UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

Trabajo de Investigación

NOMBRE DEL PROYECTO: **"***Asistente Virtual***"**

- Chávez Chico, Joel Jhotan 20210058J
- De La Cruz Valdiviezo Pedro Luis David 20172198H

PROFESOR:

- Cesar Martín Cruz Salazar

PERIODO ACADÉMICO: 2022 – I

FECHA DE ENTREGA DEL INFORME: 30 de julio de 2022

2022

Índice

1. Resumen

El presente informe realizado por estudiantes de la UNI de la espacialidad Ciencias de la Computación del curso de Arquitectura de Computadores muestra el desarrollo de un "Asistente virtual" que se ejecuta a través de un Gadget de escritorio, aplicando las bases aprendidas en programación en Python en el entorno de desarrollo de Visual Studio Code.

2. Introducción

La tecnología ha cambiado radicalmente la vida de las personas en los últimos años, por ejemplo, el uso de teléfonos móviles, las redes sociales, comprar desde casa, etc. Pues todos los días nuevas innovaciones tecnológicas están cambiando el mundo rápidamente, y gracias a ello las vidas de las personas se van haciendo cada vez más confortables. Lo que les da "vida" a la mayoría de estos proyectos es la capacidad que tiene una máquina para realizar una tarea que el ser humano no está dispuesto a desarrollar, es decir, que la máquina asiste al humano en sus labores diarias. Para que estas labores se puedan cumplir, las tareas deben de ser repetitivas y fáciles de realizar ya que estos se encuentran programados únicamente para realizar dichas tareas. Imaginemos por un momento una máquina que asista en tareas complejas, distintas, y únicas cada vez que el humano lo necesite. En el presente informe desarrollaremos las tecnologías necesarias para realizar esta máquina que funcione según las órdenes del humano, así como Siri, o Cortana.

3. Tecnología a usar

Python viene con una hermosa biblioteca estándar, que es una colección de paquetes y módulos que están disponibles en todas las máquinas que ejecutan Python [1]. Cabe mencionar que las líneas de código que se mencione en el presente informe deben de coincidir con el código fuente adjunto a este. Veremos ahora las partes más importantes del programa.

3.1 Speech Recognition

Los orígenes del reconocimiento de voz automático se remontan a 1936 en Bell Labs de AT&T [3]. La tarea del reconocimiento de voz es convertir el habla en una secuencia de palabras mediante un programa de computadora [4]. Entonces el objetivo principal de la clase Recognizer, es por supuesto el reconocimiento del habla, primero toma una señal de voz (señal analógica) lo suficientemente pequeña (<10ms) de esa forma se puede aproximar como un dato estadístico estacionario (constante en el

tiempo), este muestreo de datos se le asigna un vector de número reales que se llaman MFCC. Lo mencionado anteriormente es llamado HMM (Hidden Márkov Model) y la salida de este modelo es un arreglo de estos vectores que representan una unidad fundamental del habla(fonemas) [2].

En nuestro programa final definimos un método llamado “escuchar”, en este método es llamado cuando el usuario le da al botón “Speak” del gadget; en la línea 65 estamos activando el micrófono, mientras esto se ejecuta en la consola se muestra “escuchando…” en la línea 67, todo lo que se habla a través del micrófono se guarda en la variable “audio” y este tiene una duración máxima de 4 segundos. Aquí es cuando la clase Recognition hace su “magia” pues en la línea 70 el programa tratara de convertir ese audio a texto, debemos especificar el lenguaje que estamos trabajando, en este caso pusimos “es-PE” (español de Perú) y retornamos el mensaje de audio en texto, en caso de que no se pudo reconocer la voz (mucho ruido o que directamente no se hable) entonces se mostrará en consola “No se pudo reconocer la voz, repite por favor”. El código en este apartado es el siguiente:

```
62      def escuchar(self):
63          listener = sr.Recognizer()
64          while True:
65              with sr.Microphone() as source:
66                  print("Escuchando...")
67                  audio = listener.listen(source,phrase_time_limit=4)
68                  try:
69                      print("Reconociendo el audio...")
70                      text = listener.recognize_google(audio,Language="es-PE").lower()
71                      return text
72                  except Exception as e:
73                      print("No se pudo reconocer la voz, repite por favor")
74                      print(e)
```

Ilustración 1

De esa manera el reconocimiento de voz funciona primero detectando y luego capturando las palabras habladas (expresiones). Convierte las expresiones capturadas en una representación digital de las palabras después de eliminar los ruidos de fondo [5].

