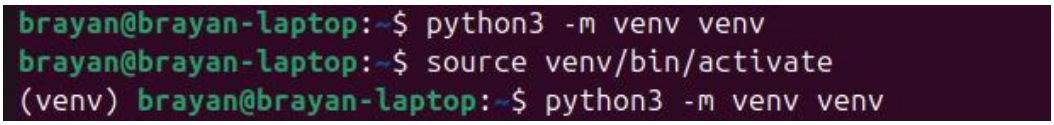
Proyecto Tercer Corte
Sistemas operativos
Brayan Poloche- Karen Garcia

- **Servidor (pc)**

1. Se crea y activa un entorno virtual de Python para trabajar de forma aislada del sistema, permitiendo gestionar dependencias específicas de un proyecto.



   - python3: Invoca el intérprete de Python versión 3.
   - -m venv venv: Usa el módulo venv para crear un entorno virtual en la carpeta venv.
   - source venv/bin/activate: Ejecuta el script de activación del entorno virtual en sistemas Unix/Linux.
   - (venv): Indica que el entorno virtual está activo en la terminal.

2. Se inicia la creación o edición de un archivo de script en Python utilizando un editor de texto directamente desde la terminal.

   - nano p.py: Abre el editor de texto Nano en la terminal para crear o editar el archivo llamado p.py.

3. Se coloca el código en el nano

```python
import pyaudio
from vosk import Model, KaldiRecognizer
import sys
import os
import json
import requests
import qi
```

- import: Carga bibliotecas externas necesarias para ejecutar funciones específicas en el código.

```python
# Configura API DeepSeek
API_KEY = "sk-53751d5c6f344a5dbc0571de9f51313e"
URL = "https://api.deepseek.com/v1/chat/completions"

PROMPT = """
Eres Chucho, un chatbot simpático, con un toque de humor y una personalidad amigable.
Te gusta ayudar a los usuarios con información técnica y general, pero siempre agregando un comentario divertido.
no envias ningun emoji
Tu tono es cercano, como si estuvieras conversando con un buen amigo.
```

- Se definen la clave y URL para usar la API del chatbot. También el prompt el cual asigna la personalidad del chtbot.

```
 GNU nano 7.2                                                    p.py
def obtener_respuesta(mensaje):
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }

    data = {
        "model": "deepseek-chat",
        "messages": [
            {"role": "system", "content": PROMPT},
            {"role": "user", "content": mensaje}
        ]
    }

    response = requests.post(URL, json=data, headers=headers)

    if response.status_code == 200:
        return response.json()["choices"][0]["message"]["content"]
    else:
        return "¡Ups! Parece que mi conexión falló. ¿Me pasas un café mientras intento arreglarlo?"
```

- requests.post(): Envía un mensaje al chatbot.

- response.status_code: Verifica si la conexión fue exitosa.
- response.json(): Extrae el contenido de la respuesta.

```
except KeyboardInterrupt:
    print("\n¡Hasta luego!")
finally:
    stream.stop_stream()
    stream.close()
    p.terminate()
    app.stop()
```

- KeyboardInterrupt: Captura Ctrl+C.
- finally: Asegura que se cierre el micrófono y conexión con Pepper.
- stop_stream(), terminate(), app.stop(): Libera recursos.

4. Se ejecuta el nano.

```
(mi_entorno) brayan@brayan-laptop:~$ python  p.py
```

- python: Invoca el intérprete de Python instalado en el sistema.

- **Cliente (pepper)**

1. Se establece una conexión segura a través de SSH con el robot Pepper para poder interactuar con él desde la computadora

```
brayan@brayan-laptop:~$ ssh nao@192.168.0.106
(nao@192.168.0.106) Password:
```

2. Se hace el nano con el siguiente código.

```python
import pyaudio
from vosk import Model, KaldiRecognizer
import sys
import os
import json
import requests
import qi

IP_PEPPER = "192.168.0.106"
PUERTO_PEPPER = 9559

# Inicializar aplicación qi y servicio TTS SOLO UNA VEZ
url = f"tcp://{IP_PEPPER}:{PUERTO_PEPPER}"
app = qi.Application(["pepper_tts", f"--qi-url={url}"])
app.start()
sesion = app.session
tts = sesion.service("ALTextToSpeech")

def pepper_habla(texto):
    try:
        tts.say(texto)
    except Exception as e:
        print("Error al mandar a hablar a Pepper:", e)

# Configura ruta modelo Vosk
model_path = "/home/brayan/Descargas/vosk-model-es-0.42"
if not os.path.exists(model_path):
    print("Descarga el modelo y ponlo en", model_path)
    sys.exit()
```

- f"tcp://{...}": Formato de cadena para definir la URL de conexión con Pepper.
- qi.Application([...]): Inicializa la aplicación para conectarse al robot.
- app.start(): Inicia la conexión.
- sesion.service("ALTextToSpeech"): Obtiene el servicio de síntesis de voz de Pepper.
- tts.say(texto): Le dice a Pepper que hable el texto.
- try/except: Manejo de errores.

- os.path.exists(): Verifica si el modelo de voz existe.

- sys.exit(): Termina el programa si no se encuentra el modelo.

```
# Configura Vosk
model = Model(model_path)
rec = KaldiRecognizer(model, 16000)

# Configura micrófono
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True, frames_per_buffer=8000)
stream.start_stream()

print("Habla ahora... (Ctrl+C para salir)")

try:
    while True:
        data = stream.read(4000, exception_on_overflow=False)
        if rec.AcceptWaveform(data):
            result = rec.Result()
            texto = json.loads(result).get("text", "")
            if texto.strip() != "":
                print("Reconocido:", texto)
                respuesta = obtener_respuesta(texto)
                print("Chucho:", respuesta)
                pepper_habla(respuesta)
```

- Model(): Carga el modelo de reconocimiento de voz.

- KaldiRecognizer(): Inicializa el reconocedor con una frecuencia de muestreo de 16 kHz.

- pyaudio.PyAudio(): Inicializa audio.

- p.open(): Abre el micrófono con formato y configuración especificada.

- stream.start_stream(): Comienza la captura de audio.

- stream.read(): Lee bloques de audio del micrófono.

- rec.AcceptWaveform(): Detecta si se dijo una frase completa.

- json.loads(): Convierte la respuesta de voz a texto.

3. Se ejecuta el nano de pepper al mismo tiempo que el del pc y se visualiza a Pepper con la conversación del chatbot.

```
Pepper [0] ~ $ python cliente_pepper.py
```

Video de Pepper: https://youtu.be/D1sjyPCZerw

- **Conclusión:**

  En este proyecto, se logró implementar un chatbot que funciona a través de la API de DeepSeek e integrarlo con el robot Pepper. Se configuró un servidor en la computadora que se encarga de procesar las peticiones del chatbot y enviar las respuestas correspondientes a Pepper. En el código, se empleó el módulo Vosk para el reconocimiento de voz, permitiendo que Pepper escuche las preguntas del usuario en tiempo real y las convierta en texto que el chatbot puede entender. Además, se integró el módulo ALTextToSpeech para que Pepper pudiera hablar y responder de manera natural, utilizando la voz incorporada del robot. Para facilitar la comunicación entre la computadora y Pepper, se utilizó una conexión SSH segura, garantizando que los datos se transfieran de forma confiable. En resumen, se integraron de manera efectiva el reconocimiento de voz, la generación de respuestas del chatbot y la conversión a voz en Pepper, logrando que el robot interactúe de forma natural y personalizada con los usuarios.