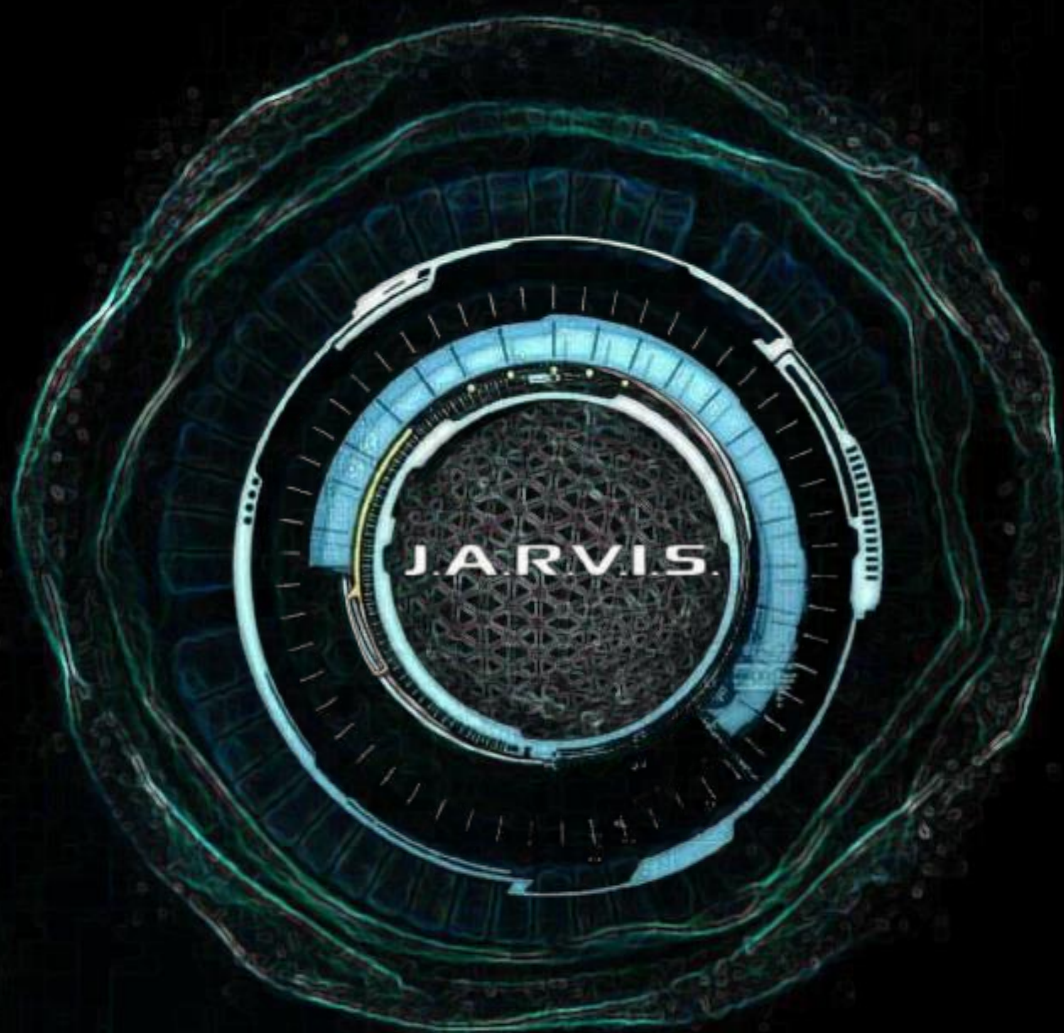




J.A.R.V.I.S.



Do Zero ao Jarvis: Criando Seu Assistente Personalizado

Carlos Garcia

Capítulo 01

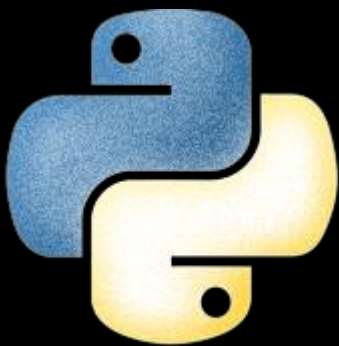
Um sonho até então distante

Olá, me chamo Carlos Garcia, um entusiasta dos filmes da Marvel, especialmente do Homem de Ferro e sua incrível inteligência artificial, o J.A.R.V.I.S que é um acrônimo para "Just a Rather Very Intelligent System". Desde a primeira vez que vi Tony Stark interagindo com seu assistente virtual, sonhei em ter um J.A.R.V.I.S. pessoal. No entanto, essa ideia parecia muito distante da minha realidade.

Com o passar dos anos, mergulhei no estudo da programação, participando de bootcamps na DIO, cursos em diversas instituições como Alura, Microsoft Learn, Open Academy, ADA Tech, dentre outras. Estou cursando Ciências de Dados na UNIVESP, além do curso técnico em Desenvolvimento de Sistemas na Fundação FAT. Todo esse aprendizado me permitiu reunir as peças necessárias para montar o quebra cabeça e iniciar este projeto ambicioso: recriar o J.A.R.V.I.S.

Neste e-Book, compartilho com você o resultado desse esforço. No momento que estou escrevendo estou com meu aplicativo na versão 1.0 do J.A.R.V.I.S. Embora seja uma versão simplista comparada ao original dos filmes, já me auxilia bastante no dia a dia e representa um passo significativo na realização de um sonho.

Este é apenas o começo. Pretendo continuar aprimorando e atualizando o meu J.A.R.V.I.S., e convido você a embarcar nessa jornada comigo. Vamos explorar juntos o fascinante mundo da inteligência artificial e descobrir como a programação pode transformar ideias em realidade.



