**Unified Productivity Hub**

**A Multifunctional Desktop Application for Alarm Management, Typing Test Evaluation, Weather Forecasting,**

**Integrated with A.I**

**Abstract**

In today's technologically advanced era, Python programming plays a crucial role in various applications. This project explores the implementation of a unified productivity hub which helps AI to increase your productivity in your work. The report discusses the problem statement, methodology, challenges faced, and outcomes, providing insights into the project's significance.

**1. Introduction**

**1.1 Background**

Python, known for its simplicity and versatility, is widely used in software development. This project leverages Python to address the challenges associated with

* **Alarm Clock:**
* Allows users to set alarms with specific hours, minutes, seconds, and periods (AM/PM).
* Provides options to activate, deactivate, and terminate alarms.
* Plays a sound (in this case, 'ss.mp3') when the alarm conditions are met.
* **Typing Test:**
* Presents users with a randomly selected paragraph for typing.
* Measures typing speed in words per minute (wpm) and counts typing errors.
* Users can start, check, and stop typing tests.
* Includes functionality to start the typing test only if the alarm has been triggered.
* **Weather Forecast:**
* Allows users to input a city name.
* Retrieves weather data from the OpenWeatherMap API using the city name.
* Displays the main weather condition and temperature in Fahrenheit for the specified city.
* The "Get Weather" button is enabled only when the typing test has been started.
* **GUI (Graphical User Interface):**
* The GUI is implemented using Tkinter and ttkbootstrap for styling.
* Organized into tabs using the Notebook widget for Alarm Clock, Typing Test, and Weather Forecast.
* Each tab contains specific widgets and functionalities.

Understanding the background is essential for grasping the project's relevance.

**1.2 Problem Statement**

The project aims to address the need for an advanced voice-activated virtual assistant capable of intelligently processing natural language queries, performing diverse tasks, and enhancing user experiences. This involves addressing the limitations of existing virtual assistants, such as restricted functionality, lack of contextual understanding, and limited integration with external services. By doing so, the project contributes to the evolution of human-computer interaction, offering users a more seamless and intuitive way to interact with technology. The core issues revolve around developing a virtual assistant that not only understands and responds to user commands accurately but also adapts to context and provides meaningful insights. This project's relevance lies in its potential to elevate user interactions with technology, making tasks more efficient, informative, and enjoyable.

**1.3 Objectives**

Natural Language Processing (NLP) Implementation:

Develop and integrate advanced NLP algorithms to enhance the virtual assistant's ability to understand and interpret natural language queries accurately.

Voice Recognition and Response:

Implement robust voice recognition capabilities to enable the virtual assistant to accurately identify spoken commands and respond with appropriate actions.

Integration with External Services:

Establish seamless integration with various external services, such as web browsers, news APIs, weather services, and AI models, to broaden the range of tasks the virtual assistant can perform.

Dynamic Context Handling:

Implement a context-aware system to enable the virtual assistant to maintain context across multiple interactions, providing more coherent and personalized responses.

AI-based Conversational Abilities:

Train the virtual assistant using advanced AI models to improve its conversational abilities, allowing for more engaging and contextually relevant interactions with users.

User Interface Design:

Develop an intuitive and user-friendly interface, possibly using a graphical user interface (GUI) or other interactive elements, to enhance the overall user experience.These objectives guide the project's development and underscore its purpose.

**2. Methodology**

**2.1 Tools and Technologies Used**

The project utilizes [AI , Alarm, Typing Test, Game and etc libraries, and technologies]. Understanding these tools is crucial for comprehending the project's technical aspects.

**2.2 Project Design**

The project design encompasses.

**2.2.1 Data Flow**

The data flow in this project involves the seamless exchange of information between different components. Here's a breakdown:

User Input Flow:

The user interacts with the system through voice commands or text queries.

The takeCommand() function captures voice input, while text queries are directly processed.

Chat Functionality:

The chat(query) function processes user queries using the OpenAI GPT-3 model.

