Documentação do Código - Assistente de Conversação Veterinária

Instalação:

Pré-requisitos:

Certifique-se de ter o Python instalado. Caso não tenha, você pode baixá-lo em python.org.

Instale o Flask e a biblioteca OpenAI utilizando os seguintes comandos no terminal:

bash
```
pip install flask
pip install --upgrade openai
```

Configuração da Chave API OpenAI:

- É necessário obter uma chave de API da OpenAI. Você pode obtê-la ao se cadastrar no site da OpenAI.
- Substitua a string `'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'` pela sua chave de API na linha correspondente no arquivo `main.py`.

Caminho do Roteiro e Arquivo de Histórico:

- Altere o caminho do roteiro e o caminho do arquivo de histórico conforme necessário nas linhas correspondentes no arquivo `main.py`.

Utilização:

Executando o Aplicativo:

Execute o arquivo `main.py` no terminal:

bash
```
python main.py
```

O servidor Flask será iniciado e estará acessível em http://127.0.0.1:5000/.

Acessando o Chat:

- Abra um navegador e acesse http://127.0.0.1:5000/.
- A interface do chat permitirá a interação com o assistente de conversação veterinária.

Conversando com o Assistente:

- Digite suas perguntas na caixa de entrada e pressione "Enviar" para obter respostas do assistente baseado no roteiro fornecido.

Encerrando a Conversa:

- Para encerrar a conversa, digite "sair" na caixa de entrada. O histórico da conversa será salvo em um arquivo de texto.

Funcionamento:

- O código utiliza Flask para criar uma aplicação web que permite a interação com um assistente de conversação veterinária baseado na API da OpenAI.
- O assistente segue um roteiro pré-definido para responder às perguntas do usuário sobre planos de saúde para animais de estimação.
- O histórico da conversa é salvo em um arquivo de texto após cada interação.

Observações:

- Certifique-se de não compartilhar sua chave de API da OpenAI publicamente.
- O código foi projetado para simular um assistente de conversação veterinária, respondendo de acordo com o roteiro fornecido.

Manutenção:

- Para reiniciar a conversa, aguarde 60 segundos de inatividade ou reinicie manualmente utilizando a rota `/submit`.

Limitações:

- O assistente responde com base no roteiro fornecido e pode não lidar eficazmente com perguntas fora do escopo ou ambíguas.

Notas:

- Este é um projeto básico e pode ser expandido para incluir mais recursos e capacidades de processamento de linguagem natural.

Explicação do Código - Assistente de Conversação Veterinária

Estrutura do Código:

1. **Importações:**
   - O código começa importando os módulos necessários, como `time`, `re`, `Flask`, `render_template`, `request`, `jsonify`, `make_response`, `redirect`, `url_for`, `Thread` e `OpenAI`.
   - Também são importadas as classes `datetime` para manipulação de datas.
2. **Configuração da Aplicação Flask e OpenAI:**
   - Uma instância da aplicação Flask é criada, e a chave de API da OpenAI é configurada para interagir com o assistente de conversação.
3. **Criação de Arquivos na OpenAI:**
   - Um arquivo é criado na OpenAI para armazenar o roteiro do assistente de conversação.
4. **Criação do Assistente na OpenAI:**
   - Um assistente é criado na OpenAI, utilizando o roteiro como referência, e é associado ao arquivo criado anteriormente.
5. **Iniciação de Threads e Listas de Mensagens:**
   - Threads são criadas para comunicação com a API da OpenAI. Listas são inicializadas para armazenar mensagens do usuário, mensagens do assistente e o histórico completo.
6. **Funções Auxiliares:**
   - Funções como `sort_messages`, `save_to_file`, e `run_chat` são definidas para ajudar na manipulação de mensagens, ordenação e interação com a API.
7. **Roteiro do Assistente:**
   - O roteiro do assistente é fornecido como uma série de perguntas e respostas definidas.
8. **Rotas da Aplicação Flask:**
   - Rotas são definidas para a página inicial (`/`) e para o envio de mensagens (`/submit`) por meio do método POST.
9. **Tempo de Inatividade e Reinício da Conversa:**
   - Um mecanismo de controle de tempo é implementado para reiniciar a conversa após um período de inatividade.
10. **Execução da Aplicação:**
    - A aplicação é executada apenas quando o script `main.py` é chamado diretamente.

Explicação Detalhada:

- **Criação do Assistente:**

    - O assistente de conversação é criado usando a API da OpenAI. As instruções e o modelo específico são fornecidos.
- **Comunicação com a API:**
    - A comunicação com a API da OpenAI é feita por meio de threads e execuções, permitindo uma interação assíncrona.
- **Manipulação de Mensagens:**
    - As mensagens do usuário e do assistente são processadas e armazenadas em listas. O histórico completo é mantido para referência.
- **Salvando o Histórico em Arquivo:**
    - O histórico da conversa é salvo em um arquivo de texto após cada interação, permitindo a persistência dos dados.
- **Controle de Tempo:**
    - Um mecanismo de controle de tempo monitora a inatividade do usuário. Se a inatividade for detectada, a conversa é reiniciada.

