

● ● ARCHITETTURA E PROGETTO DI CALCOLATORI ● ●



SPYHOLE PRESENTATION

prof. Nicola Mazzocca

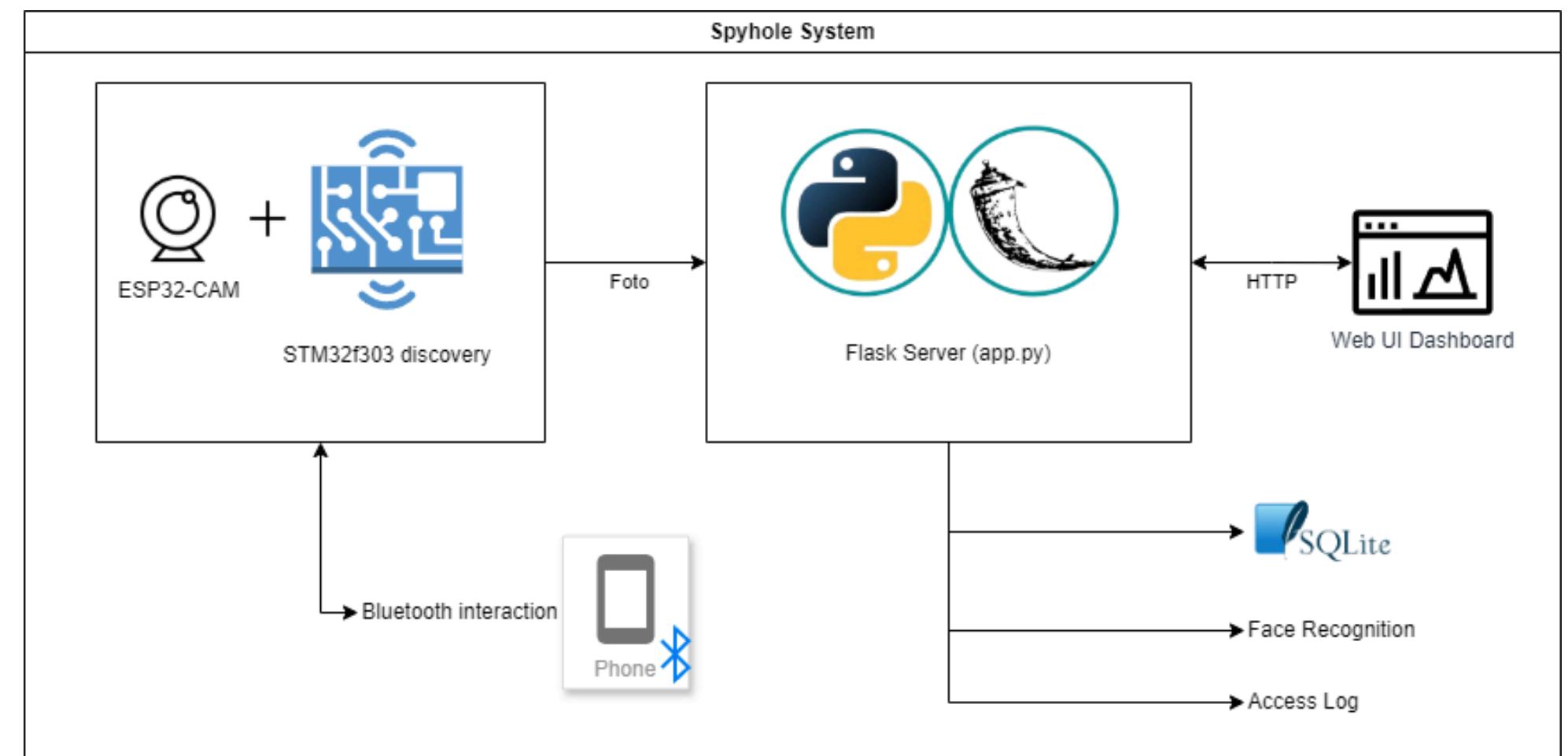
Andrea Esposito M63001650
Francesco Riccio M63001646

PANORAMICA DEL SISTEMA

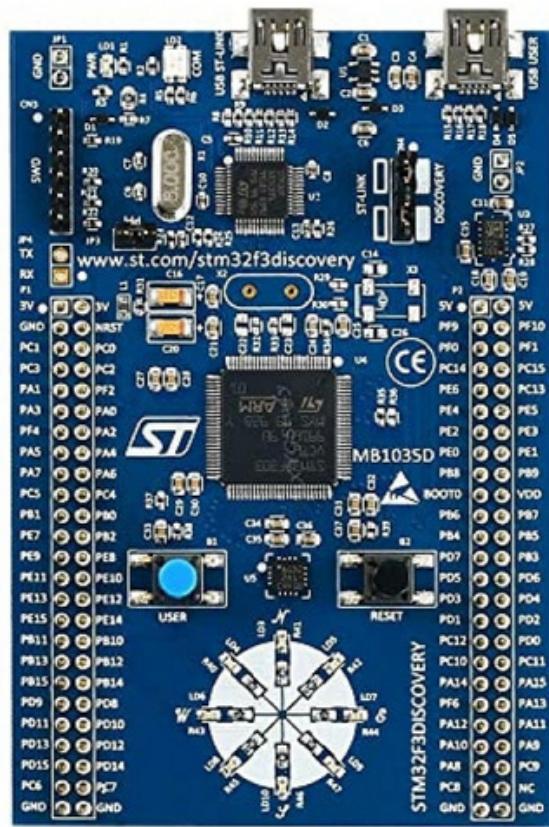
Spyhole è un sistema di controllo di accessi a due fattori:

- riconoscimento facciale
- codice PIN Bluetooth

Il tutto è gestito da una dashboard web, che consente di monitorare gli accessi in tempo reale e gestire utenti registrati



