Prototipizzazione con Arduino 2023
Ardify

Aldegheri Alessandro VR471346
Michele Cipriani VR471337

Indice

1 Introduzione **3**

2 Componenti Utilizzati **4**

3 Sviluppo **5**

4 Funzionamento Codice e Librerie **6**

5 Schema Collegamenti **13**

6 Realizzazione Fisica **14**

7 Bibliografia **15**

1 Introduzione

Ardify è un centro di controllo che interagisce con il proprio Player di Spotify mediante l'utilizzo di API ufficiali. Esso è composto da due schede comunicanti, un Arduino UNO e un ESP32 Thing Plus, con relative funzioni. Arduino si occupa dell'interazione con l'utente, dal quale riceve comandi tramite apposito telecomando ad infrarossi. Le possibili operazioni sono:

- **Pause e Resume**: permettono di fermare e far ripartire la riproduzione a proprio piacimento;
- **Skip**: permette di saltare alla canzone successiva;
- **Previous**: permette di tornare alla canzone precedente;
- **Volume**: modifica del volume a propria scelta.

Inoltre, collegato all'Arduino, è presente uno schermo LCD 16x02 il quale mostra a video il titolo della canzone corrente e l'azione ricevuta. Tramite un opportuno collegamento, Arduino comunica alla seconda scheda alcune direttive per indicare quali azioni intraprendere.

L'ESP32, una volta ricevute le informazioni, effettua le chiamate API corrispondenti. Contemporaneamente la scheda è connessa al cloud tramite ArduinoIOT, il quale ci permette di salvare e tenere aggiornate delle statistiche visionabili sulla piattaforma online. Le statistiche salvate sono:

- **Storico Device Name**: uno storico di tutti i dispositivi utilizzati;
- **Contatore Skip**: un contatore temporaneo per ogni sessione che conta le operazioni di "Skip";
- **Contatore Previous**: un contatore temporaneo per ogni sessione che conta le operazioni di "Previous";
- **Volume**: valore attuale del volume;
- **Storico Volume**: un grafico che mostra l'andamento del volume durante la sessione attuale;

