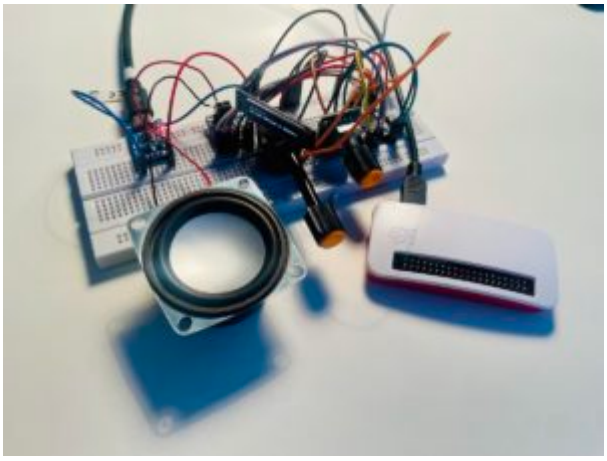
Update fürs ESP32 Internetradio: Songs in Spotify speichern



Hast du schon einmal einen Song im Radio gehört, den du dir merken wolltest, damit du ihn später auf Spotify hören kannst? Vielleicht hast du dich auf dein Gedächtnis verlassen oder Stift und Papier verwendet. Aber es geht auch eleganter:

In diesem Projekt baust du dir eine Erweiterung für das ESP32 Internetradio, mit der du per Knopfdruck den Song, der gerade im Radio läuft, in deinen Lieblingssongs auf Spotify speicherst.

Für dieses Update benötigst du:

- Raspberry Pi Zero W* oder einen Standard-Raspi plus Micro-SD-Karte

Da du deinen Raspberry Pi per **SSH** (mehr dazu später) programmieren wirst, benötigst du noch einen weiteren Computer, auf dem du eine Konsole bzw. Terminal verwenden kannst. Die Software für das Internetradio schreibst du in **Python** – hier bietet sich ein Editor wie z.B. Visual Studio Code an. Es reicht aber auch ein einfacher Texteditor. Die SD-

Karte mit dem Betriebssystem für den Raspberry Pi erstellst du am besten mit dem kostenlosen Tool **Raspberry Pi Imager**.

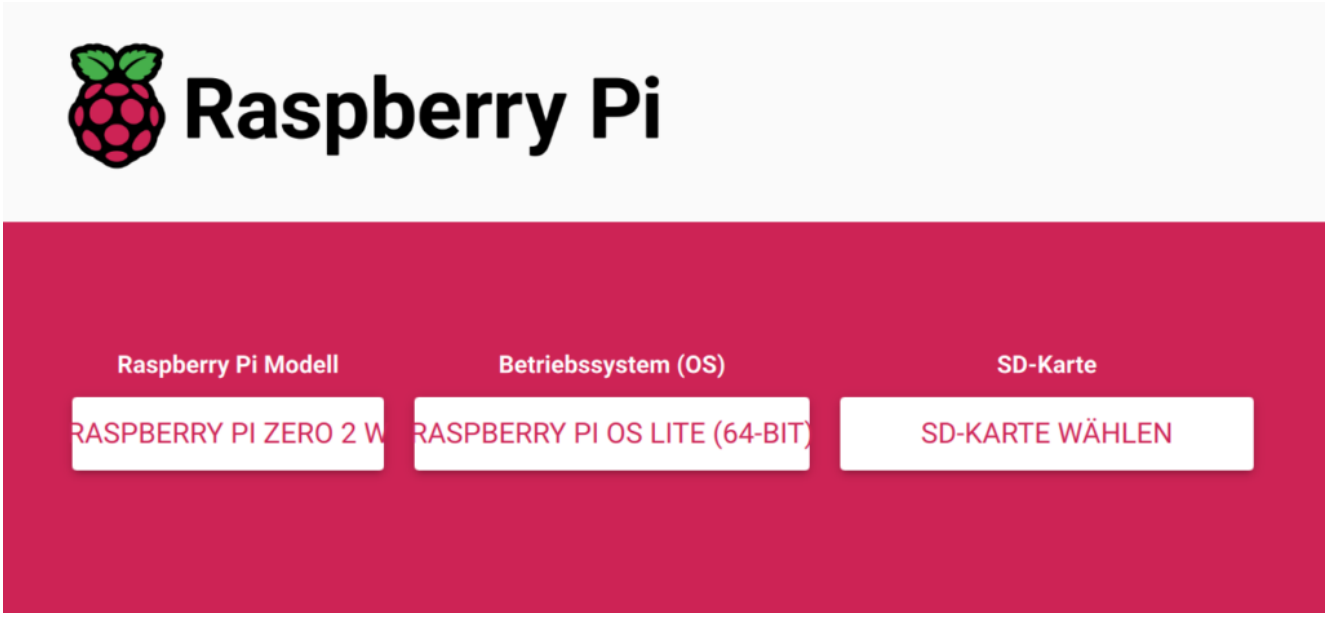
Den Raspberry Pi vorbereiten

Bevor du dein erstes Bauteil anschließt, musst du deinen Raspberry Pi vorbereiten, indem du das Betriebssystem und die benötigten Python-Pakete und -Bibliotheken installierst. Lass uns das Schritt für Schritt durchgehen:

Das Betriebssystem installieren

Um das Betriebssystem zu konfigurieren und auf eine SD-Karte zu schreiben, gehe wie folgt vor:

- Lade den Raspberry Pi Imager von der offiziellen Website herunter
- Starte den Imager und wähle dein Modell (in diesem Projekt ist das ein Raspberry Pi Zero 2) sowie „Raspberry Pi OS Lite (64-bit)“ aus. Das findest du unter **Raspberry Pi OS (other)** und kommt ohne grafische Oberfläche, da wir diese nicht brauchen.
- Wähle deine SD-Karte als Ziel



