An AI-powered virtual assistant to perform tasks such as scheduling meetings and sending emails.

3. Snapshots of the Project

Main File:

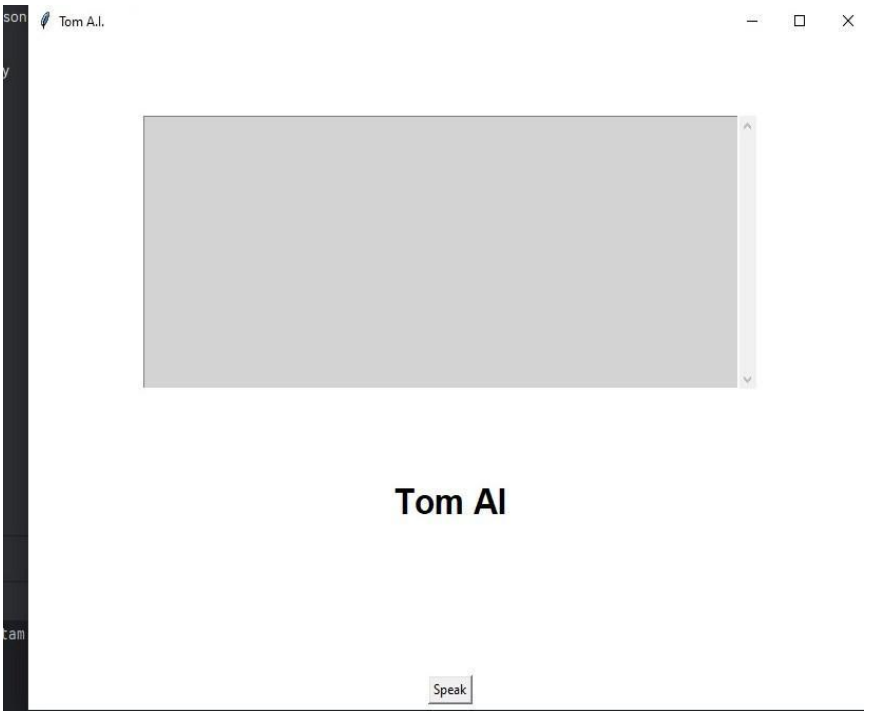
```python
import tkinter as tk
from tkinter import scrolledtext
from tkinter import messagebox
import webbrowser
import datetime
import speech_recognition as sr
import pyttsx3
import os
import openai
import smtplib
from email.message import EmailMessage
import ssl
import speech_recognition as sr
import pyttsx3
from config import apikey
from calender import create_event
```