3.2 Yfinance

Esta biblioteca se usa con el fin de obtener datos financieros en un intervalo de tiempo [6]. Esta biblioteca lo usamos en la definición del método “precioAcciones”. Cuando el programa llama a esta función el asistente dice “¿De qué acción quieres saber su precio?”, seguidamente en la línea 167 activamos el micrófono y esperamos una entrada de voz, lo convierte en texto y lo guarda en la variable nombre_accion. Entonces si en dicho texto se encuentra alguno de los nombres indicados en las condicionales elif, guarda el nombre identificado con su respectivo símbolo. El programa tratara de identificar la acción en la línea 184, se guardan los datos de 1 día en la línea 185, de esos datos se le pide el precio de cierre se guarda en la variable “precio” para finalmente hacer que el asistente lo diga en la línea 188.

```python
    def precioAcciones(self):
        self.hablar("¿De que accion quieres saber su precio?")
        nombre_accion = self.escuchar()
        if "tesla" in nombre_accion:
            nombre = "tesla"
            simbolo = "TSLA"
        elif "amazon" in nombre_accion:
            nombre = "amazon"
            simbolo = "AMZN"
        elif "facebook" in nombre_accion:
            nombre = "facebook"
            simbolo = "FB"
        elif "google" in nombre_accion:
            nombre = "google"
            simbolo = "GOOG"
        elif "apple" in nombre_accion:
            nombre = "apple"
            simbolo = "AAPL"
        try:
            ticker = yf.Ticker(simbolo)
            data = ticker.history(period='1d')
            precio = data['Close'][0]

            self.hablar(f"El precio de {nombre} es de {precio}")
        except:
            self.hablar("No entendi a que accion te refieres")
```

Ilustración 2

Esta biblioteca también se usa en el método “precioCripto”, se comporta exactamente igual que “precioAcciones” pero con los precios de algunas criptomonedas

```python
    def precioCripto(self):
        self.hablar("¿De qué criptomoneda quieres saber su precio?")
        nombre_accion = self.escuchar()
        if "doge coin" in nombre_accion:
            nombre = "doge coin"
            simbolo = "DOGE-USD"
        elif "luna" in nombre_accion:
            nombre = "luna"
            simbolo = "LUNA-USD"
        elif "ethereum" in nombre_accion:
            nombre = "ethereum"
            simbolo = "ETH-USD"
        elif "cardano" in nombre_accion:
            nombre = "ada"
            simbolo = "ADA-USD"
        elif "bitcoin" in nombre_accion:
            nombre = "bitcoin"
            simbolo = "BTC-USD"
        elif "polkadot" in nombre_accion:
            nombre = "polkadot"
            simbolo = "DOT-USD"
        try:
            ticker = yf.Ticker(simbolo)
            data = ticker.history(period='1d')
            precio = data['Close'][0]

            self.hablar(f"El precio de {nombre} es de {precio}")
        except:
            self.hablar("No entendi a que accion te refieres")
```

Ilustración 3

3.3 Pywhatkit

Python ofrece numerosas bibliotecas incorporadas para facilitar nuestro trabajo. Entre ellos, pywhatkit es una biblioteca de Python para realizar búsquedas en YB, en Google, entre otras. A continuación, se muestran algunas características del módulo pywhatkit [7]:

a. Reproducir un video de YouTube.

La función pywhatkit.playonyt() , abre YouTube en su navegador predeterminado y reproduce el video que mencionó en la función. Si pasa el nombre del tema como parámetro, se reproduce el video aleatorio sobre ese tema. Al pasar la URL del video como parámetro, abre ese video exacto.

Sintaxis: pywhatkit.playonyt("url / nombre del tema")
Parámetros:

- *URL / nombre del tema*: Mencione el nombre del tema o la URL del video de YouTube

Programa Aplicativo:

```
1  import pywhatkit
2  try:
3      pywhatkit.playonyt("Y es que sucede así - Arena Hash")
4      print("Reproduciendo...")
5
6  except:
7      print("¡¡Error de red :c!!")
```

```
Reproduciendo...
```

Función implementada:

El programa solicitará el nombre de la canción a buscar en YB, y posteriormente espera la respuesta del usuario para luego realizar la búsqueda directamente con la función pywhatkit.playonyt() explicada anteriormente.

```
157      def buscarYB(self):
158          self.hablar("Dime que quieres que busque en Youtube:")
159          busca= self.escuchar()
160          busca = busca.replace('busca en youtube', '')
161          self.hablar("Comenzando busqueda en youtube")
162          pywhatkit.playonyt(busca)
163          self.hablar(f"Busqueda exitosa")
```

Ilustración 4

b. Realizar una búsqueda en Google.

Puede realizar una búsqueda en Google usando el siguiente comando simple, este abre su navegador y busca el tema que ha dado en su código.

Sintaxis: pywhatkit.search("Palabra clave")
Parámetros:

- *Palabra clave*: Tema sobre el cual haremos la búsqueda.