- Klicke im nächsten Screen auf **Einstellungen bearbeiten** und
    - Setze einen Benutzernamen und Passwort
    - Konfiguriere dein WLAN (SSID und Passwort)
    - Aktiviere SSH im Reiter **Dienste** (Passwort zur Authentifizierung verwenden)

- Übernimm deine Einstellungen mit einem Klick auf **Ja** und schreibe das Image auf die SD-Karte

Verbinde dich per SSH mit dem Raspberry Pi

Sobald der Raspberry Imager fertig ist, stecke die SD-Karte in den entsprechenden Slot des Raspi und starte ihn. Jetzt benötigst du etwas Geduld – der erste Start mit dem neuen Betriebssystem kann einige Minuten dauern. Öffne auf deinem Computer das Terminal bzw. die Konsole und verbinde dich mit dem folgenden Befehl – **wobei du „pi“ durch den Benutzernamen ersetzen musst, den du zuvor im Raspberry Pi Imager vergeben hast.**

```
sudo ssh pi@raspberrypi.local
```

Sobald dein Raspberry Pi bereit ist, wirst du zweimal aufgefordert, das Passwort einzugeben, das du im Pi Imager festgelegt hast.

Installiere die benötigten Pakete

Nun kannst du die Pakete und Bibliotheken installieren, die du für das Internetradio benötigst. Doch zunächst kümmerst du

dich um das Update des Betriebssystems:

```
sudo apt update
sudo apt upgrade
```

Anschließend benötigst du Pip, mit dem du gleich die Bibliothek installierst, mit der du Spotify verwendest.

```
sudo apt install python3-pip
```

Eine virtuelle Umgebung einrichten

Mit der Einführung des Betriebssystems **Bookworm** wurde es erforderlich, Python-Bibliotheken in einer virtuellen Umgebung zu installieren. Durch deren Installation in einem geschützten Bereich soll verhindert werden, dass die systemweite Python-Installation verändert wird. Um eine virtuelle Umgebung einzurichten, verwende diese Befehle:

```
sudo apt install python3-venv
python3 -m venv RadioSpotify
```

Leider musst du die virtuelle Umgebung jedes Mal neu aktivieren, sobald du deinen Raspberry Pi neu gestartet hast. Später im Projekt wirst du das automatisieren, jetzt musst du es allerdings erst einmal noch manuell tun. Das geht mit diesem Befehl:

```
source RadioSpotify/bin/activate
```

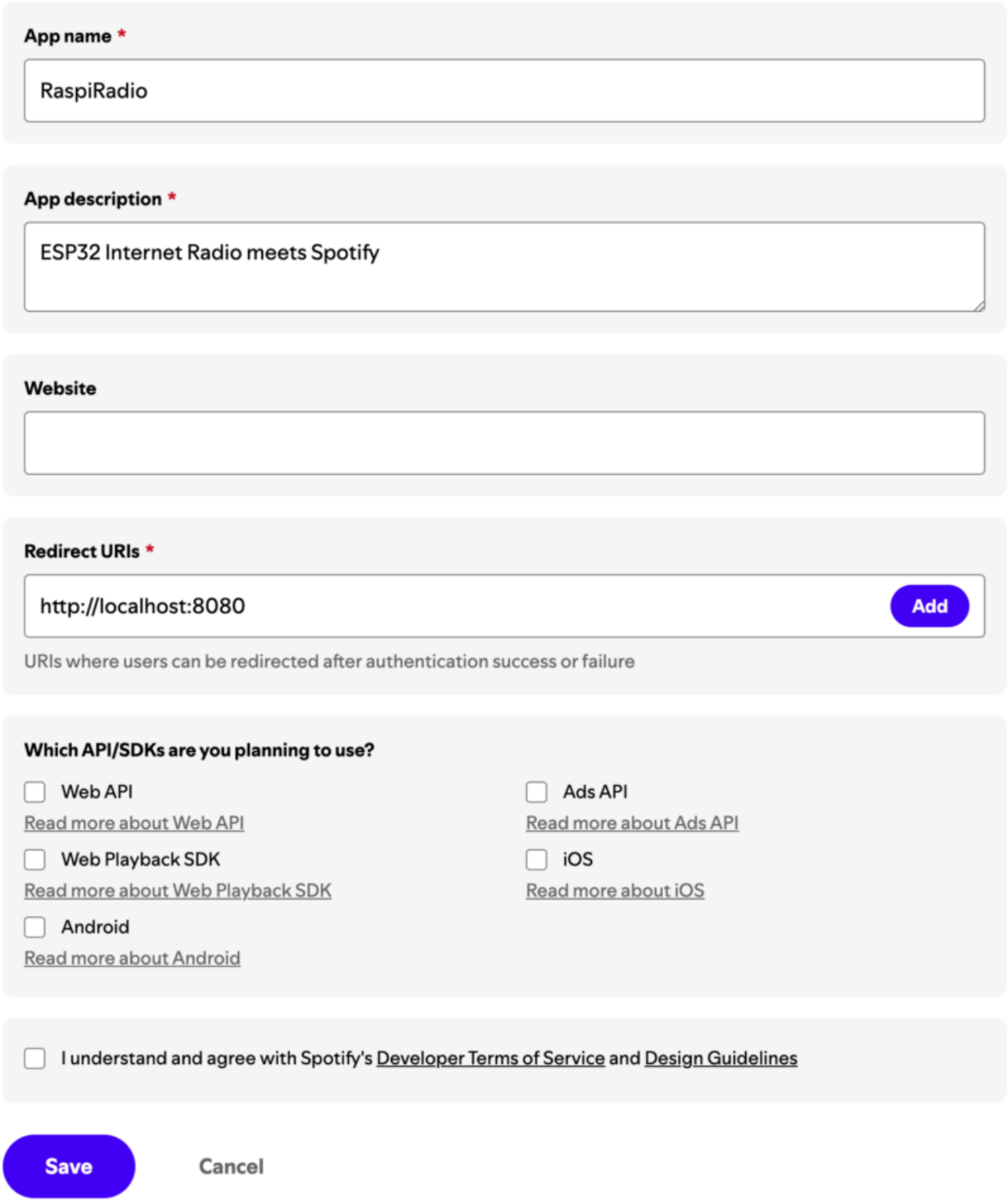
Übrigens, deaktivieren kannst du sie einfach mit dem Befehl **deactivate**. Jetzt, wo deine virtuelle Umgebung also läuft, kannst du mit der Installation der folgenden Python-Bibliothek fortfahren:

```
pip3 install spotipy
```

