utilizzato per memorizzare il tipo di carta: 1 = carta classe; 2 = carta setup WIFI
  if (!readDataFromBlock(8,data,18)) return 0;
  printDataBlock(data,8);
  if (data[0]=='1'){
    // carta classe
    // lettura dati blocco 1 - contiene il nome della classe: 2 caratteri del tipo 1A
    Serial.println(F("Carta di tipo CLASSE."));
    if (!readDataFromBlock(1,classe,18)) return 0;
    printDataBlock(classe,1);
    // verifica se classe in ingresso o in uscita dalla stanza
    if (datiStato.classe[0]==classe[0] && datiStato.classe[1]==classe[1]){
      // classe già all'interno della stanza - imposta uscita e aula vuota = "--"
      datiStato.classe[0]=0x2D;
      datiStato.classe[1]=0x2D;
      displayRGB(CLASS_OUT);
      Serial.println(F("Classe in uscita"));
    } else {
      // classe in ingresso - imposta ingresso
      copiaArr(classe,datiStato.classe);
      displayRGB(CLASS_IN);
      Serial.println(F("Classe in ingresso"));
    }
    // registra dati nella memoria flash
    writeFlash();
    // fine operazione di lettura carta
    Serial.println(F("Lettura conclusa correttamente."));
  } else if (data[0]=='2'){
    // carta setup AP
    Serial.println(F("Carta di tipo SETUP WIFI."));
    // lettura dati blocco 4 - contiene il SSID WiFi
    if (!readDataFromBlock(4,SSID,18)) return 0;
    copiaArr(SSID,datiStato.SSID);
    printDataBlock(SSID,4);
```

```
    // lettura dati blocco 5 - contiene la password WiFi
    if (!readDataFromBlock(5,password,18)) return 0;
    copiaArr(password,datiStato.password);
    printDataBlock(password,5);
    // registra dati nella memoria flash
    writeFlash();
    // fine operazioni lettura carta
    Serial.println(F("Lettura conclusa correttamente."));
    displayRGB(CARD_OK);
    // nuova connessione a WiFi
    WiFi.disconnect();
    WiFi.begin((char*)datiStato.SSID,(char*)datiStato.password);
  } else {
    Serial.println(F("Carta di tipo NON DEFINITO."));
    displayRGB(CARD_ERROR);
  }

  // ferma lettura della carta
  stopReadingCard();
  return data[0]-0x30;
}

/*
 * Ferma il rilevamento della carta
 */
void stopReadingCard(){
  // ferma il rilevamento della carta
  rfid.PICC_HaltA();

  // ferma la crittografia dei dati
  rfid.PCD_StopCrypto1();

  Serial.println(F("Rimuovere la carta."));
}

/**
 * Legge max 16 byte da un determinato blocco
 */
bool readDataFromBlock(byte block,byte data[], byte len){
  MFRC522::StatusCode status;

  status = rfid.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(rfid.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Autenticazione fallita: "));
    Serial.println(rfid.GetStatusCodeName(status));
    stopReadingCard();
    return false;
  }
```

```
  status = rfid.MIFARE_Read(block, data, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Lettura fallita: "));
    Serial.println(rfid.GetStatusCodeName(status));
    stopReadingCard();
    return false;
  }

  return true;
}

/**
 * Visualizza in esadecimale il contenuto dell'array
 */
void printHex(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
}

/**
 * Visualizza i caratteri stampabili nell'array
 */
void printText(byte *data, byte len){
  for (byte i=0;i<len;i++){
    if (data[i] >= 0x10){
      Serial.write(data[i]);
    }
  }
  Serial.println();
}

/*
 * Visualizza il contenuto di un blocco di memoria della carta
 */
void printDataBlock(byte* data, byte block){
  Serial.print(F("Dati blocco  "));
  Serial.print(block);
  Serial.print(": ");
  printText(data,16);
}

