# Progetto: Integrazione API con ChatGPT

## To-Do List

### 1. Analisi dei requisiti

- Definire il contesto del progetto e i suoi obiettivi principali.

- Identificare le funzionalità richieste (es.: gestione delle richieste API, autenticazione, memorizzazione delle risposte, ecc.).

- Identificare le dipendenze software (librerie, strumenti).

- Scegliere un ambiente di sviluppo (locale, server, ecc.).

### 2. Progettazione dell'architettura

- Decidere la struttura del progetto (moduli, directory, ecc.).

- Pianificare l’uso dell’API ChatGPT:

- - Gestione delle chiavi API.

- - Chiamate alle API (modelli disponibili, parametri, limiti di token).

- Progettare eventuali database per memorizzare conversazioni o configurazioni.

- Pianificare la gestione degli errori (es.: timeout, chiavi API non valide).

### 3. Implementazione

- Installare le dipendenze necessarie (es.: openai, requests, ecc.).

- Configurare il progetto:

- - Creare file di configurazione per le credenziali API.

- - Impostare variabili di ambiente sicure.

- Implementare la logica di chiamata alle API:

- - Funzioni per inviare richieste.

- - Gestione delle risposte.

- Implementare il frontend (se necessario):

- - CLI per interazione testuale.

- - Applicazione web (es.: Flask, FastAPI).

- Aggiungere funzionalità avanzate:

- - Salvataggio delle sessioni.

- - Creazione di prompt dinamici.

### 4. Testing

- Scrivere test unitari per le principali funzionalità.

- Simulare casi d’uso comuni.

- Testare l’uso del progetto con limiti di token o chiavi API errate.

### 5. Documentazione

- Documentare il codice (docstring).

- Creare una guida per l’installazione e l’utilizzo del progetto.

- Includere una guida per risolvere problemi comuni.

### 6. Manutenzione

- Pianificare aggiornamenti futuri (es.: supporto per nuovi modelli o funzionalità API).

- Monitorare il consumo delle API e i costi associati.

## Documento di Analisi e Sviluppo

### 1. Introduzione

Il progetto ha l'obiettivo di integrare in modo completo l'API di ChatGPT per automatizzare la gestione di conversazioni, analisi e memorizzazione delle risposte, offrendo un'interfaccia user-friendly.

### 2. Requisiti

Funzionali:

- Inviare e ricevere messaggi tramite l’API di OpenAI.

- Supportare personalizzazione dei prompt.

- Memorizzare conversazioni storiche.

Non funzionali:

- Rispetto dei limiti di token imposti dall’API.

- Prestazioni accettabili per risposte rapide.

### 3. Architettura

Componenti principali:

- Modulo di gestione delle chiamate API.

- Modulo per l’archiviazione delle conversazioni.

- Interfaccia utente (CLI o web).

Tecnologie utilizzate:

- Linguaggio: Python.

- Librerie: openai, flask/fastapi (per il frontend web).

- Database: SQLite o PostgreSQL (per la persistenza delle conversazioni).

Flusso operativo:

1. L’utente invia un messaggio.

2. Il messaggio viene elaborato e inviato all’API di ChatGPT.

3. La risposta viene processata e visualizzata.

4. I dati possono essere memorizzati in un database.

### 4. Dettagli di Implementazione

Setup iniziale:

- Creazione di un ambiente virtuale: python -m venv venv.

- Installazione delle dipendenze: pip install openai flask.

Esempio di chiamata API:

```python  
import openai  
  
openai.api\_key = "your\_api\_key"  
  
response = openai.ChatCompletion.create(  
 model="gpt-4",  
 messages=[{"role": "user", "content": "Ciao, come stai?"}]  
)  
print(response['choices'][0]['message']['content'])  
```

Gestione degli errori:

- Uso di try-except per gestire timeout o errori di rete.

### 5. Testing

Creare test automatizzati con pytest.

Simulare errori API (es.: chiavi errate).

### 6. Conclusione

Il progetto fornirà una soluzione robusta per interagire con ChatGPT, facilmente estendibile per ulteriori funzionalità.

Una volta completato, potrà essere integrato in progetti più ampi o distribuito a utenti finali.