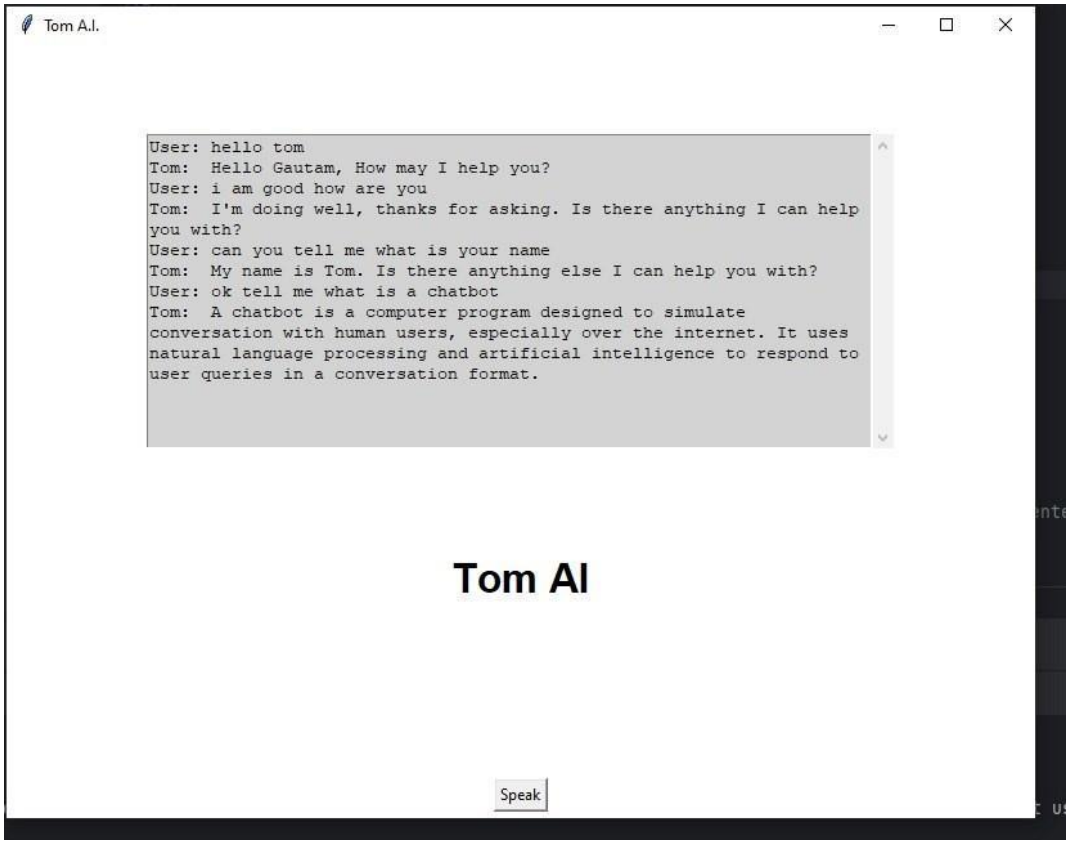
```python
# Initialize Tkinter application
app = tk.Tk()
app.title("Tom A.I.")
app.geometry("800x600")
app.configure(bg="white")

# Create a scrolled text widget to show the conversation
conversation_text = scrolledtext.ScrolledText(app, wrap=tk.WORD, width=70, height=15, bg="lightgray")
conversation_text.grid(row=0, column=0, columnspan=2, padx=10, pady=10)

# Global variables for conversation history
chatStr = ""
prev_response = ""
```

· **Speech recognition**

```python
# Function to handle voice input and display options
1 usage
def process_voice_input():
    query = listen()
    if query:
        response = chat(query)
        say(response)
        update_conversation(query, response)

        # Check if the user's query contains specific commands
        if "send email" in query or "mail" in query:
            hide_all_options()  # Hide all options first
            show_email_options()
        elif "schedule" in query or "create" in query or "make an appointment" in query or "meeting" in query or "remainder" in query:
            hide_all_options()  # Hide all options first
            show_schedule_event_options()
        else:
            hide_email_options()  # Hide email options if the query is not about sending an email
```

```python
# Function to update the conversation in the text widget
1 usage
def update_conversation(user_query, response):
    conversation_text.insert(tk.END, "User: " + user_query + "\n")
    conversation_text.insert(tk.END, "Tom: " + response + "\n")
    conversation_text.see(tk.END)  # Auto-scroll to the end
```





Sending Email:

```python
# Function to send an email
1 usage
def send_email(email_receiver, subject, body):
    email_sender = 'bishtg190@gmail.com'
    email_password = 'fwihhqallwcmnimn'

    em = EmailMessage()
    em['From'] = email_sender
    em['To'] = email_receiver
    em['Subject'] = subject
    em.set_content(body)

    context = ssl.create_default_context()

    with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
        smtp.login(email_sender, email_password)
        smtp.send_message(em)
```