/*
 * Gestione LED RGB
 */
void displayRGB(StatoLed state){
```

```
  if (currStateLed != CARD_NOT_PRESENT){
    if ((millis()-ledTimer)>ledInterval){
      currStateLed=CARD_NOT_PRESENT;
      digitalWrite(LED_R, LOW);
      digitalWrite(LED_G, LOW);
    } else {
      if (currStateLed == CARD_ERROR && state == CARD_OK){
        digitalWrite(LED_R, LOW);
        digitalWrite(LED_G, HIGH);
        currStateLed=state;
      }
    }
  } else {
    currStateLed = state;
    ledTimer = millis();
    switch (state){
      case CARD_OK:
      case CLASS_IN:
        digitalWrite(LED_R, LOW);
        digitalWrite(LED_G, HIGH);
        break;
      case CLASS_OUT:
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, HIGH);
        break;
      case CARD_ERROR:
        digitalWrite(LED_R, HIGH);
        digitalWrite(LED_G, LOW);
        break;
      default:
        digitalWrite(LED_R, LOW);
        digitalWrite(LED_G, LOW);
        break;
    }
  }
}

/*
 * Copia il contenuto da arrDa a arrA
 */
void copiaArr(byte* arrDa, byte* arrA){
  for(byte i=0;i<16;i++){
    arrA[i]=arrDa[i];
  }
}

/*
 * Leggi dati da memoria flash
```

```
 */
void readFlash(){
  EEPROM.begin(sizeof(Stato));
  if (EEPROM.percentUsed()>=0){
    EEPROM.get(0,datiStato);
    Serial.println(F("Recuperati i dati dalla memoria flash."));
    // visualizza i dati recuperati
    Serial.print(F("SSID: "));
    Serial.println((char*)datiStato.SSID);
    Serial.print(F("Password: "));
    Serial.println((char*)datiStato.password);
    Serial.print(F("Classe: "));
    Serial.println((char*)datiStato.classe);
  } else {
    Serial.println(F("Nessun dato presente nella memoria flash"));
  }
}

/*
 * Scrive i dati nella memoria flash
 */
void writeFlash(){
  EEPROM.put(0,datiStato);
  bool ok = EEPROM.commit();
  if (ok) {
    Serial.println(F("Dati salvati nella memoria flash."));
  } else {
    Serial.println(F("Non è stato possibile salvare i dati nella memoria flash"));
  }
}

/*
 * Crea l'hostname per il dispositivo WiFi
 * utilizza l'indirizzo MAC
 */
 void creaHostName(){
  String n;
  n=macAddress;
  n = "WD1-" + n;
  n.replace(":","");
  char hName[n.length()+1];
  n.toCharArray(hName, n.length()+1);
  WiFi.setHostname(hName);
  Serial.print(F("Impostato hostname: "));
  Serial.println(hName);
 }

/*
```

```
 * Gestione eventi WiFi
 */
void wifiEvent(WiFiEvent_t event) {
  if (event!=statoWiFi){
    statoWiFi=event;
    switch(event) {
        case WIFI_EVENT_STAMODE_CONNECTED:
            Serial.print(F("WiFi connesso a: "));
            Serial.println(WiFi.SSID());
            break;
        case WIFI_EVENT_STAMODE_GOT_IP:
            Serial.print(F("WiFi IP: "));
            Serial.print(WiFi.localIP());
            Serial.print(F(" Gateway: "));
            Serial.println(WiFi.gatewayIP());
            digitalWrite(BUILTIN_LED,LOW);
            break;
        case WIFI_EVENT_STAMODE_DISCONNECTED:
            Serial.println(F("WiFi disconnesso."));
            digitalWrite(BUILTIN_LED,HIGH);
            break;
        case WIFI_EVENT_STAMODE_DHCP_TIMEOUT:
            Serial.println(F("DHCP Timeout."));
            digitalWrite(BUILTIN_LED,LOW);
            break;
        default:
            break;
    }
  }
}

