Smart Garage Door

Corso di
Laboratorio IOT

By

Francesco Ranno - A13002382
Antonio Celiento - A13002205
Davide Flagiello - A13002377

Dipartimento di Ingegneria Informatica
Università degli Studi della Campania Luigi Vanvitelli

Indice

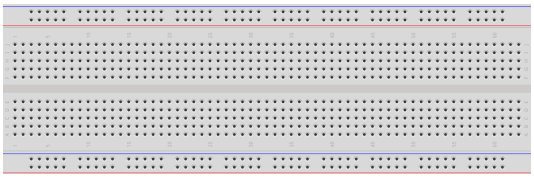
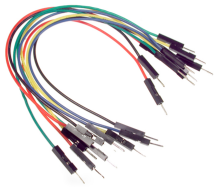
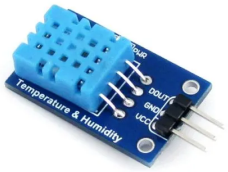
2x Breadboard	11x Jumper M/M
	
2x Servo Motor SG90	2x Freenove ESP32
	
1x DHT Sensor	1x LED Compnent
	

Tabella 1: Tabella dei componenti.

1 Introduzione

Il presente progetto propone la realizzazione di un sistema Smart Garage Door basato su ESP32 e Arduino, due tra le piattaforme più utilizzate nell'ambito dell'IOT. Attraverso l'integrazione di diverse componenti, descritte nel dettaglio poco più avanti nel documento, il sistema permetterà agli utenti di aprire, chiudere la porta del garage da remoto e anche di monitorare diversi parametri all'interno della stanza, tutto tramite un'interfaccia semplice e intuitiva. Gli utenti potranno accedere ad uno storico dettagliato dell'utilizzo attraverso l'impiego di un database per la memorizzazione di dati utili per avere una supervisione totale dell'ambiente.

2 Componenti utilizzati e collegamenti

2.1 Componenti

Nella Tabella 1 sono illustrati tutti i componenti utilizzati per questo progetto. Il componente principale è sicuramente il Servo Motor SG90 che utilizza 3 cavi: rosso, arancione e marrone. Il cavo rosso è il cavo di alimentazione positivo (+), il cavo marrone è il cavo collegato al ground (GND) (-) della nostra scheda di controllo e per ultimo il cavo arancione viene utilizzato per il segnale di controllo. Il segnale inviato tramite quest'ultimo cavo determina l'angolo di rotazione del servo motor[1]. Tutti i collegamenti sono bel visibili in Figura 1 e in Figura 2.

[1]Come descritto nella datasheet del componente il servo motor può rotare approssimativamente di 180° ossia 90° in senso orario oppure in senso antiorario

