

Prompt engineering Homework

Productivity assistant chat bot

Project Overview:

This project aims to assist users in managing their time and productivity more effectively. It leverages OpenAI's GPT to understand user prompts and perform actions such as creating calendar events, setting up tasks, or providing advice on productivity, health, and learning. The application interacts with Google Calendar and Google Tasks APIs to manage these entities directly based on user interaction, which happens in the command line interface.

Project setup:

I will give describe the steps necessary to build and run the chat bot. Since I am using Linux, here is the Linux setup guide, but other platforms should act similar.

Project dependencies:

Python and other installs:

Steps are also written here: <https://platform.openai.com/docs/quickstart?context=python>

To use the OpenAI Python library, you need at least Python 3.7.1 or newer. Steps to download are here: <https://wiki.python.org/moin/BeginnersGuide/Download>

Execute:

```
apt-get install python3 python3-dev
```

```
pip install --upgrade openai argparse python-dotenv
```

```
pip install --upgrade google-api-python-client google-auth-httpplib2 google-auth-oauthlib
```

Api key:

Create an OpenAi account, fund it with money, so various gpt-models can be queried and create a public key that you can use to authenticate yourself. You will need to use it inside the code.

Page about API keys: <https://platform.openai.com/api-keys>

API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically disable any API key that we've found has leaked publicly.

Enable tracking to see usage per API key on the [Usage page](#).

NAME	SECRET KEY	TRACKING	CREATED	LAST USED	PERMISSIONS
prompt_engineering	sk-...Fapp	Enabled	Mar 2, 2024	Mar 2, 2024	All

[+ Create new secret key](#)

Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

Personal

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

Integration with Google Calendar:

Steps are also written here: <https://developers.google.com/calendar/api/quickstart/python>

1. Create a Google Cloud project: <https://developers.google.com/workspace/guides/create-project>

Start your free trial with \$300 in credit. Don't worry – you won't be charged if you run out of credit. [Learn more](#)

Google Cloud Search (/) for resources, docs

New Project

You have 12 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)

Project name *
ProductivityChatBot

Project ID: productivitychatbot. It cannot be changed later. [EDIT](#)

Location *
No organisation [BROWSE](#)

Parent organisation or folder

[CREATE](#) [CANCEL](#)

2. Enable Google Calendar API and Google Tasks API

3. Configure the OAuth consent screen

1.

Start your free trial with \$300 in credit. Don't worry – you won't be charged if you run out of credit. [Learn more](#)

Google Cloud ProductivityChatBot Search (/) for resources, docs

APIs and services

- Enabled APIs and services
- Library
- Credentials
- OAuth consent screen**
- Page usage agreements

OAuth consent screen

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

☐ Internal

☒ External

Only available to users within your organisation. You will not need to submit your app for verification. [Learn more about user type](#)

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. [Learn more about user type](#)

[CREATE](#)

[Let us know what you think](#) about our OAuth experience

2.

Google Cloud ProductivityChatBot Search (/) for resources, docs

APIs and services

- Enabled APIs and services
- Library
- Credentials
- OAuth consent screen**
- Page usage agreements

Edit app registration

1 OAuth consent screen — 2 Scopes — 3 Test users — 4 Summary

App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *
ProductivityChatBot

The name of the app asking for consent

User support email *
gorondj.janos.job@gmail.com

For users to contact you with questions about their consent. [Learn more](#)