It sends the user query to the OpenAI API, retrieves the response, and uses text-to-speech to communicate the result.

External Services Integration:

Certain queries trigger actions like opening websites or fetching news.

Functions like open\_website(url) and get\_random\_news(category) handle interactions with external services.

Typing Test and Weather Modules:

The typing test and weather functionalities operate independently but contribute to the overall user experience.

**2.2.2 Algorithms**

Voice Recognition Algorithm (takeCommand()):

Utilizes the speech\_recognition library to capture and convert voice input to text.

Implements error handling to manage exceptions during voice recognition.

Chatbot Algorithm (chat(query)):

Leverages the OpenAI GPT-3 model for natural language processing.

Generates responses based on user prompts using the model's text completion capabilities.

Weather API Integration Algorithm (get\_weather()):

Uses the requests library to fetch real-time weather data from the OpenWeatherMap API.

Processes the API response to extract relevant weather information.

Typing Test Algorithm:

Implements a typing test functionality with a set of predefined paragraphs.

Measures typing speed and accuracy, providing feedback to the user.

**2.2.3 Architectural Design**

The architectural design of the project follows a modular and functional approach:

**Modularity:**

The project is divided into distinct modules, each responsible for specific functionalities (voice recognition, chat, external services, typing test, and weather).

This modular design enhances maintainability and allows for independent development of each feature.

**User Interface (UI):**

The UI components are structured using the tkinter library, creating a user-friendly interface for interaction.

Integration with External Services:

External services like OpenAI and OpenWeatherMap are seamlessly integrated into the project, enhancing its capabilities.

Scalability:

The architecture is designed to accommodate future extensions or additional features easily.

**2.3 Implementation Details**

Delving into the implementation process, this section highlights key Python functions and classes. Readers gain insights into the code structure and logic behind the project.

**3. Results and Discussion**

**3.1 Project Outcomes**

The project outcomes reveal.

The project outcomes showcase a successful integration of voice-based interaction, chat functionality, and external service utilization. Here's a breakdown of key aspects:

**User Interface (UI):**

The graphical user interface, developed using the tkinter library, provides an intuitive platform for users to interact with the system. The UI features:

**Voice-Driven Interaction:**

Users can command the system using voice input, processed by the speech\_recognition library.

The system responds with text and, in some cases, executes actions based on the input.

**Chat Functionality:**

Leveraging OpenAI's GPT-3 model, the chat feature enables natural language conversation with the system.

Responses are generated dynamically, providing an interactive and engaging user experience.

External Service Integration:

The system seamlessly integrates with external services, such as opening websites, fetching news, and checking the weather.

This enhances the functionality and utility of the project.

**Typing Test Feature:**

The inclusion of a typing test feature enhances the project's versatility. Users can assess their typing speed and accuracy through a simple yet effective interface. Key aspects of the typing test feature include:

**User Engagement:**

Users can start typing tests, and the system evaluates and provides feedback on their performance.

This feature adds an element of gamification to the project, making it more engaging.

Weather Information:

The project provides real-time weather information based on user queries. The weather feature includes:

**Weather Data Retrieval:**

Utilizing the OpenWeatherMap API, the system fetches and presents weather details for a specified location.

Users receive accurate and up-to-date weather information, contributing to the project's practicality.

**3.2 Challenges Faced**

While developing the project, several challenges emerged.

**Voice Recognition Accuracy:**

One of the primary challenges encountered during the project development was ensuring accurate and reliable voice recognition. The speech\_recognition library, while robust, faced difficulties in accurately interpreting certain accents or variations in pronunciation. To mitigate this, we experimented with adjusting recognition parameters and incorporating user feedback for improved performance.

**OpenAI API Interaction:**

Integrating the OpenAI GPT-3 API presented challenges related to handling API responses and managing dynamic conversations. Fine-tuning the model to generate coherent and contextually relevant responses required thorough testing and refinement. Additionally, ensuring secure and efficient handling of API keys was crucial to maintaining the integrity of the system.