Spotify vorbereiten

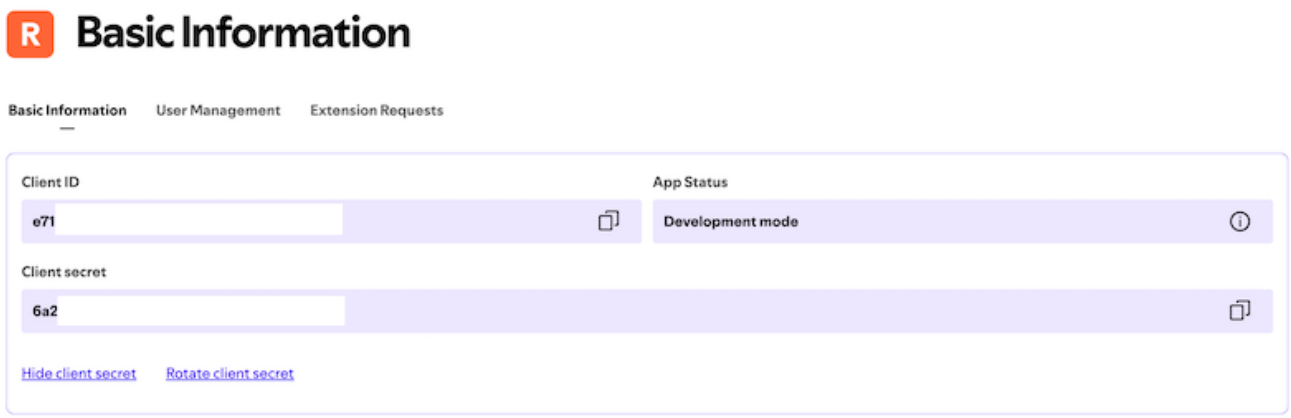
Damit dein ESP32 Internetradio bzw. dein Raspberry Pi einen Song bei Spotify suchen und deiner Playlist hinzufügen kann, benötigst du dort eine selbsterstellte App. Diese ist schnell eingerichtet:

- Erstelle ein kostenloses Entwickler-Konto auf https://developer.spotify.com
- Erstelle eine neue App im Spotify Developer Dashboard:



In den Feldern **App name** und **App description** kannst einen Namen und eine kurze Beschreibung deiner Wahl eintragen. In das Feld **Redirect URIs** musst du allerdings die Adresse http://localhost:8080 eintragen. Klicke anschließend auf **Save**.

Klicke nun auf dem nächsten Screen auf **Settings**. Dort findest du deine **Client ID** und **Client Secret**. Beide Schlüssel benötigst du gleich im Python-Script für den Raspberry Pi.



Und das war es auch schon an dieser Stelle. Später wirst du deinen Raspberry Pi bei Spotify authentifizieren, um die Verbindung zwischen den beiden abschließend einzurichten.

Das Python-Script für das Radio

Der Code für das Raspberry Pi Internetradio ist der folgende. Kopiere ihn und erstelle ein neues Script namens **radiospotify.py**