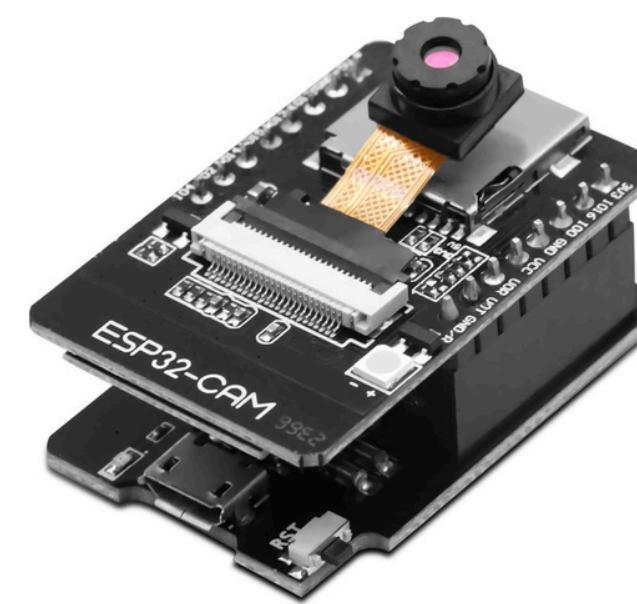
COMPONENTI UTILIZZATI



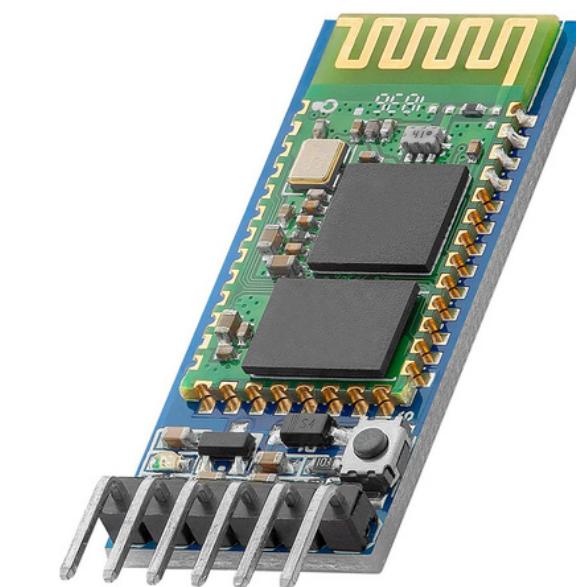
STM32F303 DISCOVERY



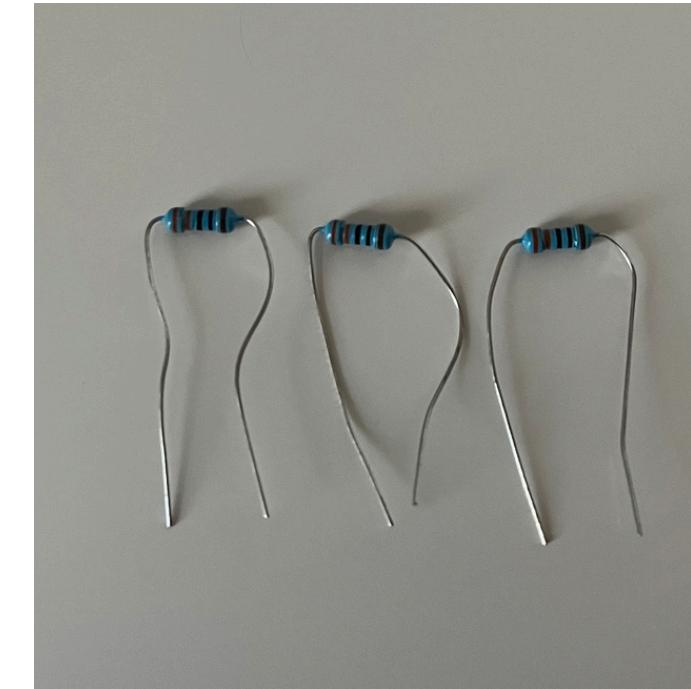
SERVOMOTORE SG90



ESP-32 CAM



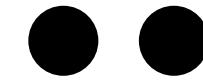
MODULO BLUETOOTH HC05



RESISTENZE 0.25 W 330 OHM

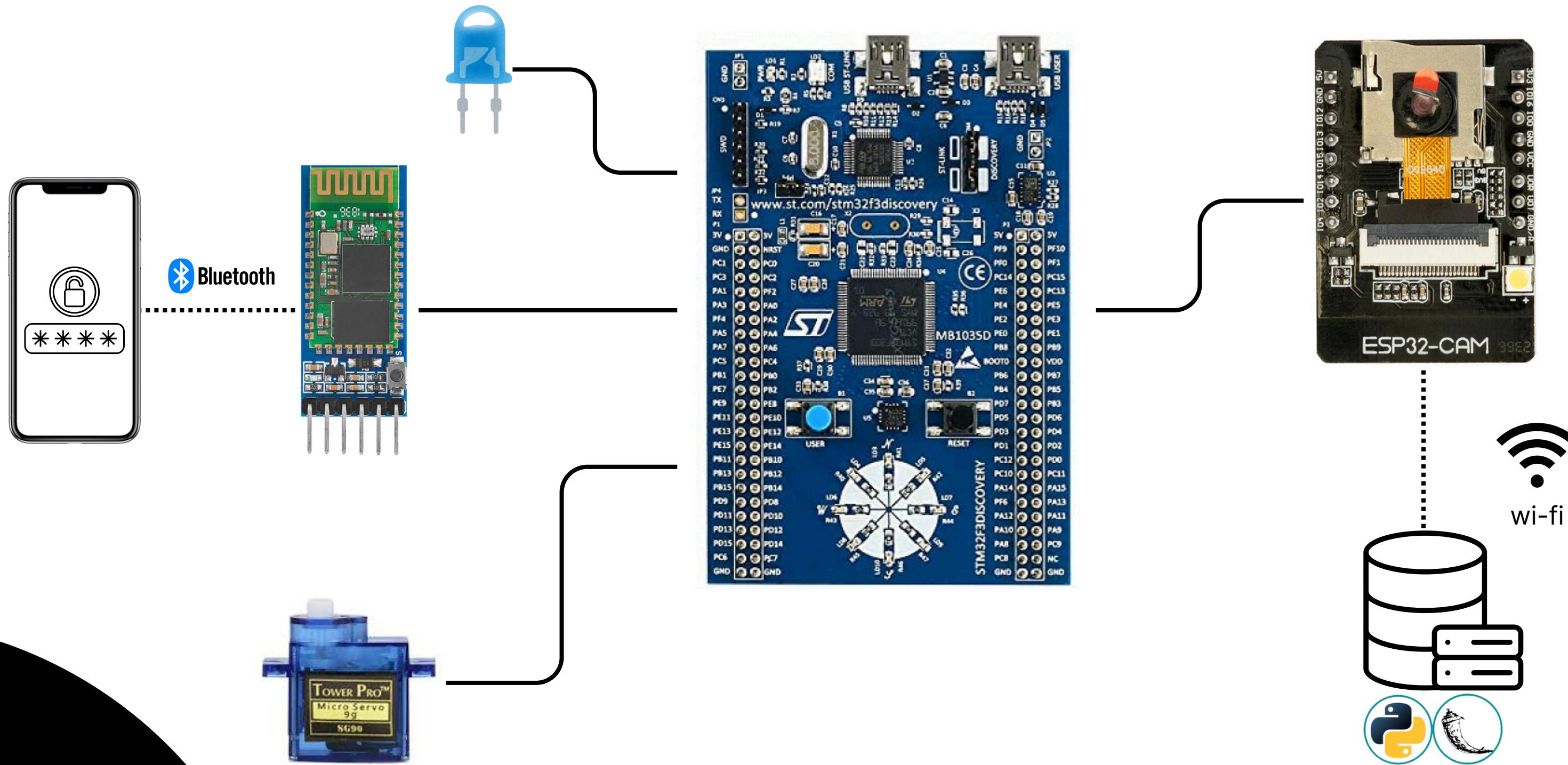


LED RGB



SCHEMA DI COLLEGAMENTO

PAGE 04





FLUSSO DI LAVORO



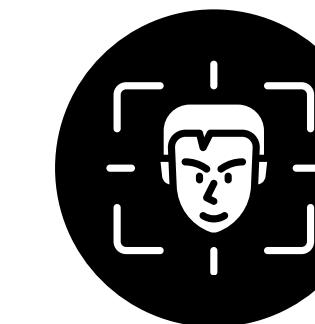
REGISTRAZIONE/LOGIN

- L'utente si autentica nel sistema o viene registrato per la prima volta.
- I dati biometrici (volto) e credenziali (email e Password) vengono associati all'account.



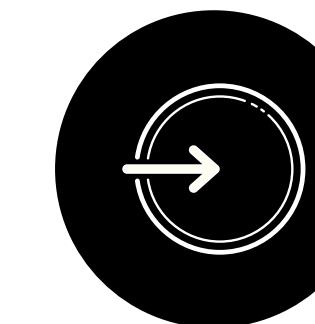
RICHIESTA D'ACCESSO

- L'utente invia il comando “Access” tramite interfaccia Bluetooth.



RICONOSCIMENTO FACCIALE

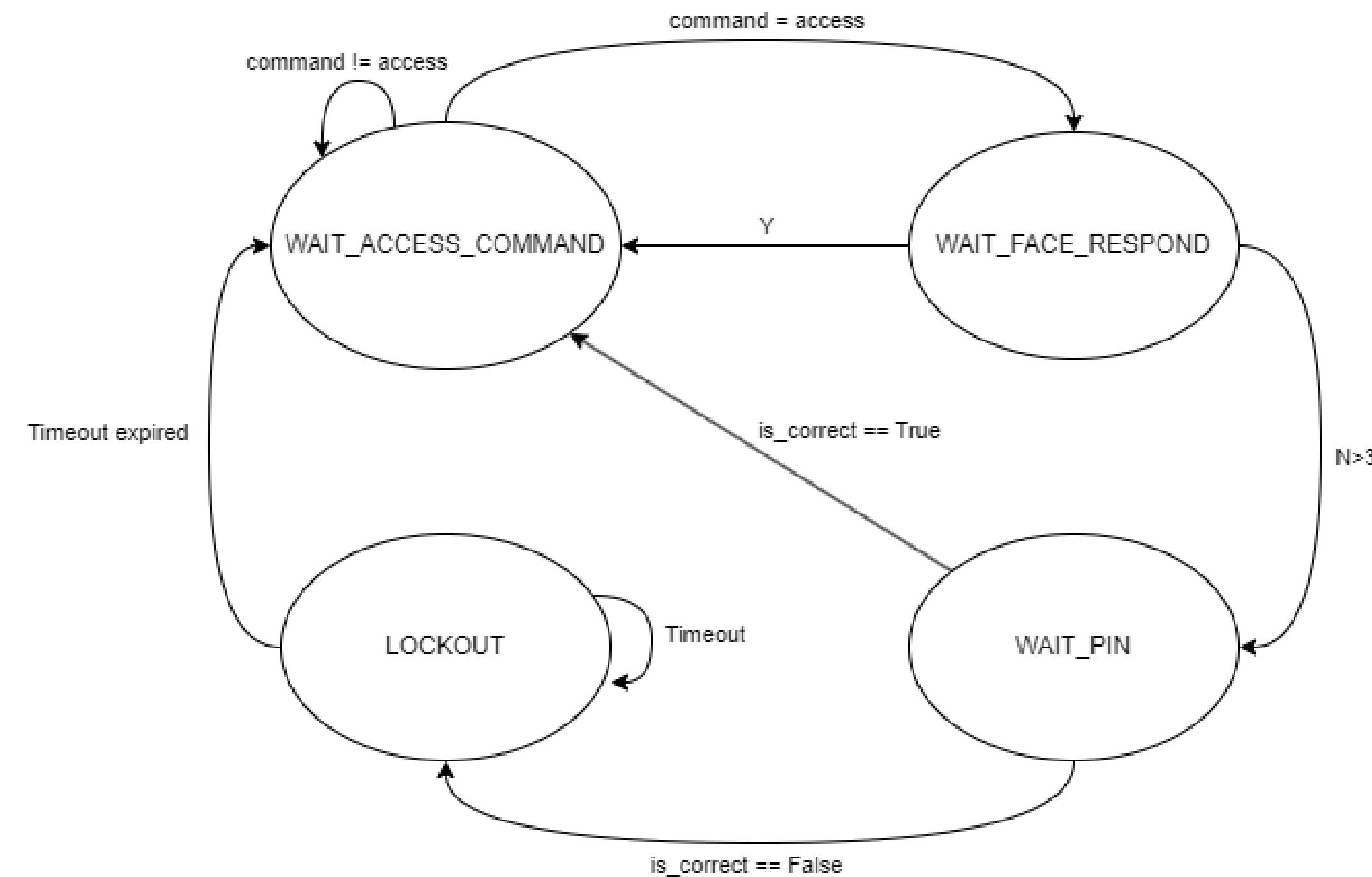
- La ESP32-CAM cattura l'immagine del volto e la confronta con i dati registrati.
- Se il riconoscimento è positivo, il servo si attiva e l'accesso è concesso.

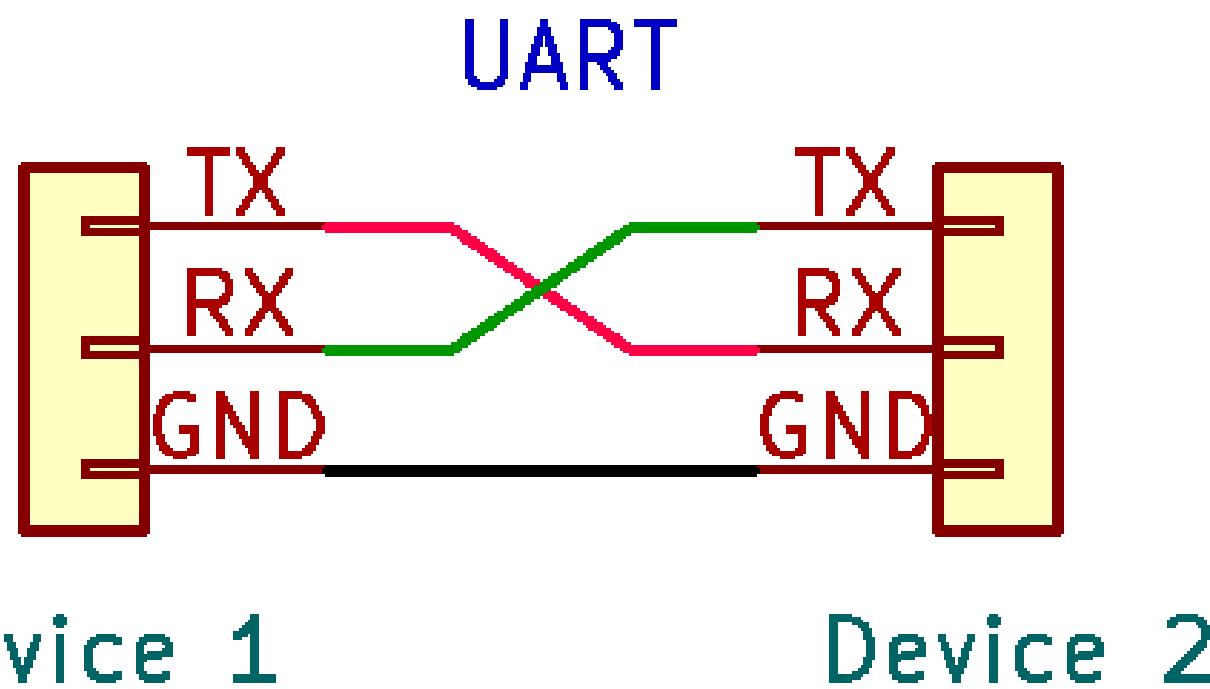


ACCESSO

- Il sistema avvia il servomotore che sblocca la serratura
- In caso di troppi errori, entra in **LOCKOUT** per un periodo di sicurezza.

MACCHINA A STATI FINITI





COMUNICAZIONE CON UART

Sia l'STM32 che l'ESP32 utilizzano un modulo UART hardware integrato, basato sullo stesso protocollo di comunicazione seriale. Grazie a questa compatibilità, i due dispositivi possono scambiarsi dati direttamente senza adattatori aggiuntivi. Inoltre, è presente anche il modulo Bluetooth HC-05, che comunica anch'esso con la STM32 tramite una seconda UART hardware. In questo modo, l'STM32 può ricevere comandi via Bluetooth dall'utente e, allo stesso tempo, dialogare con l'ESP32 per la gestione del riconoscimento facciale, mantenendo una comunicazione indipendente e parallela su due canali UART separati.

I collegamenti tra la STM32, il modulo Bluetooth HC-05 e la ESP32-CAM seguono lo schema standard UART:

- TX → RX
- RX → TX
- GND in comune tra tutti i dispositivi

La comunicazione è full-duplex, quindi trasmissione e ricezione avvengono contemporaneamente su linee separate.

