

Software Setup – Chronological Order

ECET 35901 Final Project: Spotify Voice Control System

John Danison

Fall 2025

1 Project Overview

This project implements a voice-controlled Spotify music player on a Raspberry Pi, featuring ADC volume control and MQTT integration with Node-RED for Spotify API communication.

2 System Architecture & Code Flow

2.1 Step-by-Step Logic

2.1.1 Phase 1: System Initialization

1. Virtual Environment Activation

- Script automatically detects and activates Python virtual environment (**venv/**)
- Checks if already running in venv to avoid redundant activation
- Displays warning if venv not found

2. Import Dependencies

- Load `Whisper.cpp` for speech-to-text transcription
- Import `gpiozero` for MCP3008 ADC interface
- Load `paho-mqtt` for MQTT communication
- Handle graceful degradation if components unavailable

3. Hardware Initialization

- Initialize MCP3008 ADC on channel 0 (potentiometer input)
- Test ADC availability and set flag accordingly
- Configure audio device paths: `plughw:4,0` (I2S mic), `plughw:3,0` (USB speakers)

4. Start Spotifyd Daemon

- Check if `spotifyd` process already running using `pgrep`
- If not running, launch `spotifyd` with `--no-daemon` flag
- Wait 2 seconds for daemon initialization

5. Launch MQTT Control Listener

- Create MQTT client with callback functions
- Connect to broker at `localhost:1883`
- Subscribe to `voice/control` topic for button commands
- Run listener in background thread (daemon mode)

6. Start Volume Monitoring Thread

- If ADC available, set `volume_monitoring = True`
- Launch background thread executing `volume_monitor_thread()`
- Thread reads ADC every 0.2 seconds and adjusts system volume

7. Display User Interface

- Print available commands (R/P/T/V/Q)
- Show MQTT topic information
- Enter main input loop

2.1.2 Phase 2: Main Operation Loop

Option R – Record Voice Command

1. User presses 'R' or MQTT receives `button_pressed` while not recording
2. Script displays: "Recording... speak now!"
3. Publish status to `voice/status`: "Recording"
4. Execute `arecord` command:

```
arecord -D plughw:4,0 -c1 -r 48000 -f S32_LE songrequest.wav
```

5. User speaks: "Play [Song Name] by [Artist Name]"
6. User presses CTRL+C or sends MQTT stop command
7. Audio saved as `songrequest.wav`
8. Proceed to transcription phase

Option T – Transcribe & Process

1. User presses 'T' or auto-triggered after recording stop
2. Check if `songrequest.wav` exists, display error if not
3. Publish status to `voice/status`: "Processing Request"
4. Execute Whisper.cpp transcription:

```
./whisper.cpp/build/bin/whisper-cli \
  -m ./whisper.cpp/models/ggml-tiny.en.bin \
  -f songrequest.wav -nt
```

5. Capture stdout text result (e.g., “play, Stairway to Heaven by Led Zeppelin”)
6. Call `parse_voice_command()` to format:
 - Remove “play,” or “play” prefix
 - Split on “ by ” separator
 - Extract song name and artist name
 - Capitalize each word
 - Combine: “Stairway To Heaven Led Zeppelin”
7. Call `send_to_nodered()`:
 - Create MQTT client
 - Connect to `localhost:1883`
 - Publish formatted query to `voice/spotify` topic
 - Disconnect client
8. Clear status: publish empty string to `voice/status`
9. Display success message with formatted query

Option P – Playback Last Recording

1. User presses 'P'
2. Execute `aplay` command:

```
aplay -D plughw:3,0 songrequest.wav
```

3. Audio plays through USB speakers
4. Return to main menu

Option V – Toggle Volume Monitoring

1. User presses 'V'
2. Check if ADC available, display error if not
3. If volume monitoring OFF:
 - Set `volume_monitoring = True`
 - Launch new background thread running `volume_monitor_thread()`
 - Display: “Volume monitoring: ON”
4. If volume monitoring ON:
 - Set `volume_monitoring = False`
 - Wait for thread to terminate (1 second timeout)
 - Display: “Volume monitoring: OFF”

Option Q – Quit Application

1. User presses 'Q'
2. Clear dashboard displays:
 - Publish empty string to `voice/status`
 - Publish empty string to `voice/spotify`
3. Stop volume monitoring thread if running
4. Kill `spotifyd` process: `pkill spotifyd`
5. Display goodbye message and exit program

2.1.3 Phase 3: Background Processes (Parallel Execution)

Background Thread 1: MQTT Control Listener

1. Runs continuously in daemon thread
2. Listens on `voice/control` topic
3. On message received:
 - Decode payload (e.g., “button_pressed”)
 - If currently recording: stop and transcribe
 - If not recording: start recording
4. Also accepts text commands: “record”, “stop”, “transcribe”
5. Boolean values: “true” = start, “false” = stop

Background Thread 2: Volume Monitoring

1. Runs continuously while `volume_monitoring == True`
2. Loop every 0.2 seconds:
 - Read MCP3008 ADC value (0.0 to 1.0 float)
 - Calculate display volume: `int(adc_value * 100)`
 - Map to system volume: `int(adc_value * 91) + 9` (range: 9-100%)
 - Execute: `amixer set Master [volume]%`
3. Prevents audio cutout by avoiding 0-8% range
4. Terminates when flag set to False

External Process: Node-RED Flow

1. Runs independently on Raspberry Pi
2. Subscribes to `voice/spotify` MQTT topic
3. On message received (e.g., “Stairway To Heaven Led Zeppelin”):
 - Parse search query
 - Send HTTP request to Spotify Web API
 - Search for matching track
 - Extract track URI
 - Send play command to `spotifyd` via Spotify Connect API
4. Handles authentication tokens and API rate limiting
5. Publishes playback status back to dashboard

3 Programming Languages & Tools Used

3.1 Primary Language

- **Python 3** (version 3.9+)

3.2 Key Libraries & Packages

3.2.1 Audio Processing

- **whispercpp** – OpenAI Whisper speech-to-text engine (Python bindings)
- **arecord** – ALSA audio recording tool (I2S microphone interface)
- **aplay** – ALSA audio playback tool (USB speaker output)

3.2.2 Hardware Control

- **gpiozero** – MCP3008 ADC interface for volume potentiometer
 - Reads analog voltage (0–3.3V) from potentiometer
 - Converts to digital volume control (0–100%)

3.2.3 MQTT Communication

- **paho-mqtt** – MQTT client library
 - Publishes voice commands to Node-RED
 - Subscribes to control commands from dashboard
 - Sends status updates (recording, processing, errors)

3.2.4 System Utilities

- **subprocess** – Execute shell commands (arecord, aplay, amixer)
- **threading** – Parallel execution for volume monitoring & MQTT listener
- **os, sys** – Virtual environment activation and file path management

3.3 External Tools

3.3.1 Node-RED

- **Purpose:** Spotify API integration
- **Function:** Receives MQTT messages, searches Spotify catalog, plays songs
- **Flow Files:**
 - Node-Red Spotify Flow vFinal.json
 - Spotify Test 2 Updated 2.json
- **Topics:**
 - voice/spotify (search queries)
 - voice/control (button commands)
 - voice/status (system status)

3.3.2 Spotifyd

- **Purpose:** Spotify Connect daemon
- **Function:** Allows Raspberry Pi to appear as Spotify playback device
- **Auto-start:** Launched by Python script if not running

3.3.3 MQTT Broker (Mosquitto)

- **Port:** 1883 (local)
- **Function:** Message broker between Python script and Node-RED

4 Installation & Configuration

4.1 Prerequisites

```
# 1. Update system
sudo apt update && sudo apt upgrade -y

# 2. Install ALSA audio tools
sudo apt install alsa-utils -y

# 3. Install MQTT broker
sudo apt install mosquitto mosquitto-clients -y
```

```
# 4. Install Python dependencies
pip3 install gpiozero paho-mqtt

# 5. Configure I2S microphone (add to /boot/firmware/config.txt)
sudo nano /boot/firmware/config.txt
# Add line: dtoverlay=googlevoicehat-soundcard
sudo reboot now
```

4.2 Virtual Environment Setup

```
cd ~/Documents/Final Project
python3 -m venv venv
source venv/bin/activate
pip install whispercpp gpiozero paho-mqtt
```

4.3 Whisper.cpp Installation

```
cd ~/Documents/Final Project
git clone https://github.com/ggerganov/whisper.cpp.git
cd whisper.cpp
make
# Download tiny.en model
bash ./models/download-ggml-model.sh tiny.en
```

4.4 Spotifyd Configuration

- Placed in project root directory
- Runs as foreground process (`--no-daemon` flag)
- Configured via `~/config/spotifyd/spotifyd.conf`

5 Hardware Configuration

5.1 Audio Devices

- **I2S Microphone:** `plughw:4,0` (Adafruit I2S MEMS Microphone)
 - Format: `S32_LE` (32-bit signed little-endian)
 - Sample Rate: 48000 Hz
 - Channels: 1 (mono)
- **USB Speakers:** `plughw:3,0` (USB Audio Device)
 - Playback device for recorded audio & Spotify

5.2 ADC Configuration

- **Chip:** MCP3008 (8-channel 10-bit ADC)
- **Channel:** 0 (potentiometer input)
- **Update Rate:** 200ms (0.2 seconds)
- **Volume Mapping:**
 - ADC: 0.0–1.0 → Display: 0–100%
 - System Volume: 9–100% (prevents audio cutout at low end)

6 Code Structure & Key Functions

6.1 Main File: SpotifyVoiceControl.py (524 lines)

6.1.1 Voice Command Processing

```
parse_voice_command(transcript)
# Input: "Play, Stairway to Heaven by Led Zeppelin"
# Output: "Stairway To Heaven Led Zeppelin"
# Logic:
#   - Remove "Play," or "play" prefix
#   - Remove " by " separator
#   - Capitalize each word
#   - Combine song + artist
```

6.1.2 MQTT Functions

- `send_to_nodered()` – Publish formatted search query
- `send_status_update()` – Update dashboard status display
- `on_mqtt_message()` – Handle button press commands
- `start_mqtt_listener()` – Background thread for control topic

6.1.3 Recording Functions

- `start_recording()` – Spawn arecord process (background)
- `stop_recording_and_transcribe()` – Terminate recording, run transcription
- `transcribe_audio()` – Execute whisper-cli, parse output, send to Node-RED

6.1.4 Volume Control

- `set_volume(volume_percent)` – Execute amixer command
- `volume_monitor_thread()` – Continuous ADC polling loop

7 Terminal Command Examples

7.1 Recording Audio

```
arecord -D plughw:4,0 -c1 -r 48000 -f S32_LE songrequest.wav  
# Output: Recording... (Press CTRL+C to stop)
```

7.2 Playing Audio

```
aplay -D plughw:3,0 songrequest.wav  
# Output: Playing WAVE 'songrequest.wav' : Signed 32 bit  
#         Little Endian, Rate 48000 Hz, Mono
```

7.3 Transcription (Whisper)

```
./whisper.cpp/build/bin/whisper-cli \  
-m ./whisper.cpp/models/ggml-tiny.en.bin \  
-f songrequest.wav -nt  
# Output: play, Stairway to Heaven by Led Zeppelin
```

7.4 Volume Control

```
amixer set Master 75%  
# Output: Simple mixer control 'Master',0  
#         Mono: Playback 75 [75%] [-7.50dB] [on]
```

7.5 MQTT Testing

```
# Publish test command  
mosquitto_pub -h localhost -t voice/control -m "button_pressed"  
  
# Subscribe to status updates  
mosquitto_sub -h localhost -t voice/status  
# Output: Recording  
#         Processing Request
```

7.6 Check System Status

```
# Verify MQTT broker  
sudo systemctl status mosquitto  
  
# Confirm spotifyd running  
pgrep -x spotifyd  
  
# List audio devices  
arecord -l
```

8 Running the Application

8.1 Location

~/Documents/Final Project/SpotifyVoiceControl.py

8.2 Execute Command

```
cd ~/Documents/Final Project
python3 SpotifyVoiceControl.py
```

8.3 Expected Startup Output

```
=====
ECET 35901 - Final Project
Spotify Voice Control System
=====

Initializing system...

Activating virtual environment...
* spotifyd already running
Starting MQTT control listener...
* MQTT control listener connected
  Listening on: voice/control

Starting volume control...
Volume monitoring started (running in background).

System ready!

Commands:
  R = Record voice command
  P = Play last recording
  T = Transcribe & send to Spotify
  V = Toggle volume monitoring (currently ON)
  Q = Quit

MQTT Control:
  Topic: voice/control
  Commands: 'button_pressed' (toggle record/stop)

MQTT Status:
  Topic: voice/status
  Messages: recording, Processing Request, error

Enter command (R/P/T/V/Q):
```

9 User Interaction Examples

9.1 Manual Recording

```
Enter command (R/P/T/V/Q): r

Recording... speak now!
Say: 'Play [Song Name] by [Artist Name]'
Press CTRL+C to stop.

^C
Recording manually stopped.
Saved as songrequest.wav

Enter command (R/P/T/V/Q): t

Transcribing audio... please wait.

=== TRANSCRIPTION RESULT ===
play, Stairway to Heaven by Led Zeppelin
=====

Sending to Node-RED...
  Original: 'play, Stairway to Heaven by Led Zeppelin'
  Formatted: 'Stairway To Heaven Led Zeppelin'
  Connecting to localhost:1883...
  Publishing to topic: voice/spotify
* Sent to Node-RED: 'Stairway To Heaven Led Zeppelin'
```

9.2 MQTT Button Control

```
Received command: 'button_pressed'

Recording started... (waiting for stop command)

Received command: 'button_pressed'
Stopping recording...
* Saved as songrequest.wav

Transcribing audio... please wait.
[... transcription process ...]
* Sent to Node-RED: 'Bohemian Rhapsody Queen'
```

9.3 Volume Control (Background)

Volume control runs continuously in the background. The ADC continuously reads the potentiometer and automatically adjusts system volume. No terminal output is displayed unless an error occurs.

10 File Structure

Final Project/

```
|
|-- SpotifyVoiceControl.py          # Main application (524 lines)
|-- HardwareFuncations.py          # Alternative/test version (289 lines)
|-- songrequest.wav                 # Recorded audio buffer
|
|-- venv/                           # Python virtual environment
|   +-- bin/activate_this.py        # Auto-activation script
|
|-- whisper.cpp/                    # Whisper.cpp repository
|   |-- build/bin/whisper-cli        # Transcription executable
|   +-- models/ggml-tiny.en.bin      # Tiny English model
|
|-- spotifyd                        # Spotify Connect daemon binary
|
|-- Node-Red Spotify Flow vFinal.json # Node-RED flow export
|-- Spotify Test 2 Updated 2.json    # Alternative flow version
|
|-- Commands.txt                    # Setup commands reference
+-- README.md                       # Project description
```

11 Troubleshooting Tips

11.1 Issue: “Whisper not available”

Solution: Verify virtual environment is activated and whispercpp is installed

```
source venv/bin/activate
pip install whispercpp
```

11.2 Issue: “Connection refused - MQTT broker”

Solution: Start Mosquitto broker

```
sudo systemctl start mosquitto
sudo systemctl enable mosquitto
```

11.3 Issue: “No audio devices found”

Solution: Check I2S microphone configuration

```
arecord -l # Should show plughw:4,0
sudo nano /boot/firmware/config.txt
# Verify: dtoverlay=googlevoicehat-soundcard
```

11.4 Issue: “ADC not available”

Solution: Check SPI interface enabled

```
sudo raspi-config  
# Interface Options -> SPI -> Enable
```

11.5 Issue: Spotifyd not connecting to Spotify

Solution: Verify credentials in config file

```
nano ~/.config/spotifyd/spotifyd.conf  
# Check username, password, device_name
```

12 System Requirements

- **Platform:** Raspberry Pi 4 Model B (tested)
- **OS:** Raspberry Pi OS (Debian-based, 64-bit recommended)
- **Python:** 3.9 or higher
- **Network:** WiFi or Ethernet (for Spotify API access)
- **Storage:** ~500MB for Whisper model + dependencies
- **RAM:** 2GB minimum (4GB recommended for Whisper processing)

13 Credits & References

- **Whisper.cpp:** <https://github.com/ggerganov/whisper.cpp>
- **I2S Microphone Setup:** <https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout/raspberry-pi-wiring-test>
- **Spotifyd:** <https://github.com/Spotifyd/spotifyd>
- **Paho MQTT Python:** <https://pypi.org/project/paho-mqtt/>