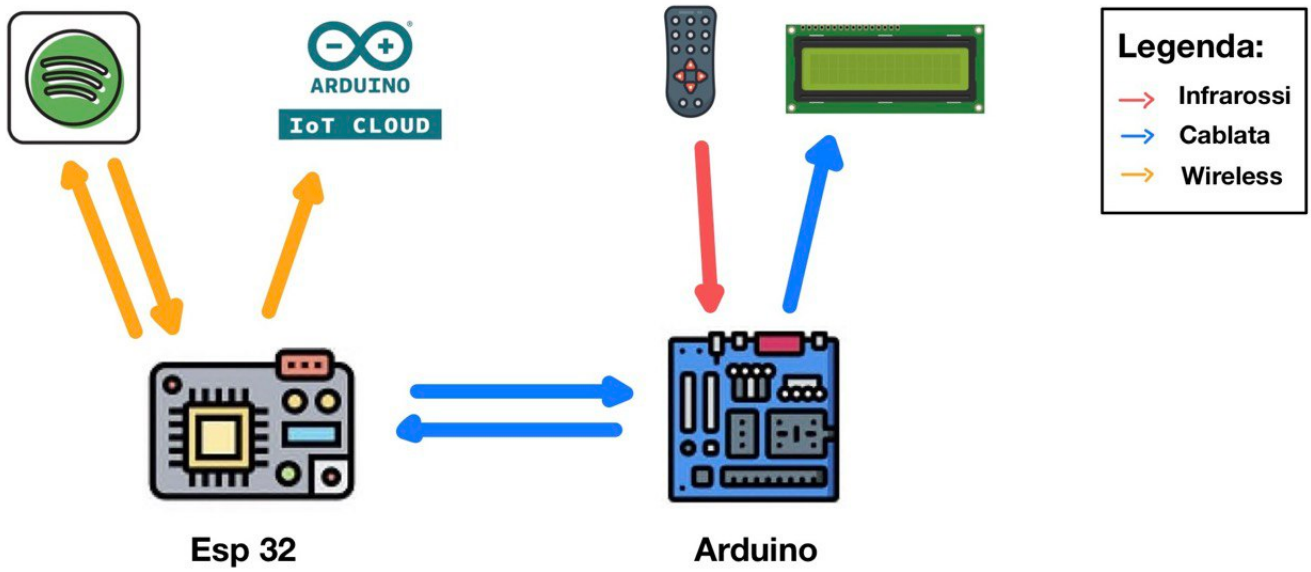

Immagine 1: Schema di Ardify

2 Componenti Utilizzati

Componenti	
Lato Arduino	Lato Esp32
Arduino Uno Schermo LCD 1602 Ricevitore IR Telecomando IR Cablaggio	Esp32 Cablaggio

3 Sviluppo

Durante il processo di sviluppo abbiamo provato varie soluzioni volte al raggiungimento del nostro obbiettivo. Una prima versione utilizzava uno script Python per la gestione delle API ma questo limitava l'utilità della seconda scheda e richiedeva un collegamento permanente tra scheda e pc. Alcune versioni successive utilizzavano rispettivamente due schede Arduino oppure due schede ESP32, ma a seguito di vari ragionamenti e considerazioni ci siamo diretti verso la soluzione attuale con una scheda Arduino e una scheda ESP32.

Come primi obbiettivi ci siamo concentrati sul funzionamento dello schermo LCD e sul ricevimento di comandi tramite il Telecomando. Per quanto riguarda lo schermo, dopo aver realizzato il circuito fisico, ci siamo interfecciati ad esso tramite la libreria LiquidCrystal.
Per la gestione del telecomando, abbiamo collegato un IR Receiver all'Arduino e tramite l'utilizzo della libreria IRremote e numerosi test abbiamo mappato i comandi a noi interessati. Una volta chiarito il funzionamento di queste componenti abbiamo rivolto la nostra attenzione verso l'interazione vera e propria con il player di Spotify.

Per poter utilizzare le API ufficiali è necessario disporre di un account Premium attivo e di un'applicazione attiva sulla dashboard degli sviluppatori di Spotify stesso. Quest'ultima fornirà i parametri necessari per svolgere le varie azioni. Le API funzionano tramite richieste GET e POST inviate tramite protocollo HTTP.

Successivamente, ci siamo concentrati sulla comunicazione tra le due schede.
Tra le varie soluzioni abbiamo provato comunicazioni Wireless e Bluetooth ma infine, per ragioni di comodità, abbiamo optato per una semplice comunicazione tramite seriale tx/rx.
Arduino, in base al comando ricevuto dal telecomando, invia determinati messaggi all'ESP32 che si occupa di effettuare la chiamata API opportuna.

Come ultima funzionalità abbiamo configurato una Dashboard sul sito di Arduino IOT Cloud.
Essa ci ha permesso, tramite apposite funzioni, di mostrare in Cloud determinate statistiche. Per poter permettere il salvataggio delle informazioni è necessario istanziare le variabili interessate nel sito e aggiungere varie procedure all'interno del codice.

Infine, per fornire una migliore esperienza, abbiamo svolto numerosi test sulla regolazione dei vari ritardi all'interno del codice per rendere l'applicazione la più reattiva possibile.
Il prodotto finale si presenta totalmente embedded e necessita solamente di una sorgente di corrente e di una connesione ad internet per poter funzionare, garantendo la portabilità del prodotto.