Nel progetto vengono utilizzate due UART distinte:

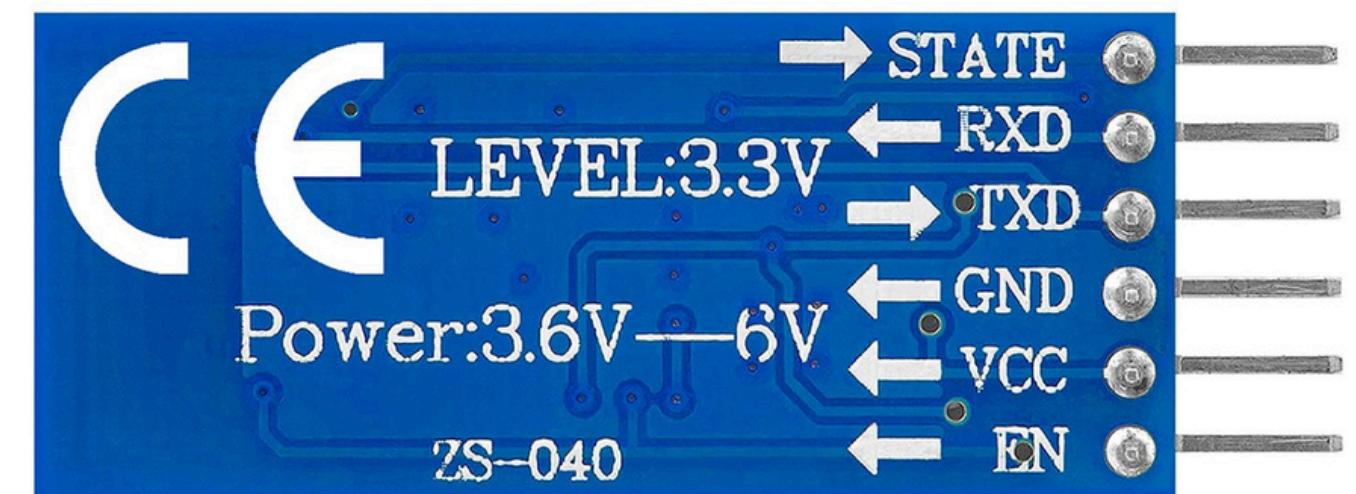
- HC-05 (Bluetooth): comunicazione con la STM32 a 9600 baud
- ESP32-CAM: comunicazione con la STM32 a 115200 baud

Sia l'HC-05 che la ESP32-CAM vengono alimentati tramite il pin a 5V della scheda STM32. Nonostante l'alimentazione a 5V, le linee di comunicazione seriale TX/RX dei due moduli operano a livello logico 3.3V, risultando quindi pienamente compatibili con la UART della STM32. Per questo motivo non è necessario utilizzare convertitori di livello tra i dispositivi.

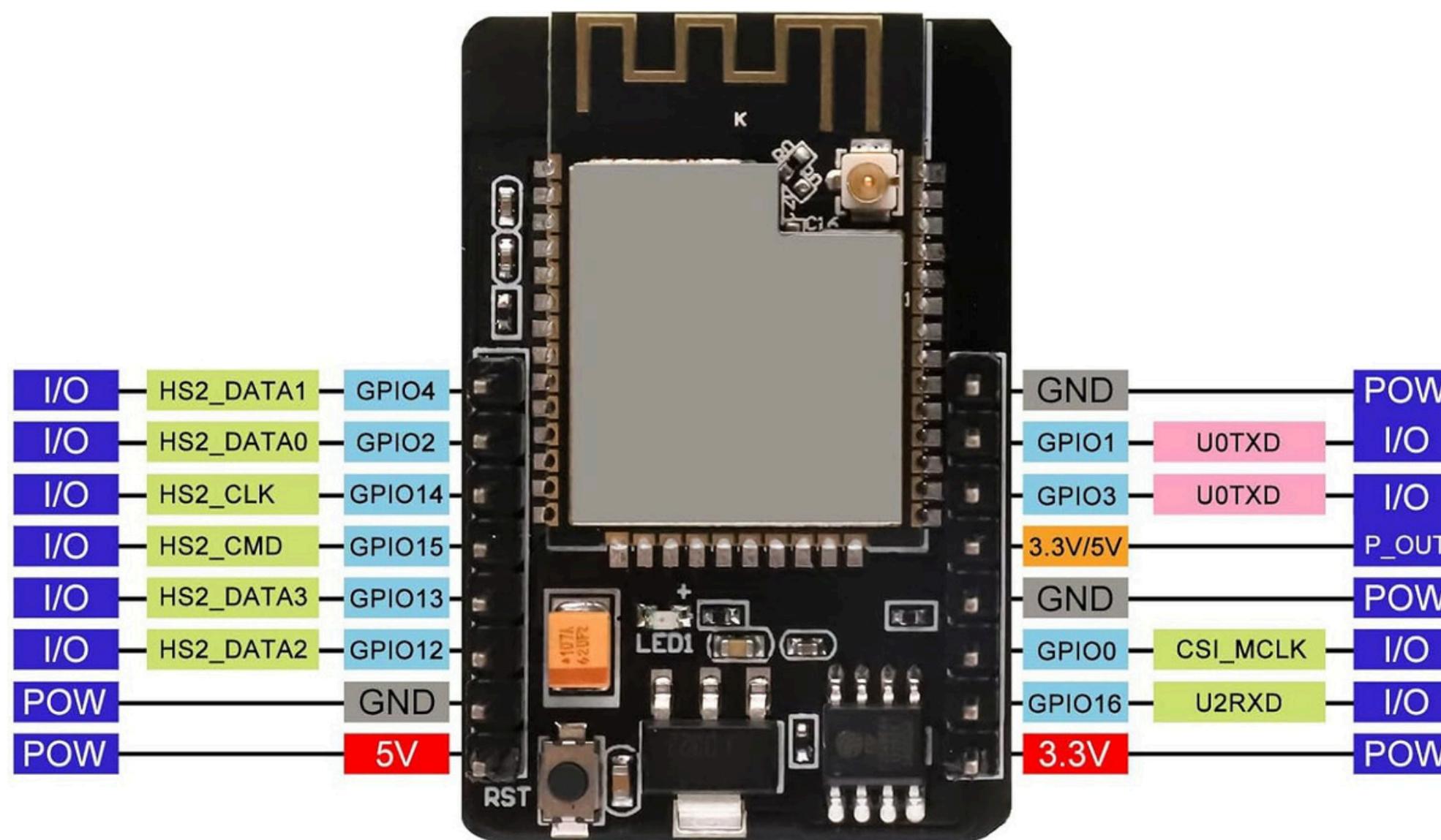


MODULO BLUETOOTH HC05

Il modulo Bluetooth HC-05 è un dispositivo pensato per aggiungere **comunicazione wireless** ai microcontrollori. È basato sul profilo SPP (Serial Port Profile), il che significa che **si comporta come una normale porta seriale, ma senza l'uso di cavi**. La comunicazione con la STM32 avviene infatti tramite l'interfaccia **UART**, utilizzando le linee TX e RX in modalità full-duplex. Il modulo viene alimentato a 5V, grazie al regolatore interno, mentre le linee di trasmissione dati operano a livello logico 3.3V, risultando pienamente compatibili con la UART della STM32 senza necessità di convertitori di livello. Il **baud rate** utilizzato nel progetto è 9600 bps, impostazione standard che permette una trasmissione stabile e affidabile. Inoltre, il modulo può essere configurato tramite comandi AT per funzionare sia come dispositivo *Master* che come *Slave*. Nel contesto del progetto, l'HC-05 permette all'utente di inviare comandi allo STM32 direttamente tramite smartphone, rendendo l'interazione semplice e immediata.



HC05 Back Side



ESP-32 CAM

La ESP32-CAM è un modulo compatto basato sul chip ESP32-S, dotato di processore dual-core a 240 MHz, 520 KB di SRAM, 4 MB di PSRAM e 4 MB di memoria Flash.

Integra connettività Wi-Fi e Bluetooth 4.2, una fotocamera OV2640 (risoluzione fino a 1600×1200) e uno slot per microSD.

Il modulo supporta formati immagine JPEG, YUV422 e RGB565 tramite interfaccia DVP.

Opera a 3.3 V, con un consumo medio in trasmissione compreso tra 160 e 260 mA, e offre interfacce come UART, SPI, I2C, PWM, ADC e DAC.

Poiché non dispone di una porta USB integrata, è necessario utilizzare un convertitore USB-Seriale per la programmazione.

La scheda è inoltre dotata di antenna Wi-Fi integrata e di connettore u.FL per un'antenna esterna opzionale.

Le sue dimensioni sono di circa 27 × 40,5 mm.



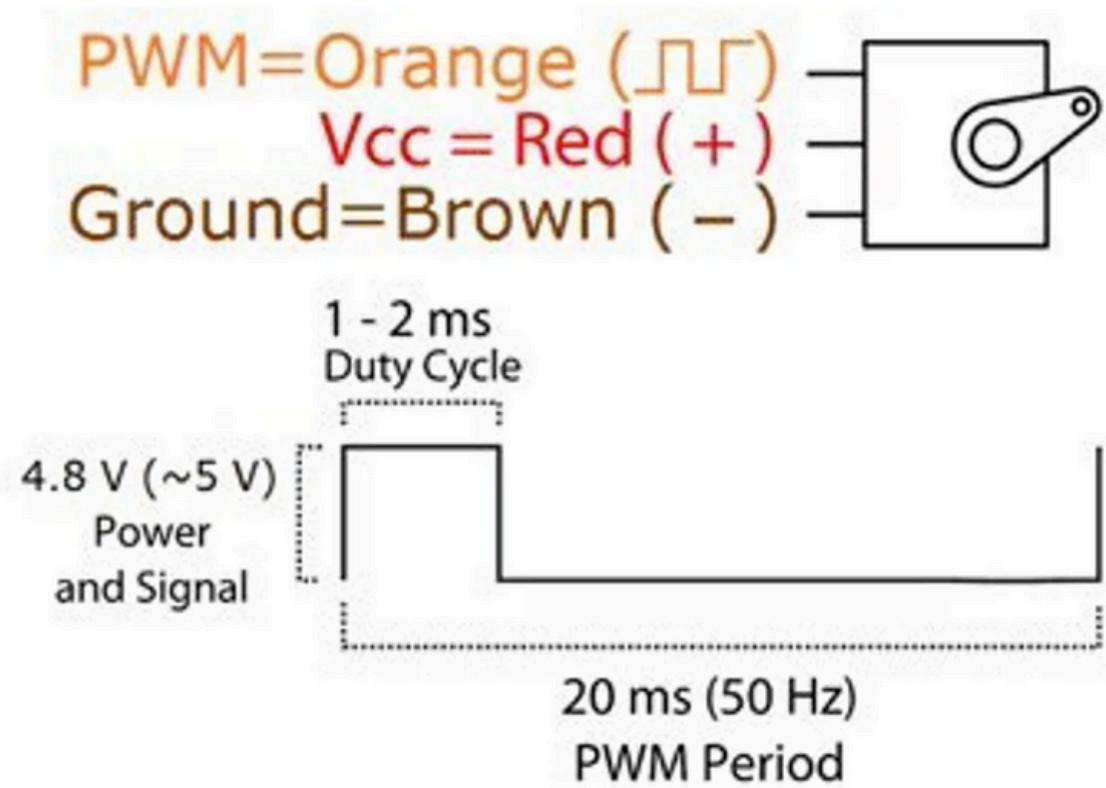
MICRO SERVOMOTORE SG90

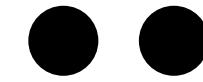
Il SG90 è un servomotore analogico compatto e preciso, controllato tramite segnale PWM (Pulse Width Modulation).

È ideale per applicazioni con Arduino, Raspberry Pi e microcontrollori embedded, in cui sono richiesti movimenti angolari controllati e reattivi.