Uso da Aplicação:

1. **Interagindo com o Assistente:**
    - Os usuários podem acessar a aplicação via navegador e interagir com o assistente, fazendo perguntas relacionadas ao roteiro.
2. **Encerrando a Conversa:**
    - O usuário pode encerrar a conversa digitando "sair". Isso aciona o salvamento do histórico e desabilita a submissão de novas mensagens.
3. **Reiniciando a Conversa:**
    - A conversa é reiniciada automaticamente após um período de inatividade ou pode ser reiniciada manualmente.
4. **Histórico e Arquivos:**
    - O histórico da conversa é salvo em um arquivo de texto, proporcionando uma referência para interações anteriores.

Limitações e Melhorias Futuras:

- **Limitações:**
    - O assistente segue um roteiro fixo e pode não lidar eficazmente com perguntas fora do escopo ou ambíguas.
    - A detecção de inatividade é baseada em um temporizador fixo.
- **Melhorias Futuras:**
    - A capacidade de lidar com perguntas mais abertas e variadas poderia ser aprimorada usando técnicas de processamento de linguagem natural mais avançadas.
    - Integração de recursos adicionais, como um banco de dados para armazenar e recuperar históricos de conversas.

Observação:

- Este código serve como uma base para a implementação de assistentes de conversação específicos. Revisões e melhorias podem ser feitas com base nas necessidades específicas do projeto ou nas atualizações das bibliotecas utilizadas.

Adição de Conteúdo ao Chat do Assistente

Este guia fornece instruções sobre como adicionar conteúdo ou dados ao chat do Assistente usando a API do OpenAI Assistants.

1. Criando um Novo Arquivo de Roteiro

Antes de começar, certifique-se de ter um arquivo de roteiro em um formato de texto, como o txt. ou até pdf. Neste exemplo, usaremos o caminho `/home/michael/Downloads/GPT(1)/GPT/Roteiro`.

python:
```
# Criação do primeiro arquivo que o assistente usará
file = client.files.create(
    file=open("/home/michael/Downloads/GPT(1)/GPT/Roteiro", "rb"),
    purpose='assistants'
)
```

Este comando cria um novo arquivo e o associa ao propósito de assistentes.

2. Criando um Assistente Inicial

Em seguida, você precisa criar um assistente inicial, adicionando o arquivo de roteiro criado anteriormente.

python:
```
# Criação do assistente inicial

my_assistant = client.beta.assistants.create(
    instructions="Você é um veterinário, ajudando o cliente a escolher um plano para seu cachorro. Responda de acordo com o roteiro, qualquer coisa que fuja do assunto, responda educadamente que não é sua função.",
    name="Vet MPC",
    tools=[{"type": "retrieval"}],
    model="gpt-4-1106-preview",
    file_ids=[file.id],
)
```

Este comando cria um assistente inicial com as instruções fornecidas, o nome "Vet MPC", uma ferramenta de recuperação e associa o arquivo de roteiro ao assistente.

3. Adicionando Novo Conteúdo ao Chat

Se desejar adicionar mais conteúdo ao chat, você pode criar um novo arquivo de roteiro e associá-lo ao assistente. Use o seguinte comando:

python

```
# Adicionando novo conteúdo ao assistente
new_file = client.files.create(
    file=open("/caminho/do/novo/arquivo", "rb"),
    purpose='assistants'
)

# Associação do novo arquivo ao assistente
assistant_file = client.beta.assistants.files.create(
    assistant_id=my_assistant.id,  # Substitua pelo ID do assistente
criado
    file_id=new_file.id
)
```

Certifique-se de substituir "/caminho/do/novo/arquivo" pelo caminho do novo arquivo que você deseja adicionar. Além disso, substitua `my_assistant.id` pelo ID do assistente que você criou.

4. Limitações

Lembre-se das limitações da API:

- Até 20 arquivos podem ser suportados.
- O tamanho total dos arquivos para uma organização pode ser de até 100 GB.
- O tamanho individual dos arquivos pode ser de no máximo 512 MB ou 2 milhões de tokens para assistentes.

5. Dúvidas e Ajuda

Se houver dúvidas ou problemas, sinta-se à vontade para solicitar ajuda ou consultar a documentação oficial do OpenAI Assistants: Documentação OpenAI Assistants.

Tipos de extensões de arquivos suportado:

1. **.c**
   - Tipo MIME: `text/x-c`
2. **.cpp**
   - Tipo MIME: `text/x-c++`
3. **.csv**
   - Tipo MIME: `application/csv`
4. **.docx**
   - Tipo MIME: `application/vnd.openxmlformats-officedocument.wordprocessingml.document`
5. **.html**
   - Tipo MIME: `text/html`
6. **.java**
   - Tipo MIME: `text/x-java`
7. **.json**
   - Tipo MIME: `application/json`
8. **.md**
   - Tipo MIME: `text/markdown`
9. **.pdf**
   - Tipo MIME: `application/pdf`
10. **.php**
    - Tipo MIME: `text/x-php`
11. **.pptx**
    - Tipo MIME: `application/vnd.openxmlformats-officedocument.presentationml.presentation`
12. **.py**
    - Tipo MIME: `text/x-python`
13. **.py**
    - Tipo MIME: `text/x-script.python`
14. **.rb**
    - Tipo MIME: `text/x-ruby`
15. **.tex**
    - Tipo MIME: `text/x-tex`
16. **.txt**
    - Tipo MIME: `text/plain`