Alessandro Picone
Giuseppe Sanfilippo
Alessandro Macaluso

Internet Of Things

(prof. Garlisi Domenico)

A.A.2021/2022

[Progetto E]: '*Si progetti un timer digitale tramite display a 7 segmenti. Predisporre l'applicazione per l'accesso alla rete wifi utilizzando l'interfaccia wireless integrata, il sistema deve integrare un webserver minimale per il settaggio del valore di timer (tempo di decremento e valore da 1 a 9), un pulsante per riavviare il timer. Allo scadere del timer bisogna avvisare l'utente con un segnale acustico.'*

Hardware: board: Esp32, Sensore: Pulsante, Attuatore: display 7 segmenti, buzzer.

Stesura

Cos'è ESP32? Per lo svolgimento della tesina abbiamo utilizzato una board ESP32, ma cos'è questa board? Prima di parlare di ESP32, bisogna introdurre il concetto di microcontrollore. Un microcontrollore è un dispositivo elettronico integrato, ottimizzato per controllare i dispositivi elettronici esterni mediante un unico chip. La tecnologia SOC (System On Chip) presente all'interno dei microcontrollori, consente di avere un sistema completo in un unico chip, differente quindi dal microprocessore, che utilizza più memorie in base alla funzione che ne necessita. Il microcontrollore possiede una memoria, un processore e pin digitali e analogici di I/O, la comunicazione tra le memorie avviene mediante BUS INTERNI, per questo i moderni microcontrollori rispettano l'architettura Harvard. Il maggior impiego dei microcontrollori, è proprio nei sistemi Embedded, oltre a trarne vantaggi in termini di architettura, si dimostrano essere più "Economici" rispetto ai microprocessori. Le board più utilizzate sono:

- ➔ Arduino
- ➔ Esp32
- ➔ Esp8266

Esp32 è, difatti, una serie di microcontrollori SOC (System On Chip), a basso costo e a bassa potenza con WiFi integrato e Bluetooth dual-mode. Sviluppati da EspressIf Systems, un'azienda cinese con sede a Shanghai, sono stati prodotti utilizzando un processo a 40nm. Possiamo dunque analizzare l'hardware presente in una board, utilizzando come valori di riferimento: Set di segnali analogico/digitali (I/O), frequenza cpu, memoria flash, memoria sram ed eventuali.

Vengono di seguito riportate le caratteristiche più importanti della board utilizzata:

Hardware

Venditore	Nome	Frequenza di lavoro (Mhz)	Memoria Flash (MiB)	pSRAM (MiB)	Alimentazione
EspressIf	ESP32-WROOM-32D	40	4	0	3.0 V ~ 3.6 V

WiFi

Protocolli	Range di Frequenza
802.11 b/g/n (802.11n fino a 150 Mbps)	2.4 GHz ~ 2.5 GHz

Bluetooth

Protocolli	Radio
Bluetooth v4.2 BR/EDR e specifiche BLE (Bluetooth Low Energy)	NZIF ricevitore con –97 dBm sensibilità. Trasmettitore di Classe-1, classe-2 classe-3. AFH (Adaptive Frequency Hopping) [Riduzione dei disturbi].

Eventuale documentazione reperibile sul sito.

La board, come sopra citato, possiede un set di porte digitali di I/O, quindi GPIO, un set di porte analogiche di Input ed infine porte seriali. Presenta inoltre i classici PIN di alimentazione 5V e 3.3V, più pin di tipo GND. Mentre l'alimentazione della Board prevista tramite cavo USB è di 5V.

Come si può utilizzare ESP32? (cenni) Come già detto, questi vengono utilizzati per alimentare e gestire SENSORI e ATTUATORI. *Come possiamo creare i nostri circuiti?* Utilizziamo una breadboard per costruire i nostri circuiti elettrici, usiamo invece dei cavi per effettuarne i collegamenti. Ogni sensore o attuatore è collegato per mezzo della breadboard ai suoi corrispettivi pin di alimentazione, GND, GPIO, etc. Una volta sviluppato il circuito abbiamo bisogno di gestirli seguendo una logica funzionale (programma) che stabiliamo noi, attraverso l'IDE (ambiente di sviluppo integrato) Arduino.

L'IDE Arduino è un'applicazione multipiattaforma in Java, che permette la stesura di codice sorgente mediante un editor di testo integrato perfettamente completo. L'editor è in grado di compilare e caricare sulla scheda collegata, il programma eseguibile (Sketch). L'IDE integra una libreria software C/C++, chiamata "Wiring", che permette una gestione ottimale delle comuni operazioni di Input e Output. Gli sketch sono scritti in un linguaggio derivato dal C/C++. Uno sketch ha bisogno di due funzioni principali, rispettivamente:

```
1  void setup{
2
3  }
4  void loop {
5
6  }
```

Prima di scrivere il nostro codice sorgente è doveroso precisare che bisogna effettuare opportune ***configurazioni per Esp32***:

1. Bisogna installare opportunamente i driver per stabilire una connessione seriale con la board.
2. Bisogna aggiungere opportunamente il file JSON (link), che estende i dispositivi compatibili con l'IDE della sezione "Gestore schede".
3. Possiamo quindi adesso installare mediante l'apposito pannello precedentemente citato, il pacchetto "esp32", questo permetterà dunque all'ide di compilare il codice sorgente e riconoscere la board di riferimento.

Sensori e Attuatori

Il progetto richiede l'utilizzo di alcuni sensori e attuatori, analizziamo quindi in dettaglio cosa sono e quali ci necessitano:

Sensori: I sensori sono dispositivi che si interfacciano con il sistema che si sta utilizzando, catturano una quantità fisica e la convertono in una quantità elettrica. Vengono usati molti effetti della fisica per costruire i sensori, quali, per esempio le proprietà meccaniche delle resistenze, la generazione di tensione in un campo magnetico, effetti foto-elettrici, etc. Di seguito sono elencati alcuni tipi di sensori: Switch, Accelerometro, Giroscopio, Magnetometro, Temperatura, Ottico, Umidità, Temperatura, Prossimità e tanti altri.

[Utilizzati nel progetto]: Pulsante.

Attuatori: Gli attuatori trasformano un segnale digitale in un effetto fisico, cioè quindi che 'attua' , mette in atto, tramite un input già dato. Di seguito sono elencati alcuni tipi di attuatori: Motori, Indicatori (LED , LCD), Altoparlanti, Relays.

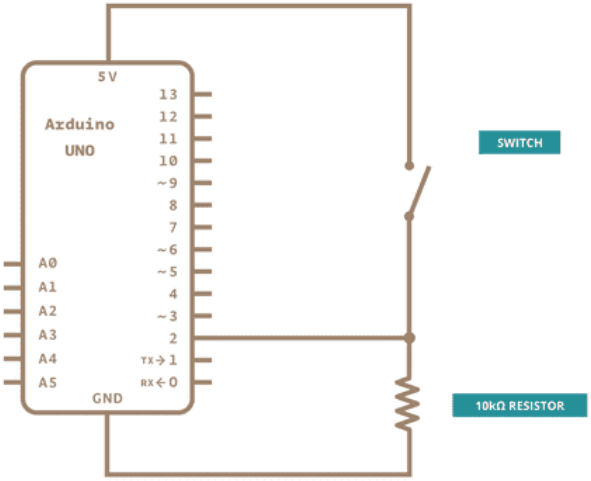
[Utilizzati nel progetto]: Display a 7 segmenti e Buzzer.

*Vengono di seguito riportate le **documentazioni dei sensori e attuatori** utilizzati nel progetto:*

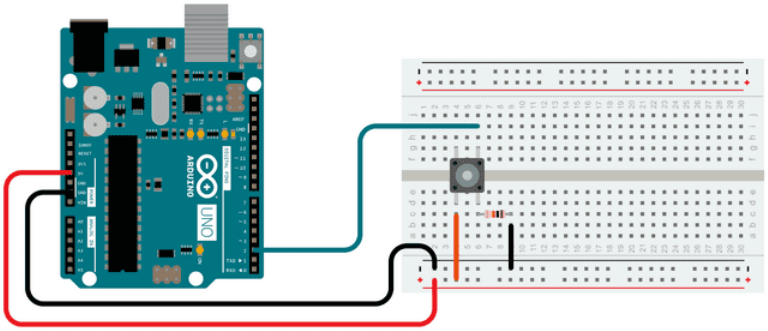
Pulsante: I pulsanti (switches) collegano due punti del circuito quando il bottone è in fase "push" ovvero quando è pigiato. Abbiamo uno stato di "LOW" sul pin utilizzato come INPUT quando il pulsante è aperto (non pigiato), pin utilizzato sarà collegato alla massa. Abbiamo invece uno stato di "HIGH" quando il pulsante è chiuso (pigiato) e fa quindi collegamento tra i piedini del bottone collegando quindi il pin a 5V. Utilizziamo inoltre una resistenza di 10kohm poiché se scollegassimo il pin di I/O , l'input sarebbe "fluttuante" e quindi restituirebbe valori casuali (HIGH e LOW).

Immagine Hardware:

Schema circuitale:



Circuito:



Nel progetto: Il pulsante nel progetto ha la funzione di riavviare il timer in qualsiasi momento, prevedo quindi di controllare ripetutamente se il pulsante sarà pigiato o meno utilizzando una `digitalRead(BUTTON)` nella funzione "loop", se mi restituirà un valore di tipo HIGH allora verrà ripristinato il display e la corrispettiva pagina html al valore iniziale.