/*
 * lettura temperatura e umidità dal sensore DHT11
 */
void readDHT(){
  int temperature;
  int humidity;
  if(millis()-timerDht>DHTInterval){
    temperature=dht11.readTemperature();
    humidity=dht11.readHumidity();
    timerDht=millis();
    if (temperature != DHT11::ERROR_CHECKSUM && temperature != DHT11::ERROR_TIMEOUT && humidity
!= DHT11::ERROR_CHECKSUM && humidity != DHT11::ERROR_TIMEOUT){
      DHTTemp=temperature;
      DHTHum=humidity;
      Serial.print(F("Temperatura: "));
      Serial.print(DHTTemp);
      Serial.print(F(" -- "));
```

```
      Serial.print(F("Umidità: "));
      Serial.println(DHTHum);
    }
    // invia dati ambientali rilevati
    inviaDati();
  }
}


/*
 * Invia dati se connessione WiFi presente
 * - identificativo dispositivo - hostname
 * - classe attualmente presente nella stanza
 * - temperatura
 * - umidità
 */
void inviaDati(){
  if (WiFi.status() == WL_CONNECTED){
    // dati da inviare
    String payload="?dev_id=" + String(WiFi.getHostname());
    payload += "&classe=" + String((char*)datiStato.classe);
    payload += "&temp=" + String(DHTTemp);
    payload += "&umid=" + String(DHTHum);
    std::unique_ptr<BearSSL::WiFiClientSecure>client(new BearSSL::WiFiClientSecure);  // set per
SSL
    client->setInsecure();
    HTTPClient http; // client http
    String url= urlScript + payload;
    Serial.println("Invio richiesta registrazione dati...");
    http.begin(*client,url.c_str());
    http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
    int httpCode = http.GET();  // richiesta con protocollo GET
    if (httpCode>0){
      // conferma di registrazione avvenuta
      Serial.println(http.getString());
    } else {
      // errore di registrazione
      Serial.print("Errore: ");
      Serial.print(httpCode);
      Serial.print(" ");
      Serial.println(http.errorToString(httpCode));
      WiFi.disconnect();
      WiFi.begin((char*)datiStato.SSID,(char*)datiStato.password);
    }
    http.end();
  }
}
```

Codice C++ del programmatore di schede RFID

```
/*
 *
 * connessioni per Arduino Uno:
 * -----------------------------------
 *             MFRC522      Arduino
 *             Reader/PCD   Uno
 * Signal      Pin          Pin
 * -----------------------------------
 * RST/Reset   RST          9
 * SPI SS      SDA(SS)      10
 * SPI MOSI    MOSI         11 / ICSP-4
 * SPI MISO    MISO         12 / ICSP-1
 * SPI SCK     SCK          13 / ICSP-3
 *
 */

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

#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN); // oggetto di classe MFRC522

MFRC522::MIFARE_Key key; // array per memorizzare le chiavi di lettura

// array per memorizzare l'UID della carta
byte nuidPICC[4];

void setup() {
  // inizializza interfaccia di collegamento con la base di lettura della carta
  Serial.begin(9600);
  delay(5000);
  SPI.begin(); // inizializza BUS SPI
  rfid.PCD_Init(); // inizializza circuito  MF-RC522

  // imposta la chiave di lettura della carta
  // i dati memorizzati nella carta sono crittografati con una chiave
  // la chiave iniziale è FFFFFFFFFFFF (hex)
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }

  Serial.println(F("Lettura carta classica MIFARE"));
  Serial.print(F("Chiave utilizzata:"));
```

```
  printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
  Serial.println();
}

void loop() {

  // Esce dal loop se non vi è nessuna carta rilevata
  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  // Esce dal loop se la carta non è stata correttamente letta
  if ( ! rfid.PICC_ReadCardSerial())
    return;

  Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
  Serial.println(rfid.PICC_GetTypeName(piccType));

  // controlla se la carta è del tipo compatibile con il lettore
  if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
    piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
    piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
    Serial.println(F("La carta non è di tipo MIFARE classic"));
    return;
  }

  // carta rilevata di formato corretto
  Serial.println(F("Carta rilevata."));