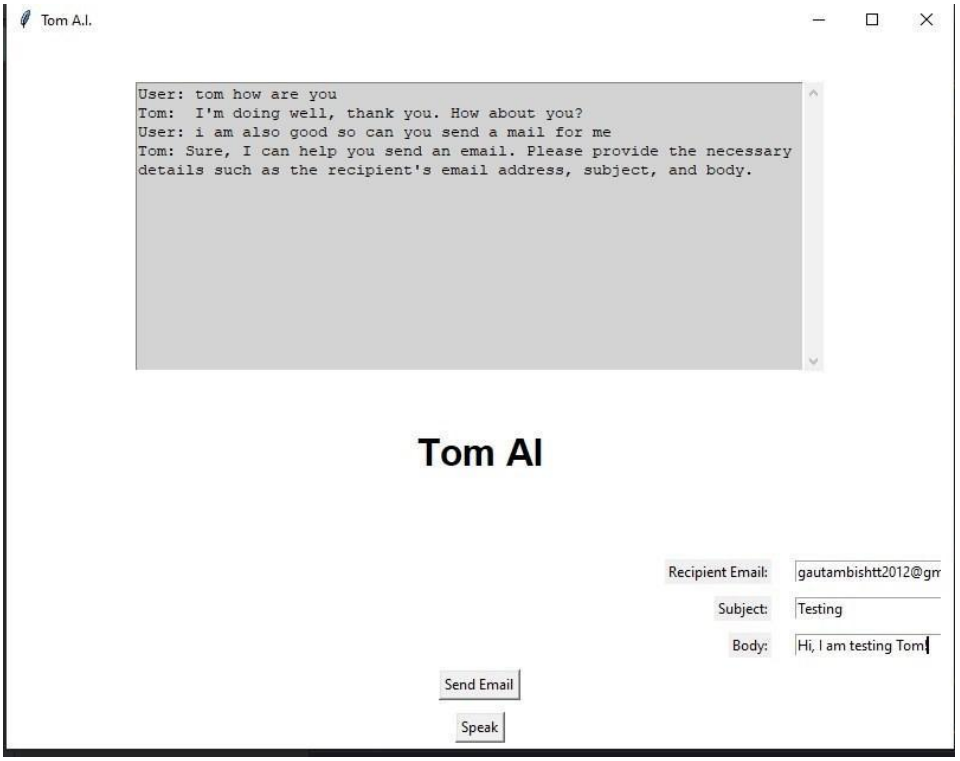
Ai Model for Sending Emails

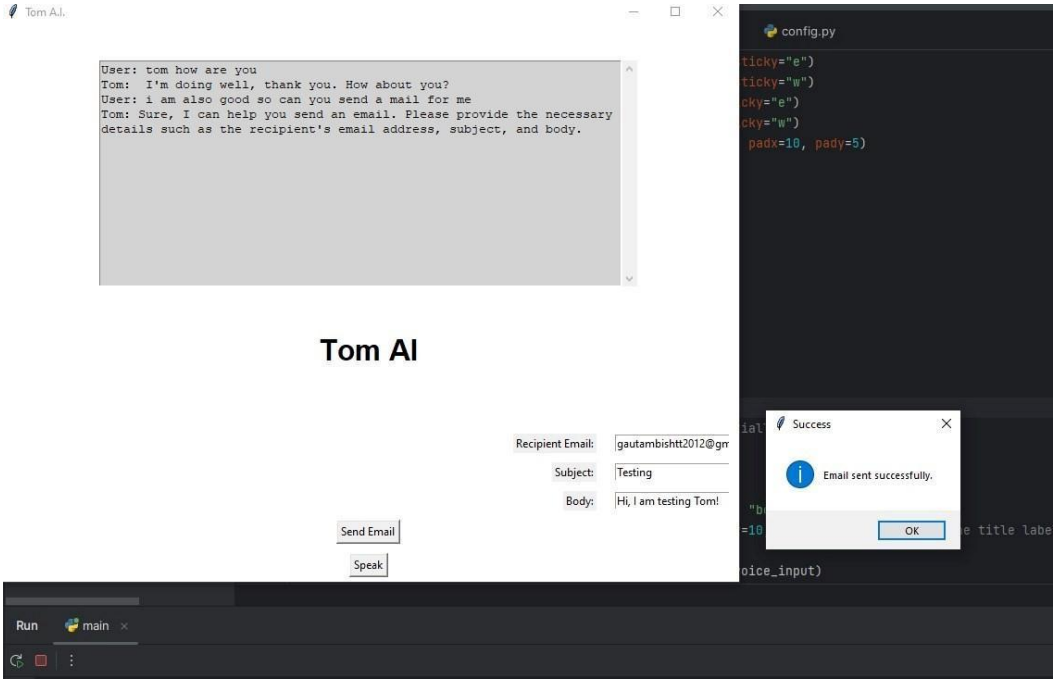
```python
# Function to interact with the chatbot
1 usage
def chat(query):
    global chatStr
    openai.api_key = apikey

    if "send email" in query or "mail" in query:
        return "Sure, I can help you send an email. Please provide the necessary details such as the recipient's email address, subject, and body."

    if "schedule event" in query or "meeting" in query:
        return "Sure, I can Schedule an event for you. Please provide the necessary details such as event date and time, the event must reflect in your calendar."

    prompt = f"{chatStr}Gautam: {query}\n Tom: "
    response = openai.Completion.create(
        model="text-davinci-003",
        prompt=prompt,
        temperature=0.7,
        max_tokens=256,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    reply = response["choices"][0]["text"]
    chatStr += f"Gautam: {query}\n Tom: {reply}\n"
    return reply
```

```python
# GUI elements for email functionality
email_receiver_label = tk.Label(app, text="Recipient Email:")
email_receiver_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")

email_receiver_entry = tk.Entry(app)
email_receiver_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")

subject_label = tk.Label(app, text="Subject:")
subject_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")

subject_entry = tk.Entry(app)
subject_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")

body_label = tk.Label(app, text="Body:")