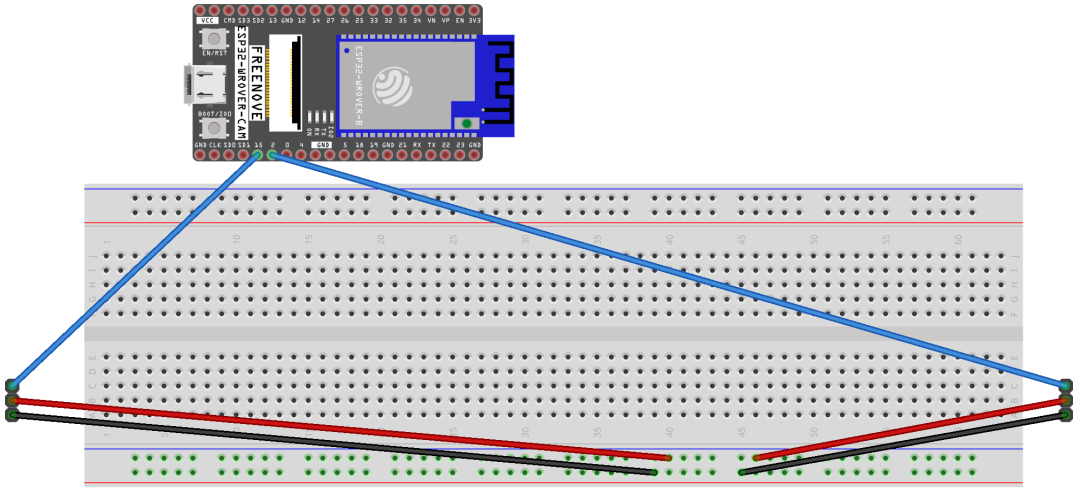


Figura 1: Collegamenti della prima esp32

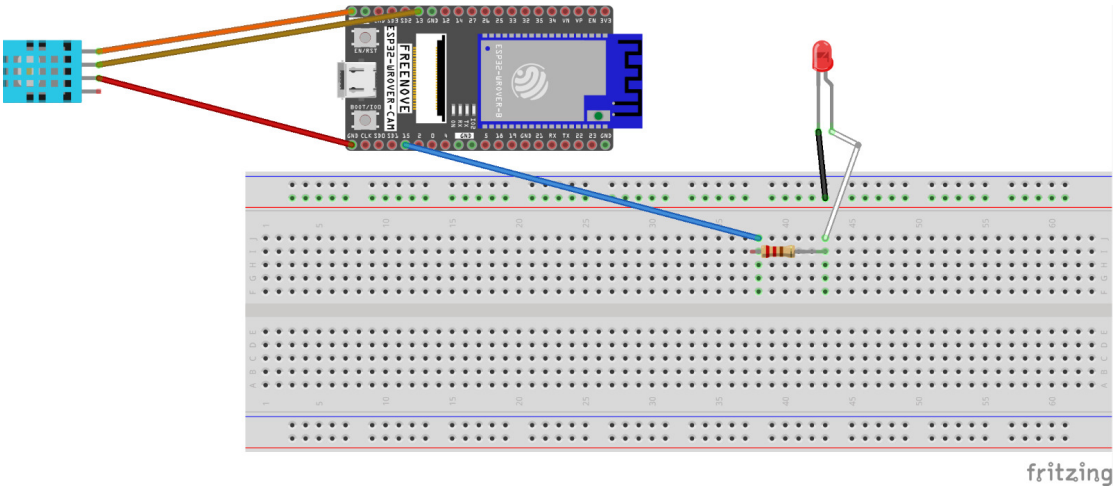


Figura 2: Collegamenti della seconda esp32

3 Base di dati

3.1 Progettazione logica

$Utenti(\mathbf{ID}, Email, Password\_hash)$
$Valori\_Sensori(\mathbf{ID}, temperatura, timestamp, umidita)$
$Stato\_Garage(\mathbf{ID}, stato\_luce, stato\_porta, timestamp)$