Pin associato:

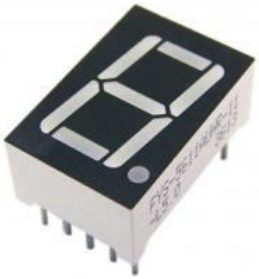
```
1. #define BUTTON 27
```

Display a 7 segmenti: Il display a 7 segmenti è un dispositivo elettronico (attuatore) in grado di visualizzare le 10 cifre numeriche (da 0 a 9), e in alcuni casi alcune lettere alfabetiche e simboli grafici, attraverso l'accensione di combinazioni di sette segmenti luminosi. Il display presenta ad ogni suo segmento un LED (Light Emitting Diode), questo consentirà quindi una gestione ottimale di ogni suo segmento associando ad ognuno di esso il suo corrispettivo pin. Bisogna specificare che di questo componente esistono due versioni: catodo comune (comune negativo) e anodo comune (comune positivo). Per rappresentare ogni cifra bisogna dunque utilizzare una configurazione predefinita di HIGH e LOW, lo schema è qui sotto riportato:

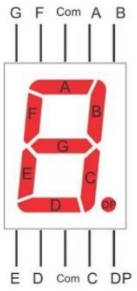
Configurazioni:

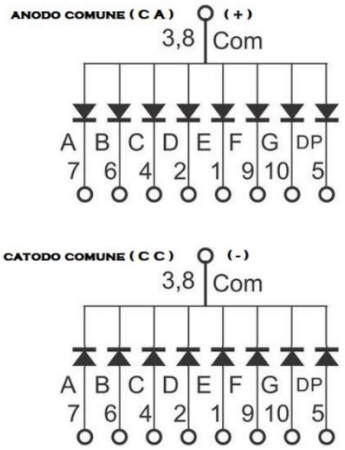
Digit	a	b	c	d	e	f	g
0	ON	ON	ON	ON	ON	ON	OFF
1	OFF	ON	ON	OFF	OFF	OFF	OFF
2	ON	ON	OFF	ON	ON	OFF	ON
3	ON	ON	ON	ON	OFF	OFF	ON
4	OFF	ON	ON	OFF	OFF	ON	ON
5	ON	OFF	ON	ON	OFF	ON	ON
6	ON	OFF	ON	ON	ON	ON	ON
7	ON	ON	ON	OFF	OFF	OFF	OFF
8	ON	ON	ON	ON	ON	ON	ON
9	ON	ON	ON	ON	OFF	ON	ON

Immagine Hardware:

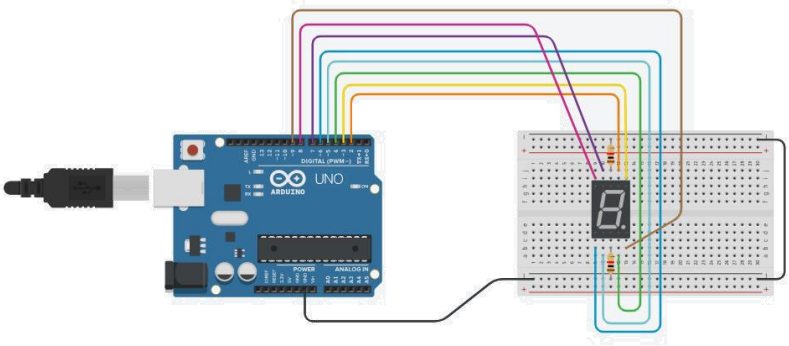


Schema del display:





Circuito:



Nel progetto: Il display a 7 segmenti assume nel progetto uno dei ruoli cardine, consente dunque la visualizzazione in tempo reale del timer. Il conto alla rovescia sarà visibile sul display decrementando di volta in volta il valore e stampandolo mediante opportune configurazioni. Ad ogni segmento sarà quindi associato un pin per la gestione delle configurazioni, mediante la funzione `digitalWrite(PIN,HIGH)` oppure `digitalWrite(PIN,LOW)` potremo stabilire l'accensione o spegnimento del segmento di riferimento. Il segmento 'dp' ovvero la rappresentazione del carattere ".", è stato escluso in quanto non necessario ai fini del progetto.

Pin associati:

```
1. #define A 2
2. #define B 15
3. #define C 16
4. #define D 17
5. #define E 18
6. #define F 4
7. #define G 5
```

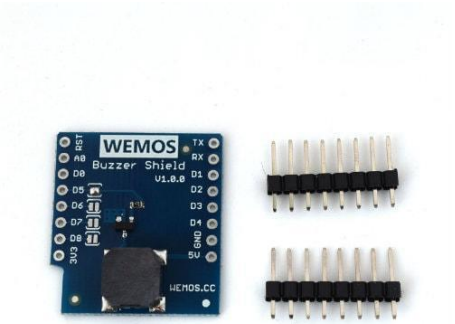
Buzzer: I buzzer sono dispositivi sonori che permettono di segnalare lo stato di un sistema. I buzzer possono essere di due tipi:

- Attivi
- Passivi

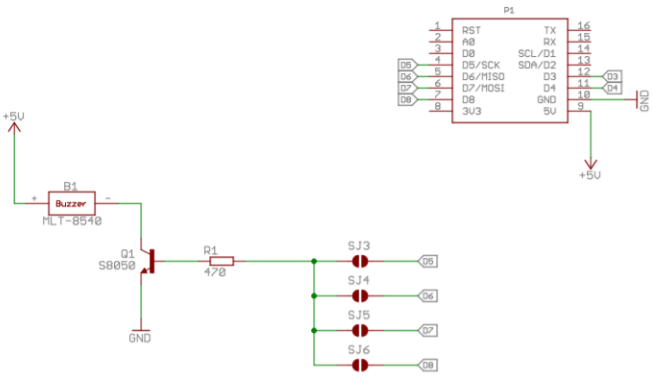
Il ***Buzzer attivo*** usa un oscillatore interno che permette di emettere un tono a frequenza fissa se viene alimentato con una tensione continua. Possiamo comandare l'emissione del suono abilitando o disabilitando l'alimentazione del buzzer. Il buzzer attivo è in grado di modificare il campo magnetico di una bobina a cui è connesso meccanicamente una membrana che oscillerà alla frequenza fissata dall'oscillatore.

Il ***Buzzer passivo*** non prevede un oscillatore interno, è quindi indispensabile un circuito esterno per generare un'onda quadra che mette in oscillazione la membrana interna del buzzer, questi attuatori potranno così emettere toni a diversa frequenza. Possiamo comandare l'emissione del suono per un certo tempo ad una determinata frequenza utilizzando la modulazione digitale PWM.

Immagine Hardware:



Schema elettrico:



Nel progetto: Il buzzer ha lo scopo di segnalare in maniera acustica, la fine del timer e di conseguenza la possibilità di potere intuire senza l'utilizzo del display a 7 segmenti, la fine del timer.

Pin associati:

```
1. #define BUZZER 19
```

Analisi progettuale

[Progetto E]: '*Si progetti un timer digitale tramite display a 7 segmenti. Predisporre l'applicazione per l'accesso alla rete wifi utilizzando l'interfaccia wireless integrata, il sistema deve integrare un webserver minimale per il settaggio del valore di timer (tempo di decremento e valore da 1 a 9), un pulsante per riavviare il timer. Allo scadere del timer bisogna avvisare l'utente con un segnale acustico.'*

Si introduce di seguito un'analisi progettuale tenendo conto delle possibili architetture e tecnologie utilizzate, documentando e formulando così una possibile soluzione: prevediamo l'accesso dell'esp32 al corrispettivo wifi di riferimento, utilizzando come parametri SSID e PASSWORD, il collegamento restituirà dunque una conferma o un avviso di errore nel caso in cui non si fosse connesso correttamente o viceversa se il collegamento è avvenuto con successo, questo è un passaggio molto importante affinchè i dispositivi della rete locale possano comunicare. Una volta accertatoci che l'accesso alla rete wifi è garantito, prevediamo un'implementazione webserver che segue dunque l'architettura client/server.

Architettura Client/Server

L'architettura Client/Server è un particolare tipo di configurazione, molto comune ed utilizzata, la quale prevede l'impiego di due elementi fondamentali per il suo funzionamento. Come si può dedurre dal nome questi elementi sono, per l'appunto, il Client e il Server.

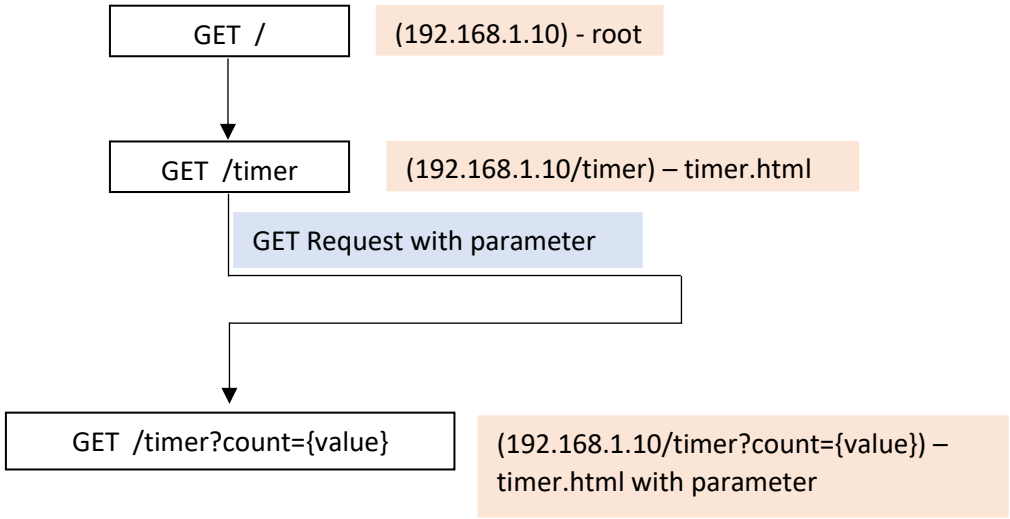
Client: all'interno di un'architettura Client/Server, il Client è colui che necessita di una risorsa che possiede il Server e che può ottenere inoltrando delle richieste a quest'ultimo. Le risorse di cui necessita possono essere di vario tipo, ad esempio, uno delle risorse fondamentali impiegate per questa progettazione è proprio il servizio web: HTTP, il quale specifica il formato con cui vengono scambiati o trasmessi i dati tra i due dispositivi. Generalmente, in questo tipo di architettura si può riscontrare la presenza di più Client che trasmettono, anche contemporaneamente, la stessa richiesta al Server. Nel nostro caso il Client può essere un qualsiasi dispositivo con il quale siamo in grado di interfacciarci con il Web Server, attraverso l'utilizzo di un'interfaccia web. Quest'interfaccia è raggiungibile, ovviamente, tramite un'apposita connessione ad internet in accoppiata ad un web browser. Per mezzo del browser siamo in grado di specificare un opportuno URL con il quale poter identificare univocamente una o più specifiche risorse.