**External Service Integration:**

Incorporating external services, such as fetching real-time weather data and news articles, posed challenges related to API compatibility and response parsing. Synchronizing these services seamlessly within the user interface required careful consideration of asynchronous operations and error handling.

**Typing Test Implementation:**

Developing the typing test feature involved addressing challenges related to accurately measuring typing speed, handling user input, and providing real-time feedback. Optimizing the user experience while ensuring the accuracy of speed calculations required iterative testing and adjustments.

**3.3 Learnings and Insights**

Working on the project yielded valuable learnings and insights.

**Continuous Iteration and Testing**:

One key lesson learned is the importance of continuous iteration and testing. As we encountered challenges, we realized the significance of regularly testing different components, refining algorithms, and making incremental improvements. This iterative approach not only enhanced the project's robustness but also allowed for more agile problem-solving.

**User-Centric Design:**

Developing a user-centric design was paramount, especially in features like voice interaction and the typing test. Understanding user behavior, preferences, and potential points of friction helped shape a more intuitive and engaging user experience. Regular user testing and feedback collection were instrumental in refining the interface.

**Secure API Key Handling:**

Handling API keys securely became a crucial aspect of the project. We learned the importance of implementing best practices for API key management, ensuring they are kept confidential, and exploring options for secure storage. This knowledge is transferable to future projects involving external APIs.

**4. Conclusion**

* The project successfully addresses a defined problem through innovative Python application development.
* Key objectives, including AI-driven interactions and external API integrations, are met, showcasing the project's purpose.
* Challenges faced, such as (mentioned above in the report) helped me , demonstrate adaptability and problem-solving skills.
* The outcomes reveal a user-friendly interface, efficient data flow, and seamless algorithmic implementations.
* Lessons learned, also mentioned above in the project , contribute to the project's success and future development endeavours.
* In summary, the project achieves its goals, combining intelligence, real-time information retrieval, and interactive elements.

**References**

YouTube: https://[www.youtube.com](http://www.youtube.com)/

Google: https://www.google.com/

**Appendices**

Appendix A GUI:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Appendix B: Source Code(of AI):

import speech\_recognition as sr  
import os  
import win32com.client  
import webbrowser  
import openai  
import datetime  
import requests  
import random  
  
chatStr = ""  
def chat(query):  
 global chatStr  
 print(chatStr)  
 openai.api\_key = "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"  
 chatStr += f"Tushar: {query}\n Jarvis: "  
  
 response = openai.Completion.create(  
 model="text-davinci-003"**,** prompt=chatStr**,** temperature=**1,** max\_tokens=**256,** top\_p=**1,** frequency\_penalty=**0,** presence\_penalty=**0** )  
 speaker.Speak(response["choices"][**0**]["text"])  
 chatStr += response["choices"][**0**]["text"]  
 return response["choices"][**0**]["text"]  
  
  
  
def businessnews(query):  
 main\_api = "https://newsapi.org/v2/top-headlines?country=in&category=business&apiKey=98665c9335254550b21ff7690b5c0bed"  
  
 news = requests.get(main\_api).json()  
 articles = news['articles']  
 news\_article = [arti['title'] for arti in articles]  
  
 # Ensure there are at least 5 articles  
 if len(news\_article) >= **5**:  
 selected\_articles = random.sample(news\_article**, 5**)  
 for article\_title in selected\_articles:  
 print(article\_title)  
 speaker.Speak(article\_title)  
 else:  
 print("Not enough articles to select 5 random news.")  
  
  
def Technews(query):  
 main\_api = "https://newsapi.org/v2/top-headlines?sources=techcrunch&apiKey=98665c9335254550b21ff7690b5c0bed"  
  
 news = requests.get(main\_api).json()  
 articles = news['articles']  
 news\_article = [arti['title'] for arti in articles]  
  