3.2 Progettazione concettuale

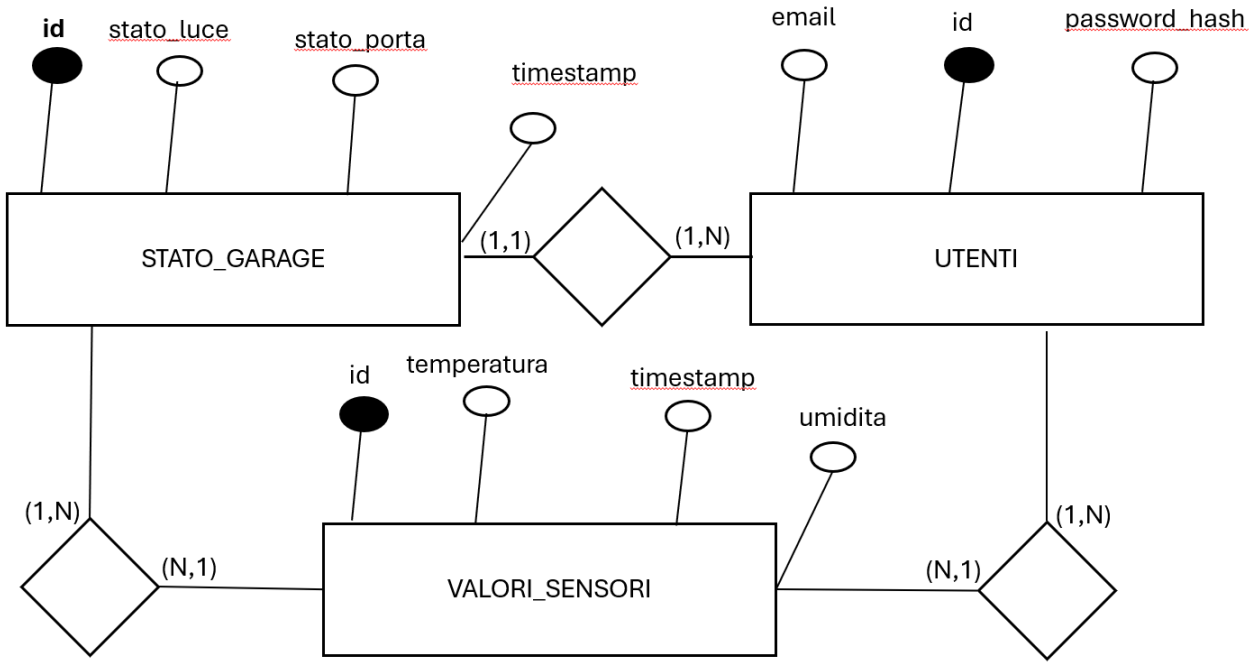


Figura 3: Progettazione concettuale

4 Trasmissione Dati

Il ciclo di vita dei dati inizia con la generazione da parte dei sensori, seguita dalla trasmissione tramite HTTP al server PHP, che infine memorizza i dati nella base di dati. La scelta del protocollo HTTP è motivata dalla sua semplicità e dalla disponibilità di supporto nelle librerie ESP32. La trasmissione dei dati avviene in maniera asincrona, attivata da specifiche richieste HTTP basate sugli eventi rilevati.

4.1 Generazione dei Dati

- **Sensore DHT11:** Misura la temperatura e l'umidità e invia i dati al microcontroller ESP32.
- **Pin di Controllo Luce (LUCE_PIN):**Controlla lo stato della luce (accesa o spenta) tramite il microcontroller ESP32.
- **Servo Motori:** Controllano l'apertura e la chiusura del garage, gestiti anch'essi dall'ESP32.

4.2 Trasmissione dei Dati

- **Connessione WiFi:** L'ESP32 si connette alla rete WiFi utilizzando le credenziali specificate (ssid e password).
- **Server Web:**L'ESP32 ospita un server web sulla porta 80, che gestisce richieste HTTP.
- **Richieste HTTP Client:** L'ESP32 invia richieste HTTP GET ai server PHP per memorizzare i dati e aggiornare gli stati.

- **Baudrate:** settato a 115200, è una velocità relativamente alta che consente una rapida trasmissione di dati tra l'ESP32 e il computer.

5 Scenari e Casi d'uso

5.1 Requisiti funzionali

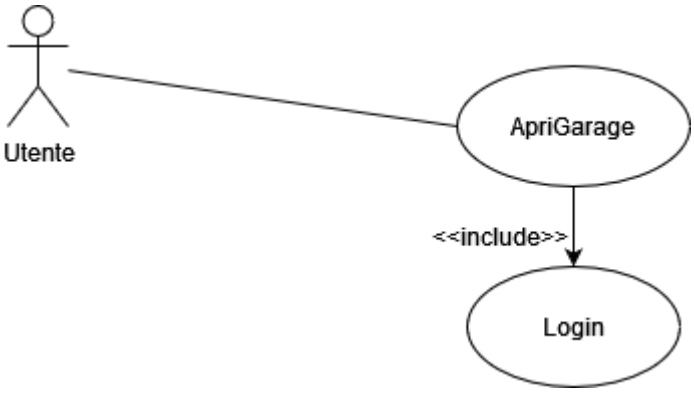
- Il sistema consentirà all'utente di aprire la porta del garage da remoto.
- Il sistema consentirà all'utente di chiudere la porta del garage da remoto.
- Il sistema consentirà all'utente di aprire la porta del garage tramite bluetooth.
- Il sistema consentirà all'utente di chiudere la porta del garage tramite bluetooth.
- Il sistema consentirà all'utente di accendere la luce all'interno del garage.
- Il sistema consentirà all'utente di spegnere la luce all'interno del garage.