Server: È l'elemento centrale dell'architettura. Il Server è quel dispositivo che si occupa di gestire le molteplici richieste inoltrate dai Client, che, tramite sempre l'utilizzo del protocollo HTTP, è in grado di trasmettere le risorse di cui dispone. Il Server è progetto per fornire più risorse differenti in base al punto di accesso definito come porta, la quale consiste in un valore numerico che identifica univocamente una specifica risorsa, in base al tipo di funzionamento che richiede la progettazione, possono essere implementate risorse di natura diversa, successivamente si parlerà in maniera più approfondita dei protocolli utilizzati per l'implementazione progettuale. Essendo l'elemento centrale dell'architettura, il Server può ricevere un numero illimitato di richieste da parte dei Client appartenenti alla rete, infatti, di base il Server può gestire più richieste contemporaneamente, tuttavia è anche una risorsa limitata, per questo motivo è possibile implementare una tecnica chiamata cluster, la quale consiste nel distribuire il carico di elaborazione ad una serie di macchine connesse tra loro che operano in parallelo, generalmente di natura omogenea tra loro, per bilanciare il numero illimitato di richieste da gestire. Nel nostro caso la situazione è molto più semplice, e non necessita di questa tecnica.

L'architettura client/server, nel progetto, prevede un client (qualunque dispositivo collegato alla rete locale) e un server capace di rispondere alle richieste provenienti dai client sulla porta 80 (ESP 32). In particolar modo dobbiamo gestire il web-server in modo tale che ad ogni richiesta ricevuta con protocollo http servizio presente nella porta 80, possa quindi ottenere una risposta contenente la risorsa cioè una pagina HTML (HyperText Markup Language) che integra CSS (Cascading Style Sheets). Analizzando una richiesta di tipo HTTP come nella foto sotto riportata

```
GET /timer HTTP/1.1
Host: 192.168.1.10
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.1.10/
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
```

notiamo che l'intestazione del pacchetto contiene quindi il percorso di riferimento richiesto dal client "GET /timer" (risorsa). Lavoriamo proprio su questa base per riconoscere quindi a quale percorso il client necessita di una risposta, restituendo quindi la pagina html (risorsa) stabilita. Il web server sarà gestito nel modo appena citato, sarà quindi in ascolto delle richieste provenienti dai client sulla porta 80 e ad ogni richiesta http, risponderà con il corrispettivo html. Il client dovrà interfacciarsi con un web browser per potere inviare le richieste, dovrà semplicemente digitare l'indirizzo ip del server nella sezione dell'URL sul browser, rimanendo quindi all'interno della rete locale (l'indirizzo ip del server potrebbe variare in quanto l'assegnazione non è di tipo statica) e inserire il percorso di riferimento se necessario. Prevediamo una gerarchia per lo sviluppo delle pagine all'interno del server:



GET /: Pagina principale, indirizzo radice del sito web, reindirizzerà il client al "GET /timer".

GET /timer: Questa pagina conterrà un input type text dove andrà inserito il corrispettivo valore di inizio timer, prevediamo quindi di gestire tutte le eccezioni che ci necessitano ovvero:
- se il valore inserito è un carattere alfanumerico allora il valore non è valido, generando così un errore del tipo "Errore input non valido".
- se il valore inserito è un valore numerico ma è minore di 0 oppure maggiore di 9 allora genera un messaggio di errore del tipo "Errore input non valido".
-se il valore immesso è valido, allora il client verrà reindirizzato al "GET /timer?count={value}", dove {value} sarà il valore del parametro "count", passato tramite url (Query String).

`GET /timer?count={value}:` La seguente pagina rappresenta il corpo del progetto in quanto gestirà non solo il timer visualizzabile nella pagina ma anche il display a 7 segmenti. Prevendiamo quindi una porzione di codice comune per sincronizzare entrambi. Il timer visualizzato verrà gestito tramite javascript (linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web), mentre il display verrà gestito attraverso le `digitalWrite(PIN,HIGH)` con le combinazioni opportune. Utilizziamo inoltre un ciclo per fare il conto alla rovescia del valore iniziale {value} (valore del parametro), decrementandolo così di volta in volta ogni secondo. Appena il timer arriverà a 0, verrà emesso un suono dal buzzer per segnalarne la fine. Viene incluso un bottone sia fisico (sopra documentato), che un bottone html, questo avrà il compito di riavviare il timer ogni qualvolta ne avremo bisogno, memorizziamo dunque il valore iniziale {value} in una variabile, in modo tale da far ripartire il timer virtuale e il display a 7 segmenti dalla variabile memorizzata.

Un'ulteriore analisi di questo progetto è stata eseguita sui protocolli utilizzati, vengono di seguito spiegati e documentati:

Cos'è un protocollo? Per consentire ai dispositivi di comunicare attraverso una rete, ognuno deve rispettare delle regole che svolgono molte funzioni nelle rete; queste regole sono chiamate protocolli. Un protocollo definisce l'ordine e il formato dei messaggi inviati e ricevuti tra le diverse entità di una rete, oltre alle azioni da compiere in seguito alla ricezione e alla trasmissione dei messaggi o di altri eventi. Tra i protocolli utilizzati abbiamo:

- ***IP:*** protocollo di rete alla base della rete Internet che si occupa di indirizzamento/instradamento dei pacchetti scambiati tra elementi della rete. La rete Internet è composta da elementi connessi, detti nodi, che trasmettono informazioni raggruppate in pacchetti e che si trovano in diversi "livelli" della rete (infatti, si distinguono reti e sotto-reti). L'obiettivo del protocollo è garantire una comunicazione tra 2 nodi qualsiasi della rete attraverso l'assegnazione di indirizzi, specificati anche negli appositi campi dei pacchetti per gestire l'istradamento di quest'ultimi. L'istradamento dei pacchetti viene gestito nello specifico da alcuni elementi detti router, che, attraverso questi indirizzi, riescono a inoltrare correttamente il pacchetto a destinazione. Questi indirizzi possono essere di due tipi: indirizzo IP, indirizzo a valenza globale, formato da 32 bit, che identifica in maniera univoca un'interfaccia connessa alla rete e serve a veicolare i pacchetti in tutte le reti e sotto-reti di Internet; indirizzo MAC, indirizzo con valenza locale, formato da 48 bit, che identifica in maniera univoca una stazione e serve a indirizzare correttamente il pacchetto quando questo arriva nella rete in cui si trova il destinatario. Nei 32 bit dell'indirizzo IP, raggruppati in 4 ottetti, abbiamo 2 parti: una parte identifica la rete a cui appartiene il dispositivo, che è uguale per tutti i dispositivi della rete; l'altra parte identifica il dispositivo stesso e le eventuali sotto-reti. Quando un dispositivo della rete vuole comunicare con un altro, questo mette nel pacchetto l'indirizzo IP del destinatario; il pacchetto arriva al router che, analizzando la parte dell'IP legata alla rete, riesce a veicolare correttamente il pacchetto nelle sotto-reti. Arrivato nella sotto-rete del destinatario, entra in gioco il protocollo ARP. Inoltre, i pacchetti presentano una "scadenza", poiché può capitare che il destinatario non sia più disponibile o che qualche tratto della rete sia interrotto, per garantire che i pacchetti "non girino all'infinito" nella rete, questi, dopo un certo numero di passaggi e tempo in cui non hanno raggiunto il destinatario, vengono cancellati.
- ***ARP:*** (Address Resolution Protocol) a partire dall'indirizzo IP, fa una "mappatura" tra indirizzo IP e indirizzo MAC e risale all'indirizzo MAC del destinatario così che il contenuto del pacchetto sia accessibile solo al destinatario che possiede quello specifico indirizzo.
- ***TCP:*** protocollo di rete che consente una comunicazione affidabile tra processi in esecuzione su host separati e fornisce trasmissioni affidabili e riconosciute che confermano il successo della consegna. Inoltre, effettua il riassemblaggio dei pacchetti e il controllo di flusso per regolare le diverse velocità di elaborazione che possono essere presenti tra mittente e destinatario. Fa parte della suite di protocolli TCP/IP, assieme ad IP, DHCP, CSMA/CA, HTTP e REST, e su di esso si appoggia gran parte delle applicazioni della rete Internet.
- ***DHCP:*** protocollo implementato come server che assegna dinamicamente ai dispositivi di una rete locale, ad ogni richiesta di accesso, un indirizzo IP permettendogli così di scambiare dati con gli altri dispositivi connessi alla rete.
- ***CSMA/CA:*** protocollo di accesso multiplo usato da Wi-Fi per evitare le collisioni nelle comunicazioni e risolvere il problema dei "nodi nascosti", secondo cui, in una rete WLAN, il ricevitore di un nodo non può

rilevare eventuali trasmissioni provenienti da altri nodi mentre il trasmettitore di un nodo è attivo, sfruttando anche il DCF (Distributed Coordination Function) per regolare determinati intervalli di tempo. Quando una stazione vuole trasmettere, ascolta il canale. Se il canale è libero, la stazione attende per un certo lasso di tempo, detto DIFS (Distributed Inter Frame Space), e, se il canale è rimasto libero per tutto il DIFS, può trasmettere il pacchetto. Completata la trasmissione, la stazione trasmittente aspetta per un tempo, detto SIFS (Short Inter Frame Space), la ricezione di un pacchetto di "acknowledgement", detto ACK, da parte della stazione ricevente che conferma la corretta ricezione del pacchetto. La durata del SIFS è inferiore a quella del DIFS e, in base alla versione dello standard 802.11 del Wi-Fi, la loro durata cambia. Durante il SIFS, le altre stazioni in ascolto sul canale, poiché questo è occupato, non invieranno trasmissioni, evitando così le collisioni, e devono aspettare per una durata casuale, detto tempo di backoff, che il canale si liberi. Le stazioni estraggono un valore casuale di CW (Collision Window) da un intervallo, che inizialmente è $0 < CW < CWmin$ e poi, dalla seconda trasmissione in poi, diventa $0 < CW < 2m * (CWmin + 1) – 1$ dove CWmin indica il valore minimo di CW e m è il numero di pacchetti ritrasmessi. Questo intervallo di valori può aumentare o diminuire in base al numero di ritrasmissioni e il valore pescato viene moltiplicato per uno slot temporale ottenendo il tempo che le stazioni devono aspettare affinché il canale si liberi. Questo protocollo, assieme alla bassa probabilità delle stazioni di estrarre lo stesso valore di CW, permette di evitare le collisioni, ma ha degli effetti negativi sul throughput, ovvero la capacità di trasmissione effettiva usata, poiché più tempo sprecano le stazioni ad aspettare minore sarà la velocità di trasmissione dati.
Per evitare anche il problema del "terminale nascosto", ovvero quando una stazione A che trasmette verso una stazione B può non essere in grado di rilevare una stazione C anch'essa impegnata in una comunicazione con B, generando così una collisione su B. Si considerano due pacchetti: RTS (Request To Send) e CTS (Clear To Send), i quali vengono scambiati tra trasmettitore e ricevitore e che annunciano alle stazioni in visibilità(che stanno ascoltando) la presenza di una trasmissione in corso. RTS e CTS vengono scambiati prima della trasmissione del pacchetto e le stazioni in visibilità, ascoltandoli, sanno che non devono trasmettere per un certo lasso di tempo, detto NAV, affinché il ciclo si completi. RTS viene mandato dal trasmettitore, esso contiene il proprio indirizzo e quello posseduto dal ricevitore in questione, mentre il CTS viene mandato dal ricevitore, il quale contiene solo l'indirizzo del ricevitore(nodo che ha trasmesso il RTS). Ci possono essere collisioni tra RTS e CTS ma, dato che sono pacchetti molto piccoli, sprechiamo meno tempo, evitando così collisioni nella trasmissione del pacchetto vero e proprio, che ha dimensioni maggiori e di conseguenza comportano un ritardo di ritrasmissione maggiore.