body_label.grid(row=7, column=0, padx=10, pady=5, sticky="e")

body_entry = tk.Entry(app)
body_entry.grid(row=7, column=1, padx=10, pady=5, sticky="w")
```

```
142 def send_email_gui():
143     email_receiver = email_receiver_entry.get()
144     subject = subject_entry.get()
145     body = body_entry.get()
146 
147     if email_receiver and subject and body:
148         send_email(email_receiver, subject, body)
149         messagebox.showinfo("Success", "Email sent successfully.")
150     else:
151         messagebox.showwarning("Incomplete Information", "Please fill all email fields.")
```

```
193 # Function to display email options
    1 usage
194 def show_email_options():
195     hide_all_options()
196     email_receiver_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")
197     email_receiver_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")
198     subject_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")
199     subject_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")
200     body_label.grid(row=7, column=0, padx=10, pady=5, sticky="e")
201     body_entry.grid(row=7, column=1, padx=10, pady=5, sticky="w")
202     send_email_button.grid(row=8, column=0, columnspan=2, padx=10, pady=5)
203 
204 # Function to hide the email options
    2 usages
205 def hide_email_options():
206     email_receiver_label.grid_remove()
207     email_receiver_entry.grid_remove()
208     subject_label.grid_remove()
209     subject_entry.grid_remove()
210     body_label.grid_remove()
211     body_entry.grid_remove()
212     send_email_button.grid_remove()
213 
```