Programa Aplicativo:

```
1   import pywhatkit
2
3   try:
4       pywhatkit.search("COVID 19")
5       print("Buscando...")
6   except:
7       print("Un error desconocido ocurrió")
```

```
Buscando...
```

Función implementada:

Similar a la función de la librería anterior el programa en este caso pide la palabra clave y se ejecuta pywhatkit.search() inmediatamente.

```
149     def buscarGoogle(self):
150         self.hablar("Dime que quieres que busque en Google:")
151         busca= self.escuchar()
152         busca = busca.replace('busca en google', '')
153         self.hablar(f"Comenzando busqueda en google")
154         pywhatkit.search(busca)
155         self.hablar(f"Busqueda exitosa")
```

Ilustración 5

3.4 Pyjokes

Pyjokes es una biblioteca de Python que se usa para crear chistes de una línea para programadores. De manera informal, también se puede denominar una biblioteca de Python divertida que es bastante simple de usar [8].

Función implementada:

En nuestro programa definimos el método “chiste”, en la línea 119 elegimos un chiste con una categoría determinada y lenguaje español, luego en la línea 120 hacemos que el asistente diga el chiste.

```
118     def chiste(self):
119         chiste = pyjokes.get_joke(language="es", category="neutral")
120         self.hablar(chiste)
```

Ilustración 6

3.5 Pyttsx3

Pyttsx3 es una biblioteca de texto a voz multiplataforma que es independiente de la plataforma. La principal ventaja de utilizar esta biblioteca para la conversión de texto a voz es que funciona sin conexión. Sin embargo, pyttsx solo admite Python 2.x. Por lo tanto, para este trabajo usaremos el pyttsx3 que está modificado para funcionar tanto en Python 2.xy Python 3.x con el mismo código [10]. Podemos usar dos formas para que nuestro texto hable:

- **Método 1**: Llamando directamente a la función speak().
- **Método 2**: Primero obtenga la referencia del motor pyttsx3 y luego use el método say(), runAndWait(). [11]

Métodos Importantes [12]:

- *pyttsx3.init*([driverName : string, debug : bool]) Obtiene una referencia a una instancia de motor que usará el controlador dado. Si el controlador solicitado ya está en uso por otra instancia de motor, se devuelve ese motor. De lo contrario, se crea un nuevo motor.

  Parámetros:

  driverName – Nombre del módulo pyttsx3.drivers para cargar y usar.

  El valor predeterminado es el mejor controlador disponible para la plataforma, actualmente:

  - sapi5 - SAPI5 on Windows
  - nsss - NSSpeechSynthesizer on Mac OS X
  - espeak - eSpeak on every other platform

  debug – Habilite la salida de depuración o no.

- *say(text : unicode, name : string):* Pone en cola un comando para pronunciar una expresión. La voz se emite de acuerdo con las propiedades establecidas antes de este comando en la cola

Parámetros:

text - Texto que desee escuchar

name - Nombre a asociar con el enunciado. Incluido en las notificaciones sobre este enunciado (opcional).

- *runAndWait()**:*** Se bloquea mientras se procesan todos los comandos actualmente en cola. Invoca devoluciones de llamada para las notificaciones del motor de forma adecuada. Devuelve cuando todos los comandos en cola antes de esta llamada se vacían de la cola.

- *speak(text: str):* Ejecuta las 3 funciones anteriores (*init(), say(text), runAndWait()*) bajo ese mismo orden, útil para programas más cortos y sencillos.
- *getProperty(name : string)*: Obtiene el valor actual de una propiedad del motor

Parámetros:

- ✓ *name* - Nombre de la propiedad a consultar.

Returns:
Valor de la propiedad al momento de esta invocación.

- *setProperty(name : string, value)*: Pone en cola un comando para establecer una propiedad del motor. El nuevo valor de propiedad afecta a todas las declaraciones en cola después de este comando.

Parámetros:

name - Nombre de la propiedad a consultar.

value – Valor a enviar

NOTA:
Los siguientes nombres de propiedad son válidos para todos los controladores:

- rate: Tasa de voz entera en palabras por minuto. El valor predeterminado es 200 palabras por minuto.
- voice: Identificador de cadena de la voz activa.
- voices: Lista de objetos descriptores de *pyttsx3.voice.Voice*

- volumen: Volumen de punto flotante en el rango de 0,0 a 1,0 inclusive. El valor predeterminado es 1.0.