___STEADY_PAYWALL___

```
import sys
import os
import fcntl
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import urlparse, parse_qs
import time
import json
import spotipy
from spotipy.oauth2 import SpotifyOAuth
import logging

from http.server import HTTPServer, BaseHTTPRequestHandler
import webbrowser
```

```
# Set up logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -
%(levelname)s - %(message)s')
# Spotify API credentials
SPOTIPY_CLIENT_ID = 'deine_client_id'
SPOTIPY_CLIENT_SECRET = 'dein_client_secret'
SPOTIPY_REDIRECT_URI = 'http://localhost:8080'
SCOPE = 'user-library-modify'
# Define cache path in user's home directory
CACHE_PATH = os.path.expanduser('~/.spotify_token_cache')
# Global variable to store the authentication code
auth_code = None
class AuthHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        global auth_code
        query_components = parse_qs(urlparse(self.path).query)
        if "code" in query_components:
            auth_code = query_components["code"][0]
            self.send_response(200)
            self.send_header('Content-type', 'text/html')
            self.end_headers()
             self.wfile.write(b"Authentication successful! You
can close this window.")
            logging.info("Received authentication code")
        else:
            self.send_response(400)
            self.send_header('Content-type', 'text/html')
            self.end_headers()
             self.wfile.write(b"Authentication failed! No code
received.")
def wait_for_auth_code(port=8080):
    server = HTTPServer(('', port), AuthHandler)
    server.handle_request()  # Handle one request then close
    return auth_code
def initialize_spotify():
    global auth_code
    auth_manager = SpotifyOAuth(
        client_id=SPOTIPY_CLIENT_ID,
        client_secret=SPOTIPY_CLIENT_SECRET,
        redirect_uri=SPOTIPY_REDIRECT_URI,
        scope=SCOPE,
```

```
        cache_path=CACHE_PATH,
        open_browser=False
    )
    # Try to get cached token
    token_info = auth_manager.get_cached_token()
                        if     not     token_info     or
auth_manager.is_token_expired(token_info):
        auth_url = auth_manager.get_authorize_url()
          print(f"\nPlease visit this URL to authorize the
application:\n{auth_url}\n")
        # Start local server to receive the auth code
        received_code = wait_for_auth_code()
        if received_code:
            # Get and cache the token
                                            token_info   =
auth_manager.get_access_token(received_code)
             logging.info("New authentication token obtained
and cached")
        else:
            logging.error("No authentication code received")
            return None
    return spotipy.Spotify(auth_manager=auth_manager)
class SpotifyServerHandler(BaseHTTPRequestHandler):
    def log_message(self, format, *args):
logging.info(f"{self.client_address[0]}:{self.client_address[1
]} - {format%args}")
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.end_headers()
        parsed_path = urlparse(self.path)
        params = parse_qs(parsed_path.query)
          logging.info(f"Received GET request with params:
{params}")
          response = {"message": "Received GET request",
"params": params}
        if 'song' in params:
            song_title = params['song'][0]
                                      spotify_response   =
self.save_to_spotify(song_title)
            response.update(spotify_response)
```

```
        self.wfile.write(json.dumps(response).encode())
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
                                            post_data     =
self.rfile.read(content_length).decode('utf-8')
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.end_headers()
        logging.info(f"Received POST data: {post_data}")
        params = parse_qs(post_data)
          response = {"message": "Received POST request",
"data": params}
        if 'song' in params:
            song_title = params['song'][0]
                                       spotify_response   =
self.save_to_spotify(song_title)
            response.update(spotify_response)
        self.wfile.write(json.dumps(response).encode())
    def save_to_spotify(self, song_title):
        global sp
        if sp is None:
            logging.error("Spotify client is not initialized")
            return {
                "spotify_status": "error",
                "message": "Spotify client is not initialized"
            }
        logging.info(f"Attempting to save song: {song_title}")
        try:
            # Search for the track
             results = sp.search(q=song_title, type='track',
limit=1)
            if results['tracks']['items']:
                track = results['tracks']['items'][0]
                # Save the track to the user's library
sp.current_user_saved_tracks_add(tracks=[track['id']])
                  logging.info(f"Successfully saved track:
{track['name']} by {track['artists'][0]['name']}")
                return {
                    "spotify_status": "success",
                      "saved_track": f"{track['name']} by
{track['artists'][0]['name']}"
```

```
                }
            else:
                logging.warning(f"Track not found on Spotify:
{song_title}")
                return {
                    "spotify_status": "not_found",
                    "message": f"Track not found on Spotify:
{song_title}"
                }
        except Exception as e:
            logging.error(f"An error occurred while saving to
Spotify: {e}")
            return {
                "spotify_status": "error",
                "message": f"An error occurred: {str(e)}"
            }
def run_server(port=8080):
    server_address = ('', port)
    try:
                httpd = HTTPServer(server_address,
SpotifyServerHandler)
        logging.info(f"Server running on port {port}")
        httpd.serve_forever()
    except OSError as e:
        if e.errno == 98:
            logging.error(f"Error: Port {port} is already in
use. Try a different port.")
        else:
            logging.error(f"Error: {e}")
        sys.exit(1)
if __name__ == '__main__':
    lock_file = '/tmp/spotify_server.lock'
    try:
        lock_handle = open(lock_file, 'w')
            fcntl.lockf(lock_handle, fcntl.LOCK_EX |
fcntl.LOCK_NB)
    except IOError:
        logging.error("Another instance of this script is
already running.")
        sys.exit(1)
    try:
```

```
        # Initialize Spotify client
        sp = initialize_spotify()
        if not sp:
              logging.error("Failed to initialize Spotify
client")
            sys.exit(1)
        port = int(sys.argv[1]) if len(sys.argv) > 1 else 8080
        run_server(port)
    except KeyboardInterrupt:
        logging.info("\nServer stopped.")
    finally:
        fcntl.lockf(lock_handle, fcntl.LOCK_UN)
        lock_handle.close()
        os.unlink(lock_file)
```

Darin musst du oben die folgenden zwei Schlüssel hinterlegen – diese findest du in deinem Spotify Developer Account.

```
SPOTIPY_CLIENT_ID = 'deine_client_id'
SPOTIPY_CLIENT_SECRET = 'dein_client_secret'
```

Und das war es schon mit den Anpassungen im Python Script. Falls du im Terminal noch auf deinem Rasperry Pi eingeloggt bist, logge dich mit **logout** aus und bewege dich in den Ordner, in dem dein gerade erstelltes Python Script liegt. Falls du unsicher bist, wie das geht, wirf einen Blick in die Tipps der Uni Düsseldorf.

Im Verzeichnis angekommen führe folgenden Befehl aus:

```
scp                                    radiospotify.py
pi@raspberrypi.local:/home/pi/RadioSpotify/
```

Den Code ausführen

Jetzt, wo der Code auf deinem Raspberry Pi ist, kannst du ihn aufrufen. Logge dich dafür zunächst wieder per SSH ein (ersetze **pi** wieder durch deinen Benutzernamen):

```
sudo ssh pi@raspberrypi.local
```

Anschließend kannst du das Script wie folgt starten:

```
source RadioSpotify/bin/activate
python3 RadioSpotify/radiospotify.py
```

Als nächstes musst deinen Rasperry Pi bei Spotify Zugriff auf dein dortiges Konto geben. **Das ist etwas umständlich – aber du musst es zum Glück nur einmal machen.** Der entsprechende Token wird auf deinem Raspberry Pi gespeichert, sodass du dich beim nächsten Start des Scripts nicht noch einmal authentifizieren musst.

Im Terminal wirst du nach dem Start des Scripts aufgefordert, eine Adresse im Browser zu öffnen – das kannst du in einem Browser deiner Wahl auf deinem Computer machen. Du wirst dort von Spotify gefragt, ob es die Verbindung zu deiner App herstellen darf – stimme dem zu. Daraufhin solltest du zwar im Browser etwas in der Art von „Webseite nicht erreichbar“ sehen – in der Ardresszeile jedoch eine andere URL stehen haben.