- ***HTTP:*** protocollo alla base del World Wide Web (WWW) poiché ne definisce l'architettura, detta "client/server", che predispone un server HTTP che rimane continuamente in ascolto per eventuali richieste di un client, ovvero qualsiasi dispositivo connesso, riguardo informazioni o entrare in una pagina web, alle quali il server risponderà inviandogli tutto ciò che chiedono. Il client riesce a identificare il server HTTP attraverso un indirizzo URL (Uniform Resource Locator), che identifica univocamente ogni risorsa disponibile sul web e specifica il protocollo da usare, costituito da: l'host name, cioè il nome del dispositivo in cui si trova il server che gestisce la risorsa, usato per inoltrare i pacchetti verso il server; il path name, che identifica la specifica risorsa nel server che stiamo raggiungendo. Con l'indirizzo IP, il client riesce a fare la richiesta al server HTTP che gli restituirà la pagina web. Vi sono 2 metodi per fare le richieste al server HTTP della pagina web: GET e POST. Nella richiesta GET, l'URL specifica il tipo di protocollo, il nome del server a cui facciamo la richiesta, la risorsa a cui vogliamo accedere (cioè il nome del file contenente la pagina web) e i parametri che ci servono per fare la richiesta con i loro valori; GET è il metodo più semplice e più usato per fare richieste a server web, infatti si usa una sola e semplice stringa. Nella richieste POST, le risorse e i parametri vengono aggiunti nella richiesta che facciamo al server web; rispetto alle richieste GET, possiamo aumentare il contenuto delle informazioni e i parametri che stiamo fornendo e renderli invisibili. Un esempio di richiesta POST è un form che invia dati personali, come in una registrazione. L'obbiettivo del protocollo HTTP è creare collegamenti ipertestuali tra i vari dispositivi connessi al server web, che risponde inviando le pagine web.
- ***REST:*** protocollo basato su HTTP che definisce un sistema di trasmissione di dati e di una o più risorse attraverso l'uso di specifici metodi HTTP per il recupero di informazioni, per la modifica ed altro. Tra i metodi HTTP abbiamo: GET, per recuperare informazioni sulle risorse o recuperare le risorse sul path; POST, per creare una risorsa all'interno di una risorsa data o per modificare risorse esistenti; PUT, per creare una nuova

risorsa sul server o modificare una già esistente; DELETE, per cancellare una risorsa. La caratteristica principale di questo protocollo è l'essere "stateless", cioè che il client, quando ha intenzione di inviare la richiesta, non necessita di conoscere lo stato in cui si trova il server.

Implementazione e commento:

Introduciamo adesso il codice utilizzato nel progetto e ne analizziamo il funzionamento: (per una migliore leggibilità alleghiamo il collegamento PasteBin)(https://pastebin.com/xMzpC4wC)

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

#define A 2
#define B 15
#define C 16
#define D 17
#define E 18
#define F 4
#define G 5
#define BUTTON 27
#define BUZZER 19

WiFiServer server(80);

const char* ssid = "iPhone";
const char* password = "IoT2021_2022";

int startTimer = 0; //SALVA L'INIZIO DEL TIMER PER IL RIAVVIO
int freq = 2000;
int channel = 0;
int resolution = 8;
```

Includiamo le librerie che ci necessitano Arduino.h e WiFi.h, la prima ci interessa poiché contiene tutte le direttive di arduino, nel caso invece di WiFi.h, utilizziamo le direttive per connetterci alla rete locale e creare il nostro WebServer. Definiamo quindi le nostre lettere "A, B, C, D, E, F, G" ai pin "2, 15, 16, 17, 18, 4, 5",per gestire il nostro display a 7 segmenti ed aggiungiamo quindi il pin per il pulsante e per il buzzer, rispettivamente a "BUTTON", "27" e a "BUZZER", "35". Dichiariamo il nostro server come tipo "WiFiServer" e lo mettiamo in ascolto nella porta 80 (http). Creiamo le nostre costanti di tipo array di caratteri "ssid = "iPhone"" e "Password= "IoT2021_2022"". Queste due costanti ci serviranno successivamente nella funzione di setup per connetterci alla nostra rete locale. Dichiaro inoltre una viariabile "startTimer" che ci servirà per memorizzare successivamente l'inizio del timer. Esp32 possiede un controller PWM con 16 canali indipendenti, che possono essere configurati per generare dei segnali PWM con diverse proprietà, questa configurazione iniziale ci servirà per utilizzare il buzzer su esp32 in quanto non supporta istruzioni quali 'analogWrite' (non integrate nella libreria interna). Scegliamo la frequenza del segnale PWM a 2000 Hz, utilizziamo il canale 0 ed una risoluzione ad 8 bit per il duty cycle (max 16 bit).

```
void setPin(){ //ISTANZIO I PIN DEL DISPLAY
    pinMode(BUTTON, INPUT);
    pinMode(A, OUTPUT);
    pinMode(B, OUTPUT);
    pinMode(C, OUTPUT);
    pinMode(D, OUTPUT);
    pinMode(E, OUTPUT);
    pinMode(F, OUTPUT);
    pinMode(G, OUTPUT);
}

void setup() {
  Serial.begin(115200);
  ledcSetup(channel, freq, resolution);
  ledcAttachPin(BUZZER, channel);
  setPin();
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");

  // In attesa di connessione
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connesso a ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}

const long timeoutTime = 2000;
String header;
```

Sviluppiamo una funzione setPin(), questa avrà al suo interno tutte le modalità dei pin (I/O), per questo scriviamo tutti i pin del display a 7 segmenti come “OUTPUT”, il buzzer come “OUTPUT” ed infine il bottone come “INPUT”, questa funzione verrà poi richiamata all’interno del setup. Entriamo adesso dentro una delle due funzioni principali, ovvero void setup(), questa verrà invocata una sola volta all’inizio del programma, analizziamone il contenuto: avremo dunque una Serial.begin(115200) , questa istruzione imposterà la velocità di trasmissione dati a 115200 bps nella porta seriale. Successivamente troviamo le istruzioni ‘`ledcSetup`’ e ‘`ledcAttachPin`’ , la prima si occuperà di configurare il PWM, che riceve in ingresso il canale, la frequenza e la risoluzione del duty cycle. Adesso invece utilizziamo ‘`ledcAttachPin`’ per collegare il canale precedentemente configurato al pin GPIO corrispettivo, nel nostro caso il 19. Successivamente effetuiamo un’invocazione al setPin() scritto precedentemente per impostare i pin di I/O, specifichiamo invece con WiFi.mode(WIFI_STA) la modalità di connessione WiFi di tipo STA cioè station, inizializziamo le impostazioni di rete,passando come parametri ssid e password con WiFi.begin(ssid,password), effettuiamo un ciclo per visualizzare quando la connessione si è istaurata con successo, stampando dei “.” durante l’attesa, una volta che lo stato sarà WL_CONNECTED cioè wifi connesso, allora uscirà dal ciclo e stamperà “Connesso a” con il nome della rete e stamperà “IP address: “ , WiFi.localIP() cioè l’ip a cui è stato associato l’esp32 nella rete locale, questo sarà l’indirizzo ip con il quale accedere successivamente alle risorse del server da parte del client. Con server.begin(), possiamo infine impostare il server in ascolto dei pacchetti in entrata nella porta di servizio 80 (http) precedentemente configurata. Dichiariamo una costante di nome “timeoutTime” che segna la chiusura di connessione tra client e server se lo stesso smette di inviare pacchetti (dopo 2 secondi dopo l’ultima ricezione) ma verrà approfondita successivamente questa procedura. Creiamo infine una variabile che chiameremo “header” che conterrà il pacchetto http.