5.2 Requisiti non funzionali

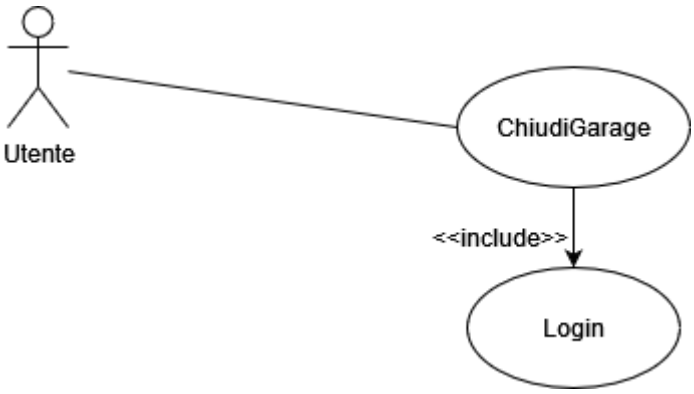
- **Affidabilità:** Il sistema deve essere affidabile, garantendo che la porta del garage si apra e si chiuda correttamente in risposta ai comandi dell'utente.
- **Facilità d'Uso:** L'interfaccia utente del sistema, inclusa l'applicazione mobile, deve essere intuitiva e di facile utilizzo per gli utenti.
- **Sicurezza:** Il sistema deve garantire la sicurezza dell'accesso al garage, impedendo l'accesso non autorizzato.

5.3 Casi d'uso

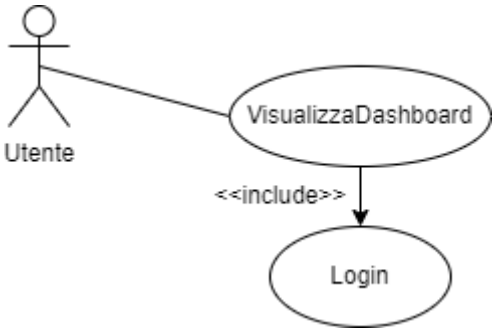
5.3.1 Apri Garage



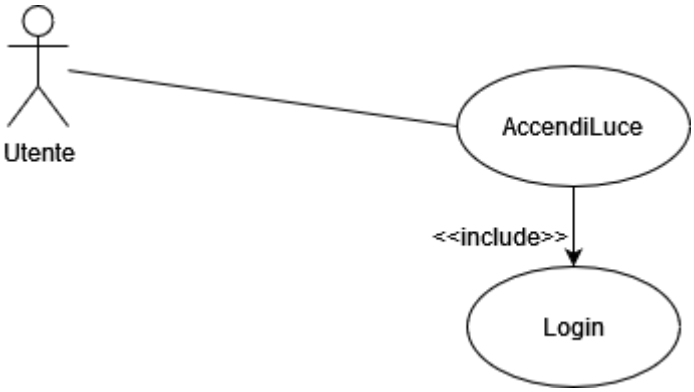
5.3.2 Chiudi Garage



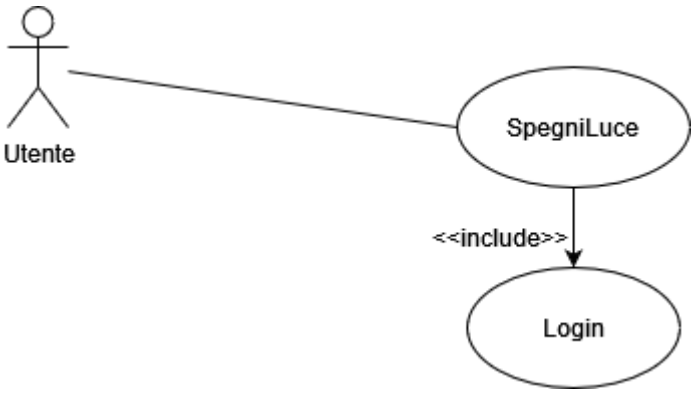
5.3.3 Visualizza Dashboard



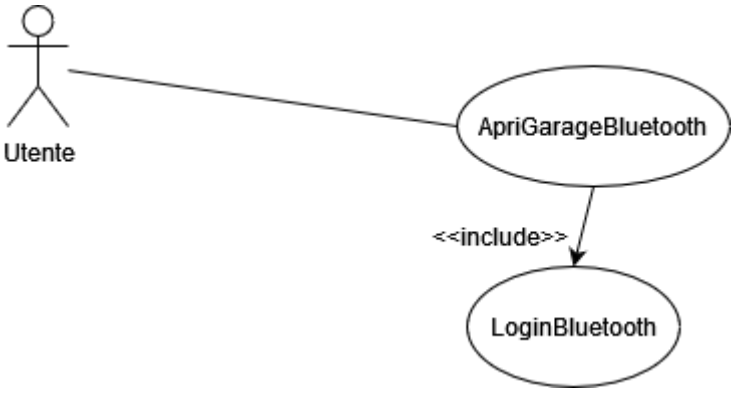
5.3.4 Accendi Luce



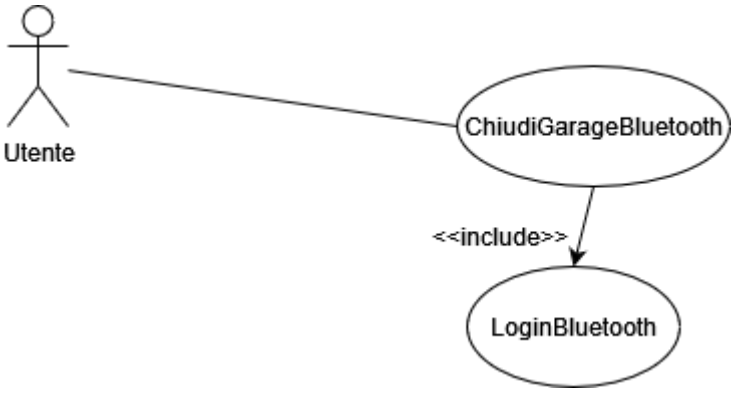
5.3.5 Spegni Luce



5.3.6 Apri Garage Bluetooth



5.3.7 Chiudi Garage Bluetooth



5.4 Scenari

5.4.1

Caso d'uso	Apri Garage
ID	ApriGarage
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente ha fatto l'accesso
Sequenza degli eventi principali	1. Il caso d'uso inizia quando l'utente preme sul pulsante "Apri" 2. Il sistema invoca il codice per l'attivazione dei motori Servo 3. Fisicamente la porta del garage si apre
Postcondizioni	La porta del garage è aperta

5.4.2

Caso d'uso	Accendi Luce
ID	AccendiLuce
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente ha fatto l'accesso
Sequenza degli eventi principali	1. Il caso d'uso inizia quando l'utente preme sul pulsante " ON Luce" 2. Il sistema invoca il codice per l'accensione della luce 3. La luce presente all'interno del garage si accende
Postcondizioni	La luce del garage è accesa

5.4.3

Caso d'uso	Spegni Luce
ID	SpegniLuce
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente ha fatto l'accesso
Sequenza degli eventi principali	1. Il caso d'uso inizia quando l'utente preme sul pulsante " OFF Luce" 2. Il sistema invoca il codice per lo spegnimento della luce 3. La luce presente all'interno del garage si spegne
Postcondizioni	La luce del garage è spenta

5.4.4

Caso d'uso	Chiudi Garage
ID	ChiudiGarage
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente ha fatto l'accesso
Sequenza degli eventi principali	1. Il caso d'uso inizia quando l'utente preme sul pulsante "Chiudi" 2. Il sistema invoca il codice per l'attivazione dei motori Servo 3. Fisicamente la porta del garage si chiude
Postcondizioni	La porta del garage è chiusa

5.4.5

Caso d'uso	Visualizza Dashboard
ID	VisualizzaDashboard
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente ha fatto l'accesso
Sequenza degli eventi principali	1. Il caso d'uso inizia quando l'utente ha fatto l'accesso. 2. Verrà stampata sulla dashboard una serie di informazioni relative ai sensori
Postcondizioni	La Dashboard è aggiornata
Sequenza degli eventi alternativa	2A. Se le informazioni dell'utente sono errate rimarrà sulla pagina di login

5.4.6

Caso d'uso	ApriGarageBluetooth
ID	ApriGarageBluetooth
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente si è collegato tramite bluetooth all'esp32
Sequenza degli eventi principali	1. L'utente digita la password sul terminale bluetooth 2. Il sistema invoca il codice per verificare le credenziali dell'utente 3. Fisicamente la porta del garage si apre
Postcondizioni	La porta del garage è aperta
Sequenza degli eventi alternativa	2A. Se le informazioni dell'utente non sono corrette verrà stampato a video "password errata"

5.4.7

Caso d'uso	ChiudiGarageBluetooth
ID	ChiudiGarageBluetooth
Attori primari	Utente
Attori secondari	ESP32
Precondizioni	L'utente si è collegato tramite bluetooth all'esp32
Sequenza degli eventi principali	1. L'utente digita la password sul terminale bluetooth 2. Il sistema invoca il codice per verificare le credenziali dell'utente 3. Fisicamente la porta del garage si chiude
Postcondizioni	La porta del garage è chiusa
Sequenza degli eventi alternativa	2A. Se le informazioni dell'utente non sono corrette verrà stampato a video "password errata"

6 Codice ARDUINO

Il codice è suddiviso in due programmi principali ognuno caricato sulla propria esp32, più avanti sono riportati i codici Arduino mentre tutto il codice è reperibile sul nostro github: Link

6.1 Librerie utilizzate

Con riferimento alla prima esp (servo motori)

- ESP32Servo.h: Libreria per controllare i servo motori con la ESP32.
- WiFi.h: Permette la connessione a una rete WiFi.
- HTTPClient.h: Permette di fare richieste HTTP.
- WebServer.h: Permette di creare un server web sulla ESP32.

Con riferimento alla seconda esp (luce e sensore DHT):

- WiFi.h
- HTTPClient.h
- WebServer.h
- DHT.h: Libreria per interfacciarsi con il sensore DHT11 (temperatura e umidità).

6.2 Codice prima esp32

```
#include <ESP32Servo.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WebServer.h>

// Credenziali WiFi
const char* ssid_router = "Hotspot 374";
const char* password_router = "123456dK";

// Indirizzo IP del server PHP
const char* serverName = "http://192.168.212.223"; // Sostituisci con l'IP del tuo
    server

// Pin dei servo motori
#define SERVO1_PIN 15
```

```
15 #define SERVO2_PIN 2

17 Servo servo1;
   Servo servo2;
19 int posVal = 0;
   int posVal2 = 0;
21
   // Server web sulla porta 80
23 WebServer server(80);

25 void setup() {
     Serial.begin(115200);
27
     // Configurazione dei servo motori
29   servo1.setPeriodHertz(50);
     servo2.setPeriodHertz(50);
31   servo1.attach(SERVO1_PIN, 500, 2500);
     servo2.attach(SERVO2_PIN, 500, 2500);
33
     // Connessione alla rete WiFi
35   WiFi.begin(ssid_router, password_router);
     Serial.println("Connessione WiFi in corso...");
37   while (WiFi.status() != WL_CONNECTED) {
       delay(500);
39     Serial.print(".");
     }
41   Serial.println("\nConnesso alla rete WiFi");
     Serial.print("Indirizzo IP: ");
43   Serial.println(WiFi.localIP());

45   // Configurazione delle route del server
     server.on("/apri_garage", handleApriGarage);
47   server.on("/chiudi_garage", handleChiudiGarage);

49   // Avvio del server
     server.begin();
51   Serial.println("Server HTTP avviato");
   }
53
   void loop() {
55   server.handleClient();
   }
57
   void handleApriGarage() {
59   while (posVal < 100 && posVal2 < 100) {
       servo1.write(100 - posVal); // Muove il servo1 in senso opposto
61     servo2.write(posVal2);
       posVal++;
63     posVal2++;
       delay(20);
65   }
     inviaStatoGarage("aperta");
67   server.send(200, "text/plain", "Garage aperto");
   }
69
   void handleChiudiGarage() {
71   while (posVal > 0 && posVal2 > 0) {
       servo1.write(100 - posVal); // Muove il servo1 in senso opposto
73     servo2.write(posVal2);
       posVal--;
75     posVal2--;
       delay(20);
77   }
     inviaStatoGarage("chiusa");
79   server.send(200, "text/plain", "Garage chiuso");
   }
81
   void inviaStatoGarage(const char* stato) {
83   if (WiFi.status() == WL_CONNECTED) {
       HTTPClient http;
85     String url = String(serverName) + "/aggiorna_stato_garage.php?stato_garage=" +
       stato;
       http.begin(url.c_str());
87     int httpResponseCode = http.GET();
       if (httpResponseCode > 0) {
89       String response = http.getString();
```

```
      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Errore nella richiesta HTTP: ");
      Serial.println(httpResponseCode);
    }
    http.end();
  }
}
```