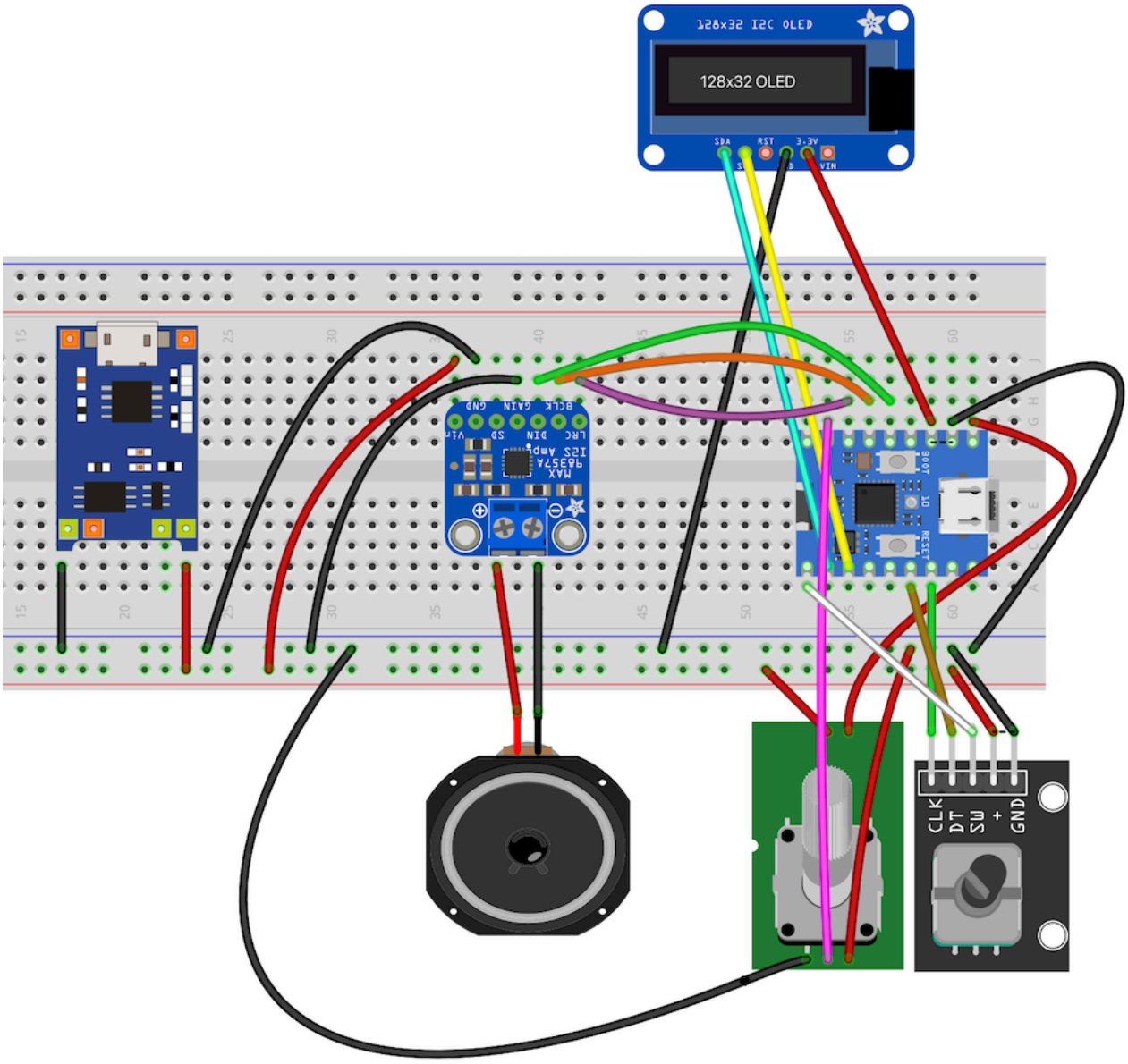
Kopiere **die gesamte neue URL** aus der Adresszeile, **öffne ein neues Terminal-Fenster auf deinem Computer**, logge dich auch darin per SSH auf deinem Raspi ein und verwende den folgenden Befehl:

```
curl "DEINE KOPIERTE URL"
```

Nun sollte in diesem Terminal-Fenster die Info **Authentication successful!** und in deinem ersten Fenster die Zeile **INFO – Server running on port 8080** erscheinen. Das bedeutet, dass der Login bei Spotify funktioniert hat und **dein Raspberry Pi nun bereit ist, Songs vom ESP32 Internetradio zu empfangen und in deiner Spotify-Playlist zu speichern.** Weiter geht es mit deinem ESP32.

Ein weiteres Kabel am ESP32

Damit du Songs an Spotify senden kannst, benötigst du einen Button, mit dem diese Funktion auslöst. Praktischerweise bringt dein Rotary Encode schon einen mit – du kannst diesen nicht nur drehen, sondern auch hörbar eindrücken. Dieser Druck wird am Pin **SW** ausgelesen. Verbinde deshalb diesen Pin mit dem GPIO 7 deines ESP32 – wenn du einen ESP32-S3 Zero verwendest, sieht die neue Verbindung so aus (das weiße Kabel ist die Verbindung vom Button zum ESP32):



Du kannst natürlich auch einen anderen Pin des ESP32 verwenden, musst das dann allerdings entsprechend im folgenden Sketch ändern.

