PROYECTO — REPRODUCTOR DE MÚSICA UTILIZANDO UN RASPBERRY PI 4 COMO SERVIDOR Y UNA INTERFAZ MOSTRADA EN UN SITIO HTML COMO CONTROL

Materia: Microprocesadores

Alumno: Guillermo Segura Elizalde

Grupo: 6CM5

ICE ESIMEZ IPN

Profesor: Ing. Roberto Galicia Galicia

Fecha: 20 de enero del 2021

Objetivo del proyecto:

Demostrar las posibilidades de utilizar una Raspberry Pi 4 con aplicaciones que lo convierten en servidor de red local, con un enfoque en el manejo de archivos multimedia y sin tener que emplear necesariamente los puertos GPIO; tomando en su lugar el puerto de salida de audio / vídeo compuesto.

Descripción:

Esta aplicación consiste en un programa escrito en el lenguaje Python, el cual utiliza el entorno Flask y la librería pygame. Flask se encarga de generar un servidor en la red local donde se ejecuten comandos Python en un apartado principal (ruta index) y donde se muestre una plantilla HTML a un navegador web; mientras que una instancia iniciada del servicio pygame (pygame.mixer) se encarga de recibir instrucciones del cliente y manejar los archivos de sonido ubicados en el directorio del programa.

Marco teórico:

En el mundo de TCP/IP las comunicaciones entre computadoras se rigen básicamente por lo que se llama modelo Cliente-Servidor, éste es un modelo que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones. El término Cliente/Servidor fue usado por primera vez en 1980 para referirse a PC's en red.

Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma.

En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio). En un sistema distribuido, cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

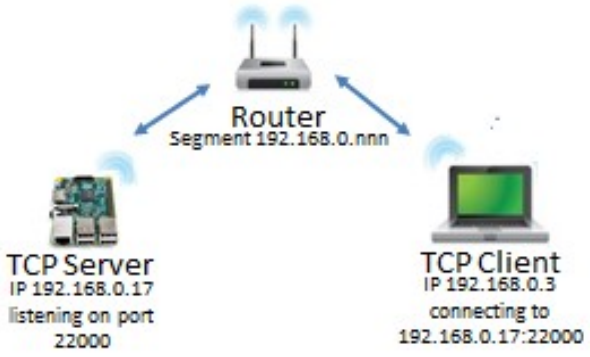


Figura 1: Ejemplo de estructura TCP/IP utilizando un Raspberry Pi 4 y un router inalámbrico.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son mas adecuadas a sus características.

Si esto se aplica tanto a clientes como servidores se entiende que la forma más estándar de aplicación y uso de sistemas Cliente/Servidor es mediante la explotación de las PC's a través de interfaces gráficas de usuario; mientras que la administración de datos y su seguridad e integridad se deja a cargo de computadoras centrales tipo mainframe. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y el o los procesos cliente sólo se ocupan de la interacción con el usuario (aunque esto puede variar). En otras palabras la arquitectura Cliente/Servidor es una extensión de programación modular en la que la base fundamental es separar una gran pieza de software en módulos con el fin de hacer más fácil el desarrollo y mejorar su mantenimiento.

¿Qué es Flask?

Flask es un microframework web escrito en Python para crear aplicaciones web. Es catalogado como microframework porque su estructura inicial es minimalista, en contraste con un Full Stack Framework, que incluye una interfaz de autenticación para el usuario, un ORM y una arquitectura definida desde el inicio como Django.

Flask es simple y se limita a lo que toda aplicación Web necesita: Un sistema de enrutamiento y un sistema de platillas, y renuncia a características que no todas las aplicaciones Web necesitan como por ejemplo: una Base de Datos, ya sea relacional o sin esquema, o un Framework Javascript para el desarrollo del Frontend.

¿Qué es Pygame?

Pygame es una biblioteca libre multiplataforma que facilita el desarrollo de juegos en tiempo real con el lenguaje de programación Python. Se distribuye bajo los términos de la LGPL GNU. Construido sobre la biblioteca SDL, permite programar la parte multimedia (gráficos, sonido y la entrada de teclado, ratón o joystick), sin encontrar las dificultades de lenguajes de bajo nivel como C y sus derivados. Esto se basa en la suposición de que la parte multimedia es suficientemente independiente de la lógica del juego para que podamos usar un lenguaje de alto nivel (en este caso el de Python) para la estructura del juego.