```
void loop() {
  //Il server rimane in ascolto di nuovi client:
  WiFiClient client = server.available();   // viene restituito un client che è
connesso e che ha informazioni disponibili per essere lette
  //Controlliamo se un nuovo client si registra e se deve inviare pacchetti
  if (client) {
    Serial.println("Nuovo client registrato!");          // stampiamo il messaggio
di avvenuta registrazione
    String currentLine = "";  // Dati che trasmette il client (il pacchetto
inizialmente è vuoto)
    unsigned long currentTime = millis();
    unsigned long previousTime = currentTime;
    while (client.connected() && currentTime - previousTime <= timeoutTime) { //
loop che verifica fino a quando il client è connesso
      currentTime = millis();
      if (client.available()) {             // Verifichiamo se ci sono dei bytes
disponibili per essere letti
        char c = client.read();             // Legge il byte successivo ricevuto dal
server a cui è connesso il client
        header += c;
        if (c == '\n') {  // se il byte è uguale a \n
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");      //viene costruito il pacchetto
HTTP di risposta
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            client.println("<!DOCTYPE html><html>");
            Serial.println(header);
            client.println("<head><link rel='stylesheet'
href='https://cdn.jsdelivr.net/npm/uikit@3.8.1/dist/css/uikit.min.css'/><title>Wemos
Timer!</title><center><h1>Timer</h1></center></head>");
            String index = header.substring(0,6); //PRELEVO LA SOTTOSTRINGA "GET /"
DALLA RICHIESTA
            index.trim(); //SE TOLGO GLI SPAZI VUOTI, LA LUNGHEZZA NON SARA' MAI
COME QUELLA DELLA HOMEPAGE
            String count = header.substring(10,17); //ESTRAGGO IL VALORE DEL
PARAMETRO COUNT (SE CONTENUTO NELL'URL)
            String timer = header.substring(0,11);  //ESTRAGGO /TIMER SE E'
CONTENUTO NELL'URL
            timer.trim(); //TOLGO GLI SPAZI DALLO /TIMER
            Serial.println("");
            Serial.println(header);   //stampo il pacchetto
            Serial.println("");
```

Passiamo adesso alla seconda funzione principale void loop(), funzione invocata ripetutamente, la cui esecuzione si interrompe solo quando si toglie l'alimentazione alla board. In questa funzione troviamo WiFiClient client = server.avaible(), istanziamo quindi un client che sarà connesso e disponibile ad inviare pacchetti. Se client sarà uguale a true significa che sta per trasmettere un pacchetto e stamperemo quindi “Nuovo client registrato!”. Dichiariamo una variabile che chiameremo “currentLine”, questa variabile ci servirà successivamente per stabilire se il pacchetto sarà di lunghezza 0. Dichiariamo altre due variabili currentTime e previousTime, dentro queste due variabili troviamo quindi millis() per la prima che rappresenta una funzione che restituisce il tempo in millisecondi dall’istante in cui il programma è in funzione sulla board, questa variabile viene aggiornata costantemente all’interno del ciclo che successivamente vedremo, mentre previousTime memorizza il valore da currentTime nel momento in cui viene istanziato espresso in millisecondi ma rimane costante. Con un ciclo eseguiremo tutto il contenuto fino a quando il client è connesso e fino a quando la differenza fra il tempo corrente ed il tempo precedente è minore o uguale di timeoutTime cioè fino a quando non sono passati 2 secondi, dopo questi due secondi è sicuro che il pacchetto è stato inviato e quindi il client viene disconnesso. Siamo nell’ipotesi in cui il client sta trasmettendo il pacchetto, aggiorniamo quindi nuovamente il currentTime per assicurarci che non siano passati 2 secondi. Viene verificato inoltre se il client ha dei bytes disponibili per essere letti con client.avaible() , nel caso in cui la risposta sia

affermativa allora dichiariamo un char c = client.read() , che legge il byte successivo ricevuto dal server, con header+=c , ricostruiamo l'intero pacchetto. Se il byte è uguale a '\n' allora verifichiamo che il pacchetto dev'essere ancora costruito con currentLine.length() uguale a 0. Con client.println() stampiamo quindi il codice html con la corrispettiva risposta da parte del server di tipo 200 (Status: pagina trovata, successo), scriviamo che la risposta sarà di tipo text/html potendo inserire così i tag html necessari. Per il progetto abbiamo incluso un framework css/js chiamato "UIKIT" che ci ha permesso di utilizzare dei bottoni stilizzati includendo solo le classi di appartenenza nei tag. Abbiamo deciso inoltre di includere un'intestazione comune alle pagine ovvero "Timer" in formato <h1></h1>, posizionato al centro della pagina .Poichè il pacchetto sarà del tipo:

```
GET /timer HTTP/1.1
Host: 192.168.1.10
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.1.10/
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
```

Estrapoliamo una sottostringa dell'header "header.substring(0,6);" cioè prendiamo i primi 7 caratteri dall'header cioè "GET / ", togliamo gli spazi con index.trim(), per una spiegazione dettagliata denominiamo questo caso come CASO ROOT, per potere utilizzarlo successivamente. Estrapoliamo poi un'altra sottostringa dell'header "header.substring(10,17);" ed estrapoliamo l'ultima sottostringa dell'header "header.substring(0,11); " denominiamo questi due casi come CASO COUNT e CASO TIMER, effettuiamo quindi una cancellazione degli spazi del CASO TIMER con timer.trim(). Effettuiamo per ultimo delle stampe di debug per leggere il contenuto dei pacchetti.

```
if(index.length()==5){  //VERIFICO SE LA RICHIESTA VIENE FATTA ALLA RADICE
            client.println("<script>document.location.href='/timer';</script>");
//NEL CASO DI ESITO POSITIVO, REINDIRIZZA ALLO /TIMER
            } else if (header.indexOf("GET /timer") >= 0 && timer.length()==10) {
//VERIFICO SE LA RICHIESTA VIENE FATTA ALLO /TIMER
              client.println("<body>"); //NEL CASO DI ESITO POSITIVO COSTRUISCO LA
PAGINA
              client.println("<br><center>AMMINISTRATORE<br><div class='uk-
margin'><div class='uk-inline'><span class='uk-form-icon' uk-icon='icon:
future'></span><input style='text-align:right;' class='uk-input' id='ptext'
type='text' placeholder='Timer' required></div></div> <button class='uk-button uk-
button-primary' onclick='bclick()'>Avvia</button></center>");
              client.println("<script>function bclick(){var val =
document.getElementById('ptext').value; if(val===''){alert('Errore, devi inserire il
valore del timer prima di proseguire!');}else{location.href =
'./timer?count='+val;}}");
              client.println("</script></body>");
            } else if (header.indexOf("GET /timer?count=")>=0 &&
count.equals("?count=")) { //VERIFICO SE LA RICHIESTA VIENE FATTA ALLO /TIMER E SE
CONTIENE PARAMETRO
                int value;
                int pos1 = header.indexOf('=');
                int pos2 = header.indexOf('HTTP');
                value = header.substring(pos1+1, pos2-4).toInt();   //ESTRAGGO IL
VALORE DEL PARAMETRO E LO CONVERTO IN NUMERO INTERO
                if(value<=9 && value>=1){  //EFFETTUO UN CONTROLLO PER VERIFICARE SE
L'INPUT IMMESSO E' VALIDO AI FINI DEL TIMER
                  startTimer = value; //MEMORIZZO IL VALORE INIZIALE NELLA VARIABILE
startTimer
                  setTimer(value,client); //AVVIO IL TIMER MEDIANTE LA FUNZIONE
setTimer()
```

CASO ROOT: consideriamo quindi un pacchetto ricevuto "GET / " a confronto con un pacchetto "GET /timer", estrapoliamo come già detto i primi caratteri potremo accorgerci facilmente che "GET /" è diverso da "GET /t", in quanto l'indirizzo radice, avrà sempre e comunque un carattere in meno rispetto a qualsiasi indirizzo dello stesso

webserver. Operiamo quindi con la lunghezza dei caratteri nella prima condizione, avendo fatto il trim sappiamo per certo che i caratteri utilizzati saranno 5 scriviamo così la nostra prima condizione. Nel caso in cui il client faccia richiesta del indirizzo radice allora otterà un redirect automatico con javascript al percorso "GET /timer"

CASO TIMER: consideriamo adesso il caso in cui il client ottenga "GET /timer", individuiamo questo percorso individuando inizialmente se è presente nella richiesta http utilizzando header.indexOf("`GET /timer`") ed in secondo luogo poiché sappiamo che l'indexOf "GET /timer" sarà identico all'indexOf di "GET /timer?count={value}", allora utilizziamo nuovamente la logica di prima, verifichiamo che "GET /timer " trim cioè "GET /timer" abbiamo una lunghezza pari a 10 che è diversa da "GET /timer?" che ha una lunghezza pari ad 11, così facendo abbiamo verificato con successo che l'utente abbia inserito solamente GET /timer. Una volta riconosciuta la pagina da visualizzare, il client visualizzerà una pagina html contenente una casella di input di tipo text, dove andrà ad inserire il valore del timer ed andrà ad azionarlo mediante il pulsante "avvia", questo valore verrà inserito tramite un codice javascript nell'url come parametro "count", e verrà eseguito un reindirizzamento della pagina, ottenendo così "GET /timer?count={value}". Inseriamo opportunamente un controllo che verifica se il campo rimane vuoto, lanciando così un alert che ne avvisa dell'errore.

CASO COUNT: consideriamo infine il caso in cui il client ottenga "GET /timer?count={value}", lo verifichiamo semplicemente facendo nuovamente l'indexOf di "GET /timer?count=" e verificando se la sottostringa "count" precedentemente citata, è uguale a "?count=", in questo modo possiamo accertarci che il percorso è corretto. Utilizziamo infine una substring per estrapolare il valore del parametro dall'url, un ultimo controllo viene dedicato al valore del parametro, per verificare se l'input immesso è un valore compreso tra 1 e 9, in caso si tratti di un esito positivo allora procediamo ad avviare il timer mediante la funzione setTimer(value,client), impostiamo inoltre il valore di timer iniziale nella variabile startTimer nel caso in cui ci serva un reset tramite bottone. Sia l'esito negativo,

```
}else{  //SE L'INPUT NON E' VALIDO ALLORA RESISTUISCE VALORE NON VALIDO
                    client.println("<center><div><div class='uk-card uk-card-primary
uk-card-hover uk-card-body uk-light'><h2 class='uk-card-title'>Valore non
valido</h2><p>Errore valore non valido: Il valore dev'essere compreso tra 9 e
1.</p></div></div></center>");

client.println("<script>window.setTimeout(function(){window.location.href =
'./timer';}, 3000);</script>");
                  }
             }else{  //SE L'URL E' DIVERSO DA TUTTE LE POSSIBILI SOLUZIONI ALLORA
RESISTUISCE UNA PAGINA DOVE DICE "ERRORE 404"
               client.println("<center><div><div class='uk-card uk-card-primary uk-
card-hover uk-card-body uk-light'><h2 class='uk-card-title'>404</h2><p>Errore 404:
Pagina non trovata.</p></div></div></center>");
             }
             client.println("<script
src='https://cdn.jsdelivr.net/npm/uikit@3.8.1/dist/js/uikit.min.js'></script><script
src='https://cdn.jsdelivr.net/npm/uikit@3.8.1/dist/js/uikit-
icons.min.js'></script>");
             client.println("</html>");
             break;
           } else {
             currentLine = "";
           }
         } else if (c != '\r') {    //SE I BYTES NON CONTENGONO /r , AGGIUNGE ALLA
CURRENT LINE TUTTI I BYTES
           currentLine += c;
         }
       }
     }
```

Nel caso in cui l'utente inserisca un valore non compreso nel range imposto, automaticamente verrà reindirizzato in una pagina in cui leggerà "Valore non valido", che, dopo 3 secondi porterà nuovamente alla pagina iniziale cioè "GET /timer". Il secondo else invece specifica che se l'url è diverso da quelli configurati, automaticamente si ritroverà una pagina di errore 404: pagina non trovata. Gli ultimi controlli gestiscono la currentLine e forniscono un riempimento con bytes nel caso in cui il byte sia diverso da '\r' cioè valore di ritorno e quindi se il byte è un carattere.

```
//IL CLIENT NON E' PIU' CONNESSO E LO DISCONNETTE
    header = "";
    delay(5000);
    client.stop();
    Serial.println("Client disconnesso.");
    Serial.println("");
  }
}
```

Infine , una volta scaduti 2 secondi imposti inizialmente, imposteremo il pacchetto vuoto, un delay (tempo di attesa) di 5 secondi e dopo disconnettiamo il client dal server, restituendo una stampa con "Client disconnesso".

```
void setTimer(int value,WiFiClient client){ //FUNZIONE CHE GESTISCE IL TIMER VIRTUALE
ED IL DISPLAY A 7 SEGMENTI
  long previousMillis = 0;
  long interval = 1000;
  long currentMillis = millis();
  client.println("<body><center><div id='countdown'></div><br><button class='uk-button
uk-button-danger' onclick='onrestart()'>Riavvia</button></center>"); //STAMPIAMO UN
BOTTONE NELLA PAGINA CHE RIAVVIA IL TIMER
    for(int i=value;i>=-1;--i){ //3*2 --> 6 //CICLO FOR CHE EFFETTUA IL CONTO ALLA
ROVESCIA
      if(digitalRead(BUTTON) == 1){          //VERIFICO CONTINUAMENTE AD OGNI CICLO SE
IL PULSANTE E' PIGGIATO
              i = startTimer; //SE E' PIGGIATO, RIPRISTINA IL CONTATORE AL VALORE
INIZIALE
              client.print("<script>countdown =");
              client.print(startTimer);
              client.print(";</script>");
       }
      delay(1000);
      switch(i){
        case 1:
            do{
                currentMillis = millis();
                if(currentMillis - previousMillis > interval){      //VERIFICO SE E'
PASSATO 1 SECONDO
                  digitalWrite(A,LOW);                              //COMBINAZIONE DEL
NUMERO 1 DEL DISPLAY A 7 SEGMENTI
                  digitalWrite(B,HIGH);
                  digitalWrite(C,HIGH);
                  digitalWrite(D,LOW);
                  digitalWrite(E,LOW);
                  digitalWrite(F,LOW);
                  digitalWrite(G,LOW);
                  client.print("<script>var countdown =");          //IMPOSTO IL TIMER
VIRTUALE AL VALORE 1
                  client.print(i);
                  client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
               }
              }while(currentMillis-previousMillis < interval+1);
              previousMillis = currentMillis; break;
```

```
case 2:
            do{
                currentMillis = millis();
                if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                  digitalWrite(A,HIGH); //COMBINAZIONE DEL NUMERO 2 DEL DISPLAY A
7 SEGMENTI
                  digitalWrite(B,HIGH);
                  digitalWrite(C,LOW);
                  digitalWrite(D,HIGH);
                  digitalWrite(E,HIGH);
                  digitalWrite(F,LOW);
                  digitalWrite(G,HIGH);
                  client.print("<script>var countdown =");  //IMPOSTO IL TIMER
VIRTUALE AL VALORE 2
                  client.print(i);
                  client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                }
               }while(currentMillis-previousMillis < interval+1);
               previousMillis = currentMillis;
        break;
        case 3:
              do{
                currentMillis = millis();
                if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                  digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 3 DEL DISPLAY A
7 SEGMENTI
                  digitalWrite(B,HIGH);
                  digitalWrite(C,HIGH);
                  digitalWrite(D,HIGH);
                  digitalWrite(E,LOW);
                  digitalWrite(F,LOW);
                  digitalWrite(G,HIGH);
                  client.print("<script>var countdown =");  //IMPOSTO IL TIMER
VIRTUALE AL VALORE 3
                  client.print(i);
                  client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                }
               }while(currentMillis-previousMillis < interval+1);
               previousMillis = currentMillis;
        break;

       case 4:
              do{
                currentMillis = millis();
                if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                  digitalWrite(A,LOW);  //COMBINAZIONE DEL NUMERO 4 DEL DISPLAY A
7 SEGMENTI
                  digitalWrite(B,HIGH);
                  digitalWrite(C,HIGH);
                  digitalWrite(D,LOW);
                  digitalWrite(E,LOW);
                  digitalWrite(F,HIGH);
                  client.print("<script>var countdown =");  //IMPOSTO IL TIMER
VIRTUALE AL VALORE 4
                  client.print(i);
                  client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                }
               }while(currentMillis-previousMillis < interval+1);
```

```
                previousMillis = currentMillis;
        break;
        case 5:
              do{
                  currentMillis = millis();
                  if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                    digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 5 DEL DISPLAY A
7 SEGMENTI
                    digitalWrite(B,LOW);
                    digitalWrite(C,HIGH);
                    digitalWrite(D,HIGH);
                    digitalWrite(E,LOW);
                    digitalWrite(F,HIGH);
                    digitalWrite(G,HIGH);
                    client.print("<script>var countdown ="); //IMPOSTO IL TIMER
VIRTUALE AL VALORE 5
                    client.print(i);
                    client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                  }
                }while(currentMillis-previousMillis < interval+1);
                previousMillis = currentMillis;
        break;
        case 6:
              do{
                  currentMillis = millis();
                  if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                    digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 6 DEL DISPLAY A
7 SEGMENTI
                    digitalWrite(B,LOW);
                    digitalWrite(C,HIGH);
                    digitalWrite(D,HIGH);
                    digitalWrite(E,HIGH);
                    digitalWrite(F,HIGH);
                    digitalWrite(G,HIGH);
                    client.print("<script>var countdown ="); //IMPOSTO IL TIMER
VIRTUALE AL VALORE 6
                    client.print(i);
                    client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                  }
                }while(currentMillis-previousMillis < interval+1);
                previousMillis = currentMillis;
        break;
          case 7:

              do{
                  currentMillis = millis();
                  if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                    digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 7 DEL DISPLAY A
7 SEGMENTI
                    digitalWrite(B,HIGH);
                    digitalWrite(C,HIGH);
                    digitalWrite(D,LOW);
                    digitalWrite(E,LOW);
                    digitalWrite(F,HIGH);
                    digitalWrite(G,LOW);
                    client.print("<script>var countdown ="); //IMPOSTO IL TIMER
VIRTUALE AL VALORE 7
                    client.print(i);
                    client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
```

```
                }
               }while(currentMillis-previousMillis < interval+1);
               previousMillis = currentMillis;
        break;
        case 8:
              do{
                  currentMillis = millis();
                  if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                    digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 8 DEL DISPLAY
A 7 SEGMENTI
                    digitalWrite(B,HIGH);
                    digitalWrite(C,HIGH);
                    digitalWrite(D,HIGH);
                    digitalWrite(E,HIGH);
                    digitalWrite(F,HIGH);
                    digitalWrite(G,HIGH);
                    client.print("<script>var countdown ="); //IMPOSTO IL TIMER
VIRTUALE AL VALORE 8
                    client.print(i);
                    client.print("; document.getElementById('countdown').innerHTML
= countdown + ' secondi rimanenti'; </script>");
                  }
               }while(currentMillis-previousMillis < interval+1);
               previousMillis = currentMillis;
          break;
        case 9:
              do{
                 currentMillis = millis();
                 if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                   digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 9 DEL DISPLAY A
7 SEGMENTI
                   digitalWrite(B,HIGH);
                   digitalWrite(C,HIGH);
                   digitalWrite(D,HIGH);
                   digitalWrite(E,LOW);
                   digitalWrite(F,HIGH);
                   digitalWrite(G,HIGH);
                   client.print("<script>var countdown ="); //IMPOSTO IL TIMER
VIRTUALE AL VALORE 9
                   client.print(i);
                   client.print("; document.getElementById('countdown').innerHTML =
countdown + ' secondi rimanenti'; </script>");
                 }
               }while(currentMillis-previousMillis < interval+1);
               previousMillis = currentMillis;
        break;
```

```
case 0:
              do{
                   currentMillis = millis();
                   if(currentMillis - previousMillis > interval){ //VERIFICO SE E'
PASSATO 1 SECONDO
                     digitalWrite(A,HIGH);  //COMBINAZIONE DEL NUMERO 0 DEL DISPLAY
A 7 SEGMENTI
                     digitalWrite(B,HIGH);
                     digitalWrite(C,HIGH);
                     digitalWrite(D,HIGH);
                     digitalWrite(E,HIGH);
                     digitalWrite(F,HIGH);
                     digitalWrite(G,LOW);