4 Funzionamento Codice e Librerie

Codice Arduino:

```
// SETUP
void setup()
{

  // code to enable the receiver that we created.
  irrecv.enableIRIn();

  // initializing the lcd
  lcd.begin(16, 2);

  // initializing the flags
  flag = 0;
  count = 1;

  // initializing the serial monitor.
  Serial.begin(9600);

  // changing the serial's timing to wait for stream data to 0.
  Serial.setTimeout(0);
}
```

Immagine 2: funzione di Setup

All'interno della funzione di Setup inizializziamo:

- la variabile per la lettura di informazioni dal telecomando;
- i pin dello schermo LCD;
- la porta seriale.

```
// MAIN-LOOP
void loop()
{

  delay(900);

  // updating the created variable to read the song from ESP32
  readSong = Serial.readString();
  if (readSong!=""){
    oldSong = readSong;
  }
  // setting the cursor on the 1602 LCD
  lcd.setCursor(0, 1);

  // print the song to the LCD display
  if (readSong==""){
    lcd.print(oldSong);
  }else {
    lcd.print(readSong);
  }


  delay(400);
```

Immagine 3: funzione di Loop

Ad ogni ciclo di loop, leggiamo il titolo della canzone da mostrare inviato dall'ESP32 tramite porta seriale e successivamente svolgiamo una serie di controlli sul suo valore per poi stampare l'informazione a schermo.

```
// creating an if-statement to decode the signal sent from the transmitter, which is the remote.
if (irrecv.decode())
{

  // withdrawing the exadecimal value sent by the remote.
  results.value = irrecv.decodedIRData.decodedRawData;

  // creating a switch to compare the value to multiple cases.
  switch (results.value)
  {

  // CASE: 1. represents the pause button.
  case 0xEA15FF00:

    // setting the cursor on the 1602 LCD
    lcd.setCursor(0, 0);

    // printing the keyword 'stop' to the serial monitor for recognition by the ESP32 script.
    Serial.println("stop");

    // checking if the song is being paused or resumed
    if (count % 2 == 0 && flag == 0)
    {
      // printing the 'Resume' string on the 1602 LCD.
      lcd.print("Resume");
    }
    else
    {
      // printing the 'Paused' string on the 1602 LCD.
      lcd.print("Paused");
      flag = 0;
    }

    count++;
    delay(250);
    break;
```

Immagine 4: funzione di Loop (cont.)

Successivamente tramite la funzione "decode" andiamo a leggere l'eventuale segnale inviato dal telecomando e in base al valore ricevuto andiamo a filtrare le varie operazioni. Grazie ad uno switch case siamo in grado di differenziare le varie operazioni ed inviare all'ESP32 istruzioni su quali API utilizzare. Il codice è speculare per le altre operazioni.

```
// creating a for-loop to add the scrolling effect.
for (int counter = 0; counter < 16; counter++)
{

  // scrolling one position to the right.
  lcd.scrollDisplayRight();

  // adding a delay to create an effect.
  delay(150);
}

// clearing the lcd after each iteration of the main-loop.
lcd.clear();
```

Immagine 5: funzione di Loop (cont.)

Infine aggiungiamo un effetto di scroll per poi pulire lo schermo.

Codice Esp32:

```
void setup()
{
  // Initializing variables
  volume_AUX = 0;
  count = 0;

  // Debug purposes Serial
  Serial.begin(9600);

  // Arduino/ESP32 Communication Serial (TX/RX)
  Serial2.begin(9600, SERIAL_8N1, 16, 17);

  // Connection to wifi connection
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
  }

  // Setting the Spotify Certificate
  client.setCACert(spotify_server_cert);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  // Arduino IOT Cloud Variables
  volume = 0;
  next_song = 0;
  previous_song = 0;
  device_name = "";
}
```

Immagine 6: funzione di Setup

Come prima cosa inizializziamo le due comunicazioni seriali per poi passare alla connessione del dispositivo alla rete. Tramite le funzioni della libreria "Wifi" ci colleghiamo alla rete wireless e successivamente chiamiamo alcuni metodi di configurazione della libreria "SpotifyArduino". Infine colleghiamo il dispositivo alla dashboard di Arduino Cloud corrispondente grazie alle funzioni definite all'interno di "thingProperties.h".

```
void loop()
{

  // Updating the Arduino Cloud Variables
  ArduinoCloud.update();

  // Spotify API Functions
  spotify.getCurrentlyPlaying(printCurrentlyPlayingToSerial, SPOTIFY_MARKET);
  spotify.getPlayerDetails(settingPlayerDetails, SPOTIFY_MARKET);

  // Setting the Arduino Cloud Volume to the API Volume, same for the device
  volume = volume_AUX;
  device_name = device;

  // Reading the command sent by the Arduino on the serial port
  command = Serial2.readString();
```