Aktualisiere den Sketch auf dem ESP32

In deinem ESP32 Internetradio fehlt nun nur noch die Funktion, den aktuellen Song zum Raspberry Pi zu senden und ihn von diesem in deinen Spotify-Lieblingssongs zu speichern. Hierfür ist ein Umbau nötig – hier der aktualisierte Sketch:

```
#include <Arduino.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Audio.h>
#include <AiEsp32RotaryEncoder.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ArduinoJson.h>

// Pin definitions bleiben unverändert
#define I2S_DOUT 2
#define I2S_BCLK 3
#define I2S_LRC 4
#define VOLUME_PIN 5

#define ROTARY_ENCODER_A_PIN 12
#define ROTARY_ENCODER_B_PIN 13
#define ROTARY_ENCODER_BUTTON_PIN 7
#define ROTARY_ENCODER_STEPS 4

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define OLED_RESET     -1
#define SCREEN_ADDRESS 0x3C

#define I2C_SDA 8
#define I2C_SCL 9
```

```
// Debug Level für ESP32
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE

// Watchdog Timeout
const int wdtTimeout = 5000;  // 5 Sekunden Watchdog Timeout

// Initialisierungs-Flags
bool isWiFiConnected = false;
bool isDisplayInitialized = false;
bool isAudioInitialized = false;

AiEsp32RotaryEncoder   rotaryEncoder(ROTARY_ENCODER_A_PIN,
ROTARY_ENCODER_B_PIN,   ROTARY_ENCODER_BUTTON_PIN,   -1,
ROTARY_ENCODER_STEPS);
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

Audio audio;

// WiFi credentials
const char ssid[] = "DEIN WLAN-NETZWERK";
const char password[] = "DEIN WLAN-PASSWORT";

// Spotify server details
const char* serverName = "IP-ADRESSE:8080";

// Radio stations bleiben unverändert
const char* stations[] = {
    "http://www.byte.fm/stream/bytefm.m3u",
    "https://st01.sslstream.dlf.de/dlf/01/128/mp3/stream.mp3",
"https://frontend.streamonkey.net/fho-schwarzwaldradiolive/mp3
-stream.m3u",
    "https://kexp-mp3-128.streamguys1.com/kexp128.mp3",
"https://eagle.streemlion.com:2199/tunein/psychedelicj.asx"
};
const char* stationNames[] = {
    "Byte.fm",
    "Deutschlandfunk",
    "Schwarzwaldradio",
    "KEXP",
    "Psychedelic Jukebox"
```

```
};
const int NUM_STATIONS = sizeof(stations) / sizeof(stations[0]);
int currentStation = 0;

// Statische Buffer statt dynamischer Strings
char streamTitle[64] = "";
char urlBuffer[256] = "";
char jsonBuffer[512] = "";

// Volume control variables bleiben unverändert
const int SAMPLES = 5;
int volumeReadings[SAMPLES];
int readIndex = 0;
int total = 0;
int average = 0;
unsigned long lastVolumeCheck = 0;
const unsigned long VOLUME_CHECK_INTERVAL = 500;

void IRAM_ATTR readEncoderISR() {
    rotaryEncoder.readEncoder_ISR();
}

// Optimierte String-Ersetzungsfunktion mit statischem Buffer
void replaceSpecialChars(const char* input, char* output, size_t outputSize) {
    size_t i = 0, j = 0;
    while (input[i] && j < outputSize - 1) {
        char c = input[i++];
        switch (c) {
            case 'ä': memcpy(&output[j], "a", 1); j += 1; break;
            case 'ö': memcpy(&output[j], "o", 1); j += 1; break;
            case 'ü': memcpy(&output[j], "u", 1); j += 1; break;
            case 'Ä': memcpy(&output[j], "A", 1); j += 1; break;
            case 'Ö': memcpy(&output[j], "O", 1); j += 1; break;
            case 'Ü': memcpy(&output[j], "U", 1); j += 1;
```

```
break;
             case 'ß': memcpy(&output[j], "ss", 2); j += 2;
break;
            default: output[j++] = c;
        }
    }
    output[j] = '\0';
}

void setup() {
    Serial.begin(115200);
    // Debug Level setzen
    esp_log_level_set("*", ESP_LOG_VERBOSE);
    Serial.println(F("ESP32-S3 Internet Radio starting..."));
      Serial.printf("Initial free heap: %d bytes\n",
ESP.getFreeHeap());
    // Basis-Setup
    pinMode(VOLUME_PIN, INPUT);
    // Encoder Setup
    rotaryEncoder.begin();
    rotaryEncoder.setup(readEncoderISR);
    rotaryEncoder.setBoundaries(0, NUM_STATIONS - 1, true);
    rotaryEncoder.setAcceleration(0);
    // Volume readings initialisieren
    for (int i = 0; i < SAMPLES; i++) {
        volumeReadings[i] = 0;
    }
    // Wire begin - grundlegende I2C-Initialisierung
    Wire.begin(I2C_SDA, I2C_SCL);
}

void loop() {
    static unsigned long lastInitAttempt = 0;
    const unsigned long initInterval = 5000;
    // Heap-Überwachung
    static unsigned long lastHeapCheck = 0;
     if (millis() - lastHeapCheck > 10000) {  // Alle 10
Sekunden
             Serial.printf("Free heap: %d bytes\n",
ESP.getFreeHeap());
        lastHeapCheck = millis();
```

```
    }
    // Verzögerte Initialisierung
    if (!isDisplayInitialized && millis() - lastInitAttempt >
initInterval) {
        initializeDisplay();
        lastInitAttempt = millis();
    }
     if (!isWiFiConnected && millis() - lastInitAttempt >
initInterval) {
        connectToWiFi();
        lastInitAttempt = millis();
    }
    if (isWiFiConnected && !isAudioInitialized && millis() -
lastInitAttempt > initInterval) {
        initializeAudio();
        lastInitAttempt = millis();
    }
      // Normale Loop-Funktionalität nur wenn alles
initialisiert ist
      if (isDisplayInitialized && isWiFiConnected &&
isAudioInitialized) {
        audio.loop();
        yield();
        checkEncoder();
        yield();
        checkVolumeControl();
        yield();
    }
    delay(10);  // Kleine Pause für Stabilität
}

void initializeDisplay() {
    Serial.println(F("Initializing OLED display..."));
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD1306 initialization failed"));
        return;  // Statt Endlosschleife einfach zurückkehren
    }
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0,0);
```

```
    display.println(F("Initializing..."));
    display.display();
    isDisplayInitialized = true;
    Serial.println(F("Display initialized successfully"));
}

void connectToWiFi() {
    Serial.println(F("Connecting to WiFi..."));
    WiFi.begin(ssid, password);
    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 20) {
        delay(500);
        Serial.print(".");
        attempts++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        Serial.println(F("\nWiFi connected"));