client.print("<script>document.getElementById('countdown').innerHTML = 'Timer
terminato'; </script>"); //IMPOSTO IL TIMER VIRTUALE A "Timer terminato".
                   }
                 }while(currentMillis-previousMillis < interval+1);
                 previousMillis = currentMillis;
         ledcWrite(channel, 125);
         ledcWriteTone(channel, 1255);
         delay(2000);
         ledcWriteTone(channel, 0);
        break;
        default:
        do{
                   currentMillis = millis();
                   if(digitalRead(BUTTON) == 1){ //VERIFICO SE SCADUTO IL TIMER,
L'UTENTE PIGIA IL BOTTONE DI RESET
                     i = startTimer;
                     client.print("<script>countdown =");
                     client.print(startTimer);
                     client.print(";</script>");
                   }
                 }while(currentMillis-previousMillis < interval*7);  //VERIFICO PER 7
SECONDI SE IL PULSANTE E' PIGIATO
                 previousMillis = currentMillis;
       break;
      }
    client.print("<script>function onrestart(){
location.reload();document.getElementById('countdown').innerHTML = 'Alla scadenza
del timer verr&agrave eseguito un riavvio automatico...';}</script>"); //SE CLICCA
SUL RIAVVIO VIRTUALE RICARICA LA PAGINA
    }
    client.println("</body>");
}
```

Quest'ultima funzione setTimer(value,client), richiede due parametri il valore d'inizio del timer e la reference del client, questo perché abbiamo bisogno di gestire il client.println all'interno della funzione. Abbiamo 3 variabili dichiarate: previousMillis = 0, interval = 1000 e currentMillis = millis() (tempo di esecuzione), questi 3 valori come nel caso già presentatosi, servono per gestire il tempo in quanto ogni volta che passa 1 secondo, il timer viene decrementato di 1. Effettuiamo il conto alla rovescia con un for, partendo dal parametro specificato (value) quando si richiama la funzione. Attraverso uno switch gestiamo i casi che vanno da 1 a 9, inserendo non solo le corrispettive configurazioni del display a 7 segmenti ma anche un codice javascript che gestisce il conto alla rovescia virtuale, andando a modificare di volta in volta il div "countdown" nella pagina. All'interno del ciclo for includiamo inoltre l'analisi del bottone fisico in modo tale che se è pigiato, cioè se la digitalRead(BUTTON) restituisce 1 allora automaticamente viene prelevato lo startTimer e viene impostato il valore di "i = startTimer", in modo tale che il timer al ciclo successivo, riprende di nuovo dal valore iniziale e quindi si riavvia, il tutto gestito sempre con un codice javascript che modifica il valore di countdown. Nel caso 0, oltre alla configurazione del display, definiamo anche il suono del buzzer, per avvisare che il timer è appena terminato: con 'ledcWrite' impostiamo il duty cycle