Programa Aplicativo:

```python
import pyttsx3

engine = pyttsx3.init()

#Para definir una voz en particular
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[2].id)

# Establezca la tasa de pronunciación, el valor predeterminado es 200
rate = engine.getProperty('rate')
engine.setProperty('rate', rate - 50)

# Establece el tamaño de la pronunciación, el rango es [0.0,1.0]
volume = engine.getProperty('volume')
engine.setProperty('volume', 0.8)

msg = '¡Hola que tal! ¿Cómo estás?'

engine.say(text=msg, name="Saludo")

engine.runAndWait()
```

Función Implementada

Para la fase final se implementa la siguiente función que dará voz a nuestro chatbot con las propiedades asignadas por defecto, de manera sencilla podemos entender que esta función recibe un mensaje por escrito y es reproducido inmediatamente en la aplicación.

```python
    def hablar(self,msg):
        engine = pyttsx3.init()
        #engine.setProperty('rate', 200)
        engine.say(text=msg, name="Saludo")
        engine.runAndWait()
```

3.6 Pyautogui

Es una biblioteca para automatizar tareas en múltiples sistemas operativos. Entiéndase por Automatizar como controlar el mouse y el teclado, aunque esta librería incluye herramientas como cuadros de dialogo y capturas de pantalla. La API del módulo es bastante simple. Lamentablemente las funciones no siguen las nomenclaturas recomendadas por la P8P (guía de estilo del lenguaje)

- *size():* Obtiene el tamaño del monitor principal. Retorna una tupla (x, y) en unidades de pixeles.

- *Position():* Obtiene la posición del mouse. Retorna en una tupla(x, y) en unidades de pixeles.

- *moveTo(x, y, t):* Mueve el mouse de la posición actual a la posición (x, y) en t segundos.

- *click(x, y, n, i, tipo, tiempo):* Mueve el mouse de la posición actual a la posición (x, y) dando n clics de 'tipo' (primary, secondary), en un tiempo total 'tiempo' segundos. Si el argumento es nombre de una imagen, 'imagen.png' dará clic al centro de la imagen en ventana.

- *dragTo(x, t, tipo, tiempo):* Mueve el mouse de la posición actual a la posición (x, y) manteniendo presionado un tipo de clic 'tipo' (left, middle, right) durante un 'tiempo' segundos

- *write(str, i):* Escribe ,con una pausa de i segundos entre cada tecla, la cadena str.

- *alert(str):* Genera una alerta en 'box' pausando el programa hasta que se de clic 'ok'

Programa aplicativo:

```
import pyautogui
pyautogui.alert('Una vez cerrado esta ventana, se hara una espiral cuadrada\n con el mouse')

distance = 50
while distance > 0:
    pyautogui.drag(distance, 0, duration=0.5)   # move right
    distance -= 5
    pyautogui.drag(0, distance, duration=0.5)   # move down
    pyautogui.drag(-distance, 0, duration=0.5)  # move left
    distance -= 5
    pyautogui.drag(0, -distance, duration=0.5)  # move up
```

3.7 Wikipedia

Es una biblioteca básica que facilita el acceso y análisis de datos de Wikipedia. Cuando una persona quiere hacer una búsqueda usando el asistente, el programa llamara al método "buscarWikipedia", primero el asistente hablará diciendo "Dime que quieres que busque en Wikipedia" seguidamente escucha y lo guarda en la variable "buscar". En la línea 142, si se encuentra contenido la frase "busca en Wikipedia" en buscar entonces se removerá de la variable. Luego le pedimos a la biblioteca que busque el termino pedido, lo guardamos en resultado para finalmente el asistente lo hable en la línea 144.

Función Implementada:

```
136     def buscarWikipedia(self):
137         self.hablar("Dime que quieres que busque en Wikipedia:")
138         busca=self.escuchar()
139         busca = busca.replace('busca en wikipedia', '')
140         try:
141             wikipedia.set_lang('es')
142             busca = busca.replace('busca en wikipedia', '')
143             resultado = wikipedia.summary(busca, sentences=3)
144             self.hablar(f"Esto es lo que dice wikipedia: {resultado}")
145             return 0
146         except:
147             self.hablar("No entendi bien lo que quieres buscar, vuelve a intentarlo")
```

Ilustración 7

3.8 Tkinter

Python ofrece múltiples opciones para desarrollar GUI(interfaz gráfica de usuario). De todos los métodos de GUI, tkinter es el método más utilizado. Es una interfaz estándar de Python para el kit de herramientas Tk GUI que se envía con Python. Python con tkinter es la forma más rápida y sencilla de crear aplicaciones GUI. De todas las librerías anteriores está la podríamos considerar como la librería más importante, ya que es la que dará vida a nuestro asistente con la que podremos interactuar y sobre la cual se colocará las instrucciones deseadas.

Métodos Importantes

- tkinter.Tk(screenName=None, baseName=None, className='Tk', useTk=1). Construya un widget Tk de nivel superior, que suele ser la ventana principal de una aplicación, e inicializa un intérprete Tcl para este widget. Cada instancia tiene su propio intérprete Tcl asociado. La clase Tk normalmente se instancia utilizando todos los valores predeterminados. Sin embargo, actualmente se reconocen los siguientes argumentos de palabras clave.
  - screenname: Cuando se proporciona (como una cadena), establece la variable de entorno DISPLAY. (solo X11)
  - baseName: Nombre del archivo de perfil.
  - className: Nombre de la clase de widget.
  - useTk: Si es Verdadero, inicialice el subsistema Tk. La función tkinter.Tcl() establece esto en False..

- mainloop().Genera un bucle infinito que se usa para ejecutar la aplicación, esperar a que ocurra un evento y procesar el evento siempre que la ventana no esté cerrada.

Tkinter también ofrece acceso a la configuración geométrica de los widgets que pueden organizar los widgets en las ventanas principales. Hay principalmente tres clases de gestión de geometría.

- **Método pack()**: Organiza los widgets en bloques antes de colocarlos en el widget padre.
- **Método grid():** organiza los widgets en cuadrícula(estructura similar a una tabla) antes de colocarlos en el widget padre.
- **Método place():** Organiza los widgets colocándolos en posiciones específicas dirigidas por el programador.

Además, hay una serie de widgets que puede colocar en su aplicación tkinter tales como botones, entradas de texto, frames, etiquetas, paneles. Etc.

Para el desarrollo de nuestra aplicación usaremos principalmente los botones que activarán las funciones implementadas que activarán los métodos más resaltantes de todas las librerías definidas en los puntos anteriores, asimismo para el la configuración visual usaremos el método grid() por su sencillez y familiaridad de la estructura por cursos anteriores.

4. Estructura del Programa

Dadas las librerías que usaremos en el presente proyecto, veamos una estructura general del programa que nos ayudará a organizar el desarrollo del código fuente. Para ello nos apoyaremos los diagramas UML específicamente, en los diagramas de uso. Particularmente el diagrama de uso para este gadget será:

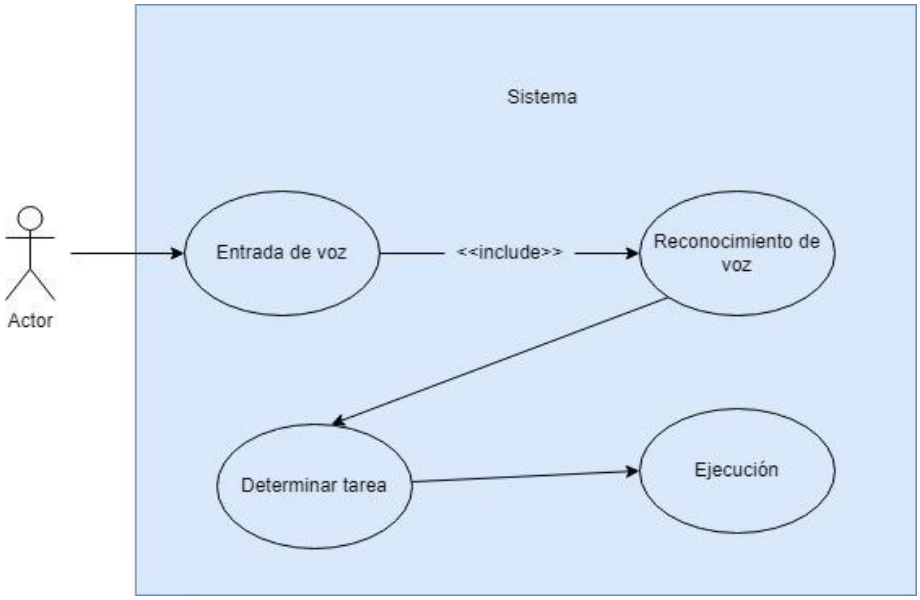


Ilustración 8

El actor es un usuario que a través de un micrófono le da una orden al programa, el programa recibe la entrada analógica e interpreta la tarea a realizar, finalmente lo ejecuta mostrando al usuario.

En primer lugar hay que tener en claro que para programar nuestro código si bien es cierto se puede realizar con una programación estructurada secuencial simple, es mucho mejor debido a la implementación de una interfaz con tkinter ya que podríamos generar

una clase myBot donde tendríamos en el método constructor todas las configuraciones iniciales de nuestra ventana, la división de la ventana mediante frames, la implementación de los botones que dan origen a nuestros métodos y algunas etiquetas que facilitaran la manipulación de nuestra aplicación. Una vez aclarado esto luego de importar todas las librerías a usar definimos nuestra clase myBot con todas las características anteriores como se puede observar a continuación.

```
14  class myBot:
15      def __init__(self,window):
16          self.bienvenida()
17          self.wind = window
18          self.wind.title('Mi asistente Virtual- ChatBOT')
19
20          # Frame Container
21          frame = LabelFrame(self.wind, text = '¿Qué quieres que diga?')
22          frame.grid(row=0, column=0,columnspan=3, pady=20)
23
24          # Imput
25          Label(frame,text='Ingrese una frase: ').grid(row=1, column=0)
26          self.name = Entry(frame)
27          self.name.focus() #parpadea el cursos
28          self.name.grid(row=1, column=1)
29
30          # Boton
31          ttk.Button(frame, text = 'Speak', command = lambda: self.hablar(self.name.get()) ).grid(row=2,columnspan=2,sticky = W+E)
32
33          self.img = ttk.PhotoImage(file='fondo.png')
34          ttk.Label(self.wind,image=self.img).grid(row=2,column=0)
35          ttk.Button(self.wind, text = 'Activar Micrófono', command = self.activarMicrofono ).grid(row=3,columnspan=2)
36
37          # Frame Container 2
38          frame2 = LabelFrame(self.wind, text = 'Indique alguna función')
39          frame2.grid(row=4, column=0,columnspan=3, pady=10)
```

A continuación se muestra la colocación de los botones con un pequeño texto que nos indica que es lo que se realiza, así como la implementación de la función definida para cada caso, como también otros aspectos como el contenedor sobre el cual se va a colocar, la posición relativa al frame, el espaciado total o parcial según la orientación.

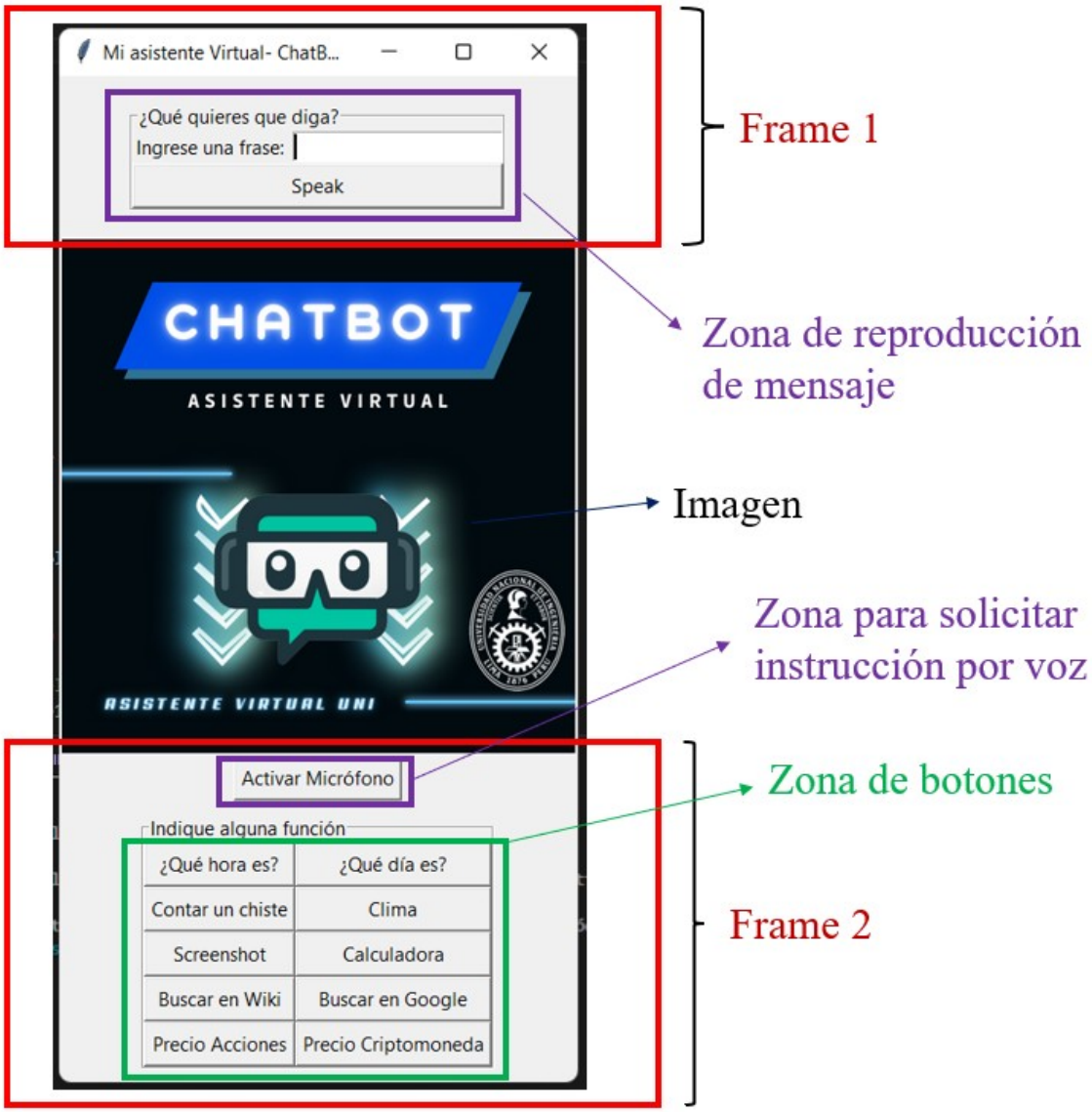
```
41          ttk.Button(frame2, text = '¿Qué hora es?', command = self.hora).grid(row=0,column=0,sticky=W+E)
42          ttk.Button(frame2, text = '¿Qué día es?', command = self.dia).grid(row=0,column=1,sticky=W+E)
43          ttk.Button(frame2, text = 'Contar un chiste', command = self.chiste).grid(row=1,column=0,sticky=W+E)
44          ttk.Button(frame2, text = 'Clima', command = self.clima).grid(row=1,column=1,sticky=W+E)
45          ttk.Button(frame2, text = 'Screenshot', command = self.capturarPantalla).grid(row=2,column=0,sticky=W+E)
46          ttk.Button(frame2, text = 'Calculadora', command = self.calculadora).grid(row=2,column=1,sticky=W+E)
47          ttk.Button(frame2, text = 'Buscar en Wiki', command = self.buscarWikipedia).grid(row=3,column=0,sticky=W+E)
48          ttk.Button(frame2, text = 'Buscar en Google', command = self.buscarGoogle).grid(row=3,column=1,sticky=W+E)
49          ttk.Button(frame2, text = 'Buscar en YB', command = self.buscarYB).grid(row=4,column=0,sticky=W+E)
50          ttk.Button(frame2, text = 'Precio Acciones', command = self.precioAcciones).grid(row=5,column=0,sticky=W+E)
51          ttk.Button(frame2, text = 'Precio Criptomoneda', command = self.precioCripto).grid(row=5,column=1,sticky=W+E)
```