Specifiche principali:

- Tensione di funzionamento: 4.8 – 6.0 V
- Angolo di rotazione: circa 180° (\approx 90° per lato)
- Tipo di controllo: PWM
- Larghezza impulso: 500 – 2400 μ s (1500 μ s \approx posizione centrale)
- Coppia: \sim 1.8 kg·cm a 4.8 V
- Velocità: \sim 0.1 s/60° a 4.8 V (senza carico)
- Assorbimento di corrente:
 - A riposo: < 10 mA
 - In movimento: \sim 100 – 250 mA (in base al carico)





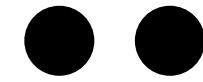
SEGNALE PWM CONTROLLO SERVO

PAGE 11

Il servo motore viene controllato tramite un segnale **PWM** a frequenza fissa di 50 Hz, cioè un impulso ripetuto ogni 20 millisecondi. La posizione del servo non dipende dalla frequenza, che rimane costante, ma dalla **durata dell'impulso** all'interno di ogni periodo. Un impulso di circa *1 millisecondo* porta il servo verso una posizione di *fine corsa*, uno di circa *1.5 millisecondi* lo posiziona al *centro*, mentre un impulso di circa *2 millisecondi* lo porta all'*altro estremo* della rotazione.

Nel programma il **Timer1** è configurato con un *prescaler* che divide il clock principale fino a ottenere un timer che conta a 1 MHz, quindi ogni tick corrisponde a 1 microsecondo. Il periodo del timer è impostato a 20000, che corrisponde a 20000 microsecondi, cioè ai 20 millisecondi necessari per ottenere i 50 Hz richiesti dal servo. Questo permette di controllare la durata dell'impulso semplicemente scrivendo nel registro di comparazione (**CCR1**) il valore corrispondente in microsecondi. Ad esempio, impostando 1000 si ottiene un impulso di 1 ms, 1500 produce un impulso di 1.5 ms, e così via.

La funzione **Servo_Move()** modifica direttamente questo valore e quindi varia la posizione del servo. Di conseguenza, il controllo del servo si riduce a scegliere il valore numerico che rappresenta la durata dell'impulso desiderata, mantenendo il movimento preciso e stabile.



IMPLEMENTAZIONE DEL

PAGE 12

MECCANISMO DELLE INTERRUZIONI

Il sistema è progettato per gestire eventi in modo efficiente tramite l'uso di **interruzioni**, evitando attese attive e migliorando le prestazioni del microcontrollore.

1. Ricezione **Bluetooth** tramite UART

- La comunicazione Bluetooth è gestita tramite la **ISR HAL_UART_RxCpltCallback()** per ricevere i comandi.
- Quando arriva un comando via Bluetooth (ad esempio "Access"), l'interruzione gestisce il comando e cambia lo stato del sistema.

2. Riconoscimento Facciale tramite UART (**ESP32-CAM**)

- La comunicazione con la ESP32-CAM avviene tramite UART e la stessa interruzione **HAL_UART_RxCpltCallback()** gestisce i caratteri ricevuti ('Y' o 'N') che indicano se il riconoscimento facciale è riuscito o meno.
- A seconda della risposta (ad esempio, "Access Granted" o "Face Not Recognized"), il sistema passa a uno stato successivo.

3. Controllo del **Servo**

- Quando un comando di accesso è valido (PIN o riconoscimento facciale), il servo viene mosso tramite l'uso di PWM, e il movimento viene gestito tramite un timer con un controllo temporizzato.

4. **Lockout** e Gestione del Tempo

- In caso di accesso negato, viene avviato un timer di lockout (10 secondi), gestito dalla funzione principale (while(1)), che impedisce ulteriori tentativi di accesso fino al termine del lockout.
- Il sistema esegue il controllo del tempo (**HAL_GetTick()**) e ripristina lo stato di accesso al termine del periodo di blocco.

5. **Riattivazione** del Ricevimento dei Dati

- Ogni volta che un byte viene ricevuto (sia via Bluetooth che dalla ESP32-CAM), viene riavviata la ricezione dell'interruzione per garantire una comunicazione continua e senza interruzioni.

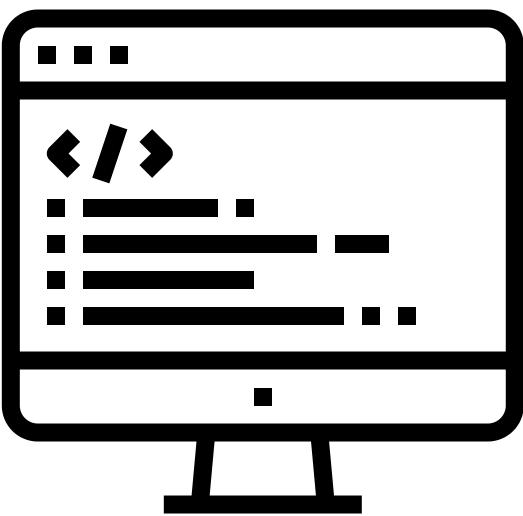


● ●

CONFIGURAZIONE

STM32F303

DICHIARAZIONI VARIABILI



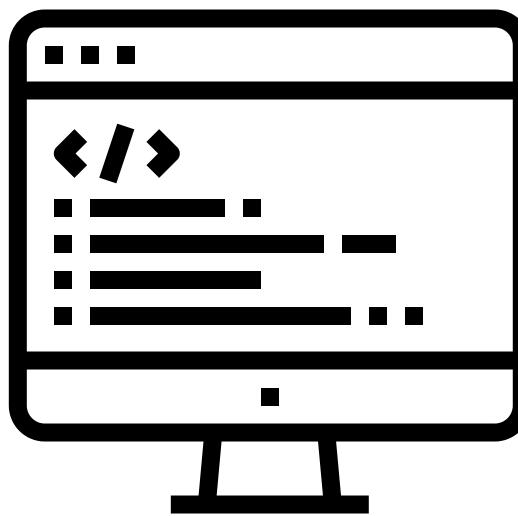
```
5  /* Defines */
6 #define SERVO_STOP    1400
7 #define SERVO_OPEN    1000
8 #define SERVO_CLOSE   1000
9 #define BUFFER_SIZE   64
10 #define MAX_FACE_ATTEMPTS 3
11 #define LOCKOUT_TIME 10000 // 10 secondi
12 #define SERVO_OPEN_TIME 200
13 #define LED_FAIL_TIME 2000
14
15 /* Typedef */
16 typedef enum {
17     WAIT_ACCESS_COMMAND,
18     WAIT_FACE_RESPONSE,
19     WAIT_PIN,
20     LOCKOUT
21 } AccessState;
22
23 /* Private variables */
24 TIM_HandleTypeDef htim1;
25 UART_HandleTypeDef huart2; // Bluetooth
26 UART_HandleTypeDef huart3; // ESP32-CAM
27
28 AccessState access_state = WAIT_ACCESS_COMMAND;
29
30 /* Bluetooth variables */
31 char bt_char;
32 char bt_buffer[BUFFER_SIZE];
33 uint8_t bt_index = 0;
34
35 /* CAM variables */
36 uint8_t cam_byte;
37
38 /* State management */
39 int face_attempts = 0;
40 int message_sent = 0;
41 uint32_t lockout_timer = 0;
42 uint32_t action_timer = 0;
43 uint8_t action_state = 0; // 0 idle, 1 success, 2 failure
```

● ●

CONFIGURAZIONE

STM32F303

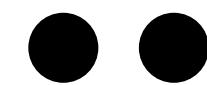
MAIN



```

210  /* Main */
211  int main(void)
212  {
213      HAL_Init();
214      SystemClock_Config();
215      MX_GPIO_Init();
216      MX_TIM1_Init();
217      MX_USART2_UART_Init();
218      MX_USART3_UART_Init();
219
220      HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
221      Servo_Move(SERVO_STOP);
222      LED_Blue();
223
224      HAL_UART_Receive_IT(&huart2, &bt_char, 1);
225      HAL_UART_Receive_IT(&huart3, &cam_byte, 1);
226
227      const char msg[] = "WRITE 'ACCESS' TO START FACIAL RECOGNIZE\r\n";
228      HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);
229
230      while(1)
231      {
232          // gestione servo/LED temporizzati
233          if(action_state == 1 && HAL_GetTick() - action_timer >= SERVO_OPEN_TIME)
234          {
235              Servo_Move(SERVO_CLOSE);    // apertura inversa per simulare chiusura
236              HAL_Delay(200);           // breve pausa per completare il movimento
237              Servo_Move(SERVO_STOP);   // ferma il servo
238              LED_Blue();
239              action_state = 0;
240          }
241
242          else if(action_state == 2 && HAL_GetTick() - action_timer >= LED_FAIL_TIME)
243          {
244              LED_Blue();
245              action_state = 0;
246          }
247
248          // gestione lockout
249          if(access_state == LOCKOUT)
250          {
251              if(HAL_GetTick() - lockout_timer >= LOCKOUT_TIME)
252              {
253                  access_state = WAIT_ACCESS_COMMAND;
254                  message_sent = 0;
255
256                  LED_Blue();
257
258                  const char msg2[] = "WRITE 'ACCESS' TO START FACIAL RECOGNIZE\r\n";
259                  HAL_UART_Transmit(&huart2, (uint8_t*)msg2, sizeof(msg2)-1, 100);
260
261              }
262          }
263      }
264  }

```



CONFIGURAZIONE STM32F303

NVIC-INTERRUPTS

PAGE 15

The screenshot shows the STM32CubeMX software interface for the project "PROGETTO_ESAME_APP.ioc". The main window displays the "Pinout & Configuration" tab, specifically the "NVIC Mode and Configuration" section. On the left, a sidebar lists various system components under "System Core", with "NVIC" selected. The main configuration area shows a table of interrupt types and their settings:

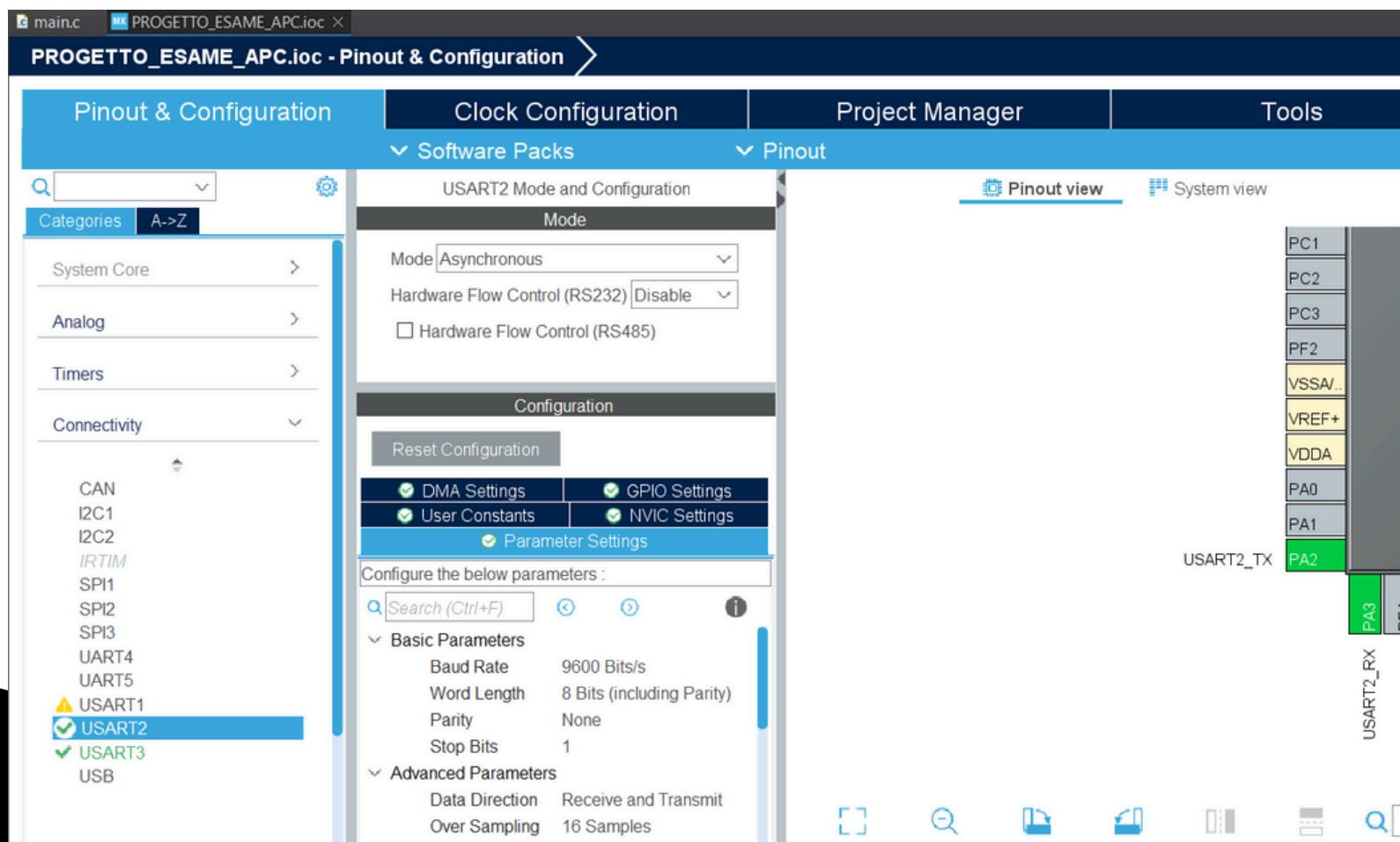
| Interrupt Type | Enabled | Preemption Priority | Sub Priority |
|---|-------------------------------------|---------------------|--------------|
| Pre-fetch fault, memory access fault | <input checked="" type="checkbox"/> | 0 | 0 |
| Undefined instruction or illegal state | <input checked="" type="checkbox"/> | 0 | 0 |
| System service call via SWI instruction | <input checked="" type="checkbox"/> | 0 | 0 |
| Debug monitor | <input checked="" type="checkbox"/> | 0 | 0 |
| Pendable request for system service | <input checked="" type="checkbox"/> | 0 | 0 |
| Time base: System tick timer | <input checked="" type="checkbox"/> | 15 | 0 |
| PVD interrupt through EXTI line16 | <input type="checkbox"/> | 0 | 0 |
| Flash global interrupt | <input type="checkbox"/> | 0 | 0 |
| RCC global interrupt | <input type="checkbox"/> | 0 | 0 |
| TIM1 break and TIM15 interrupts | <input type="checkbox"/> | 0 | 0 |
| TIM1 update and TIM16 interrupts | <input checked="" type="checkbox"/> | 0 | 0 |
| TIM1 trigger, commutation and TIM17 interrupts | <input type="checkbox"/> | 0 | 0 |
| TIM1 capture compare interrupt | <input type="checkbox"/> | 0 | 0 |
| USART2 global interrupt / USART2 wake-up interrupt through EXTI lin.. | <input checked="" type="checkbox"/> | 0 | 0 |
| USART3 global interrupt / USART3 wake-up interrupt through EXTI lin.. | <input checked="" type="checkbox"/> | 0 | 0 |
| Floating point unit interrupt | <input type="checkbox"/> | 0 | 0 |

At the bottom of the configuration table, there are dropdown menus for "Enabled", "Preemption Priority", and "Sub Priority".

CONFIGURAZIONE

STM32F303

UART BLUETOOTH



```

/* UART callback */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART2) // Bluetooth
    {
        if(access_state != LOCKOUT)
        {
            if(bt_char == '\r' || bt_char == '\n')
            {
                bt_buffer[bt_index] = 0;
                bt_index = 0;

                if(access_state == WAIT_ACCESS_COMMAND)
                {
                    if(strcasecmp(bt_buffer, "Access") == 0)
                    {
                        LED_Blue();
                        uint8_t shoot_cmd[] = "2\r\n";
                        HAL_UART_Transmit(&huart3, shoot_cmd, sizeof(shoot_cmd)-1, 50);
                        access_state = WAIT_FACE_RESPONSE;
                    }
                    const char msg[] = "TRYING FACE RECOGNITION...\r\n";
                    HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);
                    message_sent = 0;
                }
                else if(message_sent == 0)
                {
                    const char msg[] = "WRITE 'ACCESS' TO START FACIAL RECOGNIZE\r\n";
                    HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);
                    message_sent = 1;
                }
                else if(access_state == WAIT_PIN)
                {
                    if(strcmp(bt_buffer, "1234") == 0)
                    {
                        LED_Green();
                        Servo_Move(SERVO_OPEN);
                        action_timer = HAL_GetTick();
                        action_state = 1;

                        const char msg[] = "ACCESS GRANTED\r\n";
                        HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);

                        face_attempts = 0;
                        access_state = WAIT_ACCESS_COMMAND;
                        message_sent = 0;
                    }
                    else
                    {
                        LED_Red();
                        const char msg[] = "ACCESS DENIED. WAIT 10 SECONDS... \r\n";
                        HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);

                        face_attempts = 0;
                        access_state = LOCKOUT;
                        lockout_timer = HAL_GetTick();
                    }
                }
                else
                {
                    bt_buffer[bt_index++] = bt_char;
                    if(bt_index >= sizeof(bt_buffer)) bt_index = 0;
                }
            }
            HAL_UART_Receive_IT(&huart2, &bt_char, 1); // riattiva sempre
        }
    }
}

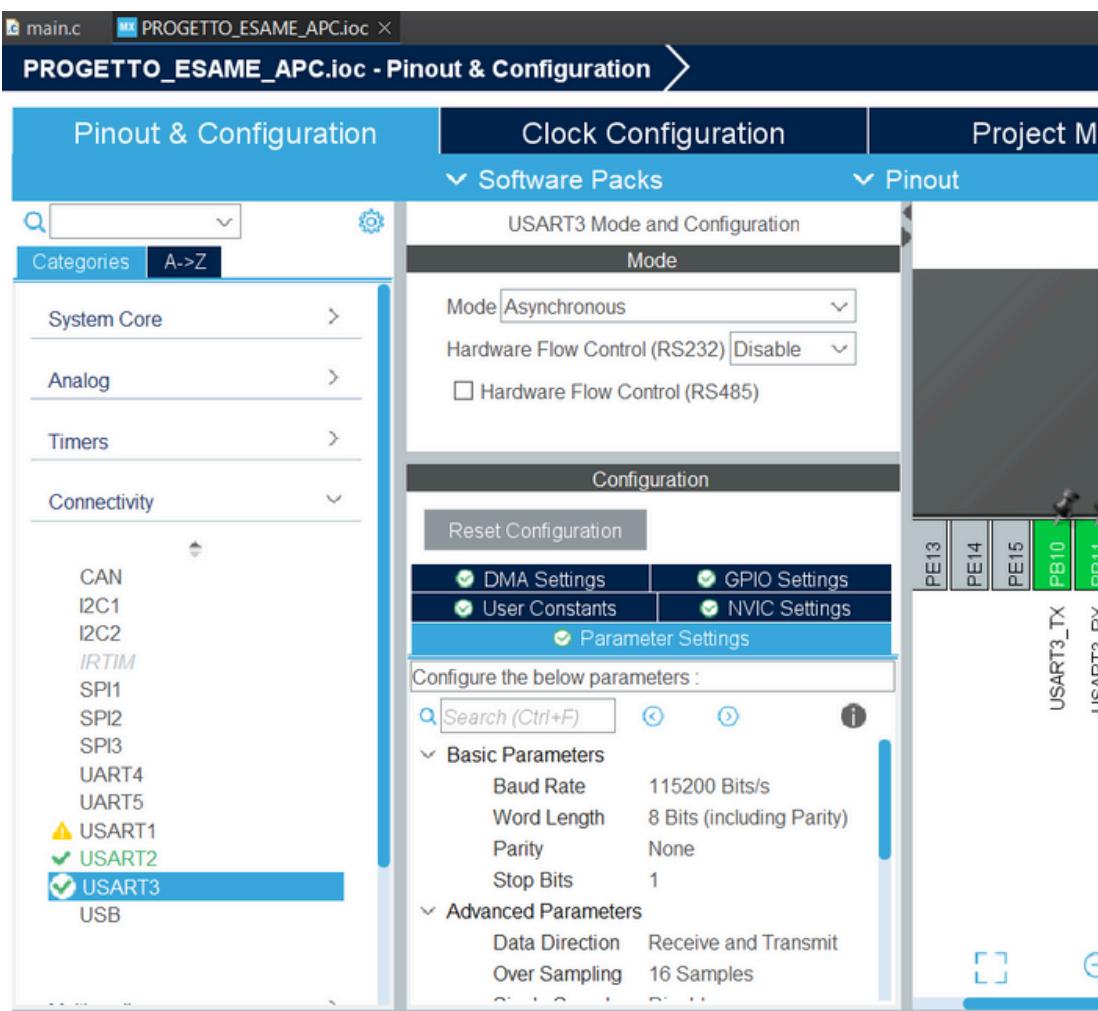
```

