Prabhjot Singh

IT-229-A

December6,2024

Project Documentation: Jarvis Virtual Assistant

1. Introduction

Overview:
This project is a voice-activated assistant named "Jarvis," which integrates multiple functionalities to assist users in performing various tasks. The assistant can play music and anything on Youtube, search on Google, open applications, and interact with users in natural language. The project uses Python for backend logic, with HTML, CSS, Bootstrap and JavaScript for the frontend interface, and Eel for connecting both the backend and frontend.

Purpose:
The purpose of this project is to provide an intelligent assistant that allows users to perform daily tasks through voice commands, such as searching for information and providing us as much we want by being adaptable, opening applications, playing videos, and interacting with web platforms. The project aims to provide an interactive and user-friendly way to interface with the computer.

Main Features:

- **Voice Recognition**: Understands spoken commands and responds accordingly.
- **YouTube Video Playback**: Automatically plays the video related to the search term on YouTube.
- **Web Search**: Executes web searches on Google for any query.(eg opening specific person LinkedIn on google)
- **Application Launch**: Opens applications installed on the system via voice commands.
- **Interactive Frontend**: Web interface using HTML, CSS,Bootsrap, and JavaScript to provide real-time feedback to users.
-

2. Explanation and Details

How the Project Works:

1. **Voice Command Input**:
   The user speaks a command, which is captured by a microphone. The speech is then converted into text using Google's Speech Recognition API.
2. **Command Processing**:
   The backend (Python) processes the command and performs the appropriate action (e.g., playing a video, opening an application, or performing a search).
3. **Frontend Interaction**:
   The frontend is a web interface built using HTML, CSS,Bootsrap,and JavaScript. It provides real-time feedback to the user, such as showing the spoken command or displaying search results.
4. **Backend Operations**:
   The backend uses several libraries to perform tasks such as:
   - **Playsound=1.2.2** for voice assistant
   - **Pyaudio** to recognise the voice command
   - **Speech Recognition** to recognise and convert into text
   - **pywhatkit** to play YouTube videos.
   - **web browser** to perform Google searches.
   - **os** to open applications installed on the system.
   - **pyttsx3** for text-to-speech functionality.
   - **Groq** as a API for providing AI functionalities
   - **Time** to give little delay in automation and searches
   - **Eel** to connect backend(python) to the frontend(HTML,CSS,Javascript)
5. **Communication Between Backend and Frontend**:
   The frontend and backend communicate using **Eel**, a Python library that allows for the creation of desktop apps with web technologies.

3. Setup and Libraries

To successfully run this project, several Python libraries need to be installed. Follow the steps below to set up the environment and install the required libraries.

Step1:Install Python

For macOS:

1. **Download Python from the official website:**
    - **Visit the Python Downloads page and download the latest version for macOS.**
    - **The download will provide a .pkg installer.**
2. **Install Python:**
    - **Open the .pkg file and follow the installation instructions.**
3. **Verify the installation:**
    - **Open the Terminal and check the Python version by typing:**

Try:

```
bash

python --version
```

If the above command doesn't work, try:

Then try this:

```
bash

python3 --version
```

4-Make sure you are using Python version to ensure compatibility with the libraries.

For Windows:

1. **Download Python:**

- **Visit the Python Downloads page and download the latest version for Windows.**

2. **Run the installer:**
   - **Double-click the downloaded .exe file.**
   - **Ensure to check the box that says "Add Python to PATH" before clicking "Install Now".**
3. **Verify the installation:**
   - **Open Command Prompt (press Win + R, type cmd, and hit Enter).**
   - **Check the Python version by typing:**

Try:

```
bash

python --version
```

If the above command doesn't work, try:

Then try this:

```
bash

python3 --version
```

4-Make sure you are using Python version to ensure compatibility with the libraries.

Step 2: Install Required Libraries

Once your Python version is confirmed, you can proceed with installing the necessary libraries. Use pip (if you have python normal version)or pip3(if you have python3 version) to install the libraries as needed, depending on your Python setup.

1. **playsound (v1.2.2):** This library is used to play sound files, enabling the voice assistant to produce audio cues and responses.

Installation command:

2. **pyaudio:** pyaudio enables voice command recognition by capturing audio input from the user's microphone.

Installation command:

```
(.venv) 10-3-31-237:Jarvis prabhsandhu$ pip install pyaudio
Collecting pyaudio
  Using cached PyAudio-0.2.14-cp310-cp310-macosx_15_0_arm64.whl
Installing collected packages: pyaudio
Successfully installed pyaudio-0.2.14
```

3. **SpeechRecognition:** This library is used to convert spoken words into text, allowing the voice assistant to understand and process user commands.