        isWiFiConnected = true;
        if (isDisplayInitialized) {
            display.clearDisplay();
            display.setCursor(0,0);
            display.println(F("WiFi connected"));
            display.display();
        }
    } else {
        Serial.println(F("\nWiFi connection failed"));
    }
}

void initializeAudio() {
    audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
    audio.setVolume(10);
    connectToStation(currentStation);
    isAudioInitialized = true;
    Serial.println(F("Audio initialized"));
}

void checkEncoder() {
    if (rotaryEncoder.encoderChanged()) {
        currentStation = rotaryEncoder.readEncoder();
        connectToStation(currentStation);
```

```
    }
    if (rotaryEncoder.isEncoderButtonClicked()) {
        Serial.println(F("Encoder button clicked"));
        sendToSpotify();
    }
}

void connectToStation(int stationIndex) {
    audio.stopSong();
    audio.connecttohost(stations[stationIndex]);
    updateDisplay();
}

void checkVolumeControl() {
    unsigned long currentMillis = millis();
          if  (currentMillis  -  lastVolumeCheck  >=
VOLUME_CHECK_INTERVAL) {
        lastVolumeCheck = currentMillis;
        total = total - volumeReadings[readIndex];
        volumeReadings[readIndex] = analogRead(VOLUME_PIN);
        total = total + volumeReadings[readIndex];
        readIndex = (readIndex + 1) % SAMPLES;
        average = total / SAMPLES;
        int volume = map(average, 0, 4095, 5, 23);
        static int lastVolume = -1;
        if (volume != lastVolume) {
            audio.setVolume(volume);
            lastVolume = volume;
            updateDisplay();
        }
    }
}

void updateDisplay() {
    if (!isDisplayInitialized) return;
    display.clearDisplay();
    display.setCursor(0,0);
    char buffer[64];
     replaceSpecialChars(stationNames[currentStation], buffer,
sizeof(buffer));
    display.println(buffer);
```

```
    display.println();
    replaceSpecialChars(streamTitle, buffer, sizeof(buffer));
    display.println(buffer);
    display.display();
}

// Optimierte URL-Encoding Funktion mit statischem Buffer
void urlEncode(const char* input, char* output, size_t
outputSize) {
    size_t j = 0;
    for (size_t i = 0; input[i] && j < outputSize - 4; i++) {
        char c = input[i];
        if (isalnum(c)) {
            output[j++] = c;
        } else if (c == ' ') {
            output[j++] = '+';
        } else {
            if (j + 3 >= outputSize) break;
            sprintf(&output[j], "%%%02X", c);
            j += 3;
        }
          yield();  // Watchdog füttern während langer
Operationen
    }
    output[j] = '\0';
}

void sendToSpotify() {
    static unsigned long lastRequestTime = 0;
    unsigned long currentTime = millis();
    if (currentTime - lastRequestTime < 5000) {
        Serial.println(F("Request blocked: Too soon since last
request"));
        return;
    }
    lastRequestTime = currentTime;
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        // URL-Encoding mit statischem Buffer
        char encodedTitle[128];
                urlEncode(streamTitle,  encodedTitle,
```

```
sizeof(encodedTitle));
        // URL zusammenbauen mit snprintf
         snprintf(urlBuffer, sizeof(urlBuffer), "%s/?song=%s",
serverName, encodedTitle);
        Serial.println(F("--------- New Request ---------"));
        Serial.print(F("Request URL: "));
        Serial.println(urlBuffer);
        http.begin(urlBuffer);
        http.setTimeout(5000);  // 5 Sekunden Timeout
                  http.setReuse(false);       //  Keine
Verbindungswiederverwendung
        yield();  // Watchdog füttern
        int httpResponseCode = http.GET();
        if (httpResponseCode > 0) {
            // Statisches JSON-Dokument
            StaticJsonDocument<512> doc;
            // Response lesen
            String payload = http.getString();
            yield();  // Watchdog füttern
             DeserializationError error = deserializeJson(doc,
payload);
            if (!error) {
                const char* message = doc["message"];
                Serial.print(F("Server message: "));
                Serial.println(message);
                if (doc.containsKey("spotify_status")) {
                            const char* spotifyStatus =
doc["spotify_status"];
                    Serial.print(F("Spotify status: "));
                    Serial.println(spotifyStatus);
                }
            }
        } else {
            Serial.print(F("Error on HTTP request: "));
            Serial.println(httpResponseCode);
        }
        http.end();
        Serial.println(F("Connection closed"));
    }
    yield();  // Watchdog füttern am Ende
}
```

```
// Audio callback functions
void audio_info(const char *info) {
    Serial.print(F("info        ")); Serial.println(info);
}

void audio_id3data(const char *info) {
    Serial.print(F("id3data     ")); Serial.println(info);
}

void audio_eof_mp3(const char *info) {
    Serial.print(F("eof_mp3     ")); Serial.println(info);
}

void audio_showstation(const char *info) {
    Serial.print(F("station     ")); Serial.println(info);
}

void audio_showstreaminfo(const char *info) {
    Serial.print(F("streaminfo  ")); Serial.println(info);
}

void audio_showstreamtitle(const char *info) {
    Serial.print(F("streamtitle: ")); Serial.println(info);
    strncpy(streamTitle, info, sizeof(streamTitle) - 1);
    streamTitle[sizeof(streamTitle) - 1] = '\0';
    updateDisplay();
}

void audio_bitrate(const char *info) {
    Serial.print(F("bitrate     ")); Serial.println(info);
}

void audio_commercial(const char *info) {
    Serial.print(F("commercial  ")); Serial.println(info);
}

void audio_icyurl(const char *info) {
    Serial.print(F("icyurl      ")); Serial.println(info);
}

void audio_lasthost(const char *info) {
```

```
    Serial.print(F("lasthost    ")); Serial.println(info);
}

void audio_eof_speech(const char *info) {
    Serial.print(F("eof_speech  ")); Serial.println(info);
}
```

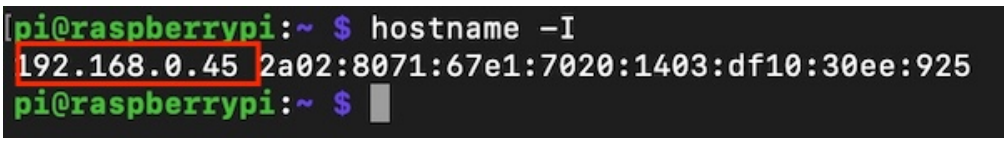
Im obigen Sketch musst du zunächst deine eigenen WLAN-Zugangsdaten eintragen:

```
const char ssid[] = "DEIN WLAN-NETZWERK";
const char password[] = "DEIN WLAN-PASSWORT";
```

Außerdem benötigst du die IP-Adresse deines Raspberry Pi. Diese findest du zum Beispiel mit dem folgenden Befehl heraus, den du im Terminal eingibst (während du per SSH mit ihm verbunden bist):

```
hostname -I
```

Daraufhin erscheint die IP-Adresse im Terminal (im rot markierten Teil):



Diese Adresse trägst du dann im Sketch hier ein, versehen mit dem Port **:8080**

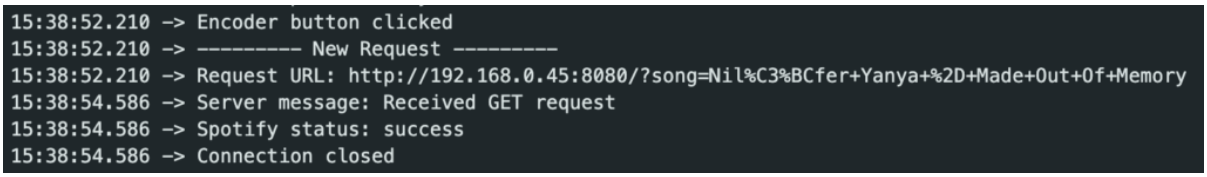
```
const char* serverName = "http://192.168.0.45:8080";
```

Die Wahl der Radiostationen bzw. deren Stream-Adressen und Namen bleibt dir natürlich selbst überlassen – hier hast du sicherlich bestimmt schon eine Liste erstellt und kannst sie in diesen Sketch übernehmen.

Wenn du alles erledigt hast, kannst du diesen Sketch auf deinen ESP32 hochladen und damit wie gewohnt Radio hören.

Einen Song an Spotifiy übertragen

Jetzt wird es Zeit für den Test deiner neuen Funktion! Sobald du einen Song hörst, dessen Interpret und Titel du auf dem Display siehst, drücke den Button an deinem Rotary Encode. Der Song sollte daraufhin kurz abbrechen und dann wieder einsetzen. In der Zwischenzeit gibt dir der Serielle Monitor Auskunft darüber, was gerade passiert ist:



Für die Übertragung wurde die IP-Adresse deines Raspberry Pi um die Song-Informationen erweitert und von deinem ESP32 aufgerufen. Dein Raspberry hat den Song erfolgreich an Spotify weitergeleitet und sendet dem ESP32 ein **Spotify status: success**

Wirf nun in Spotify einen Blick in deine Playlist **Lieblingssongs**. Hier sollte der gerade übertragene Song bereits zu finden sein.

Das Script automatisch starten

Noch ein letzter Baustein für dieses Projekt: Es ist natürlich sehr unpraktisch, wenn du dein neues Feature immer erst aktivieren musst, indem du auf dem Raspberry Pi die virtuelle Umgebung aktivierst und das Python-Script manuell startest. Deshalb wirst du nun dafür sorgen, dass das Script automatisch startet, sobald du deinen Raspi hochfährst. Das geht folgendermaßen:

- Erstelle eine Service-Datei:

```
sudo nano /etc/systemd/system/spotify-radio.service
```

- Kopiere den den folgenden Code, füge ihn ein und speichere es ab mit STRG+O, STRG+X.

```
[Unit]
Description=Spotify Radio Service
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/RadioSpotify
ExecStart=/home/pi/RadioSpotify/bin/python
/home/pi/RadioSpotify/radiospotify.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

- Lade die systemd-Konfiguration neu:

```
sudo systemctl daemon-reload
```

- Aktiviere den Service für den Autostart:

```
sudo systemctl enable spotify-radio
```

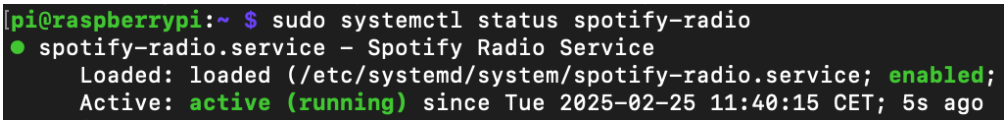
- Starte nun den Service:

```
sudo systemctl start spotify-radio
```

Ob der Service läuft, kannst du abschließend mit folgendem Befehl überprüfen:

```
sudo systemctl status spotify-radio
```

Im Terminal sollte nun etwas in dieser Art anzeigen, dass dein Spotify-Service läuft:



Starte testweise deinen Raspberry Pi neu und führe noch einmal die obige Prüfung aus. In deinem Terminal sollte wieder stehen, dass der Service aktiv ist.

Und das war es! Dein ESP32 Internetradio ist nun direkt mit Spotify verbunden und du hast die Möglichkeit, interessante Songs dort in deinen Lieblingssongs speichern. Viel Spaß damit!