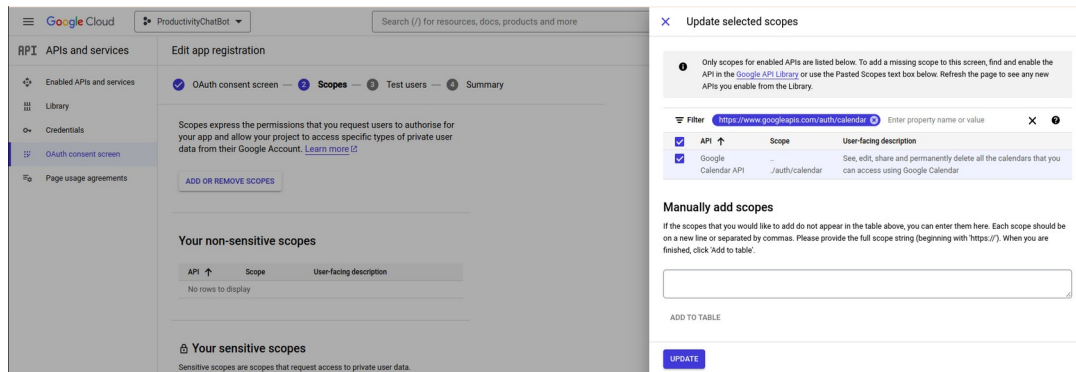
Developer contact information

Email addresses *
gorondj.janos.job@gmail.com

These email addresses are for Google to notify you about any changes to your project.

[SAVE AND CONTINUE](#) [CANCEL](#)

3. Also select Google Tasks API here



Google Cloud | ProductivityChatBot | Search (/) for resources, docs, products and more

APIs and services | Edit app registration

Enabled APIs and services | Library | Credentials | OAuth consent screen | Page usage agreements

OAuth consent screen | Scopes | Test users | Summary

Scopes express the permissions that you request users to authorize for your app and allow your project to access specific types of private user data from their Google Account. [Learn more](#)

ADD OR REMOVE SCOPES

Your non-sensitive scopes

API	Scope	User-facing description
Google Calendar API	/auth/calendar	See, edit, share and permanently delete all the calendars that you can access using Google Calendar

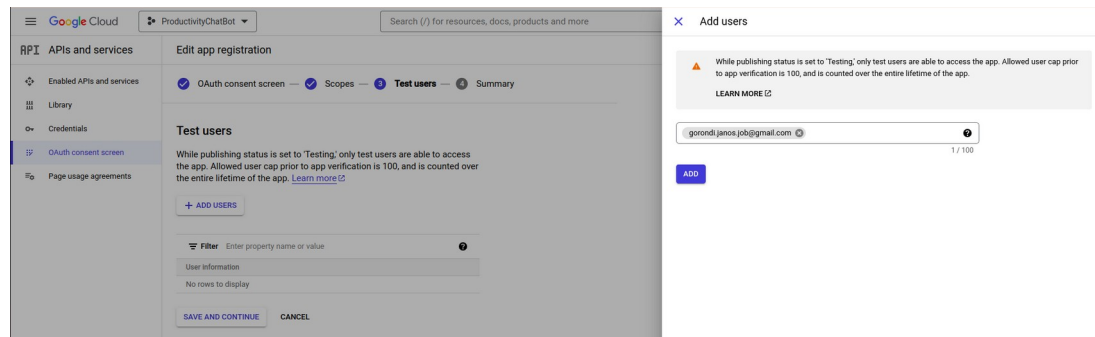
Manually add scopes

If the scopes that you would like to add do not appear in the table above, you can enter them here. Each scope should be on a new line or separated by commas. Please provide the full scope string (beginning with 'https://'). When you are finished, click 'Add to table'.

ADD TO TABLE

UPDATE

4.



Google Cloud | ProductivityChatBot | Search (/) for resources, docs, products and more

APIs and services | Edit app registration

Enabled APIs and services | Library | Credentials | OAuth consent screen | Page usage agreements

OAuth consent screen | Scopes | Test users | Summary

Test users

While publishing status is set to 'Testing', only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. [Learn more](#)

+ ADD USERS

Filter | Enter property name or value

User Information

No rows to display

SAVE AND CONTINUE | CANCEL

Add users

While publishing status is set to 'Testing', only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. [LEARN MORE](#)

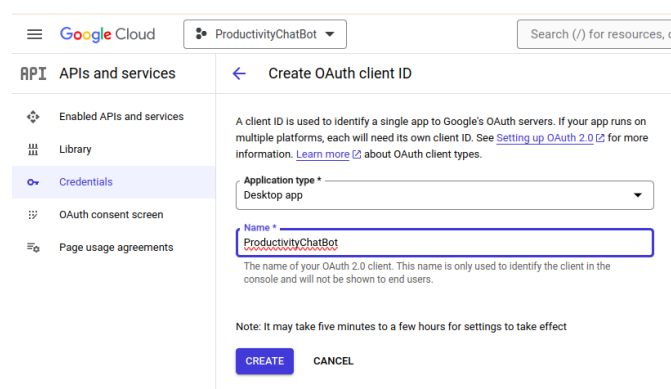
gorond.janos.job@gmail.com

1 / 100

ADD

4. Authorize credentials

1.



Google Cloud | ProductivityChatBot | Search (/) for resources, docs, products and more

APIs and services | Create OAuth client ID

Enabled APIs and services | Library | Credentials | OAuth consent screen | Page usage agreements

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type *
Desktop app

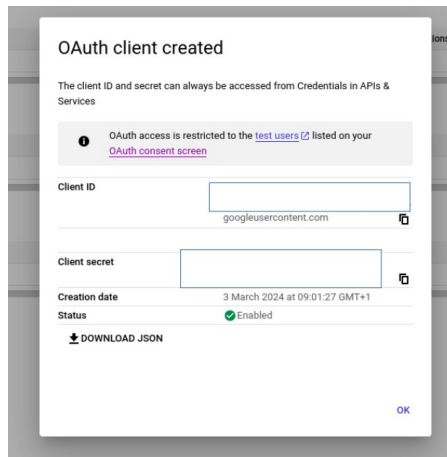
Name *
ProductivityChatBot

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

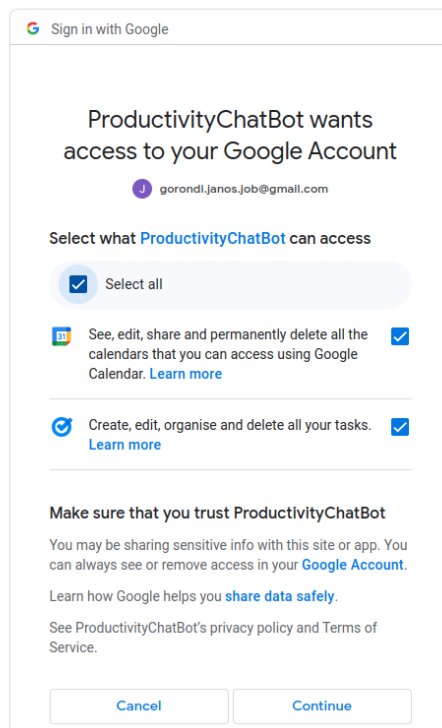
Note: It may take five minutes to a few hours for settings to take effect

CREATE | CANCEL

- Here you have to download the json file, copy it to the chat bot projects root directory/auth folder and rename it to credentials.json. This will be needed later on. Because the code will look for it



3. After you started your first prompt, Google will ask for verification again. Just enable everything.



Chat bot architecture:

Configuration (config.py)

- **Class Config:** Central configuration hub for the application. It parses command-line arguments and initializes settings that control the application's behavior.
- **Attributes:**
 - `debug`: Enables detailed logging if set.
 - `model`: Specifies the GPT model to use for generating responses.
 - `date`: Defines a reference date for operations that require temporal context.
 - `timeZone`: Sets the default time zone for event and task scheduling.

- Other attributes store user options, context structures for different actions (Events, Todos, Advice), reminder intervals, maximum conversation history size, and paths for saving .ics files.

Main Application Flow (main.py)

- **Google and GPT Client Setup:**
 - Initializes the Google Calendar and Tasks clients, along with the GPT client, using credentials loaded from environment variables and stored tokens.
- **Message Memory Management:**
 - Handles the accumulation of messages exchanged during the session. It acts as a queue to maintain a history of interaction within a specified size limit.
- **User Interaction:**
 - Captures user input through the command line and processes it through GPT to determine the desired action. Based on GPT's response, it performs operations such as creating calendar events, adding tasks, or delivering advice.
- **Debugging Support:**
 - Provides functionality to print debug messages and the current state of `message_memory` based on the `-d` or `--debug` flag.
- **Handling GPT Responses:**
 - Decodes GPT's structured responses to extract actionable data. It then validates the user's intended action (Event, Todo, Advice) and executes it accordingly. This includes interfacing with Google APIs to create calendar events or tasks and handling dialog outputs.
- **Saving Events:**
 - If requested by the user and appropriate for the action, generates and saves an .ics file representation of a calendar event for import into Google Calendar.

Functions

- **setup_google_credentials:** Sets up authentication for Google APIs.
- **setup_calendar_client:** Initializes the Google Calendar API client.
- **setup_tasks_client:** Initializes the Google Tasks API client.
- **setup_gpt_client:** Configures the OpenAI GPT client.
- **print_debug_text:** Conditional logging based on the debug mode.
- **add_prompt_to_message_memory:** Manages message history, ensuring it doesn't exceed the configured limit.
- **get_user_prompt:** Interacts with the user, capturing and logging their input.

- **remind_gpt_if_needed:** Inserts reminder context into `message_memory` based on a defined interval.
- **send_gpt_request:** Sends the accumulated conversation history to GPT for processing.
- **save_google_calendar_event_to_file:** Saves event data to an `.ics` file when requested.
- **valid_user_action:** Validates the action determined by GPT against configured options.
- **create_google_calendar_event:** Creates a Google Calendar event based on GPT's response.
- **create_google_tasks_task:** Adds a task to Google Tasks based on GPT's response.
- **handle_gpt_response:** Routes GPT's response to appropriate handlers based on the intended action.
- **main:** Orchestrates the application flow, handling user input, GPT interaction, and response processing.

Error Handling

- Throughout the application, error handling is implemented to gracefully manage exceptions arising from API calls, JSON parsing, and file operations. This ensures the application remains robust across various operational scenarios.

Configuration Prompts in `config.py`

The `Config` class includes several multi-line string attributes designed as prompts or context passages for interactions with OpenAI's GPT model. These strings serve various roles, from setting the stage for the assistant's behavior to structuring responses for specific user actions. Here's an overview of each:

1. **purpose_instructions:** This string introduces the assistant's capabilities and persona. It outlines that the assistant is knowledgeable in areas like time management, productivity, self-development, health, and training, emphasizing its role as a personal life improvement coach. This foundational context is likely used to prime the GPT model for the type of advice and assistance it should provide, ensuring responses are in line with the assistant's defined character and expertise.
2. **user_action_instructions:** This prompt guides the GPT model on how to interpret user actions from the input. It mentions that the assistant should deduce whether the user intends to create a calendar event, a to-do task, or seek advice, based on the input provided. This instruction helps in tailoring the assistant's response to the user's implied or explicit request without necessitating follow-up questions for clarification.
3. **user_save_instructions:** This part instructs the GPT model on deciding if a user's request should result in saving an event to a file. It clarifies that saving is contingent upon the user's explicit request and is only applicable to calendar events. This conditionally influences the assistant's behavior in generating `.ics` files for events.
4. **event_structure, todo_structure, and advice_structure:** These strings outline the expected response formats for different user actions (Event, Todo, Advice). They provide templates or schemas that the assistant's responses should conform to, ensuring

consistency in how advice, event details, or to-do tasks are presented back to the user. These structures likely serve as guides for both the GPT model and the application logic to format the generated content appropriately.

5. **starting_context:** Combines the above instructions and structures into a comprehensive initial context for the GPT model. This amalgamated prompt sets the stage for the interaction, embedding the assistant's role, operational guidelines, and response formatting expectations directly into the conversation's outset. This initial context is crucial for priming the GPT model with the assistant's objectives, capabilities, and the format of its outputs.
6. **save_context:** Specifically designed for generating the content of a `.ics` file, this prompt guides the GPT model to produce an output formatted for direct inclusion in such a file. It is likely invoked when the user has requested an event be saved, influencing the assistant to return a calendar event's details in the standard `.ics` file format.
7. **reminder_context:** Intended to remind the GPT model of the assistant's purpose and operational guidelines at predetermined intervals. This recurring prompt helps maintain the assistant's focus and consistency in its responses, especially during prolonged interactions or when the conversation veers into less structured territories.