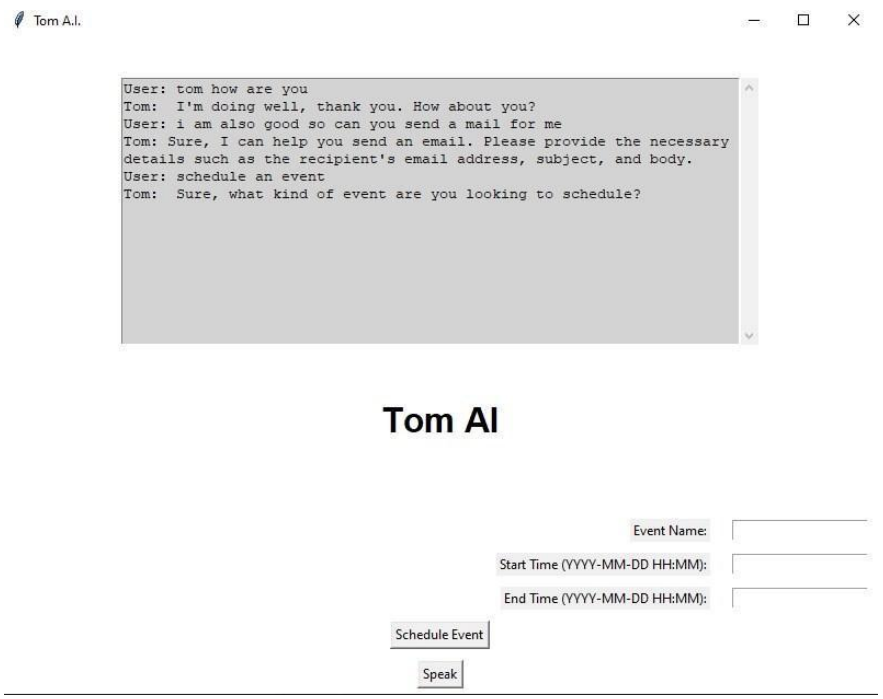


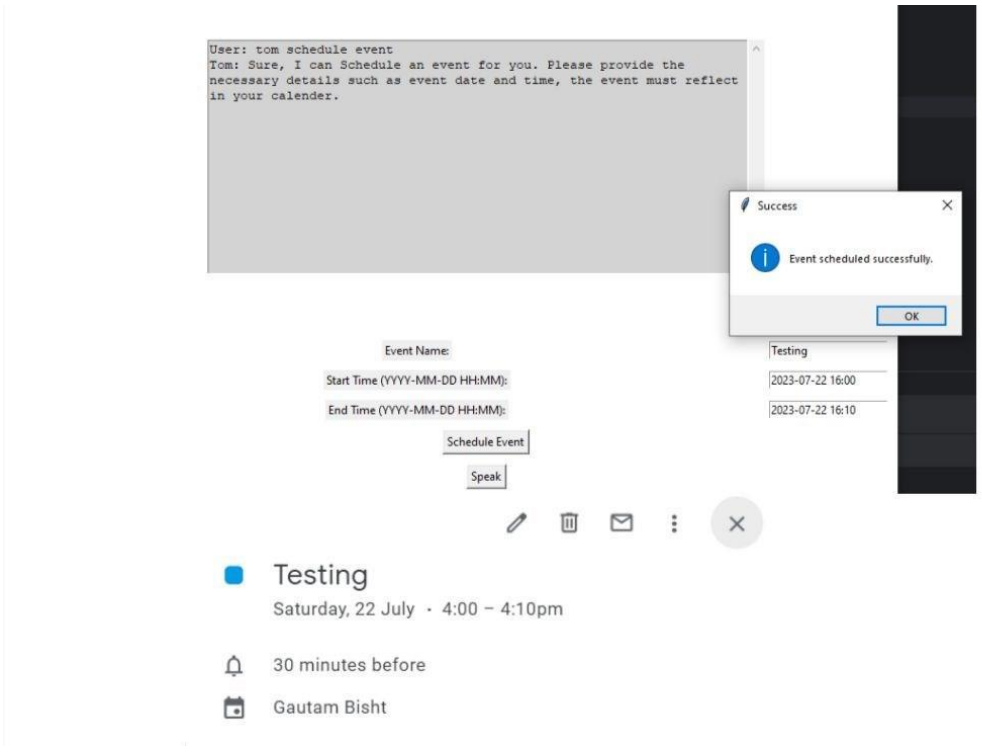
Scheduling Event:

```
1 usage
def show_schedule_event_options():
    hide_all_options()
    event_summary_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")
    event_summary_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")
    start_time_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")
    start_time_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")
    end_time_label.grid(row=7, column=0, padx=10, pady=5, sticky="e")
    end_time_entry.grid(row=7, column=1, padx=10, pady=5, sticky="w")
    schedule_event_button.grid(row=8, column=0, columnspan=2, padx=10, pady=5)
```

```
send_email_button = tk.Button(app, text="Send Email", command=send_email_gui)

# GUI elements for event scheduling functionality
event_summary_label = tk.Label(app, text="Event Name:")
event_summary_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")

event_summary_entry = tk.Entry(app)
event_summary_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")

start_time_label = tk.Label(app, text="Start Time (YYYY-MM-DD HH:MM):")
start_time_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")

start_time_entry = tk.Entry(app)
start_time_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")

end_time_label = tk.Label(app, text="End Time (YYYY-MM-DD HH:MM):")
end_time_label.grid(row=7, column=0, padx=10, pady=5, sticky="e")

end_time_entry = tk.Entry(app)
end_time_entry.grid(row=7, column=1, padx=10, pady=5, sticky="w")