  // visualizza l'UID in esadecimale
  Serial.print(F("NUID carta: "));
  printHex(rfid.uid.uidByte, rfid.uid.size);
  Serial.println();

  // memorizza l'UID letto
  for (byte i = 0; i < 4; i++) {
    nuidPICC[i] = rfid.uid.uidByte[i];
  }

  // legge alcuni dati dai blocchi di memoria della carta
  byte data[18]; // buffer dati da scrivere o leggere nella carta - max numero di caratteri
per un blocco: 16

  // lettura dati blocco 1 - utilizzato per scrivere la classe nella carta di tipo classe
  if (!readDataFromBlock(1,data,18)) return;
  printDataBlock(data,1, "Classe");
  // lettura dati blocco 4 - utilizzato per scrivere il SSID nella carta di tipo setup AP
  if (!readDataFromBlock(4,data,18)) return;
```

```
  printDataBlock(data,4, "SSID WiFi");
  // lettura dati blocco 5 - utilizzato per scrivere la password dell'AP nella carta di tipo setup AP
  if (!readDataFromBlock(5,data,18)) return;
  printDataBlock(data,5, "Password WiFi");
  // lettura dati blocco 8 - utilizzato per scrivere il tipo di carta (1 = carta classe 2 = carta setup AP)
  if (!readDataFromBlock(8,data,18)) return;
  printDataBlock(data,8, "Tipo di carta");

  // richiede eventuale scrittura di nuovi dati nel settore indicato
  userInput((char*) data,2,F("Scrivere nuovi dati? (s/n + invio)"));
  if (data[0]=='s'){
    // richiede che tipo di carta scrivere
    userInput((char*) data,2,F("Che tipo di carta vuoi programmare (1=carta classe, 2=carta setup)? (1/2 + invio)"));
    if (data[0]=='1'){
      // carta di tipo CLASSE
      writeDataToBlock(8,data,1); // scrive il tipo di carta nel blocco 8 della carta
      userInput((char*) data,3,F("Inserisci nome classe (2 caratteri + invio)"));
      fillWithZeros(data,2);
      writeDataToBlock(1,data,2); // scrive la classe nel blocco 1 della carta
    } else if (data[0]=='2'){
      // carta di tipo SETUP
      writeDataToBlock(8,data,1); // scrive il tipo di carta nel blocco 8 della carta
      byte c=userInput((char*) data,17,F("Inserisci SSID AP (max 16 caratteri + invio)"));
      fillWithZeros(data,c);
      writeDataToBlock(4,data,c); // scrive SSID nel blocco 4 della carta
      c=userInput((char*) data,17,F("Inserisci password AP (max 16 caratteri + invio)"));
      fillWithZeros(data,c);
      writeDataToBlock(5,data,c); // scrive la password WiFi nel blocco 5 della carta
    }
  }

  // fine delle operazioni di lettura/scrittura
  stopReadingCard();
}


/**
 * Legge i 16 byte da un determinato blocco
 */
bool readDataFromBlock(byte block,byte data[], byte len){
  MFRC522::StatusCode status;

  status = rfid.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(rfid.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Autenticazione fallita: "));
```

```
    Serial.println(rfid.GetStatusCodeName(status));
    Serial.println(F("Rimuovere la carta."));
    return false;
  }

  status = rfid.MIFARE_Read(block, data, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Lettura fallita: "));
    Serial.println(rfid.GetStatusCodeName(status));
    Serial.println(F("Rimuovere la carta."));
    return false;
  }

  return true;
}

/**
 * Scrive i 16 byte da un determinato blocco
 */
bool writeDataToBlock(byte block, byte data[], byte len){
  MFRC522::StatusCode status;

  status = rfid.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, block, &key, &(rfid.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Autenticazione fallita: "));
    Serial.println(rfid.GetStatusCodeName(status));
    Serial.println(F("Rimuovere la carta."));
    return false;
  }

  status = rfid.MIFARE_Write(block, data, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Scrittura fallita: "));
    Serial.println(rfid.GetStatusCodeName(status));
    Serial.println(F("Rimuovere la carta."));
    return false;
  } else {
    Serial.println();
    Serial.println(F("Scrittura effettuata con successo."));
    return true;
  }
}