Immagine 7: funzione di Loop

Ad ogni ciclo di loop andiamo ad aggiornare i valori delle variabili in cloud per poi inviare sulla porta seriale il titolo della canzone corrente. Inoltre, tramite la funzione "getPlayerDetails" andiamo a salvarci in locale le informazioni necessarie per il funzionamento delle API successive.
In base al comando ricevuto tramite comunicazione seriale andiamo a svolgere l'operazione richiesta.

```
// If the command is Stop
if (command.indexOf("stop") >= 0)
{
  // Checking if the song is to be paused or resumed
  if (count % 2 == 0)
  {
    // Spotify API to pause the song
    if (!spotify.pause())
    {
      Serial.println("done!");
    }
  }
  else
  {
    // Spotify API to play the song
    if (!spotify.play())
    {
      Serial.println("done!");
    }
  }
  count++;
} // If the command is Next
```

Immagine 8: funzione di Loop (cont.)

Tramite un semplice contatore andiamo a fermare/riprendere la riproduzione della canzone.

```
else if (command.indexOf("next") >= 0)
{
  // Spotify API to play the next song
  if (!spotify.nextTrack())
  {
    // Updating the counter
    next_song++;
    Serial.println("done!");
  }
} // If the command is Back
else if (command.indexOf("back") >= 0)
{
  // Spotify API to go back to the previous song
  if (!spotify.previousTrack())
  {
    // Updating the counter
    previous_song++;
    Serial.println("done!");
  }
```

Immagine 9: funzione di Loop (cont.)
Gestione del salto alla canzone successiva e ritorno a canzone precedente. E' possibile notare anche l'aggiornamento dei valori delle variabili di Arduino Cloud.

```
else if (command.indexOf("up") >= 0)
{
  // Getting the current volume
  spotify.getPlayerDetails(settingPlayerDetails, SPOTIFY_MARKET);
  // Spotify API to set the volume of the player
  if (!spotify.setVolume(volume_AUX + 10 /*10% more*/))
  {
    Serial.println("done!");
  }
} // If the command is Down
else if (command.indexOf("down") >= 0)
{
  // Getting the current volume
  spotify.getPlayerDetails(settingPlayerDetails, SPOTIFY_MARKET);
  // Spotify API to set the volume of the player
  if (!spotify.setVolume(volume_AUX - 10 /*10% less*/))
  {
    Serial.println("done!");
  }
}
```

Immagine 10: funzione di Loop (cont.)
Per quanto riguarda la gestione del volume, utilizzando le informazioni salvate in precedenza andiamo ad aumentare o diminuire la percentuale del volume di un 10%.

```
void printCurrentlyPlayingToSerial(CurrentlyPlaying currentlyPlaying)
{
  // Sending to Arduino the current TrackName
  Serial2.print(currentlyPlaying.trackName);
}
```

Immagine 11: funzione ausiliaria
Funzione che stampa sulla porta seriale condivisa con l'Arduino il titolo della canzone corrente.

```
void settingPlayerDetails(PlayerDetails playerDetails)
{
  // Setting the device name and the current Volume
  device = playerDetails.device.name;
  volume_AUX = playerDetails.device.volumePercent;
}
```

Immagine 12: funzione ausiliaria
Funzione che salva in variabili locali il nome dell'attuale device in ascolto e la percentuale del volume.

Librerie utilizzate:

- **LiquidCrystal** : per la gestione dello schermo LCD
- **IRremote** : per la gestione del ricevitore ad infrarossi
- **Wifi** : per poter connettere l'esp32 a internet
- **SpotifyArduino** : per gestire il player di Spotify tramite API
- **SoftwareSerial** : per aprire una comunicazione seriale sui pin TX e RX
- **ArduinoJson** : per la gestione dei valori di ritorno delle API
- **thingProperties** : per gestire l'aggiornamento delle variabili in cloud

5 Schema Collegamenti

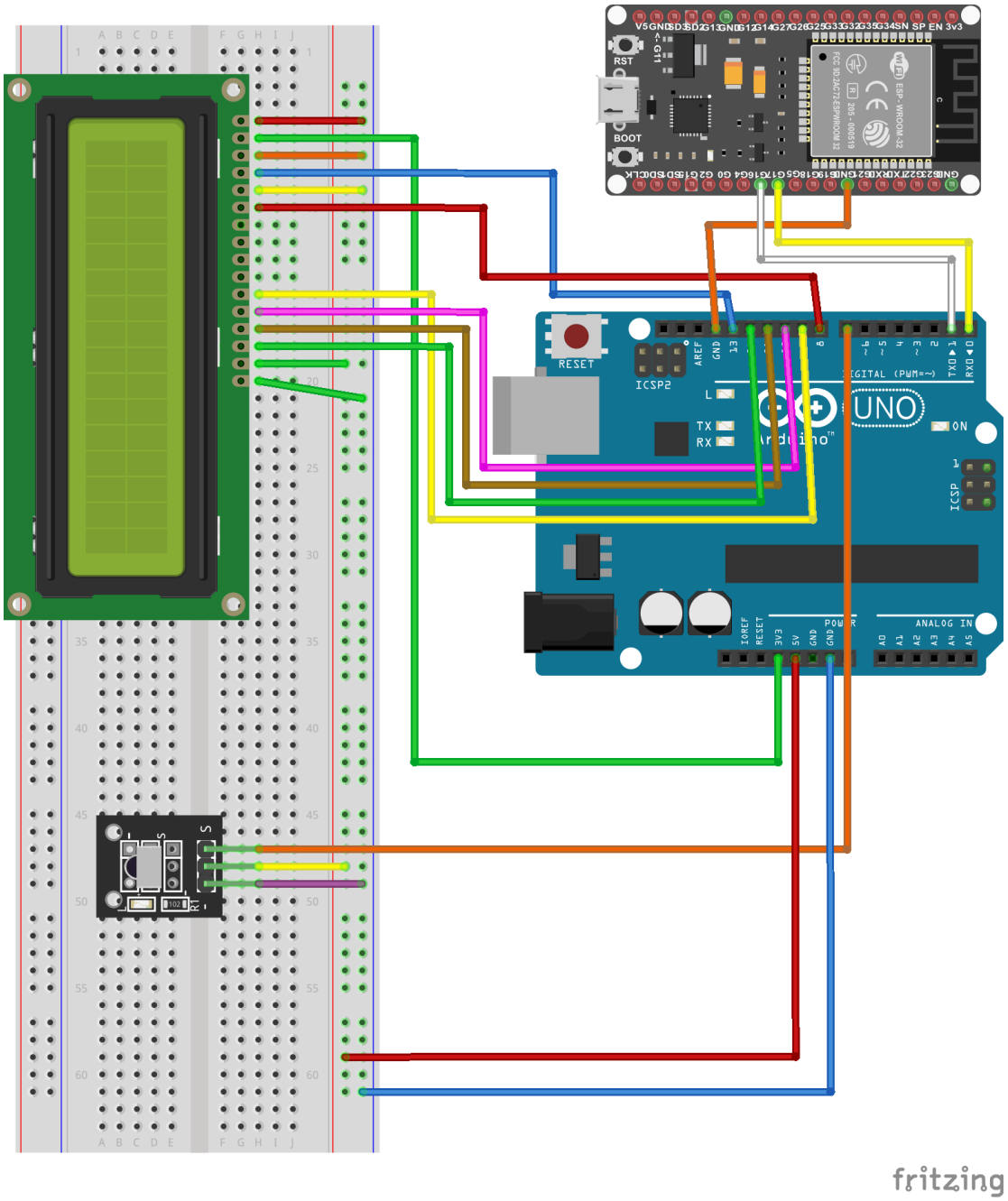

Immagine 12: Schema dei collegamenti del circuito.