● ●

CONFIGURAZIONE

STM32F303

UART CAM



```

else if(huart->Instance == USART3) // ESP32-CAM
{
    if(access_state == WAIT_FACE_RESPONSE)
    {
        if(cam_byte == 'Y')
        {
            LED_Green();
            Servo_Move(SERVO_OPEN);
            action_timer = HAL_GetTick();
            action_state = 1;

            face_attempts = 0;
            access_state = WAIT_ACCESS_COMMAND;

            const char msg[] = "ACCESS GRANTED\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);
            message_sent = 0;
        }
        else if(cam_byte == 'N')
        {
            face_attempts++;

            if(face_attempts < MAX_FACE_ATTEMPTS)
            {
                const char msg[] = "FACE NOT RECOGNIZED. TRY AGAIN\r\n";
                HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);

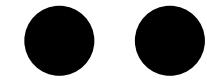
                LED_Red();
                action_timer = HAL_GetTick();
                action_state = 2;

                access_state = WAIT_ACCESS_COMMAND;
                message_sent = 0;
            }
            else
            {
                access_state = WAIT_PIN;
                face_attempts = 0;

                const char msg[] = "MAX ATTEMPTS REACHED. INSERT PIN\r\n";
                HAL_UART_Transmit(&huart2, (uint8_t*)msg, sizeof(msg)-1, 100);

                LED_Red();
                message_sent = 0;
            }
        }
    }
    HAL_UART_Receive_IT(&huart3, &cam_byte, 1); // riattiva sempre
}
}

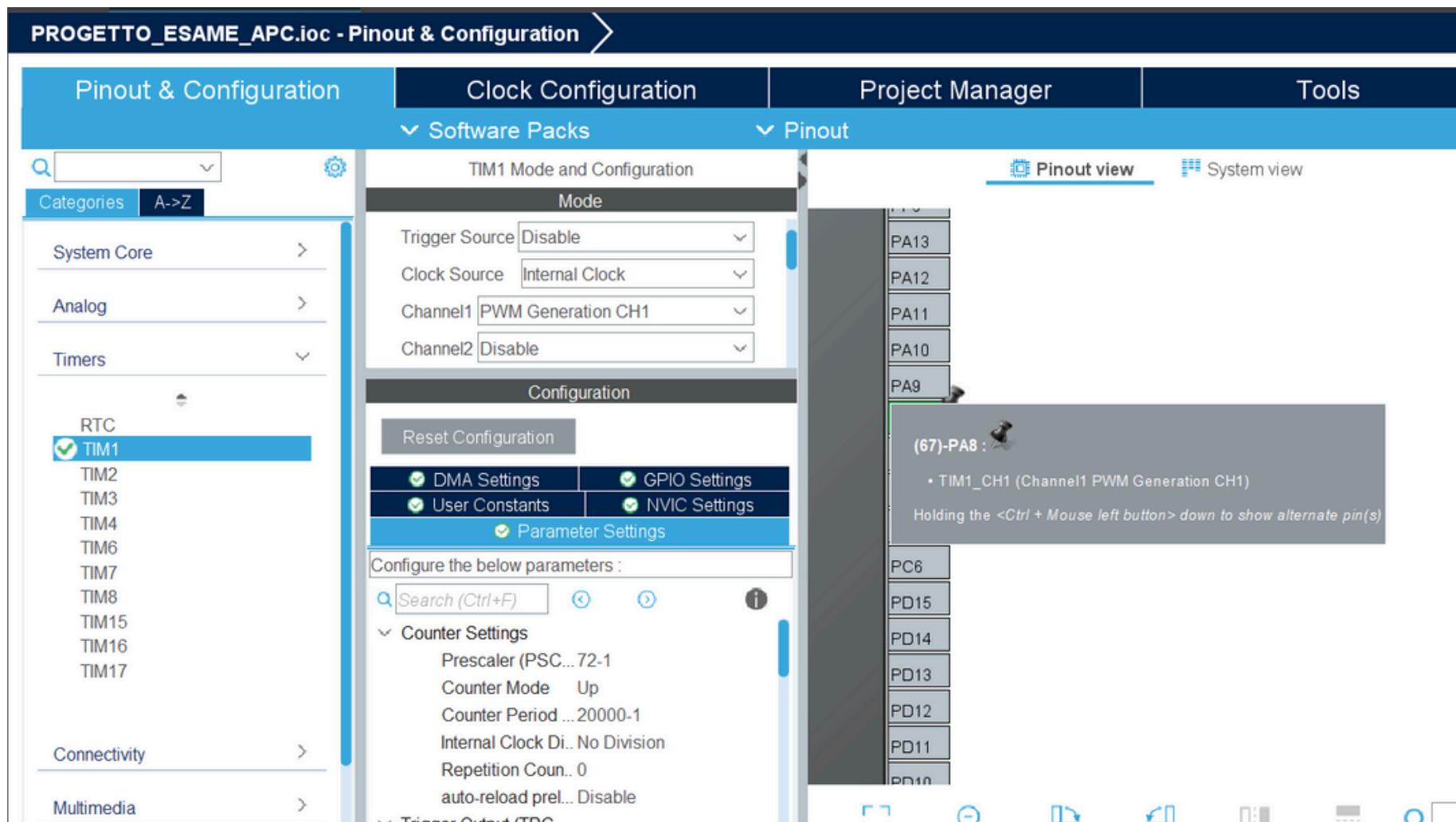
```



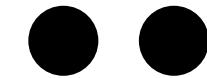
CONFIGURAZIONE STM32F303

SERVOMOTORE

PAGE 18



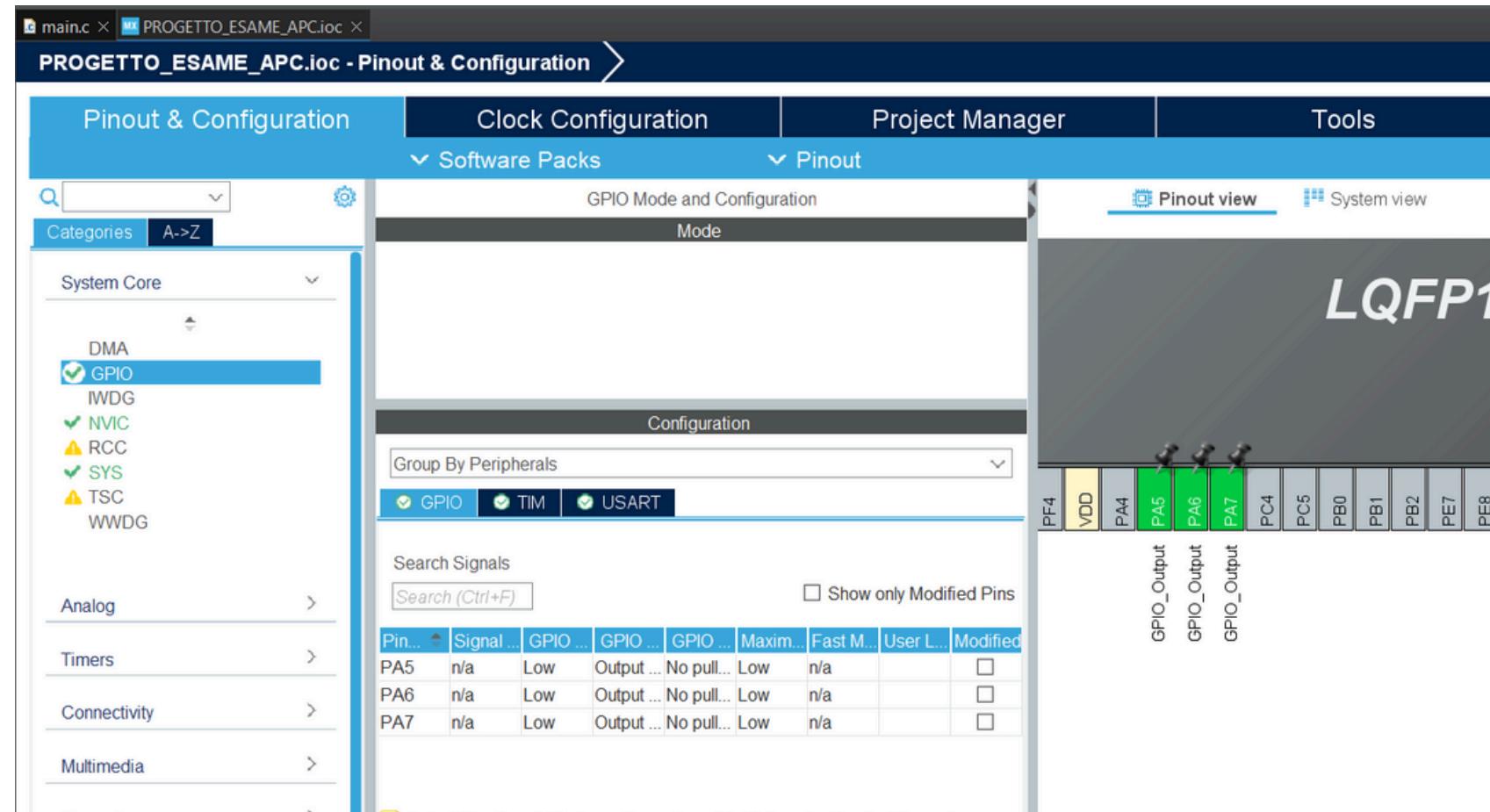
```
void Servo_Move(uint16_t pulse_val) {
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pulse_val);
}
```



CONFIGURAZIONE STM32F303

PAGE 19

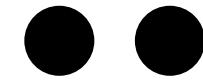
LED RGB



```
void LED_Red(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
}

void LED_Green(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
}

void LED_Blue(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
}
```



CONFIGURAZIONE CAMERA ESP-32

PAGE 20

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

// Dati rete WiFi
const char* ssid = "andrea";
const char* password = "12345678";

// URL del server a cui fare POST
const char* serverUrl = "http://172.20.10.2:5000/upload";

#define LED_PIN 4
// Setup camera per modulo AI Thinker (modifica se usi altro modello)
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27

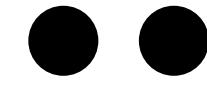
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

void startCamera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sscb_sda = SIOD_GPIO_NUM;
    config.pin_sscb_scl = SIOC_GPIO_NUM;
    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;
    config.xclk_freq_hz = 20000000;
    config.pixel_format = PIXFORMAT_JPEG;

    if(psramFound()){
        config.frame_size = FRAMESIZE_SVGA; // 800x600, qualità migliore
        //config.frame_size = FRAMESIZE_VGA;
        //config.jpeg_quality = 10;
        config.jpeg_quality = 8;
        config.fb_count = 2;
    } else {
        config.frame_size = FRAMESIZE_CIF;
        config.jpeg_quality = 12;
        config.fb_count = 1;
    }

    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("Camera init failed with error 0x%x\n", err);
    }
}

void setup() {
    Serial.begin(115200);
    delay(1000);
    pinMode(LED_PIN, OUTPUT); // <<== Imposta il pin del LED come uscita
    WiFi.begin(ssid, password);
    Serial.print("Connessione WiFi...");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }
}
```



CONFIGURAZIONE CAMERA ESP-32

PAGE 21

```
    Serial.print(".");
}

Serial.println(" connesso!");

startCamera();
xTaskCreatePinnedToCore(
    uartTask,           // funzione
    "UART Task",        // nome task
    4096,               // stack size
    NULL,               // parametri
    1,                  // priorità
    NULL,               // handle
    1                  // core (1 = core App, 0 = core Pro)
);

}

void uartTask(void * parameter) {
    Serial.setTimeout(100); // importante: timeout breve per non bloccare
    while (true) {
        if (Serial.available()) {
            String command = Serial.readStringUntil('\n');
            command.trim();

            if (command == "2") {
                //DEBUG]Serial.println("Scatto foto...");
                digitalWrite(LED_PIN, HIGH);

                camera_fb_t * fb = esp_camera_fb_get();
                if (!fb || fb->format != PIXFORMAT_JPEG) {
                    //DEBUG]Serial.println("Errore acquisizione!");
                    if (fb) esp_camera_fb_return(fb);
                    digitalWrite(LED_PIN, LOW);
                    continue;
                }

                HTTPClient http;
                http.setTimeout(10000);
                http.begin(serverUrl);
            }
        }
    }
}

int httpResponseCode = http.POST(fb->buf, fb->len);
if (httpResponseCode > 0) {
    String response = http.getString();
    //DEBUG]Serial.println("Server risponde: " + response);
    if (response.indexOf("not ok") == -1){
        Serial.write('Y');
    }
    else{
        Serial.write('N');
    }
} else {
    //DEBUG]Serial.printf("Errore invio POST: %d\n", httpResponseCode);
    Serial.write('N');
}

http.end();
esp_camera_fb_return(fb);
digitalWrite(LED_PIN, LOW);
}

vTaskDelay(10 / portTICK_PERIOD_MS); // evita uso eccessivo della CPU
}

void loop() {
    //attesa attiva
    delay(100);
}
```

Quando viene ricevuto l'apposito comando tramite UART, si accende il led della board e viene scattata una foto.
Questa sarà poi inviata tramite il metodo http POST ad un server flask remoto, da cui si attende una risposta, da inviare tramite uart.



CODICE UPLOAD SERVER FLASK

Ogni volta che viene ricevuta un'immagine tramite post, la route salva l'immagine in un apposito folder e richiama la funzione recognize face, che ritorna un valore booleano.

```
@app.route('/upload', methods=['POST'])
def upload_image():
    """
    Endpoint per ricevere immagini dal sistema di riconoscimento facciale.
    Salva l'immagine, esegue il riconoscimento e registra l'accesso.
    Outputs JSON con status del riconoscimento.
    """

    try:
        image_bytes = request.data
        if not image_bytes:
            return jsonify({'error': 'No data received'}), 400

        # Salva immagine ricevuta con timestamp
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        image_path = os.path.join(UPLOAD_FOLDER, f"image_{timestamp}.jpg")
        image = Image.open(io.BytesIO(image_bytes))
        image.save(image_path)

        # Riconoscimento facciale
        recognized, name_or_msg = recognize_face(image_bytes)
        result = {
            'timestamp': timestamp,
            'filename': f"image_{timestamp}.jpg",
            'recognized': recognized,
            'name': name_or_msg if recognized else "Unknown"
        }
        access_log.append(result)

        if recognized:
            return jsonify({'status': 'ok', 'name': name_or_msg}), 200
        else:
            return jsonify({'status': 'not ok', 'reason': name_or_msg}), 200

    except Exception as e:
        return jsonify({'error': 'File non valido o danneggiato', 'detail': str(e)}), 400
```

FACE RECOGNITION

All'avvio, il sistema carica i volti **registrati** che corrispondono alle identità degli utenti autorizzati, prelevandoli da una cartella specifica gestita dall'amministratore del server. La funzione **recognize_face**, fondamentale per il corretto funzionamento dell'applicazione, utilizza le funzioni della libreria Python **face_recognition**. In particolare, viene impiegata una funzione di confronto che confronta il volto acquisito con tutte le immagini archiviate nella cartella (database).

```
def load_known_faces():
    """
    Ogni immagine viene codificata e il nome viene estratto dal filename.
    """

    for filename in os.listdir(KNOWN_FOLDER):
        if filename.lower().endswith('.jpg', '.jpeg', '.png'):
            path = os.path.join(KNOWN_FOLDER, filename)
            image = face_recognition.load_image_file(path)
            encodings = face_recognition.face_encodings(image)
            if encodings:
                known_face_encodings.append(encodings[0])
                known_face_names.append(os.path.splitext(filename)[0])
    print(f"[INFO] {len(known_face_encodings)} known faces loaded.")

load_known_faces()

def recognize_face(image_bytes):
    """
    Riconosce un volto comparandolo con quelli noti.
    """

    # Carica immagine ricevuta
    unknown_image = face_recognition.load_image_file(io.BytesIO(image_bytes))
    unknown_encodings = face_recognition.face_encodings(unknown_image)

    if not unknown_encodings:
        return False, "No face found"

    unknown_encoding = unknown_encodings[0]

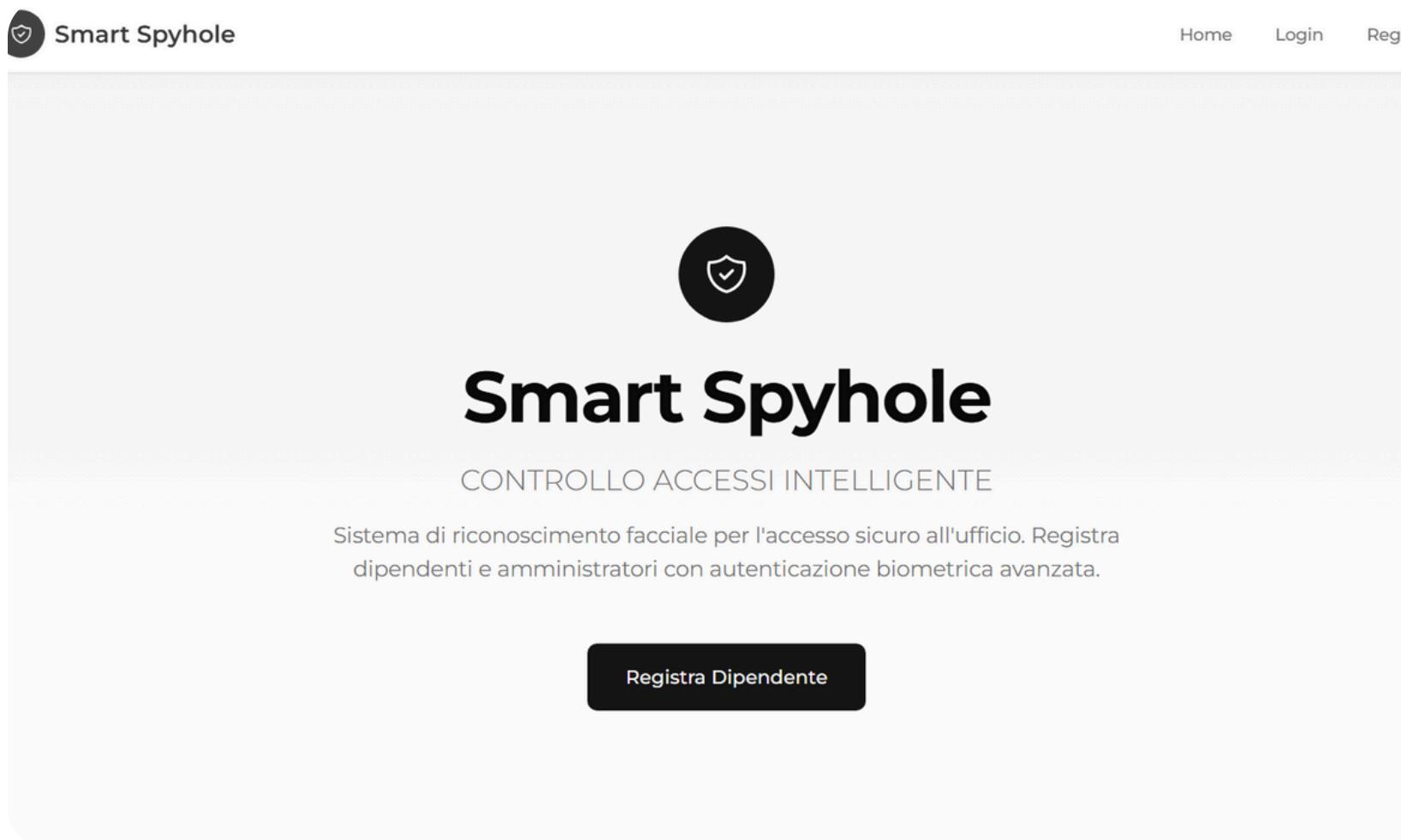
    # Calcola la distanza tra volti (soglia: 0.4 per maggiore precisione)
    face_distances = face_recognition.face_distance(known_face_encodings, unknown_encoding)
    best_match_index = face_distances.argmin()
    best_distance = face_distances[best_match_index]