/**
 * Visualizza in esadecimale il contenuto dell'array
 */
void printHex(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
```

```
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
}

/**
 * Visualizza in decimale il contenuto dell'array
 */
void printDec(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(' ');
    Serial.print(buffer[i], DEC);
  }
}

/**
 * Visualizza i caratteri stampabili nell'array
 */
void printText(byte *data, byte len){
  for (byte i=0;i<len;i++){
    if (data[i] >= 0x10){
      Serial.write(data[i]);
    }
  }
  Serial.println();
}

/*
 * acquisisce da terminale l'input dell'utente e lo memorizza nell'array data
 * n è il numero massimo di caratteri da acquisire
 * prompt è il messaggio di prompt
 */
byte userInput(char * data, byte n, String prompt){
  byte c;
  Serial.print(prompt);
  Serial.setTimeout(20000);
  c=Serial.readBytesUntil('\n',data,n);
  Serial.println();
  return c;
}

/*
 * Stampa il contenuto di un blocco in esadecimale e in caratteri ASCII
 */
void printDataBlock(byte* data, byte block, String etichetta){
  Serial.print(F("Dati blocco  "));
  Serial.print(block);
  Serial.print(F(": "));
```

```
    printHex(data, 16);
    Serial.println();
    Serial.print(etichetta);
    Serial.print(F(": "));
    printText(data,16);
    Serial.println();
  }

  /*
   * riempie con zeri gli elementi di un vettore a partire da un determinato indice
   */
  void fillWithZeros(byte* data, byte startFrom){
    for (byte i = startFrom; i<18; i++){
      data[i]=0;
    }
  }

  /*
   * Ferma il rilevamento della carta
   */
  void stopReadingCard(){
    // ferma il rilevamento della carta
    rfid.PICC_HaltA();

    // ferma la crittografia dei dati
    rfid.PCD_StopCrypto1();

    Serial.println(F("Rimuovere la carta."));
  }
```

Codice dell'applicazione web sviluppata con Apps Script di Google

```
const spreadSheetID = "..." // id del foglio dati Google

/*
  Elabora i dati provenienti da una richiesta con protocollo HTTP GET
*/
function doGet (e) {
  const lock = LockService.getScriptLock()
  lock.tryLock(10000) // blocca il foglio

  try {
    // elabora la richiesta e verifica se è una richiesta di lettura o di registrazione
    // se contiene il campo getAula allora è una richiesta di lettura altrimenti di registrazione
    if (e.parameter['getAula']!=undefined){
```

```
      // richiesta di lettura
      lock.releaseLock()
      return ContentService.createTextOutput(leggiDati(e)).setMimeType(ContentService.MimeType.JSON)
    } else {
      // richiesta di registrazione
      registraDati(e)
      // toglie il blocco al foglio
      lock.releaseLock()
      return ContentService.createTextOutput('Dati registrati')
    }
  }

  catch (e) {
    // restituisce un messaggio di errore in formato JSON
    return ContentService.createTextOutput('Errore: ' + e)
  }

}

/*
  Legge i dati di una determinata aula richiesta con il parametro getAula
*/

function leggiDati(e){
  // riporta i dati richiesti in formato JSON
  const doc = SpreadsheetApp.openById(spreadSheetID)
  let sheet = doc.getSheetByName('generale')

  if (e.parameter['getAula']==0){
    // restituisce i dati di tutte le aule
    let index = sheet.getLastRow()
    let datiAule=[]
    for( let i =2; i<=index; i++){
      datiAule.push({
        aula: sheet.getRange(i,3).getValue().toString(),
        classe: sheet.getRange(i,4).getValue().toString(),
        temp: sheet.getRange(i,5).getValue().toString(),
        umid: sheet.getRange(i,6).getValue().toString(),
        devid: sheet.getRange(i,1).getValue().toString()
      })
    }
    return JSON.stringify({
      'result' : 'ok',
```

```
      'dati' : datiAule
    })
  } else {
    // restituisce i dati dell'aula specificata
    let aule=sheet.getRange("C:C").getValues()  // colonna dei numeri delle aule
    let index=aule.findIndex(function(em){return em==e.parameter['getAula']})
    if (index == -1){
      // aula non trovata: oggetto JSON con msg di errore
      return JSON.stringify({
        'result': 'error',
        'error': 'Aula inesistente'
        })
    } else {
      // aula trovata: costruisce oggetto JSON da restituire
      index++
      return JSON.stringify({
        'result': 'ok',
        'dati':[
          {
            aula: sheet.getRange(index,3).getValue().toString(),
            classe: sheet.getRange(index,4).getValue().toString(),
            temp: sheet.getRange(index,5).getValue().toString(),
            umid: sheet.getRange(index,6).getValue().toString(),
            devid: sheet.getRange(index,1).getValue().toString()
          }
        ]
        })
    }
  }

}



/*
  Registra i dati ricevuti
  formato dei campi del POST:
  - dev_id : id del dispositivo Wemos D1 [stringa]
  - stanza : numero della stanza in cui il dispositivo è posizionato [stringa]
  - classe : classe presente nella stanza indicata [stringa]
  - temp   : temperatura rilevata nella stanza [°C]
  - umid   : umidità rilevata nella stanza [%]
*/
function registraDati(e){
```

```
  const doc = SpreadsheetApp.openById(spreadSheetID)
  let sheet = doc.getSheetByName('generale')

  let codici=sheet.getRange("A:A").getValues()  // colonna dei codici ID dei dispositivi
  let index=codici.findIndex(function(em){return em==e.parameter['dev_id']})
  if (index == -1){
    // dispositivo non registrato
    // crea una nuova riga nel foglio generale
    index = sheet.getLastRow() + 1
  } else {
    index +=1;
  }
  // memorizza i dati nel foglio generale
  sheet = doc.getSheetByName('generale')
  sheet.getRange(index,2).setValue(new Date())
  const idCell = sheet.getRange(index,1)
  sheet.getRange(index,4).setValue(e.parameter['classe'])
  sheet.getRange(index,5).setValue(e.parameter['temp'])
  sheet.getRange(index,6).setValue(e.parameter['umid'])
  // memorizza i dati nel foglio dello storico del dispositivo specifico
  sheet = doc.getSheetByName(e.parameter['dev_id'])
  if (!sheet){
    // crea un nuovo foglio per lo storico delle registrazioni
    creaNuovoFoglio(e.parameter['dev_id'])
  }
  sheet = doc.getSheetByName(e.parameter['dev_id'])
  idCell.setValue(e.parameter['dev_id'])
  const hrefValue =
SpreadsheetApp.newRichTextValue().setText(e.parameter['dev_id']).setLinkUrl(doc.getUrl()+"#gid=" +
sheet.getSheetId()).build()
  idCell.setRichTextValue(hrefValue)
  index = sheet.getLastRow() + 1
  sheet.getRange("D3").setValue(e.parameter['dev_id'])
  sheet.getRange(index,1).setValue(new Date())
  sheet.getRange(index,2).setValue(e.parameter['temp'])
  sheet.getRange(index,3).setValue(e.parameter['umid'])
  sheet.getRange(index,4).setValue(e.parameter['classe'])
}

/*
 * Crea un nuovo foglio di lavoro copia del foglio modello
 */
function creaNuovoFoglio(nome){
```

```
  const doc = SpreadsheetApp.openById(spreadSheetID)
  const modello = doc.getSheetByName('modello')
  doc.insertSheet(nome,doc.getSheets().length,{template: modello})
  doc.getActiveSheet().showSheet()
}
```

Codice HTML, CSS e Javascript delle pagine del sito web

Il codice è disponibile nella cartella zip:
SOLACE _HtmlCssJs.zip