6 Realizzazione Fisica

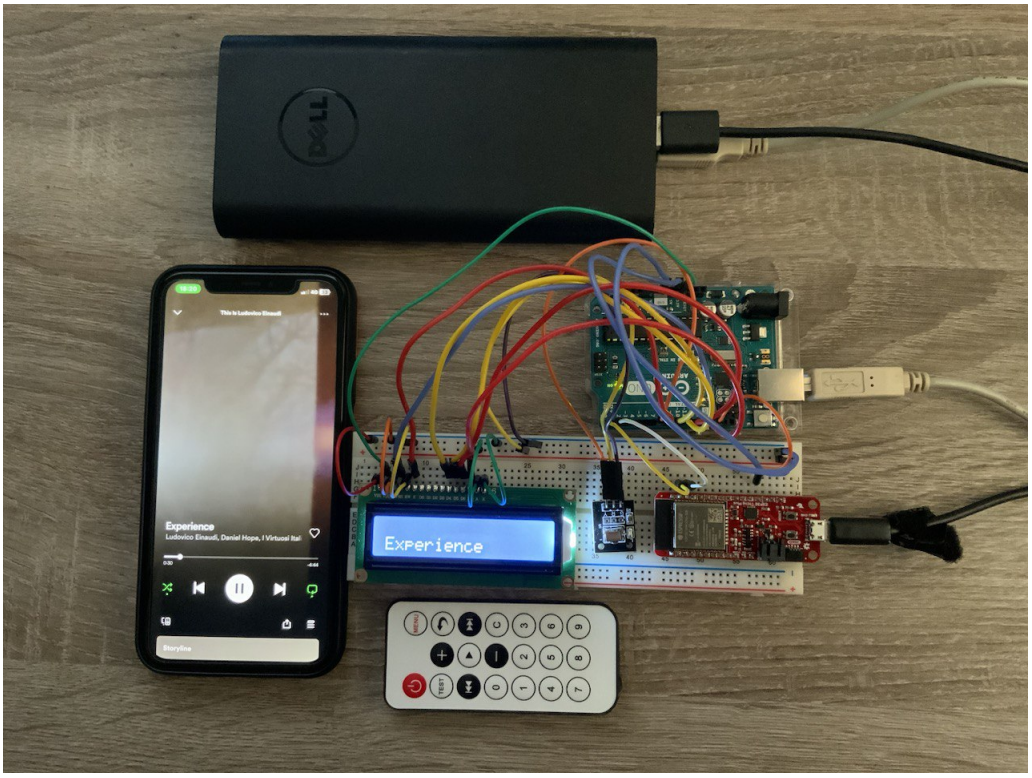

Immagine 13: Foto progetto.

Immagine 14: Azione di "Skip" a prossima canzone.

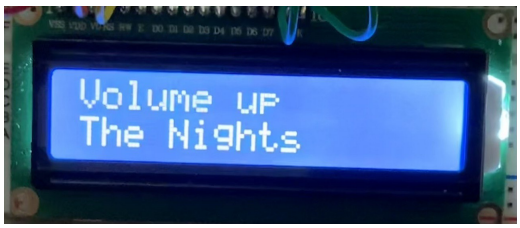

Immagine 15: Azione di "Volume Up" a canzone corrente.

7 Bibliografia

Per la gestione delle API di Spotify ci siamo appoggiati ad una libreria già esistente:
`https://github.com/witnessmenow/spotify-api-arduino`

Per il funzionamento e la gestione dello schermo LCD ci siamo appoggiati alla documentazione ufficiale:
`https://buildmedia.readthedocs.org/media/pdf/arduinoliquidcrystal/latest/arduinoliquidcrystal.pdf`

Per l'utilizzo della libreria IRreceiver ci siamo appoggiati a vari tutorial tra cui:
`https://www.pjrc.com/teensy/td_libs_IRremote.html`

Per il collegamento e l'interfaccia ad Arduino Cloud abbiamo principalmente seguito questa guida:
`https://docs.arduino.cc/arduino-cloud/getting-started/esp-32-cloud`