Installation command:

```
(.venv) 10-3-31-237:Jarvis prabhsandhu$ pip install SpeechRecognition
Collecting SpeechRecognition
  Using cached SpeechRecognition-3.11.0-py2.py3-none-any.whl.metadata (28 kB)
Requirement already satisfied: requests>=2.26.0 in ./.venv/lib/python3.10/site-packages (from SpeechRecognition) (2.32
.3)
Requirement already satisfied: typing-extensions in ./.venv/lib/python3.10/site-packages (from SpeechRecognition) (4.1
2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in ./.venv/lib/python3.10/site-packages (from requests>=2.26.0
->SpeechRecognition) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in ./.venv/lib/python3.10/site-packages (from requests>=2.26.0->SpeechReco
gnition) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./.venv/lib/python3.10/site-packages (from requests>=2.26.0->Spee
chRecognition) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in ./.venv/lib/python3.10/site-packages (from requests>=2.26.0->Spee
chRecognition) (2024.8.30)
Using cached SpeechRecognition-3.11.0-py2.py3-none-any.whl (32.8 MB)
Installing collected packages: SpeechRecognition
Successfully installed SpeechRecognition-3.11.0
```

4. **pywhatkit:** pywhatkit is used to play YouTube videos by sending commands directly to the platform, based on voice or text input.

Installation command:

```
(.venv) 10-3-31-237:Jarvis prabhsandhu$ pip install pywhatkit
Requirement already satisfied: pywhatkit in ./.venv/lib/python3.10/site-packages (5.4)
Requirement already satisfied: Pillow in ./.venv/lib/python3.10/site-packages (from pywhatkit) (11.0.0)
Requirement already satisfied: pyautogui in ./.venv/lib/python3.10/site-packages (from pywhatkit) (0.9.54)
Requirement already satisfied: requests in ./.venv/lib/python3.10/site-packages (from pywhatkit) (2.32.3)
Requirement already satisfied: wikipedia in ./.venv/lib/python3.10/site-packages (from pywhatkit) (1.4.0)
Requirement already satisfied: Flask in ./.venv/lib/python3.10/site-packages (from pywhatkit) (3.1.0)
Requirement already satisfied: Werkzeug>=3.1 in ./.venv/lib/python3.10/site-packages (from Flask->pywhatkit) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in ./.venv/lib/python3.10/site-packages (from Flask->pywhatkit) (3.1.4)
Requirement already satisfied: itsdangerous>=2.2 in ./.venv/lib/python3.10/site-packages (from Flask->pywhatkit) (2.2
0)
Requirement already satisfied: click>=8.1.3 in ./.venv/lib/python3.10/site-packages (from Flask->pywhatkit) (8.1.7)
```

5. **pyttsx3:** This library provides text-to-speech functionality, converting text input into spoken words and enhancing the user experience.

Installation command:

```
(.venv) 10-3-31-237:Jarvis prabhsandhu$ pip install pyttsx3
Collecting pyttsx3
  Using cached pyttsx3-2.98-py3-none-any.whl.metadata (3.8 kB)
Collecting pyobjc>=2.4 (from pyttsx3)
  Using cached pyobjc-10.3.1-py3-none-any.whl.metadata (26 kB)
Requirement already satisfied: pyobjc-core==10.3.1 in ./.venv/lib/python3.10/site-packages (from pyobjc>=2.4->pyttsx3)
 (10.3.1)
Collecting pyobjc-framework-AddressBook==10.3.1 (from pyobjc>=2.4->pyttsx3)
  Using cached pyobjc_framework_AddressBook-10.3.1-cp36-abi3-macosx_11_0_universal2.whl.metadata (2.4 kB)
Collecting pyobjc-framework-AppleScriptKit==10.3.1 (from pyobjc>=2.4->pyttsx3)
  Using cached pyobjc_framework_AppleScriptKit-10.3.1-py2.py3-none-any.whl.metadata (2.3 kB)
Collecting pyobjc-framework-ApplicationServices==10.3.1 (from pyobjc>=2.4->pyttsx3)
  Using cached pyobjc_framework_ApplicationServices-10.3.1-cp310-cp310-macosx_10_9_universal2.whl.metadata (2.6 kB)
Collecting pyobjc-framework-Automator==10.3.1 (from pyobjc>=2.4->pyttsx3)
  Using cached pyobjc_framework_Automator-10.3.1-cp36-abi3-macosx_11_0_universal2.whl.metadata (2.5 kB)
Collecting pyobjc-framework-CFNetwork==10.3.1 (from pyobjc>=2.4->pyttsx3)
  Using cached pyobjc_framework_CFNetwork-10.3.1-cp36-abi3-macosx_11_0_universal2.whl.metadata (2.5 kB)
Requirement already satisfied: pyobjc-framework-Cocoa==10.3.1 in ./.venv/lib/python3.10/site-packages (from pyobjc>=2.
4->pyttsx3) (10.3.1)
Collecting pyobjc-framework-CoreAudio==10.3.1 (from pyobjc>=2.4->pyttsx3)
```

6. **webbrowser:** webbrowser is used to facilitate Google searches by automatically opening the browser and performing searches directly from within the application. This library is built-in with Python, so you don't need to install it.

Usage:

```
python

import webbrowser
```

7. **os:** The os library allows the system to open and manage applications installed on the user's device, enabling operations like launching software or opening files. This is another built-in Python library, so no installation is required.

Usage:

```
python

import os
```

8. **groq:** groq integrates AI functionalities through an API, allowing the application to perform tasks such as answering questions or providing insights based on natural language inputs.

Installation command:

9. **time:** The time library introduces small delays between automated operations, ensuring smooth execution and preventing system overload. This is also a built-in Python library, so no installation is required

Usage:

```
python

import time
```

10. **eel:** eel is a Python library used to connect the backend (Python) with the frontend (HTML, CSS, JavaScript), allowing for a seamless user interface experience with real-time interaction.

Installation command:

```
(envjarvis) 10-3-31-237:Jarvis prabhsandhu$ pip3 install eel
Requirement already satisfied: eel in ./envjarvis/lib/python3.10/site-packages (0.17.0)
Requirement already satisfied: bottle in ./envjarvis/lib/python3.10/site-packages (from eel) (0.13.2)
Requirement already satisfied: bottle-websocket in ./envjarvis/lib/python3.10/site-packages (from eel) (0.2.9)
Requirement already satisfied: future in ./envjarvis/lib/python3.10/site-packages (from eel) (1.0.0)
Requirement already satisfied: pyparsing in ./envjarvis/lib/python3.10/site-packages (from eel) (3.2.0)
Requirement already satisfied: whichcraft in ./envjarvis/lib/python3.10/site-packages (from eel) (0.6.1)
Requirement already satisfied: gevent-websocket in ./envjarvis/lib/python3.10/site-packages (from bottle-websocket->eel
) (0.10.1)
Requirement already satisfied: gevent in ./envjarvis/lib/python3.10/site-packages (from gevent-websocket->bottle-websoc
ket->eel) (24.10.3)
Requirement already satisfied: zope.event in ./envjarvis/lib/python3.10/site-packages (from gevent->gevent-websocket->b
ottle-websocket->eel) (5.0)
Requirement already satisfied: zope.interface in ./envjarvis/lib/python3.10/site-packages (from gevent->gevent-websocke
t->bottle-websocket->eel) (7.1.1)
Requirement already satisfied: greenlet>=3.1.1 in ./envjarvis/lib/python3.10/site-packages (from gevent->gevent-websock
et->bottle-websocket->eel) (3.1.1)
Requirement already satisfied: setuptools in ./envjarvis/lib/python3.10/site-packages (from zope.event->gevent->gevent-
websocket->bottle-websocket->eel) (74.1.2)
```

4-Some most Important functions with code-:

1-Voice Command Recognition Function

Code:

```python
def takecommand():

    r = sr.Recognizer()

    with sr.Microphone() as source:
        print('listening....')
        eel.DisplayMessage('listening....')
        r.pause_threshold = 1
        r.adjust_for_ambient_noise(source)

        audio = r.listen(source, timeout=15, phrase_time_limit=10)


    try:
        print('recognizing')
        eel.DisplayMessage('recognizing')
        query = r.recognize_google(audio, language='en-us')
        print(f"user said: {query}")
        eel.DisplayMessage(query)
        time.sleep(2)



    except Exception as e:
        return ""

    return query.lower()
```

Explanation:

What it does:
This function uses the speech_recognition library to enable the assistant to listen to the user's voice input through a microphone, convert it into text, and process it as a command. First, a sr.Recognizer() object is created to handle the speech recognition process.
The pause_threshold is set to 1 seconds, which means that if there is a pause in speech longer than 1 seconds, the recognizer will stop listening and process the input. The microphone is used as the audio source, and the recognizer adjusts for ambient noise to ensure accurate speech detection. The assistant listens to the user's speech using r.listen(), captures the audio, and then sends it to Google's speech recognition service via the recognize_google() method to transcribe the speech into text. If successful, the transcribed text is printed; otherwise, error handling ensures the assistant responds appropriately if it cannot understand the speech or encounters an issue with the Google API. This functionality is crucial for enabling voice-based interaction, making the assistant responsive to user commands.

Why it's important:
The Recognizer class is the core component in the speech_recognition library for transforming spoken words into text. It acts as the interface between your program and the microphone. Without it, your program would not be able to understand voice commands, which is a key functionality of your assistant. By using Recognizer, your assistant can

interpret the user's spoken input and act on it, such as performing a task, giving feedback, or processing commands.

2-Command for performing various task(like opening apps and search)

```python
@eel.expose
def allCommands(message=1):
    if message == 1:
        query = takecommand()
        if not query:
            print("No command recognized.")
            eel.DisplayMessage("I didn't catch that. Could you repeat?")
            return
        eel.senderText(query)
    else:
        query = message
        eel.senderText(query)
    try:
        if "open" in query or "play" in query or "on youtube" in query:
            from engine.features import openCommand
            print(f"Executing openCommand for query: {query}")
            openCommand(query)
        else:
            from engine.features import chatBot
            print(f"Processing with chatBot for query: {query}")
            chatBot(query)
    except Exception as e:
        print(f"Error occurred: {e}")


    eel.ShowHood()
```

Explanation:

The allCommands() function is crucial for managing and executing various user commands within the assistant. This function processes input from the frontend, either from voice or text, and determines the appropriate action to take based on the content of the query. It first captures the user's voice input through the takecommand() function or directly uses the provided message. The function then checks the query for specific keywords like "open" or "on youtube," triggering respective actions such as opening applications or playing YouTube videos. If neither of these conditions is met, it defaults to processing the query using the chatbot API for general questions or tasks. The function also includes error handling to ensure that, if something goes wrong, the process doesn't break and provides feedback to the user. Finally, the UI is updated using eel.ShowHood() to reflect the outcome of the command. This function is essential as it serves as the core processing unit that allows the assistant to interact with the user, execute tasks, and respond accordingly. Without it, the assistant would be limited in functionality, unable to perform commands or provide appropriate responses based on user input.

3-AI chatbot API integration using Groq

```python
client = Groq(
    api_key="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"
)
def chatBot(query):
    # Use Groq client to create a chat completion request
    chat_completion = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": query
            }
        ],
        model="llama3-8b-8192",
    )

    # Retrieve and print the response
    response = chat_completion.choices[0].message.content
    print(response)
    speak(response)
    return response
```

Explanation:

The line client Groq(api_key="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") initializes a connection to the Groq API by creating an instance of the Groq class and providing it with an API key. The API key acts as a unique identifier that allows the application to authenticate and interact with the Groq service. Groq is used in this case to integrate AI functionalities into the assistant, such as answering questions or providing insights based on natural language inputs. By setting up the client object, the application can send requests to Groq and receive responses that enhance the assistant's ability to process and understand user commands. This integration is crucial for adding advanced AI features, improving the assistant's versatility, and making it capable of handling more complex user interactions.

5-The final setup to run all functionalities together

```
import os
import eel
from engine.features import *
from engine.command import *
eel.init("www")
playAssistantSound()
os.system('open -a Safari http://localhost:8000/index.html')

eel.start('index.html', mode=None, host='localhost', block=True)
```

Explanation: This code serves as the starting point for initializing and running the voice assistant application. It begins by importing key libraries: import os for system-level operations, import eel for connecting the Python backend with the HTML/JavaScript frontend, and custom modules (engine.features and engine.command) for the assistant's core functionalities. The frontend is initialized using eel.init("www"), which sets the location of the all the files such as command.py and all the functions we performing.
The playAssistantSound() function plays an introductory audio cue to indicate that the assistant is ready. The code then uses os.system to open the default web browser (Safari) and load the assistant's interface hosted locally at http://localhost:8000/index.html. Finally, the eel.start() function begins the backend/frontend interaction, ensuring that the application runs smoothly with real-time communication between user commands and assistant responses. This setup is crucial for establishing a functional and interactive environment for users to engage with the assistant.

5. Future Improvements

1. **Enhanced Speech Recognition**:
    - **Current Limitation**: The current voice recognition might only support specific commands or struggle with accents/noise.
    - **Future Improvement**: Implementing more advanced speech-to-text models that support multiple languages, accents, and noisy environments would increase accuracy and user satisfaction.
2. **Improved AI Functionality**:
    - **Current Limitation**: The AI's ability to provide intelligent responses could be basic, relying on predefined patterns or rules.
    - **Future Improvement**: I will enhance the AI's ability to understand context better and incorporate more sophisticated Natural Language Processing (NLP)

models, allowing the assistant to handle more complex queries or tasks autonomously.

3. **Integrating More APIs**:

   Expand functionality: Currently, my assistant is limited to chatbot capabilities. By adding APIs that handle specific tasks (like smart home control, weather updates, etc.), I could greatly enhance its usefulness and make it more adaptable to various user needs.

4. **Voice Command Customization**:
   - **Current Limitation**: The current system might have a set range of commands or lack personalization.
   - **Future Improvement**: Allowing users to define their own custom voice commands could personalize the assistant further, improving the user experience.

5. **Error Handling and Robustness**:
   - **Current Limitation**: The system may have some weak points in handling errors, especially when encountering unexpected inputs or system failures.
   - **Future Improvement**: Implementing better error handling, logging, and user notifications would make the assistant more stable and user-friendly.