    # Soglia più severa per sicurezza
    if best_distance < 0.4:
        return True, known_face_names[best_match_index]
    else:
        return False, "Unknown"
```

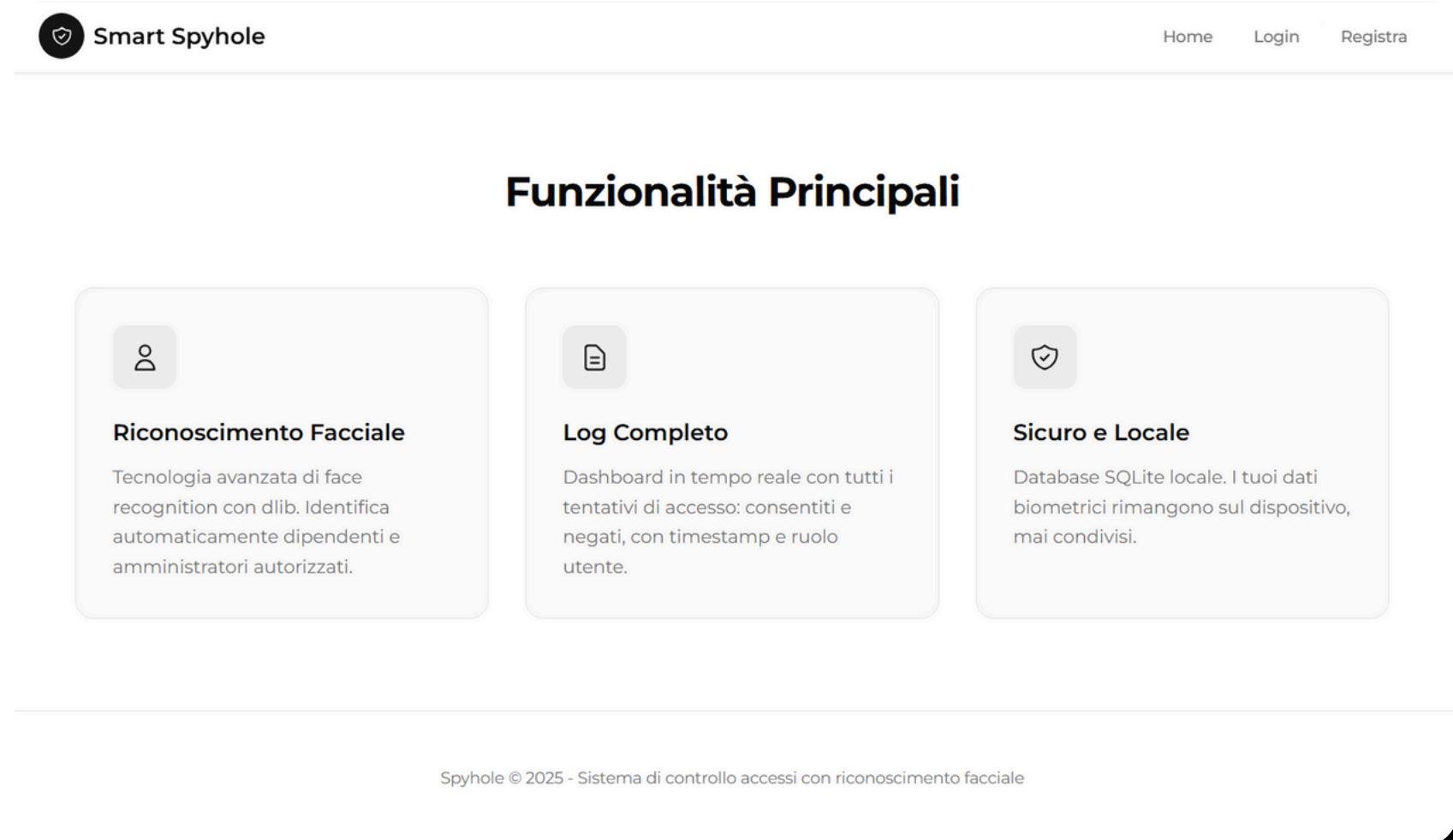


DASHBOARD E WEB APP-HOME

L'applicazione si presenta con una schermata iniziale in cui vengono mostrate le funzionalità principali messe a disposizione degli utenti.



The screenshot shows the homepage of the Smart Spyhole application. At the top left is the logo "Smart Spyhole" with a shield icon. At the top right are three navigation links: "Home", "Login", and "Registra". The main content area features a large central heading "Smart Spyhole" with a subtitle "CONTROLLO ACCESSI INTELLIGENTE". Below this, a descriptive text reads: "Sistema di riconoscimento facciale per l'accesso sicuro all'ufficio. Registra dipendenti e amministratori con autenticazione biometrica avanzata." At the bottom is a black button labeled "Registra Dipendente".



The screenshot shows the "Funzionalità Principali" (Main Features) page of the Smart Spyhole application. At the top left is the logo "Smart Spyhole". At the top right are three navigation links: "Home", "Login", and "Registra". The main content area is titled "Funzionalità Principali" and contains three cards:

- Riconoscimento Facciale**: Tecnologia avanzata di face recognition con dlib. Identifica automaticamente dipendenti e amministratori autorizzati.
- Log Completo**: Dashboard in tempo reale con tutti i tentativi di accesso: consentiti e negati, con timestamp e ruolo utente.
- Sicuro e Locale**: Database SQLite locale. I tuoi dati biometrici rimangono sul dispositivo, mai condivisi.

At the bottom of the page is a copyright notice: "Spyhole © 2025 - Sistema di controllo accessi con riconoscimento facciale".



WEB APP REGISTRAZIONE

La procedura di registrazione consente agli utenti di registrarsi al sistema, aggiungendo oltre ad username, password e ruolo anche una propria foto che sarà sfruttata per il riconoscimento facciale.

Smart Spyhole

Home Login Registra



Registrazione Utente Autorizzato

Carica una foto del volto (un solovolto nella foto). Puoi usare la webcam o caricare un file.

Nome (username)

Password

Ruolo

User (Utente)

Formati accettati: JPG, JPEG, PNG (max 5MB)

Scegli Foto (JPG, PNG)

Smart Spyhole

Home Login Registra

Formati accettati: JPG, JPEG, PNG (max 5MB)

Scegli Foto (JPG, PNG)



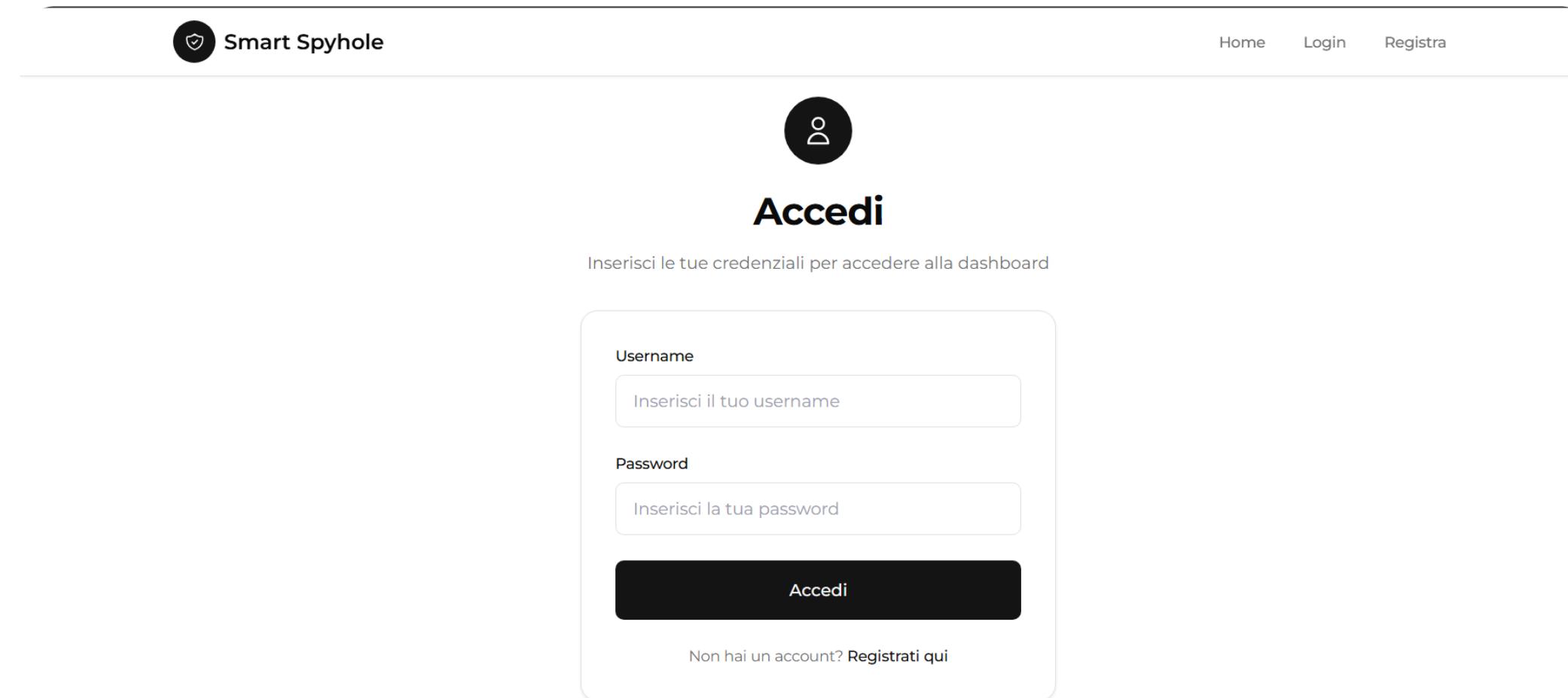
oppure trascina qui una foto

Consigli: fonte ben illuminato, niente occhiali scuri, uno solo volto nella foto.



WEB APP LOGIN

Attraverso il login ciascun utente può consultare il registro degli accessi, in modo da rilevare eventuali tentativi di accesso da parte di sconosciuti.



The screenshot shows the login page of a web application named "Smart Spyhole". At the top, there is a navigation bar with the logo "Smart Spyhole" and links for "Home", "Login", and "Registra". Below the navigation bar, there is a large circular icon containing a user profile symbol. The main heading "Accedi" is centered above a form field placeholder text "Inserisci le tue credenziali per accedere alla dashboard". The form itself is contained within a light gray rounded rectangle and includes two input fields labeled "Username" and "Password", both with placeholder text "Inserisci il tuo username" and "Inserisci la tua password" respectively. Below these fields is a large black rectangular button with the white text "Accedi". At the bottom of the form, there is a small link "Non hai un account? Registrati qui".



REGISTRO ACCESSI

Vengono mostrati in tempo reale gli accessi al sistema.

Smart Spyhole

Home Dashboard Callejon Logout

Registro Accessi

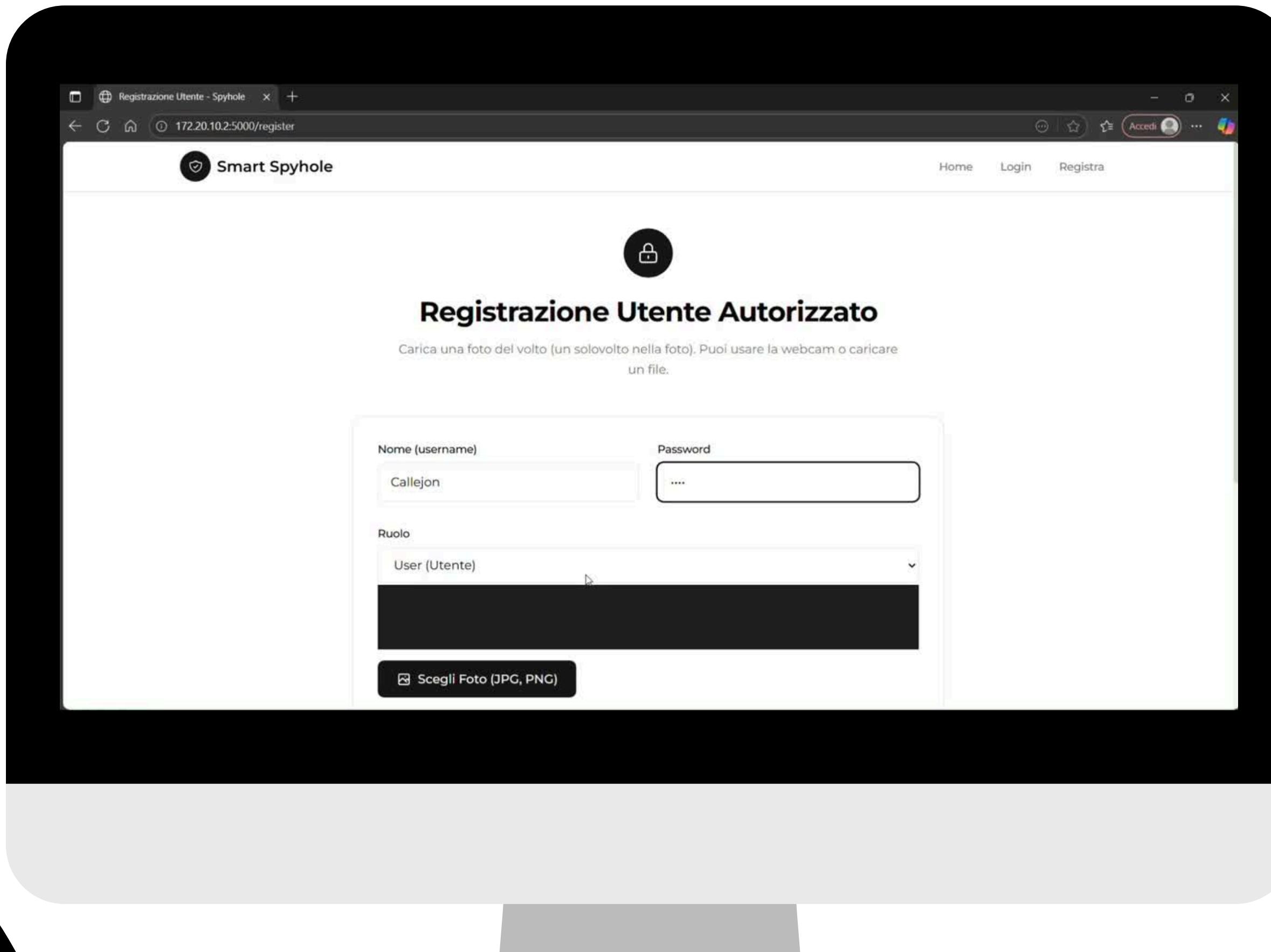
Monitoraggio in tempo reale degli accessi riconosciuti

● Aggiornamento automatico

| Accesso | Riconoscimento |
|---|--------------------|
|  Sconosciuto ⌚ 08/11/2025 10:23:59 ✖ Accesso negato | X Non riconosciuto |
|  Callejon ⌚ 08/11/2025 10:22:45 ✓ Accesso autorizzato | ✓ Riconosciuto |

DEMO DEL PRODOTTO

PAGE 28



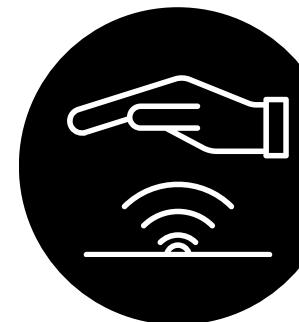


CONCLUSIONI E SVILUPPI FUTURI

Il sistema Spyhole integra riconoscimento facciale, autenticazione Bluetooth e controllo meccanico tramite servomotore, offrendo una soluzione completa per il controllo accessi. L'architettura distribuita, composta da **STM32**, **ESP32-CAM**, **HC-05** e **server Flask**, garantisce comunicazioni efficienti e affidabili tra tutti i moduli, permettendo alla STM32 di coordinare comandi Bluetooth e dati per il riconoscimento facciale in modo stabile. L'integrazione della dashboard web consente il monitoraggio in tempo reale e la gestione centralizzata degli utenti. Di seguito sono riportati alcuni ***sviluppi futuri***:



Implementazione di un database remoto per la gestione scalabile delle identità e dei log di accesso.



Introduzione di sensori aggiuntivi (es. RFID, impronte digitali) per l'autenticazione multifattoriale.



Aggiunta di cifratura dei dati e protocolli di sicurezza per le comunicazioni Bluetooth e Wi-Fi.



Notifiche e alert: invio automatico di alert via email o app in caso di tentativi di accesso non autorizzati.



**GRAZIE PER
L'ATTENZIONE**