passando in input sia il canale che il valore del duty cycle da impostare (nel nostro caso 0 e 125). Con l'istruzione `'ledcWriteTone'` effettuiamo l'emissione del suono passando come parametri il canale e la frequenza del suono da emettere, impostiamo quindi un delay di 2 secondi che corrisponde alla durata dell'emissione del suono ed infine lo blocchiamo utilizzando l'istruzione precedente ed impostandola alla frequenza 0. Specifichiamo inoltre che il ciclo va da i=n fino ad i=-1, poiché nel momento in cui il display trova allo stato 0, al ciclo successivo passa al caso 'default' (mantenendo il display su 0) fornendo 7 secondi di tempo per potere resettare di nuovo il timer allo startTimer (mediante bottone fisico). Infine includiamo il classico bottone virtuale per potere riavviare il timer con un semplice refresh tramite javascript.

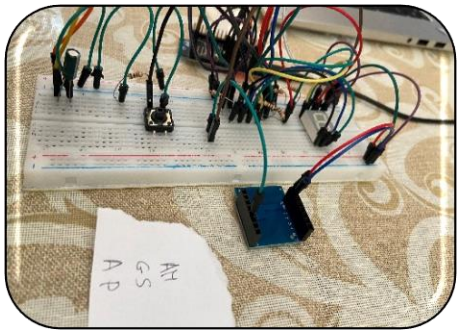
Precisazioni: Riavvio istantaneo con bottone fisico (anche durante il conto alla rovescia), riavvio dopo la scadenza del timer con bottone virtuale. Il bottone virtuale non era richiesto nel progetto ma è stato comunque implementato.

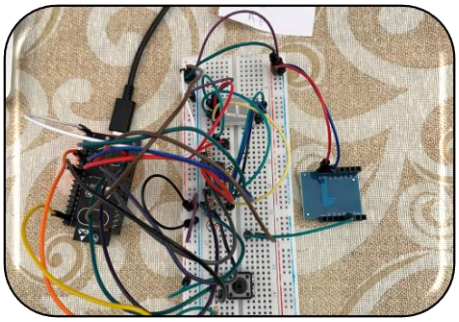
L'intero codice è visionabile su PASTEBIN:

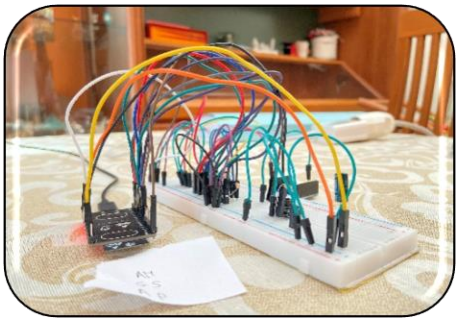
(per una migliore leggibilità) https://pastebin.com/xMzpC4wC

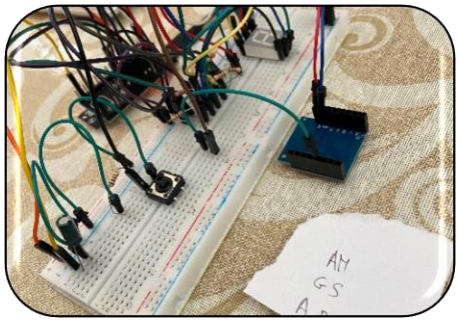
Foto del circuito:

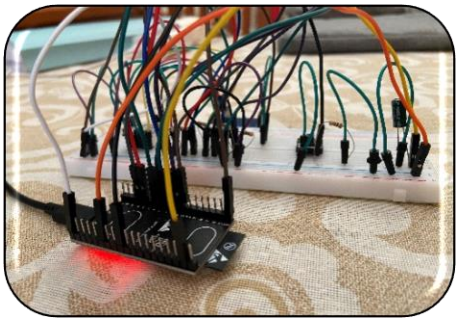
Su quasi ogni immagine è presente un foglietto di carta con le iniziali (nome e cognome) del gruppo ESP32 per dimostrare che le foto sono state scattate e non provengono da fonti di terze parti



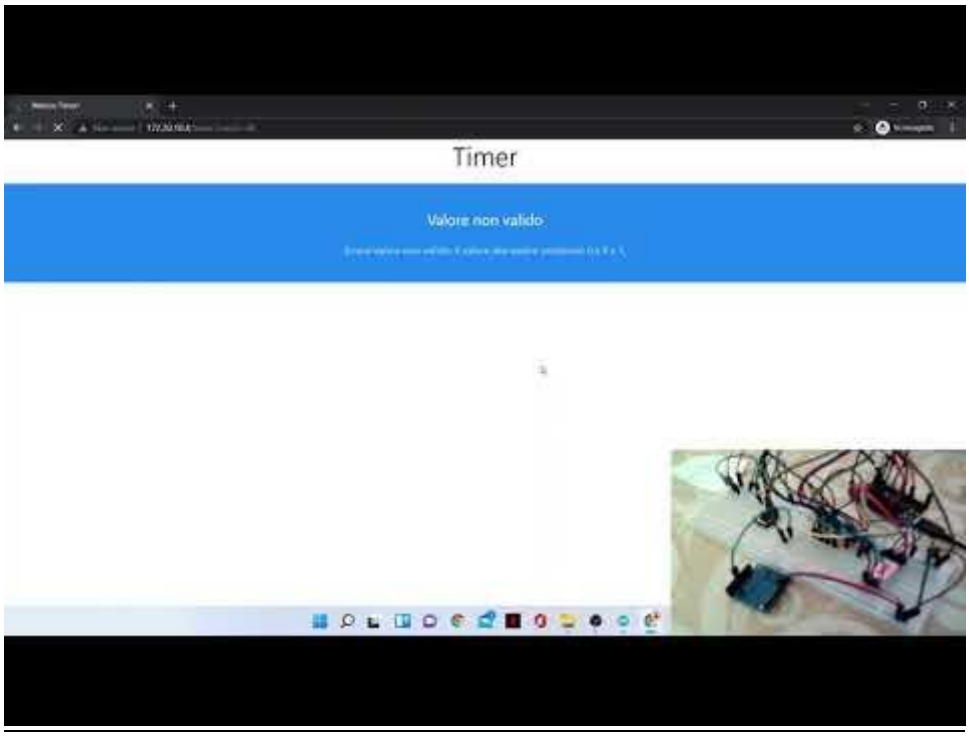








Video dimostrativo:

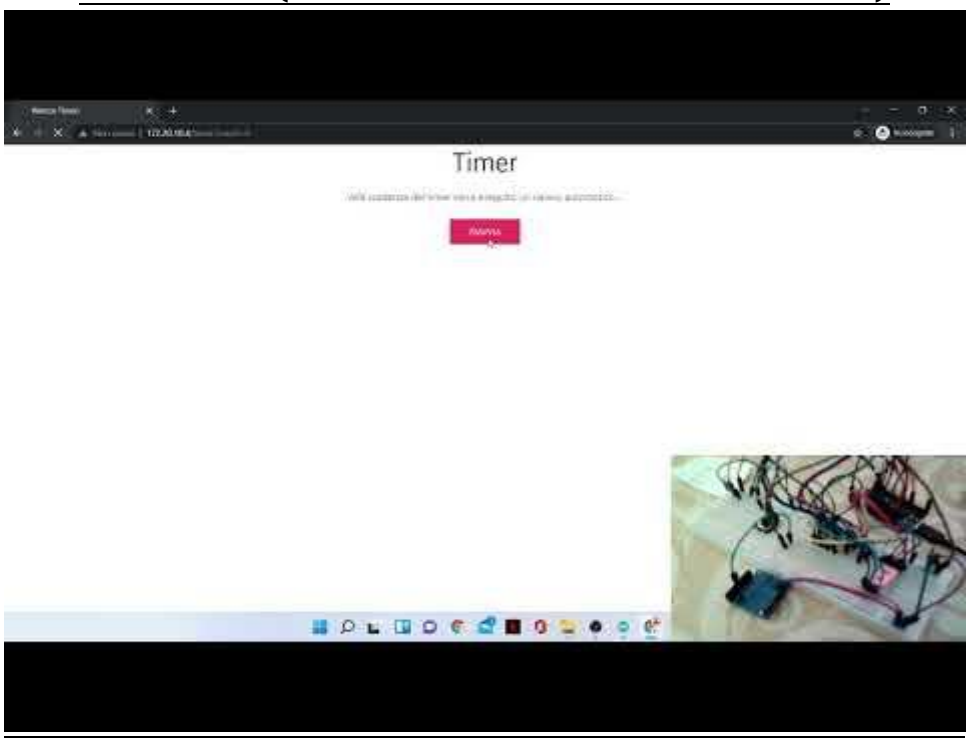


(https://www.youtube.com/watch?v=sa081y2zwTk)

Nel video dimostrativo appena osservato, vengono analizzati e gestiti tutti i casi sopra studiati. Il progetto risulta quindi finito e pronto per essere utilizzato. Questa fase come osservato nel video, rispetta i requisiti richiesti nel Progetto E, citato ad inizio stesura.

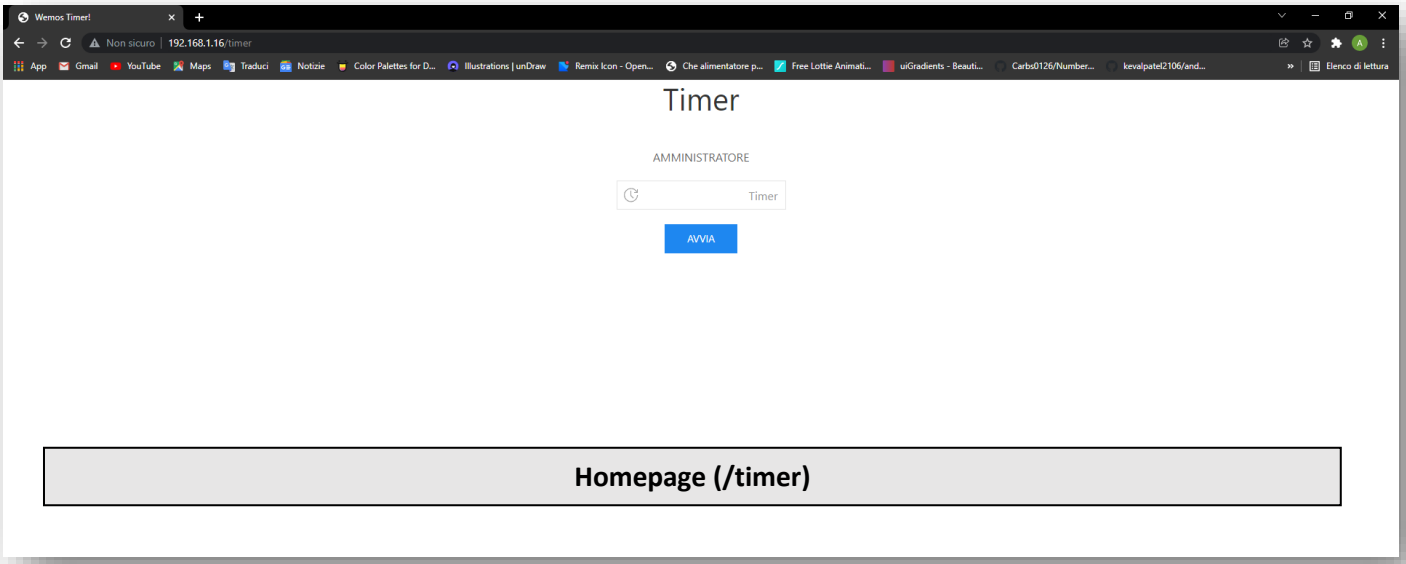
Viene inoltre allegato un video Extra per il funzionamento del bottone virtuale:

VideoExtra:(riavvio mediante bottone virtuale)

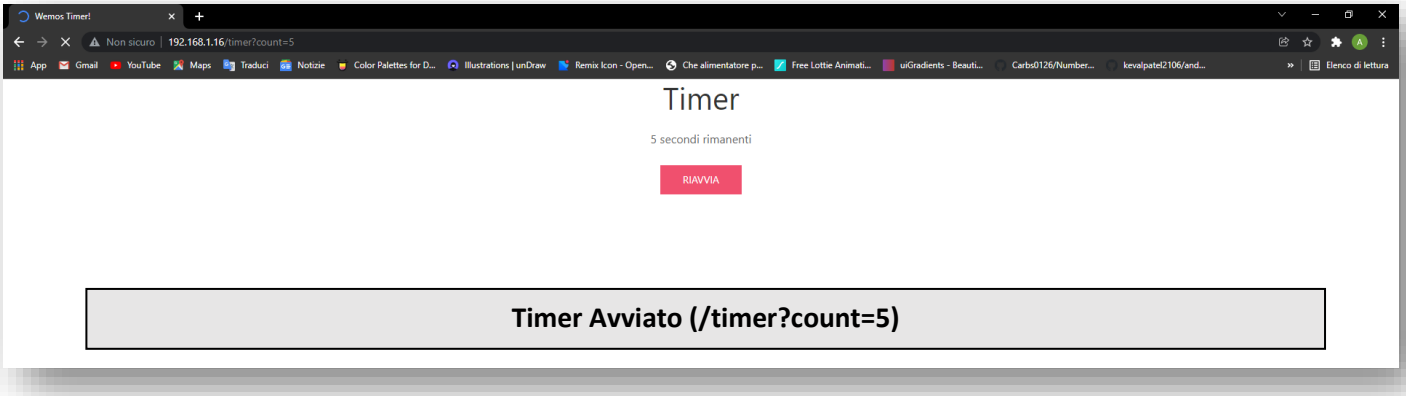


(https://www.youtube.com/watch?v=kHfPwz8rl0w)

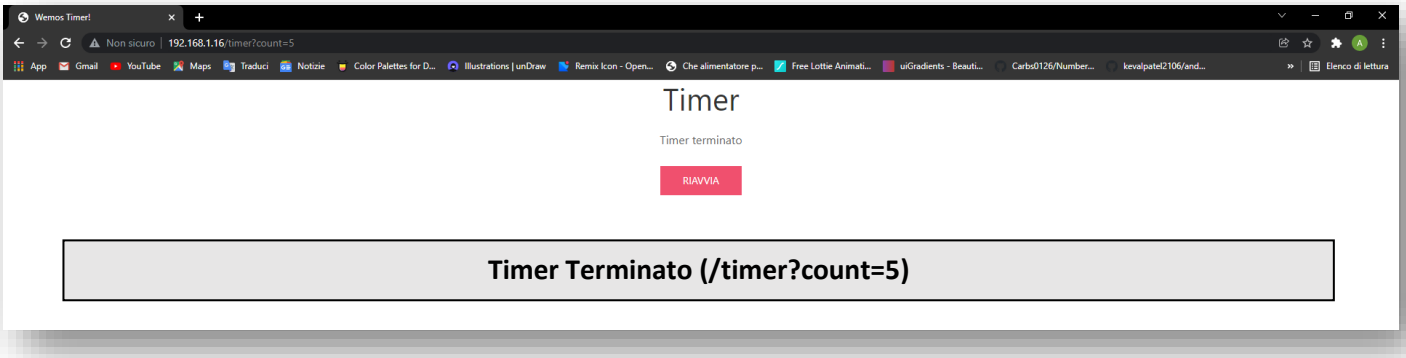
Screenshot WebServer:



Homepage (/timer)

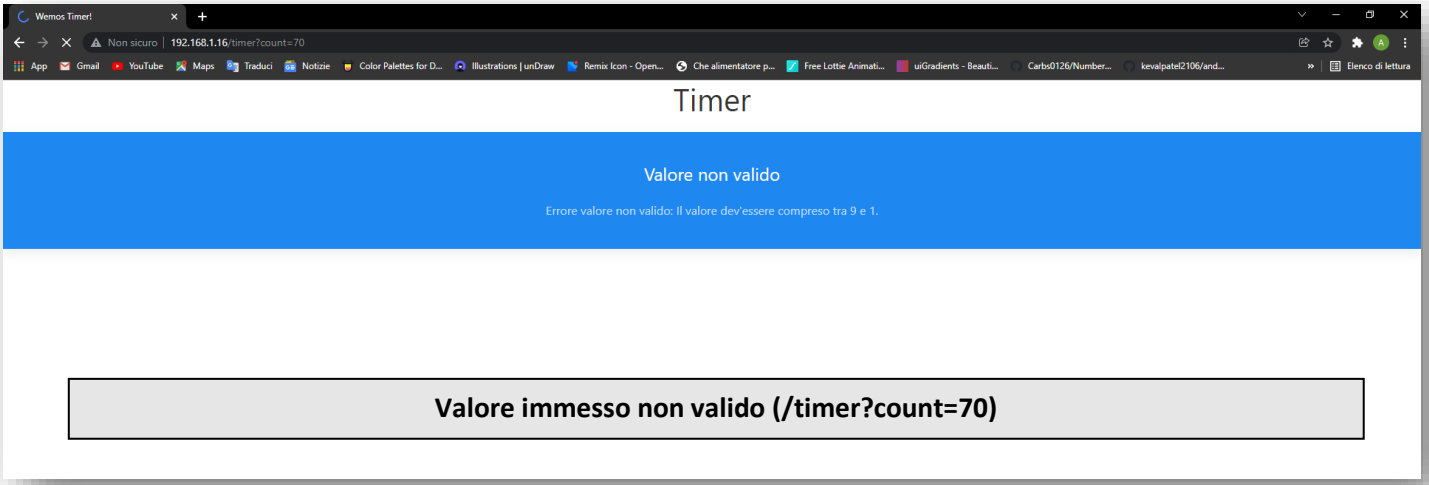


Timer Avviato (/timer?count=5)

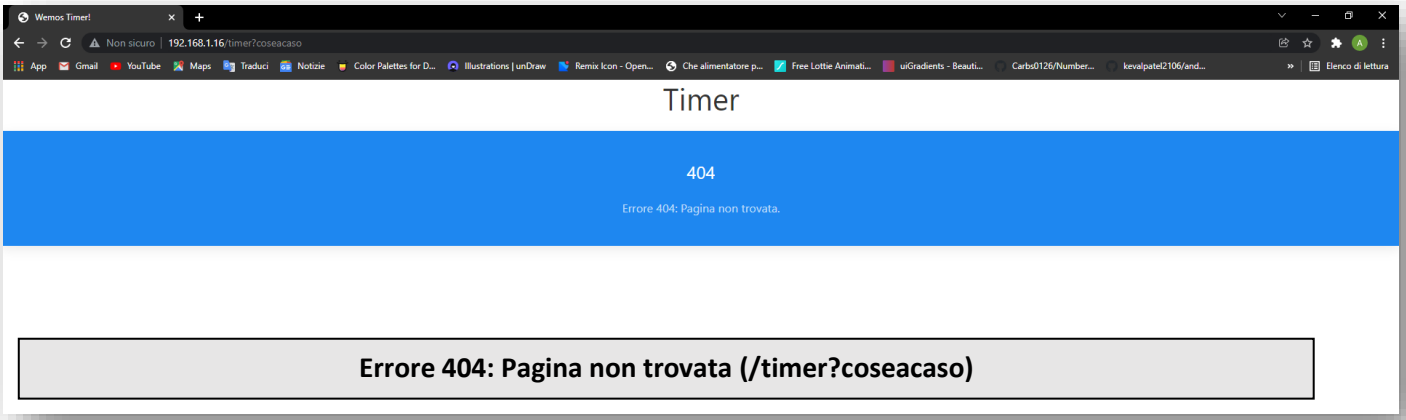


Timer Terminato (/timer?count=5)

Timer Riavviato con bottone virtuale (/timer?count=5)



Valore immesso non valido (/timer?count=70)



Errore 404: Pagina non trovata (/timer?coseacaso)

Conclusioni:

Il timer può avere molteplici impieghi: si pensi ad esempio l'utilizzo di un timer per una gara sportiva oppure l'impiego previsto come un semaforo stradale, che, con un segnale acustico ed un display a 7 segmenti, ne segnala l'attraversamento pedonale. Concludiamo quindi il progetto, avendo documentato e dimostrato i meccanismi ed i suoi funzionamenti, rispettando così la tesi proposta.