Capítulo 02

Bibliotecas Utilizadas

Uma biblioteca em Python é uma coleção de módulos e pacotes que contém código reutilizável criado por um outro autor, desenvolvido para facilitar a implementação de funcionalidades específicas. As bibliotecas permitem que desenvolvedores acessem uma ampla gama de funcionalidades sem a necessidade de escrever código do zero.

As bibliotecas em Python são ferramentas essenciais que ajudam os desenvolvedores a criar aplicações robustas e eficientes com menos esforço, elas me ajudaram com diversas funcionalidades como o reconhecimento da fala, tanto para falar quanto para escrever o que é dito, dentre tantas outras funcionalidades que não precisei me preocupar durante o desenvolvimento, focando apenas na criação das funcionalidades do meu projeto.

Bibliotecas Utilizadas no Projeto:

1. Tkinter:

É a biblioteca padrão do Python para criar interfaces gráficas de usuário (GUI). Com ela você pode criar janelas, botões, caixas de texto e outros elementos de interface de maneira simples e eficaz.

2. Threading (importada como Thread):

Permite criar e controlar threads em Python.

Threads são unidades de execução que podem rodar simultaneamente. Com Thread, você pode executar tarefas em paralelo, o que é útil para operações que precisam rodar em segundo plano sem bloquear a execução do programa principal.

3. Speech_recognition:

É uma biblioteca para reconhecimento de fala em Python que permite que você converta áudio falado em texto. É compatível com várias APIs e mecanismos de reconhecimento de fala, como Google Web Speech API, Microsoft Bing Voice Recognition, e outros.

4. OS:

Fornece uma maneira de interagir com o sistema operacional. executando comandos do sistema, manipular arquivos e diretórios, obter informações sobre o ambiente de execução e muito mais. É uma biblioteca essencial para scripts que precisam interagir com o sistema operacional subjacente.

5. Pyttsx3:

É uma biblioteca para conversão de texto em fala (TTS) em Python. Ela permite que você converta texto em fala usando mecanismos TTS instalados no sistema. Funciona offline e é compatível com Windows, MacOS e Linux, permitindo que os programas Python "falem" com os usuários.

6. Google.generativeai:

É uma biblioteca para acessar as funcionalidades de inteligência artificial generativa da Google. Esta biblioteca pode ser usada para interagir com modelos de IA da Google que geram conteúdo, como texto, imagens e outros. É uma ferramenta poderosa para integrar capacidades de geração de conteúdo automatizado em seus projetos.

7. Datetime & Time:

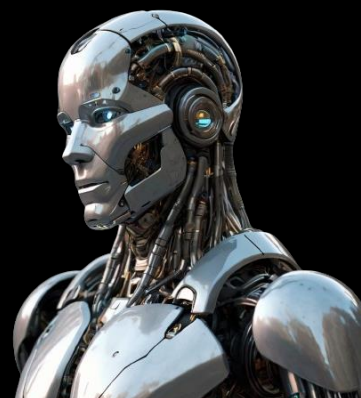
Fornece classes para manipular datas e horas. Criar, formatar, e manipular datas e horas. É útil para operações como calcular diferenças entre datas, formatar data/hora em strings específicas, e outras funcionalidades relacionadas a tempo.

8. Webbrowser:

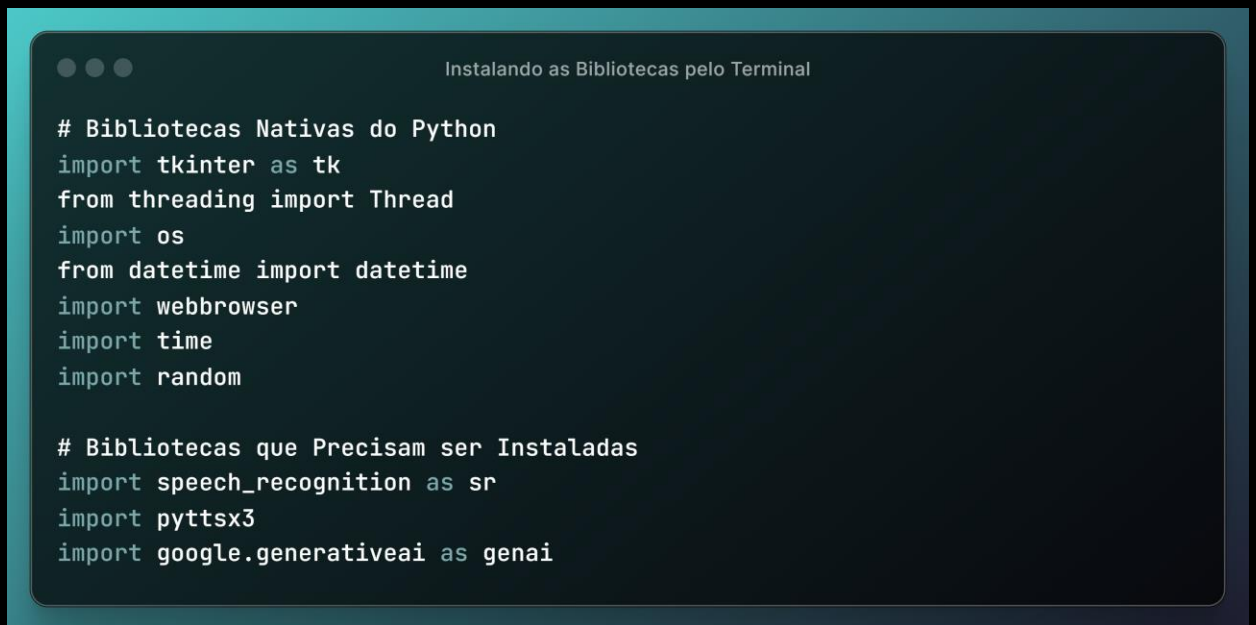
Fornece uma interface para exibir documentos web aos usuários. Com esta biblioteca você pode abrir URLs no navegador padrão do usuário diretamente a partir de seu script Python. É útil para automação de tarefas web ou para criar atalhos rápidos para páginas da internet.

6. Random:

Gera números pseudo-aleatórios. Com ela você pode realizar operações como escolher elementos aleatoriamente de uma lista, embaralhar sequências, gerar números aleatórios dentro de um intervalo específico, e outras funcionalidades que envolvem aleatoriedade.



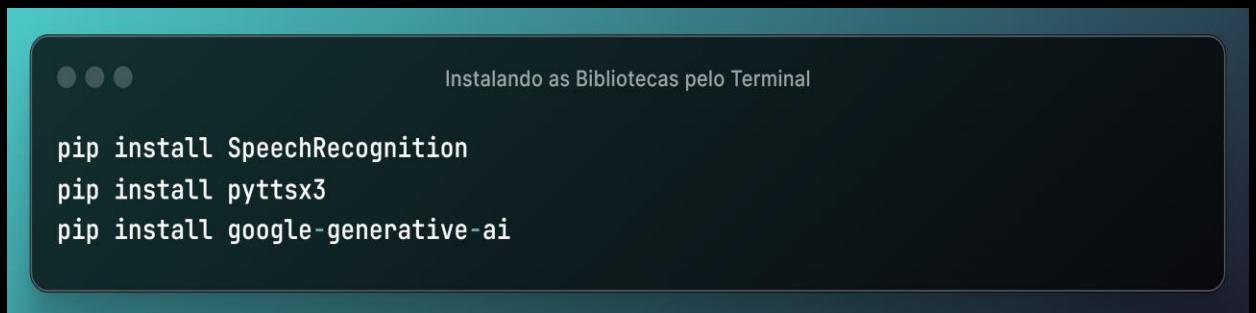
Abaixo, eu separei todas as bibliotecas usadas em duas listas. A primeira com bibliotecas nativas do Python e outra com bibliotecas externas utilizadas neste projeto, que necessitam de instalação, através do comando `pip install` via terminal:



```
● ● ● Instalando as Bibliotecas pelo Terminal

# Bibliotecas Nativas do Python
import tkinter as tk
from threading import Thread
import os
from datetime import datetime
import webbrowser
import time
import random

# Bibliotecas que Precisam ser Instaladas
import speech_recognition as sr
import pyttsx3
import google.generativeai as genai
```



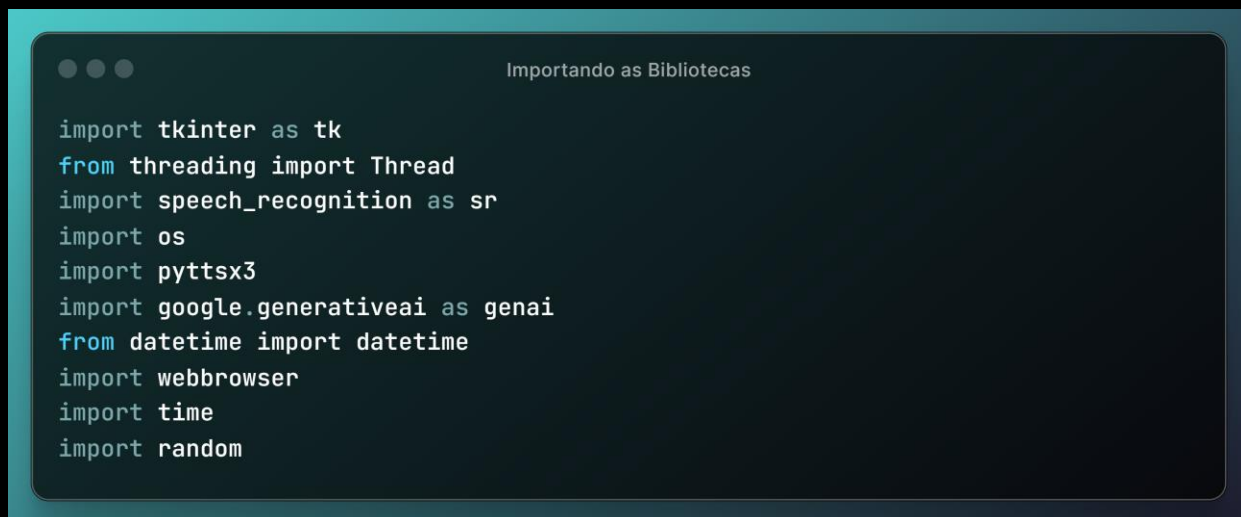
```
● ● ● Instalando as Bibliotecas pelo Terminal

pip install SpeechRecognition
pip install pyttsx3
pip install google-generative-ai
```


Capítulo 03

Falar é Fácil, Mostre-me o Código!

Importando as Bibliotecas:



```
import tkinter as tk
from threading import Thread
import speech_recognition as sr
import os
import pyttsx3
import google.generativeai as genai
from datetime import datetime
import webbrowser
import time
import random
```

- tkinter: Utilizada para criar interfaces gráficas;
- threading: Permite a execução de tarefas em paralelo, essencial para o funcionamento do assistente sem travar a interface;
- speech_recognition: Usada para captar e reconhecer comandos de voz;
- os: Permite executar comandos do sistema operacional, como abrir aplicativos;
- pyttsx3: Biblioteca para conversão de texto em fala;
- google.generativeai: Integração com a API de IA do Google para pesquisas avançadas;
- datetime: Para manipulação e formatação de datas e horas;
- webbrowser: Abre URLs no navegador padrão;
- time: Usada para operações de delay (pausas temporais);
- random: Para selecionar respostas de forma aleatória.

Classe JarvisApp - Inicialização (__init__)

```
Classe JarvisApp

class JarvisApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Jarvis")
        self.root.geometry("200x190")
        # Definindo as cores
        bg_color = "#006400"
        fg_color = "#000000"
        self.root.configure(bg=bg_color)

        self.label = tk.Label(root, text="Assistente de Voz - Carlos-CGS",
                               bg=bg_color, fg=fg_color)
        self.label.pack(pady=10)

        # Criar o círculo brilhante neon
        self.canvas = tk.Canvas(root, width=100, height=100, bg=bg_color,
                                highlightthickness=0)
        self.circle = self.canvas.create_oval(30, 30, 70, 70, outline="black",
                                                width=5)
        self.canvas.pack()

        self.btn_iniciar = tk.Button(root, text="Iniciar Assistente",
                                       command=self.iniciar_assistente, bg=fg_color, fg=bg_color)
        self.btn_iniciar.pack(pady=5)

        self.thread = None
        self.running = False
        self.listening = False
        self.speaking = False
```

Nesta parte do código eu inicio e defino todos os quesitos visuais do projeto:

- Configura a janela principal (self.root) com título e tamanho.
- Define a cor de fundo e de texto.
- Adiciona um rótulo (self.label) para exibir mensagens.
- Adiciona um Canvas com um círculo neon que será usado para indicar atividades.

- Adiciona um botão (self.btn_iniciar) para iniciar o assistente.
- Estados Iniciais:
- self.thread: Armazena a thread do assistente.
- self.running, self.listening, self.speaking: Flags para controlar o estado do assistente.

Iniciar Assistente (iniciar_assistente)

```
def iniciar_assistente(self):
    if not self.running:
        self.thread = Thread(target=self.executar_assistente)
        self.thread.start()
        self.running = True
        self.label.config(text="Assistente Executando...")
```

Nesta outra parte, eu inicio o assistente em uma nova thread, permitindo que o assistente funcione sem travar a interface gráfica. Funcionamento:

- Verifica se o assistente já está em execução.
- Cria uma nova thread para executar a def executar_assistente.
- Inicia a thread e atualiza o estado.

Executar Assistente (executar_assistente)

```
def executar_assistente(self):
    engine = pyttsx3.init()

    def falar(texto):
        mudar_cor_circulo("#00FF00")
        self.speaking = True
        engine.say(texto)
        engine.runAndWait()
        self.speaking = False
        mudar_cor_circulo("#00FFFF")
```

Nesta parte eu definir toda a lógica principal do assistente.

- Sub-função falar converter o texto em fala, permitir que o assistente comunique respostas e informações ao usuário.
- Funcionamento:
- Altera a cor do círculo para indicar que o assistente está falando.
- Utiliza pyttsx3 para converter o texto em fala.
- Restaura a cor do círculo após a fala.

Mudar Cor do Círculo (mudar_cor_circulo)

```
Função Mudar a Cor do Círculo da Interface Visual do Jarvis  
  
def mudar_cor_circulo(cor):  
    self.canvas.itemconfig(self.circle, outline=cor)
```

Esta função alterar a cor do contorno do círculo no Canvas, fornecendo um feedback visual ao usuário sobre o estado do assistente (por exemplo, quando está falando).



Ouvir Microfone (ouvir_microfone)

```
Função para Ouvir o Microfone

def ouvir_microfone(primeira_vez):
    microfone = sr.Recognizer()
    if primeira_vez:
        apresentacao()
        primeira_vez = False

    while self.running:
        with sr.Microphone() as source:
            microfone.adjust_for_ambient_noise(source)
            self.label.config(text="Esperando pelo comando 'Jarvis'...")
            audio = microfone.listen(source)

            try:
                frase = microfone.recognize_google(audio, language='pt-BR')
                print(f"Comando reconhecido: {frase}")

                respostas = [
                    "Como posso te ajudar?",
                    "Pois não?",
                    "Sim?",
                    "O que você precisa?",
                    "O que você deseja?",
                    "Em que posso ser útil?"
                ]
                if "Jarvis" in frase:
                    resposta = random.choice(respostas)
                    self.label.config(text=resposta)
                    falar(resposta)

                    with sr.Microphone() as source:
                        audio = microfone.listen(source)

                    try:
                        comando = microfone.recognize_google(audio,
language='pt-BR')
                        ...
                    except sr.UnknownValueError:
                        pass
            return False
```

Neste trecho eu capturo e reconhecer comandos de voz do usuário, gerando assim a interação principal com o usuário. Funcionamento:

- Ajusta o microfone para ruído ambiente.
- Aguarda o comando de ativação ("Jarvis").
- Ao reconhecer o comando, responde com uma das respostas aleatórias.

Animar Círculo (animar_circulo)

```
Função para Animar o Círculo

def animar_circulo():
    if self.speaking:
        for i in range(5, 18, 5):
            if not self.speaking:
                break
            self.canvas.coords(self.circle, 20 - i, 20 - i, 80 + i, 80 + i)
            self.canvas.update()
            time.sleep(0.05)
        for i in range(17, 4, -5):
            if not self.speaking:
                break
            self.canvas.coords(self.circle, 20 - i, 20 - i, 80 + i, 80 + i)
            self.canvas.update()
            time.sleep(0.05)
    self.root.after(47, animar_circulo)
```

Nesta função eu crio uma animação no círculo enquanto o assistente está falando, melhorando a interação visual com o usuário, indicando que o assistente está ativo e processando informações.

Apresentação (apresentacao)

```
Função Apresentação - Fala inicial do Jarvis

def apresentacao():
    hora_atual = datetime.now().hour
    animar_circulo()
    if 0 <= hora_atual < 12:
        saudacao = "Bom dia"
    elif 12 <= hora_atual < 18:
        saudacao = "Boa tarde"
    else:
        saudacao = "Boa noite"
    falar(f"{saudacao} Carlos. Hoje é {datetime.now().strftime('%d/%m/%Y')}, são {datetime.now().strftime('%H:%M')}. Me chamo Jarvis.")
    self.label.config(text="Se precisar de algo, fale meu nome....")
    falar("Se precisar de algo, fale meu nome.")
```


A função apresentação fornece uma introdução ao iniciar o assistente, cumprimentar o usuário com base na hora do dia e fornecer informações atuais.

Funcionamento:

- Obtém a hora atual e determina a saudação adequada (bom dia, boa tarde, boa noite).
- Utiliza a função falar para comunicar a saudação e a data/hora atual.

Comando "pesquisar"

```
Comando Pesquisar

if "pesquisar" in comando:
    pesquisa = comando.split("pesquisar", 1)[1].strip()
    if pesquisa:
        self.label.config(text="Um momento, estou pesquisando.")
        falar("Um momento, estou pesquisando.")
        resposta = pesquisar_no_google(f"Resposta em forma discursiva, variando
entre 20 e 40 palavras dependendo da complexidade da pergunta, falando da mesma
forma que o Jarvis a ia do homem de ferro, sobre o seguinte assunto:
{pesquisa}")
        falar(resposta)
    else:
        self.label.config(text="Sobre o que deseja saber?")
        falar("Sobre o que deseja saber?")
        with sr.Microphone() as source:
            audio = microfone.listen(source)
        try:
            pesquisa = microfone.recognize_google(audio, language='pt-BR')
            self.label.config(text="Só um momento, estou pesquisando.")
            falar("Só um momento, estou pesquisando.")
            resposta = pesquisar_no_google(f"Resposta em forma discursiva,
variando entre 20 e 40 palavras dependendo da complexidade da pergunta, falando
da mesma forma que o Jarvis a ia do homem de ferro, sobre o seguinte assunto:
{pesquisa}")
            falar(resposta)
        except sr.UnknownValueError:
            self.label.config(text="Não entendi, repita o que deseja.")
            falar("Não consegui entender a pesquisa.")
```

O Comando pesquisara permite que o assistente faça pesquisas na web quando o usuário diz "pesquisar", fornecer respostas informativas e úteis sobre qualquer assunto que o usuário queira saber.

Verificação do Comando:

- O código verifica se a palavra "pesquisar" está presente no comando do usuário.
- Se "pesquisar" estiver presente, o comando é dividido a partir dessa palavra, e o resto do comando é considerado como o termo de pesquisa.
- Checagem do Termo de Pesquisa:
- Se o termo de pesquisa não estiver vazio, o assistente configura a interface para indicar que está pesquisando e chama a função `pesquisar_no_google` para obter uma resposta sobre o assunto.
- Se o termo de pesquisa estiver vazio, o assistente pede ao usuário para especificar sobre o que deseja saber e escuta novamente.

Pesquisa e Resposta:

- A função `pesquisar_no_google` é chamada para buscar uma resposta detalhada sobre o assunto.
- A resposta é falada pelo assistente.
- Tratamento de Erros:
- Caso o assistente não consiga entender o comando ou a pesquisa, uma mensagem de erro é mostrada e falada, solicitando ao usuário que repita.

Comando "tocar música"

```
Comando Tocar Música

elif "tocar música" in comando:
    reproduzir_musica(microfone)
```

Este elif tocar uma música específica ou uma playlist, acionando a função reproduzir_musica.

Comando "Abrir [aplicativo]"

```
Comando Abrir Aplicativo

elif "Abrir navegador" in comando:
    os.system("start chrome.exe")
    falar("Abrindo o navegador")
# Outros aplicativos seguem a mesma estrutura
```

Este elif executa a abertura de aplicativos do Windows utilizando a biblioteca OS, como navegador, calculadora, Paint, bloco de notas, Excel, Word, CMD, VS Code.

Comando "Concertar Internet"

```
Comando Concertar Internet

elif "Concertar internet" in comando:
    falar("Iniciando o solucionador de problemas de rede")
    os.system("msdt.exe /id NetworkDiagnosticsNetworkAdapter")
```

Este elif iniciar o solucionador de problemas de rede.

Comando "Que horas são & Que dia é hoje"

```
Comando Que Horas São & Dia é Hoje

elif "Que horas são" in comando:
    hora_atual = datetime.now().strftime("%H:%M")
    falar(f"Agora são {hora_atual}.")
elif "Que dia é hoje" in comando:
    data_atual = datetime.now().strftime("%d/%m/%Y")
    falar(f"Hoje é {data_atual}.")
```

Este elif Informar a hora atual e a data atual.

Comando "transcrever"

```
Comando Transcrever

elif "transcrever" in comando:
    falar("Qual mensagem você gostaria de salvar?")
    with sr.Microphone() as source:
        audio = microfone.listen(source)
    try:
        mensagem = microfone.recognize_google(audio, language='pt-BR')
        caminho_arquivo = r"C:\Users\CARLOS GARCIA\Desktop\mensagens_salvas.txt"

        with open(caminho_arquivo, "a") as arquivo:
            arquivo.write(f"{datetime.now().strftime('%d/%m/%Y %H:%M:%S')} - {mensagem}\n")
        falar("Mensagem salva com sucesso.")
    except sr.UnknownValueError:
        falar("Não consegui entender a mensagem. Tente novamente.")
```

Este elif transcrever uma mensagem falada pelo usuário e salvá-la em um arquivo de texto, permitir ao usuário guardar notas e mensagens importantes de forma prática.

Comando “Desligar Sistema”

```
Comando Desligar Sistema

elif "desligar sistema" in comando:
    falar("Finalizando o sistema. Até logo.")
    self.running = False
    self.label.config(text="Assistente Desligado.")
```

Este elif desligar o assistente, permitindo ao usuário encerrar a sessão do assistente de maneira controlada.

Comando “Não Reconhecido”

```
Comando Não Reconhecido

else:
    falar(f"O comando {comando} não foi reconhecido, tente novamente.")
```

Finalizo a estrutura de decisão com este else, que informar ao usuário que o comando não foi reconhecido.

Tratamento de Erros Gerais

```
Tratamentos de Erros Gerais

except sr.UnknownValueError:
    self.label.config(text="Não entendi, repita o que deseja.")
    falar("Não entendi, repita o que deseja.")
```

Utilizo esse exception para capturar e lidar com erros de reconhecimento de fala, garantindo que o assistente possa continuar funcionando mesmo se encontrar problemas ao tentar entender o usuário.

Reproduzir Música (reproduzir_musica)

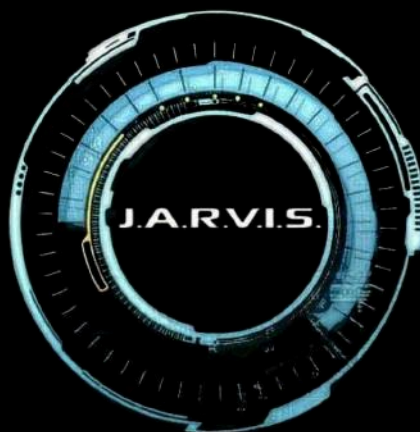
Função Reproduzir Musica

```
def reproduzir_musica(microfone):  
    falar("Qual música você deseja ouvir?")  
    with sr.Microphone() as source:  
        audio = microfone.listen(source)  
    try:  
        musica = microfone.recognize_google(audio, language='pt-BR')  
        if not musica:  
            falar("Por favor, diga o nome da música.")  
            with sr.Microphone() as source:  
                audio = microfone.listen(source)  
                musica = microfone.recognize_google(audio, language='pt-BR')  
        musica_formatada = musica.replace(" ", "+")  
        url_youtube = f"https://www.youtube.com/results?search_query={musica_formatada}"  
        webbrowser.open(url_youtube)  
    except sr.UnknownValueError:  
        falar("Desculpe, não consegui entender. Por favor, repita.")
```

Reproduzir músicas do YouTube pelo navegador

Funcionamento:

- Solicita o nome da música ao usuário.
- Formata a string de busca para o YouTube.
- Abre o navegador com a busca da música no YouTube.



Pesquisar no Google (pesquisar_no_google)

```
Pesquisa API Google

def pesquisar_no_google(pesquisa):
    GOOGLE_AI_KEY = "Cole_sua_chave_api_aqui"
    genai.configure(api_key=GOOGLE_AI_KEY)
    # Configurando a Temperatura das Respostas
    configurar_geracao = {
        "candidate_count": 1,
        "temperature": 0.9,
    }
    # Configurando os níveis de segurança das respostas (Ofensivas, Raciais,
    Sexuais, etc.)
    configurar_seguranca = {
        "HARASSMENT": "BLOCK_NONE",
        "HATE": "BLOCK_NONE",
        "SEXUAL": "BLOCK_NONE",
        "DANGEROUS": "BLOCK_NONE",
    }
    # Definindo o Modelo Usado para Pesquisa
    model = genai.GenerativeModel(model_name="gemini-1.0-pro",
                                   generation_config=configurar_geracao,
                                   safety_settings=configurar_seguranca)
    # Configurando o Histórico de Pesquisa
    chat = model.start_chat(history=[])
    response = chat.send_message(pesquisa)
    return response.text
```

Neste trecho do código eu defino todos os parâmetros e configurações para conexão com a API do Google utilizando a API GenAI, permitindo que o assistente forneça informações pesquisadas ao usuário.

Funcionamento:

- Configura a chave de API do Google e as configurações de geração e segurança.
- Inicializa o modelo de geração gemini-1.0-pro.
- Inicia uma conversa no modelo sem histórico.
- Envia a pesquisa como mensagem e retorna

Loop do Assistente JARVIS

```
Loop do Assistente JARVIS

# Loop do Assistente Jarbas
primeira_vez = True
while self.running:
    if ouvir_microfone(primeira_vez):
        break
    primeira_vez = False
```

Este loop serve para manter o assistente de voz ativo, continuamente ouvindo e respondendo a comandos do usuário. A primeira vez que o assistente é ativado, ele realiza uma apresentação. Após isso, ele escuta e responde a comandos até que seja instruído a parar (por exemplo, pelo comando de desligar o sistema).

Interface Gráfica

```
Iniciando Loop Infinito da Interface Gráfica

# Criar a Interface Gráfica
root = tk.Tk()
app = JarvisApp(root)
root.mainloop()
```

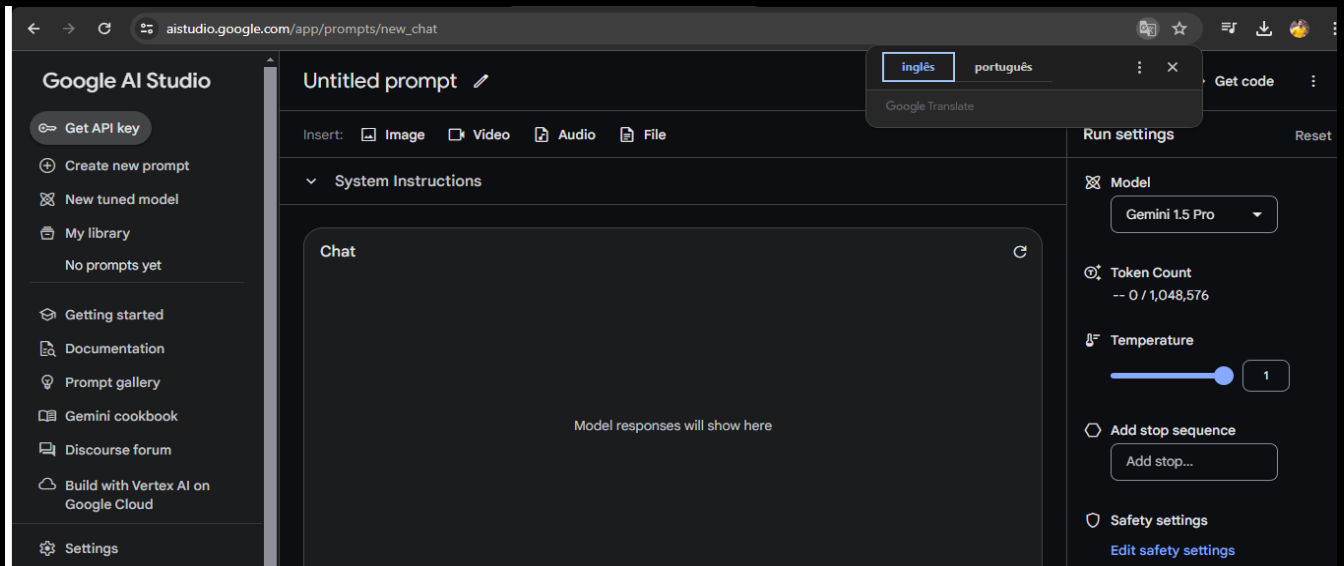
Na parte final, eu inicializar a interface gráfica do assistente.
Funcionamento:

- Cria uma nova janela tkinter.
- Inicializa a classe JarvisApp passando a janela como argumento.

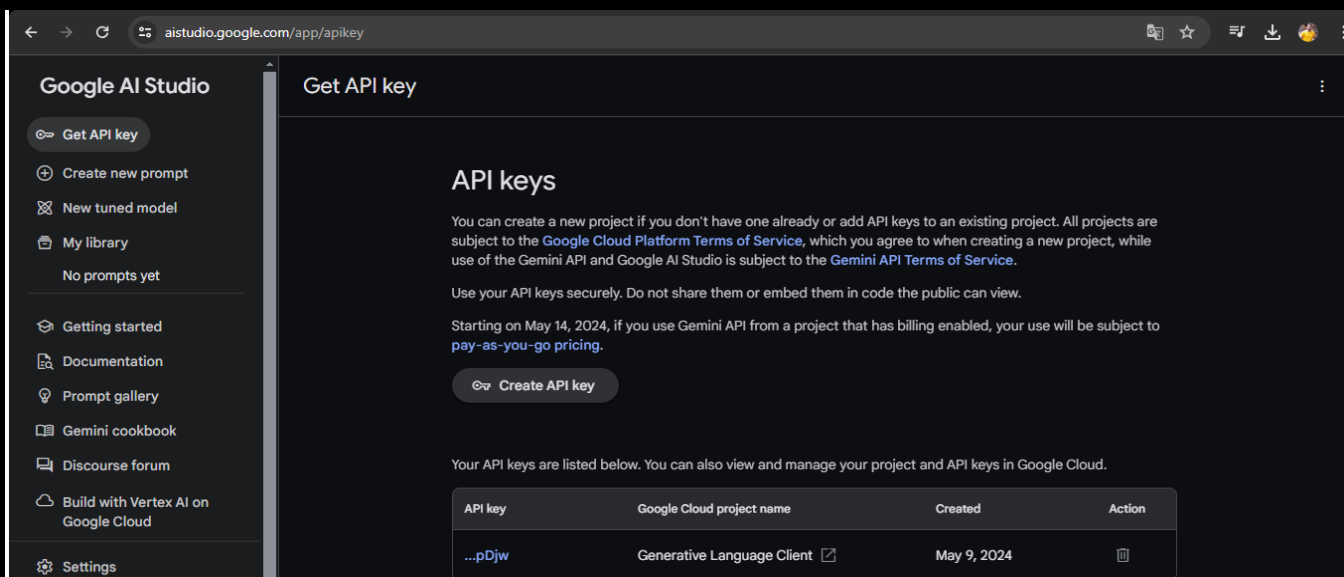
Como Gerar sua Própria API_KEY

Para utilizar os serviços do Google, incluindo o Google Generative AI, você precisa de uma chave de API. Aqui vou te mostrar de forma prática como gerar sua chave de API no Google AI Studio:

- Acesse o Console do Google AI Studio
- Primeiro, acesse o Console do Google AI Studio https://aistudio.google.com/app/prompts/new_chat

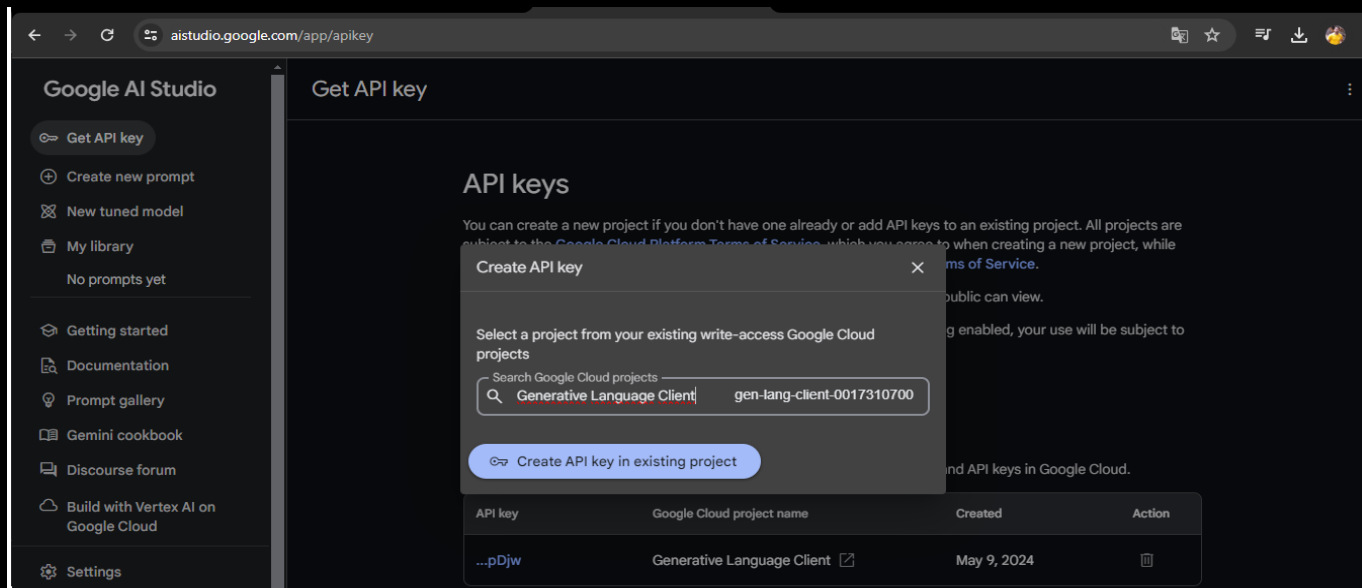


- Clique em Get API Key
- Selecione o botão do lado esquerdo superior com o desenho



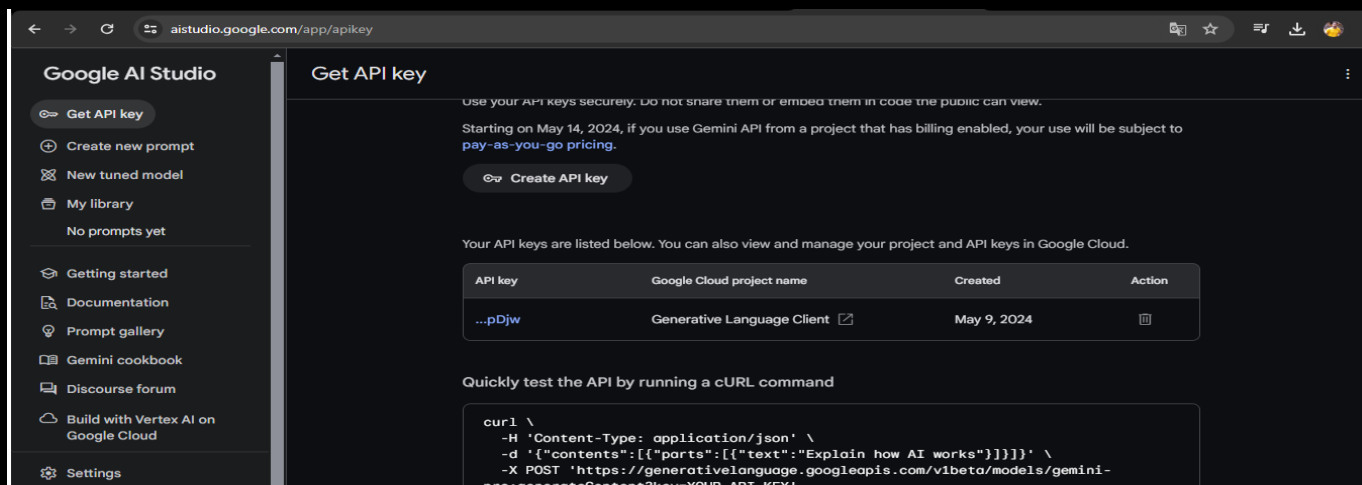
Crie sua API Key

Por fim clique em Create API Key in existing Project, ou caso não tenha nenhum projeto criado, crie um projeto e logo após crie sua API Key



Acessando sua API Key

- Você poderá acessar sua API Key futuramente voltando neste mesmo local, onde ela ficará disponível como na imagem abaixo:



ATENÇÃO - Nunca divulgue suas chave de API, não compartilhando com ninguém, pois com elas em mãos qualquer um poderá acessar seus recursos no Google Cloud Platform.

Conclusão

Para finalizar, espero ter contribuído com a sua jornada e ter lhe inspirado a prosseguir em sua trajetória como programador, pois quando criamos códigos que realmente fazem sentido em nossas vidas e nos desafiam no dia a dia em sua implementação. Quando o superamos e resolvemos, acontece algo mágico, como se nossa bateria é totalmente recarregada e continuamos firmes em nossas caminhada.

Estou trabalhando em novas implementações para este código, para que algum dia ele seja 1% do JARVIS real do filme homem de ferro, facilitando cada vez mais meu dia a dia através de auxílios executados por comando de voz.

Vamos Disseminar os Conhecimentos e Transbordar Tudo o que Aprendemos!



Segue acima o link para acessar todo o código fonte deste projeto em um repositório meu no GitHub, assim como meu LinkedIn.

Carlos-CGS
Carlos Garcia