 # Ensure there are at least 5 articles  
 if len(news\_article) >= **5**:  
 selected\_articles = random.sample(news\_article**, 5**)  
 for article\_title in selected\_articles:  
 print(article\_title)  
 speaker.Speak(article\_title)  
 else:  
 print("Not enough articles to select 5 random news.")  
  
  
def weather(query):  
 weather.apikey = "30d4741c779ba94c470ca1f63045390a"  
 city = ''.join(query.split('in')[**1**:])  
 weather\_data = requests.get(  
 f"https://api.openweathermap.org/data/2.5/weather?q={city}&units=imperial&APPID={weather.apikey}"  
 )  
 weater = weather\_data.json()['weather'][**0**]['main']  
 temp = round(weather\_data.json()['main']['temp'])  
 print(f"The weather in {city} is: {weater}, and the temperature in {city} is: {temp} degrees fahrenheit")  
 speaker.Speak(f"The weather in {city} is: {weater}, and the temperature in {city} is: {temp}degrees fahrenheit")  
  
  
def ai(prompt):  
 openai.api\_key = "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"  
 text = f"Openai response for prompt: {prompt} \n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"  
  
 response = openai.Completion.create(  
 model="text-davinci-003"**,** prompt=prompt**,** temperature=**1,** max\_tokens=**256,** top\_p=**1,** frequency\_penalty=**0,** presence\_penalty=**0** )  
  
 try:  
 print(response["choices"][**0**]["text"])  
 text += response["choices"][**0**]["text"]  
 if not os.path.exists("Openai"):  
 os.mkdir("Openai")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
 with open(f"Openai/prompt- {''.join(prompt.split('intelligence')[**1**:])}.txt"**,** "w") as f:  
 f.write(text)  
  
  
speaker = win32com.client.Dispatch("SAPI.SpVoice")  
  
  
def say(text):  
 os.system(f"say{text}")  
  
  
def takeCommand():  
 r = sr.Recognizer()  
 with sr.Microphone() as source:  
 r.pause\_threshold = **0.6** audio = r.listen(source)  
 try:  
  
 print("Recognizing....")  
 query = r.recognize\_google(audio**,** language="en-in")  
 print(f"User said: {query}")  
 return query  
 except Exception as e:  
 return "Some Error Occurred. Sorry from Jarvis"  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('PyCharm')  
 speaker.Speak("good evening sir,Jarvis here")  
  
 while True:  
  
 print("Listening.....")  
  
 query = takeCommand()  
 sites = [["youtube"**,** "https://www.youtube.com"]**,** ["wikipedia"**,** "https://www.wikipedia.com"]**,** ["google"**,** "https://www.Google.com"]**,** ["armorgames"**,** "https://armorgames.com/"]]  
 for site in sites:  
  
 if f"Open {site[**0**]}".lower() in query.lower():  
 speaker.Speak(f"Opening {site[**0**]} sir.... ")  
 webbrowser.open(site[**1**])  
  
 if "open music" in query:  
 speaker.Speak("Opening Spotify sir")  
 Spotify = "https://open.spotify.com/"  
 webbrowser.open("Spotify")  
  
 elif "open alarm" in query:  
 speaker.Speak("Opening alarm sir")  
 os.system("python jarvis\_alarm.py")  
  
 elif "the time" in query:  
 hour = datetime.datetime.now().strftime("%H")  
 min = datetime.datetime.now().strftime("%M")  
 speaker.Speak(f"Sir time is {hour} bajjkkkkee {min} minutes")  
  
 elif "close".lower() in query.lower():  
 speaker.Speak("Shutting Down")  
 exit()  
  
 elif "Using artificial intelligence".lower() in query.lower():  
 ai(prompt=query)  
  
 elif "what's the weather" in query:  
 weather(query)  
  
 elif "the business news" in query:  
 businessnews(query)  
 elif "the technology news" in query:  
 Technews(query)  
 elif "game".lower() in query.lower():  
 speaker.Speak("Opening game")  
 os.system("python ticc2222.py")  
 else:  
 chat(query)

Appendix C Output:

All The queries asked to the AI and answers received:

