**CameraWebSever.ino**

**1. Inclusione delle librerie**

cpp

CopiaModifica

#include "esp\_camera.h"

#include <WiFi.h>

* esp\_camera.h contiene tutte le funzioni per configurare e usare la fotocamera dell’ESP32.
* WiFi.h serve per connettere l’ESP32 a una rete WiFi.

**2. Configurazione della fotocamera e scelta modello**

* Ci sono varie definizioni di modelli di camera commentate.
* Viene attivata la linea:

cpp

CopiaModifica

#define CAMERA\_MODEL\_AI\_THINKER

* Questo dice al programma quale configurazione di pin e caratteristiche hardware deve usare (l’AI Thinker ESP32-CAM è molto diffuso).

**3. Configurazione WiFi**

cpp

CopiaModifica

const char \*ssid = "UNIMORE";

const char \*password = "292456";

* Qui si impostano nome e password della rete WiFi a cui connettersi.

**4. Funzioni dichiarate**

cpp

CopiaModifica

void startCameraServer();

void setupLedFlash(int pin);

* startCameraServer() è la funzione (definita altrove) che avvia il server web per gestire lo streaming e i comandi via HTTP.
* setupLedFlash() serve per configurare il LED flash della camera se presente.

**5. Setup principale (setup())**

* **Inizializzazione seriale** (utile per debug via monitor seriale):

cpp

CopiaModifica

Serial.begin(115200);

Serial.setDebugOutput(true);

* **Configurazione pin e parametri della fotocamera**

cpp

CopiaModifica

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

...

config.frame\_size = FRAMESIZE\_UXGA;

config.pixel\_format = PIXFORMAT\_JPEG;

...

config.fb\_location = CAMERA\_FB\_IN\_PSRAM;

* Qui si dicono quali pin GPIO usa la fotocamera per i dati, clock, sincronismi.
* Si imposta la risoluzione (UXGA = 1600x1200), formato JPEG.
* Se è presente la PSRAM (RAM esterna, importante per immagini ad alta risoluzione) si aumentano qualità e buffer.
* **Inizializzazione della camera**

cpp

CopiaModifica

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK) {

Serial.printf("Camera init failed with error 0x%x", err);

return;

}

* Inizializza il driver e hardware fotocamera con i parametri impostati. Se fallisce, stampa errore e ferma.
* **Configurazione del sensore (es. rotazione immagine)**

cpp

CopiaModifica

sensor\_t \*s = esp\_camera\_sensor\_get();

if (s->id.PID == OV3660\_PID) {

s->set\_vflip(s, 1);

s->set\_brightness(s, 1);

s->set\_saturation(s, -2);

}

...

if (config.pixel\_format == PIXFORMAT\_JPEG) {

s->set\_framesize(s, FRAMESIZE\_QVGA);

}

* Alcuni modelli di sensore hanno immagini girate o colori sbagliati, qui si correggono.
* **Setup LED Flash (se definito)**

cpp

CopiaModifica

#if defined(LED\_GPIO\_NUM)

setupLedFlash(LED\_GPIO\_NUM);

#endif

* Se la board ha un LED flash, viene configurato.
* **Connessione WiFi**

cpp

CopiaModifica

WiFi.begin(ssid, password);

WiFi.setSleep(false);

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

delay(500);

Serial.print(".");

}

Serial.println("WiFi connected");

* Si connette alla rete WiFi specificata, con debug seriale.
* **Avvio server camera**

cpp

CopiaModifica

startCameraServer();

* Qui parte il server HTTP che gestirà streaming e controllo della camera.
* **Stampa IP per collegarsi**

cpp

CopiaModifica

Serial.print("Camera Ready! Use 'http://");

Serial.print(WiFi.localIP());

Serial.println("' to connect");

**6. Loop principale (loop())**

cpp

CopiaModifica

void loop() {

// Do nothing. Everything is done in another task by the web server

delay(10000);

}

* Non fa nulla di attivo. Tutto il lavoro viene gestito da task separati (ad esempio il server web creato da startCameraServer).
* Il delay serve a non sovraccaricare il processore inutilmente.

**Riassunto molto semplice**

* Configura e inizializza la fotocamera con parametri corretti.
* Si connette alla WiFi.
* Avvia un server web che permette di vedere lo stream video e inviare comandi.
* Il programma principale poi “dorme” e lascia tutto a processi interni.