6.3 Codice seconda esp32

```
    #include <WiFi.h>
#include <HTTPClient.h>




#include <WebServer.h>



#include "DHT.h"

#define DHTPIN 13
#define DHTTYPE DHT11

#define LUCE_PIN 15

const char* ssid = "Hotspot 374";
const char* password = "123456dK";
const char* serverName = "http://192.168.212.223/save_data.php"; // IP del computer

WebServer server(80);
DHT dht(DHTPIN, DHTTYPE);

String stato_luce = "OFF"; // Dichiarazione della variabile stato_luce

void setup() {
  Serial.begin(115200);
  setup_wifi();

  server.on("/luce", handleLuce);
  server.on("/dati_sensori", handleDatiSensori);
  server.on("/stato", handleStato);

  server.begin();

  pinMode(LUCE_PIN, OUTPUT);
  dht.begin();
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP()); // Stampa l'indirizzo IP della ESP32
}

void handleLuce() {
```

```
  String cmd = server.arg("cmd");
  if (cmd == "accendi") {
    digitalWrite(LUCE_PIN, HIGH);
    stato_luce = "ON";
    server.send(200, "text/plain", "Luce accesa");
  } else if (cmd == "spegni") {
    digitalWrite(LUCE_PIN, LOW);
    stato_luce = "OFF";
    server.send(200, "text/plain", "Luce spenta");
  } else {
    server.send(400, "text/plain", "Comando non valido");
  }
}

void handleDatiSensori() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
    server.send(500, "text/plain", "Errore nella lettura del sensore DHT");
    return;
  }
  String data = "Temperatura: " + String(t) + " C , Umidit : " + String(h) + "%";
  server.send(200, "text/plain", data);
  sendDataToServer(t, h);
}

void handleStato() {
  String stato = "{\"luce\":\"" + stato_luce + "\"}";
  server.send(200, "application/json", stato);
}

void sendDataToServer(float temperatura, float umidita) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String serverPath = String(serverName) + "?temperatura=" + String(temperatura)
    + "&umidita=" + String(umidita);

    http.begin(serverPath.c_str());
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Errore nella richiesta HTTP: ");
      Serial.println(httpResponseCode);
    }
    http.end();
  } else {
    Serial.println("WiFi Disconnesso");
  }
}

void loop() {
  server.handleClient();
}
```

6.4 Codice per apertura/chiusura Bluetooth

```
   #include <ESP32Servo.h>
#include <BluetoothSerial.h>

// Pin dei servo motori
#define SERVO1_PIN 15
#define SERVO2_PIN 2

Servo servo1;
Servo servo2;
int posVal = 0;
int posVal2 = 0;

```

```
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32_Garage"); // Nome Bluetooth

  // Configurazione dei servo motori
  servo1.setPeriodHertz(50);
  servo2.setPeriodHertz(50);
  servo1.attach(SERVO1_PIN, 500, 2500);
  servo2.attach(SERVO2_PIN, 500, 2500);

  Serial.println("Il dispositivo    pronto per la connessione Bluetooth.");
}

void loop() {
  if (SerialBT.available()) {
    String command = SerialBT.readStringUntil('\n');
    command.trim();

    if (command.equals("apri_garage")) {
      handleApriGarage();
    } else if (command.equals("chiudi_garage")) {
      handleChiudiGarage();
    }
  }
}

void handleApriGarage() {
  while (posVal < 100 && posVal2 < 100) {
    servo1.write(100 - posVal); // Muove il servo1 in senso opposto
    servo2.write(posVal2);
    posVal++;
    posVal2++;
    delay(20);
  }
  SerialBT.println("Garage aperto");
}

void handleChiudiGarage() {
  while (posVal > 0 && posVal2 > 0) {
    servo1.write(100 - posVal); // Muove il servo1 in senso opposto
    servo2.write(posVal2);
    posVal--;
    posVal2--;
    delay(20);
  }
  SerialBT.println("Garage chiuso");
}
```

7 Conclusione

Degli eventuali sviluppi futuri del progetto possono essere:

- **Crittografia delle comunicazioni:** Utilizzare HTTPS anziché HTTP per garantire che i dati trasmessi siano crittografati.
- **Automazione:** Implementare logiche di automazione basate su condizioni specifiche, ad esempio accendere la luce automaticamente quando l'umidità supera una certa soglia.
- **Risparmio energetico:** Tramite l'implementazione di Mosquitto.
- **App mobile:** Sviluppare un'app mobile dedicata per un accesso e controllo più intuitivi.