Usage Within the Application

These prompts are primarily utilized in interactions with the GPT model to ensure the generated responses adhere to the defined persona, understand user intents, and format outputs according to the application's needs. They're woven into the conversational flow, serving as contextual anchors or templates that guide the GPT model's output towards the desired structure and content, thereby enhancing the relevance and usability of its responses within the scope of the application's functionality.

The core mechanism for maintaining an efficient and cost-effective interaction with GPT is through a meticulously managed `message_memory` list, which captures the dialogue history between the user and the GPT.

Optimizing Token Usage and Costs

To address the potential rapid growth of `message_memory` and its impact on token usage and costs, I implemented the `conversation_history_size` variable. This effectively caps the size of the `message_memory`, ensuring that older messages are removed to make room for new interactions. This capping mechanism is crucial for optimizing the number of tokens used in each GPT request, directly contributing to cost efficiency.

Preserving Essential Context

Despite the dynamic nature of `message_memory`, the application is designed to retain the initial context and incorporate periodic reminders. The initial context, which outlines the assistant's capabilities and sets the tone for user interaction, remains a constant in `message_memory`. This foundational message ensures that GPT's responses remain aligned with the application's purpose.

Implementing Reminder Mechanism

Moreover, the application introduces reminders at specified intervals (controlled by `reminder_interval`), refreshing GPT's understanding of its objectives and the expected response format. These reminders are strategically placed to prevent drift in GPT's responses and maintain a focused interaction that aligns with user prompts.

Application Flow and Mechanisms

- **Initialization:** Upon startup, the application configures the necessary clients for interacting with OpenAI's GPT and Google's APIs, guided by settings defined in `Config`.
- **Message Memory Management:** Throughout the user interaction, `message_memory` collects the exchange between the user and GPT. This list is carefully managed to adhere to a predetermined size limit, ensuring efficiency.
- **User Prompts:** Users can enter prompts, which are then processed with the aid of GPT, considering the cumulative context built up in `message_memory`.
- **Action Execution:** Based on GPT's responses, the application executes actions such as creating calendar events or tasks, or delivering customized advice, with the flexibility to save events when requested by the user.

Key Features

- **Cost-Efficiency:** By limiting the conversation history, the application minimizes the number of tokens used per request.
- **Consistency:** The persistent initial context and reminder messages ensure GPT's responses remain on track, facilitating consistent and relevant interactions.
- **Flexibility:** Users have the freedom to inquire and command within the scope of productivity, benefiting from an AI assistant that adapts to their needs.

Example prompts:

- Set up an event for me to exercise from 7am to 8am tomorrow. It's going to be a yoga session to start the day right. Please make sure it doesn't repeat. Save it in a file.
- I need to organize a weekly book club meeting. Create an event every Thursday from 6pm to 8pm starting this week. We'll be discussing new books each week and sharing insights.
- Add a to-do for me to pay the electricity and internet bills on the 15th of this month. It's important to avoid late fees, so a reminder would be great.
- I'm trying to build a habit of meditating daily. Create a to-do for me to meditate starting tomorrow from 6:30am to 7am.
- Can you give me some tricks and tips about eating healthy, being happy while maintaining a busy university schedule?

- Can you set up an event for a weekend getaway? It should start on Friday at 5pm and end on Sunday at 8pm. We'll be going camping in the mountains to relax and enjoy nature. This should be a one-time event.

Demo video of chat bot:

<https://drive.google.com/file/d/1xiFBFw69Neev-l6iuBgdlN1-rE3EwQnu/view?usp=sharing>

Github repository:

<https://github.com/GJanos/productivity-chat-bot/tree/main>