Pygame, además de la adaptación de la SDL para Python también proporciona algunas funciones específicas para el desarrollo del juego.

También podemos notar que Pygame ya no se utiliza exclusivamente para videojuegos, sino también para diversas aplicaciones que requieren gráficos.

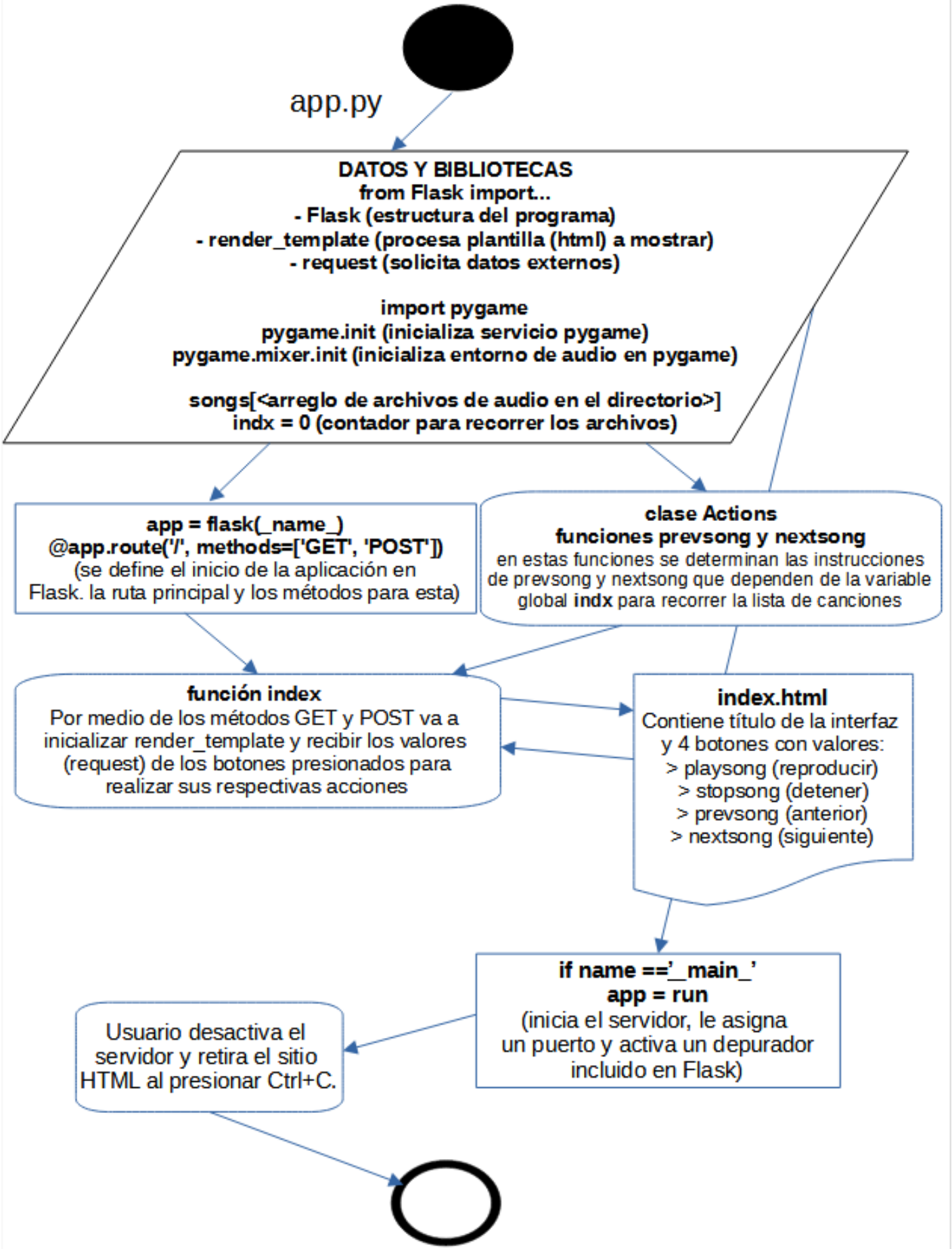
Desarrollo

Para el desarrollo de la práctica se cuenta con los siguientes materiales:

- Computadora Raspberry Pi (¾) con tarjeta SD y un sistema operativo Linux instalado
- Cable de audio TRS de 3.5mm
- Cualquier bocina o auricular que tenga entrada o conector de 3.5mm
- Una red inalámbrica Wi-fi

No se requiere ningún otro material para que funcione el reproductor de audio. El único requisito es que la Raspberry Pi esté previamente conectada a la red inalámbrica, y que se conozca su dirección IP local asignada por el router.
Todos los elementos de la aplicación van a premanecer dentro de un mismo directorio o carpeta, y la plantilla HTML que se va a mostrar al cliente va a ser incluida en una subcarpeta.

El diagrama de flujo del programa planteado se muestra en la siguiente página.


app.py
DATOS Y BIBLIOTECAS
from Flask import...
- Flask (estructura del programa)
- render_template (procesa plantilla (html) a mostrar)
- request (solicita datos externos)
import pygame
pygame.init (inicializa servicio pygame)
pygame.mixer.init (inicializa entorno de audio en pygame)
songs[<arreglo de archivos de audio en el directorio>]
indx = 0 (contador para recorrer los archivos)
app = flask(_name_)
@app.route('/', methods=['GET', 'POST'])
(se define el inicio de la aplicación en Flask. la ruta principal y los métodos para esta)
clase Actions
funciones prevsong y nextsong
en estas funciones se determinan las instrucciones de prevsong y nextsong que dependen de la variable global indx para recorrer la lista de canciones
función index
Por medio de los métodos GET y POST va a inicializar render_template y recibir los valores (request) de los botones presionados para realizar sus respectivas acciones
index.html
Contiene título de la interfaz y 4 botones con valores:
> playsong (reproducir)
> stopsong (detener)
> prevsong (anterior)
> nextsong (siguiente)
if name ==’_main_’
app = run
(inicia el servidor, le asigna un puerto y activa un depurador incluido en Flask)
Usuario desactiva el servidor y retira el sitio HTML al presionar Ctrl+C.

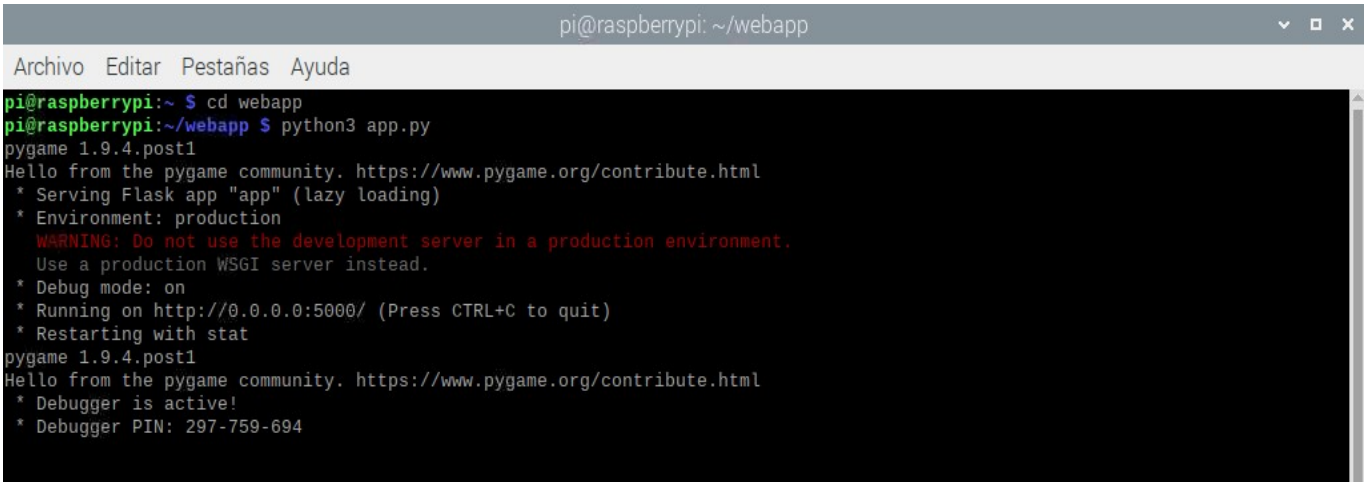
Este es el código de la parte principal del programa, app.py, escrito en lenguaje Python:

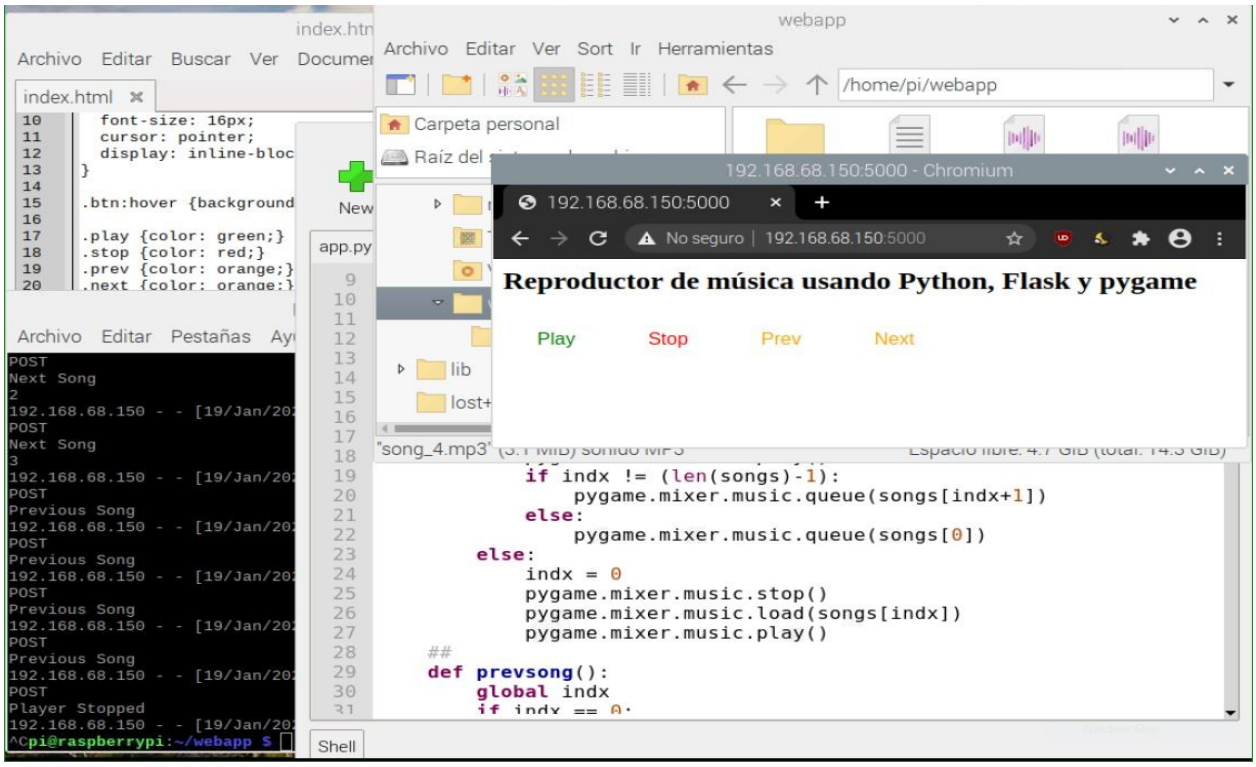
```
| import pygame
| ##import os
| pygame.init()
|
| pygame.mixer.init()
|
| songs = ["song_1.mp3","song_2.mp3","song_3.mp3","song_4.mp3"]
| indx = 0
|
| class Actions:
|     ##corregir nextsong
|     def nextsong():
|         global indx
|         if indx < len(songs):
|             indx += 1
|             pygame.mixer.music.load(songs[indx])
|             pygame.mixer.music.play()
|             if indx != (len(songs)-1):
|                 pygame.mixer.music.queue(songs[indx+1])
|             else:
|                 pygame.mixer.music.queue(songs[0])
|         else:
|             indx = 0
|             pygame.mixer.music.stop()
|             pygame.mixer.music.load(songs[indx])
|             pygame.mixer.music.play()
|     ##
|     def prevsong():
|         global indx
|         if indx == 0:
|             pygame.mixer.music.stop()
|             pygame.mixer.music.load(songs[0])
|             pygame.mixer.music.play()
|         else:
|             indx -= 1
|             pygame.mixer.music.load(songs[indx])
|             pygame.mixer.music.play()
|             pygame.mixer.music.queue(songs[indx+1])
|
| ##Arranque Flask y creación del servidor
| app = Flask(__name__)
|
| @app.route('/', methods=['GET', 'POST'])
| def index():
|     global indx
|     print(request.method)
|     if request.method == 'POST': ##Botones del cliente
|             if request.form.get('playsong') == 'playsong':
|                 pygame.mixer.music.load(songs[0])
```

```
|                 pygame.mixer.music.play()
|                 pygame.mixer.music.queue(songs[1])
|                 print("Playing Song")
|             elif  request.form.get('stopsong') == 'stopsong':
|                 ##
|                 pygame.mixer.music.stop()
|                 index = 0
|                 print("Player Stopped")
|             elif  request.form.get('prevsong') == 'prevsong':
|                 Actions.prevsong()
|                 print("Previous Song")
|             elif  request.form.get('nextsong') == 'nextsong':
|                 Actions.nextsong()
|                 print("Next Song")
|                 print(indx)
|             else:
|                 return render_template("index.html")
|     elif request.method == 'GET':
|             print("No Post Action")
|     return render_template('index.html')
|
|
| if __name__ == '__main__': ##Inicia servidor
|     app.run(debug=True, host='0.0.0.0')
```

Pruebas y costo

Se conectó el Raspberry al adaptador de corriente, se ejecutó el programa desde LXTerminal, se ingresó la dirección IP con el puerto provisto después de la ejecución y lo que se observó fue lo siguiente:





Como se puede observar en la primera captura, la consola LXTerminal imprime el momento en el que se inicializa el servidor y el servicio pygame en la Raspberry Pi. Posteriormente, imprime los estados de los métodos GET y POST, tanto al iniciar el programa como al oprimir algun botón en la plantilla HTML (la interfaz gráfica).

Si el programa de extensión .py se ubicó correctamente en el directorio *webapp* (como se observa en la segunda imagen, pero puede tener otro nombre) junto con la carpeta *template (*que contiene el archivo index.html) y los archivos de sonido a reproducir (con sus nombres escritos correctamente en el arreglo del programa, de extensión .mp3, .wav o .ogg), se debería de escuchar la música proveniente del dispositivo conectado a la entrada A/V de la Raspberry Pi.

El costo de todos los componentes se muestra a continuación:

- Cable de audio TRS 3.5mm (cable auxiliar de audio) $50 (previamente adquirido)
- Bocina de batería recargable con conexiones múltiples $180 (previamente adquirida)

TOTAL: **$**230 (el precio puede variar)

Conclusiones

Este proyecto, además de servir como referente para todas las posibilidades que existen para manejar un servidor que reproduce archivos de sonido dentro de una Raspberry Pi, puede visualizarse en una posible aplicación de megafonía o publidifusión, donde se requiere controlar un sistema PA utilizado para reproducir música para ambientación o anuncios, ya sea a manera de bucle infinito o seleccionando el archivo de sonido a reproducir.

De la misma manera en la que se implementa el servicio de Pygame en el entorno Flask para utilizar instrucciones escritas en Python, se puede usar cualquier otra aplicación de sonido que pueda ser controlada localmente desde la terminal de Linux (VLC, Clementine, entre otras), así como se puede implementar el servidor desde cualquier otro entorno que pueda alojar plantillas HTML y que pueda operar en una Raspberry Pi (Apache, Node.js, etc.).

Referencias y recursos consultados:

http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

https://rico-schmidt.name/atix/atix24.pdf

https://unipython.com/curso-python-videojuegos-pygame/

https://www.pygame.org/docs/ref/music.html

https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/

https://stackoverflow.com/questions/46516349/python-3-pygame-mixer-next-song

Enlace del repositorio del programa (GitHub):

https://github.com/WSegura63/pygame-flask_rpi_player