```

```
# Function to hide all email and event scheduling options
5 usages
def hide_all_options():
    hide_email_options()
    event_summary_label.grid_remove()
    event_summary_entry.grid_remove()
    start_time_label.grid_remove()
    start_time_entry.grid_remove()
    end_time_label.grid_remove()
    end_time_entry.grid_remove()
    schedule_event_button.grid_remove()

# Call the hide_all_options function to hide all options initially
hide_all_options()

# GUI elements for the "Speak" button
title_label = tk.Label(app, text="Tom AI", font=("Arial", 24, "bold"), bg="white")
title_label.grid(row=1, column=0, columnspan=2, padx=10, pady=10, sticky="ew")  # Center the title label horizontally

speak_button = tk.Button(app, text="Speak", command=process_voice_input)
speak_button.grid(row=9, column=0, columnspan=2, padx=10, pady=5)

# Add padding to create a border around the GUI
app.grid_rowconfigure(0, weight=1)  # Allow row 0 to expand vertically
app.grid_columnconfigure(0, weight=1)  # Allow column 0 to expand horizontally
app.grid_rowconfigure(2, weight=1)  # Allow row 2 to expand vertically

# Run the main application loop
app.mainloop()
```

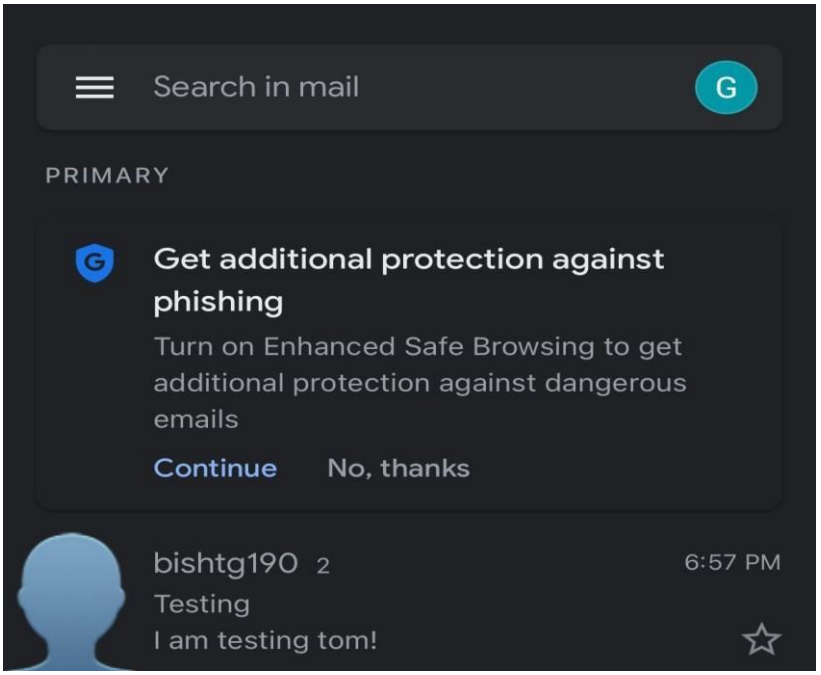




Receiving Email:

```python
import datetime
import pickle
from google.auth.transport.requests import Request
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
import os.path

# If modifying the scope, delete the token.pickle file.
SCOPES = ['https://www.googleapis.com/auth/calendar']

1 usage
def get_credentials():
    creds = None
    # The file token.pickle stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials .json.', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)
    return creds
```

```python
def create_event(summary, start_time, end_time):
    creds = get_credentials()
    service = build('calendar', 'v3', credentials=creds)

    event = {
        'summary': summary,
        'start': {
            'dateTime': start_time.strftime('%Y-%m-%dT%H:%M:%S'),
            'timeZone': 'Asia/Kolkata',
        },
        'end': {
            'dateTime': end_time.strftime('%Y-%m-%dT%H:%M:%S'),
            'timeZone': 'Asia/Kolkata',
        },
    }

    try:
        event = service.events().insert(calendarId='primary', body=event).execu
        print('Event created: %s' % (event.get('htmlLink')))
    except HttpError as e:
        print('Error creating event:', e)
```



4.Conclusions

The AI Virtual Assistant project introduces a powerful and user-friendly tool that combines advanced technologies to streamline tasks, enhance productivity, and provide a personalized user experience. Through a well-designed architecture, seamless interactions, and integration with external services, the project demonstrates the potential of AI-driven solutions. With a focus on accuracy, security, and real-time responsiveness, the AI Virtual Assistant offers a glimpse into the future of efficient task management and intelligent assistance.