Estos botones es una manera de como acceder a los métodos definidos previamente pero no es la única forma, también se ha implementada un botón llamado “Activar Micrófono”, este botón hará que se active el método “activarMicrofono”, en este método el asistente habla diciendo “Micrófono activado, indíqueme una instrucción” luego escucha y el resultado se guarda en forma de texto en la variable “instrucción”. Luego tratara de identificar palabras claves de la variable “instrucción”, si encuentra palabras claves entonces ejecutara los métodos mencionados anteriormente dependiendo de las condicionales, en caso de que no coincida con ninguna condicional el asistente dirá “No se pudo reconocer la voz, repita por favor”.

```python
    def activarMicrofono(self):
        self.hablar("Micrófono activado, indíqueme una instrucción")
        instruccion = self.escuchar()
        print(instruccion)
        try:
            if (instruccion.find("hora")>=0):
                self.hora()
            if (instruccion.find("día")>=0):
                self.dia()
            if (instruccion.find("chiste")>=0):
                self.chiste()
            if (instruccion.find("clima")>=0):
                self.clima()
            if (instruccion.find("captura")>=0 or instruccion.find( "screenshot")>=0):
                self.capturarPantalla()
            if (instruccion.find("calcula")>=0 or instruccion.find("calcular")>=0):
                self.calculadora()
            if (instruccion.find("wiki")>=0):
                self.buscarWikipedia()
            if (instruccion.find("google")>=0):
                self.buscarGoogle()
            if (instruccion.find("youtube")>=0):
                self.buscarYB()
            if (instruccion.find("acciones")>=0 or instruccion.find("acción")>=0):
                self.precioAcciones()
            if (instruccion.find("criptomoneda")>=0):
                self.precioCripto()
            if (instruccion.find("nombre")>=0 or instruccion.find("llamo")>=0):
                self.nombre()
        except:
            self.hablar("No se pudo reconocer la voz, repite por favor")
```

Esta otra forma de interactuar con el programa puede facilitar al usuario si en caso no pueda acceder a los botones o simplemente prefiere hacer uso de su voz dado el contexto en el que está, es más incluso el asistente podría ser mejorado y esté a la escucha en todo momento para que mediante un

Interfaz final:



5. Posibles mejoras a futuro

Si bien es cierto se mencionó que existe una segunda forma de interactuar con el programa mediante el botón “Activar Micrófono”, lo cual es una buena alternativa, pero el detalle aquí es que de todas formas cada vez que se necesite del bot tendremos que pulsar este botón ahora, para una futura versión sería mucho mejor que apenas se inicie el programa esté a la escucha en todo momento a una única palabra clave, para ello podríamos asignarle un nombre a nuestro asistente, y que este atenta a su llamado, al momento de que se verifique que el llamado fue hecho correctamente, se seguiría con el procedimiento como si la llamada activara el botón “Activar Micrófono” y prosigue con las instrucciones ya establecidas anteriormente.

Incluso el que la asistente sea llamado solo con la voz beneficiaría mucho si se implementase una gran variedad de librerías con nuevos métodos a usar, ya que añadir un botón tras otro saturaría mucho nuestra interfaz y haría que su aspecto sencillo se

pierda, sin mencionar que sería necesario más líneas de código para cada botón perdiendo la optimización de nuestro programa, por otro lado si se tratase de una orden hecha por voz habría solo cambios en el repertorio de palabras clave y la implementación del nuevo método con las nuevas características deseadas.

6. Dificultades encontradas

En un inicio no estaba tan claro sobre que lenguaje diseñar nuestro proyecto, estaba pensado en trabajarlo en C++ o en Java, pero al ver la variedad de librerías en Python y cantidad de información relacionada acerca de nuestro proyecto, optamos por incursionar sobre este lenguaje a pesar de tener muy poca practica sobre este en comparación de los lenguajes mencionados anteriores, lo cual se pudo lograr gracias al conocimiento ya adquirido por ciclos anteriores y la lógica de programación desarrollada en todo este tiempo. Asimismo el implementar estas nuevas librerías no fue una tarea sencilla, ya que algunas de ellas fueron hechas por terceros con una estructura fuertemente diseñada con algunos términos nuevos para nosotros, incluso algunas de estas no son muy conocidas entre la comunidad y la única fuente que hubo era de la página oficial, por ello fue un poco complicada la instalación e implementación del código en el IDE de Visual Studio, asimismo mediante pequeños programas aplicativos fue que se pudo comprender los métodos más resaltantes de cada librería para luego ser insertada en el código principal.

Por otro lado, para construir nuestra interfaz en Pyhton se pudo encontrar diversas opciones como las librerías Kivy, PyQY, PySide, entre otros, algunas de estas cuentan métodos poco difíciles de manejar, sin mencionar que algunas quedaron atrás tras pasar los años. Tras numerosas pruebas sobre diferentes librerías decidimos escoger la librería Tkinter ya que a parte de ser una librería interna del mismo lenguaje cuenta con mayor información y es por excelencia la mejor librería si se trata de inicar en este mundo de interfaces gráficas de usuario GUI.

7. Conclusiones

Notamos que agregarle funcionalidades al asistente es relativamente sencillo, sin embargo, cuando las instrucciones son mas complejas, la dificultad del método para que realice dicha acción aumenta. También notamos que la entrada de voz debe de darse de las mejores condiciones posibles, en caso contrario es muy probable que el asistente entienda otra cosa, o simplemente no se ejecute adecuadamente. Otra problemática es el tiempo que se encuentra activado el micrófono, que solo son 4 segundos, es decir que, si le damos una instrucción de 2 segundos, debemos esperar 2 segundos mas el tiempo de ejecución (que es mucho tiempo) pero si quisiéramos darle una entrada de voz mas largo el tiempo de escuchar no es suficiente. A pesar de estos inconvenientes nuestro Asistente Virtual es totalmente funcional siempre y cuando las condiciones sean adecuadas.

6. Referencias bibliográficas

[1] Steven F. Lott, Dusty Phillips. Python Object-Oriented Programming: Build robust and maintainable object-oriented Python application and libraries. 4th Edition 2021

[2] Vibha Tiwari. MFCC and its applications in speaker recognition. 2015

[3] B.H. Juang# & Lawrence R. Rabiner. Automatic Speech Recognition – A Brief History of the Technology Development. 2004

[4] Nitin Indurkhya, Fred J. Damerau. Manual de procesamiento del lenguaje natural. 2011

[5] Annabel Z. Dodd. The Essential Guide to Telecommunications. 2010

[6] Smigel, L. Yfinance Python tutorial (2022). Analyzing Alpha. https://analyzingalpha.com/yfinance-python. 2022, enero 11

[7] Introducción al módulo pywhatkit – Acervo Lima. (s/f). Acervolima.com. https://es.acervolima.com/introduccion-al-modulo-pywhatkit/. 1 de junio de 2022

[8] GeeksforGeeks. Python Script to create random jokes using pyjokes. https://www.geeksforgeeks.org/python-script-to-create-random-jokes-using-pyjokes/. 2020, December 2

[9] Pyjokes Society. (s/f). API reference. Pyjok. https://pyjok.es/api/. 1 de junio de 2022

[10] Texto a voz en Python. (s/f). Acervolima.com. Recuperado el, de https://es.acervolima.com/texto-a-voz-en-python-modulo-pyttsx/. 1 de junio de 2022

[11] (S/f). Cppsecrets.com. Recuperado el 1 de junio de 2022, de https://cppsecrets.com/users/121939710997110505653100101101112641031099710510846991111109/Python-pyttsx3-init-say-runAndWait.php

[12] Using pyttsx3 — pyttsx3 2.6 documentation. (s/f). https://pyttsx3.readthedocs.io/en/latest/engine.html. 29 de mayo de 2022

[13] Welcome to PyAutoGUI's documentation! — PyAutoGUI documentation. (n.d.). Retrieved, from https://pyautogui.readthedocs.io/en/latest/. May 29, 2022

[14] Wikipedia. (n.d.). PyPI. Retrieved, from https://pypi.org/project/wikipedia. May 29, 2022

[15] Wikipedia – Wikipedia 0.9 documentation. from https://wikipedia.readthedocs.io/en/latest/code.html. June 3, 2022