Test tecnico Software Engineer 🍕

Parte 1

Sviluppa una piccola libreria Python per l'interazione con Large Language Models (LLMs).

Release 0.1.0

- Implementa una classe astratta `LLMClient` che definisce l'interfaccia per:
  - Gestione di un LLM (caricamento, configurazione)
  - Response handling
  - Error handling
- Implementa due classi mock:
  - `OpenAIMockClient`: simula risposte di un LLM OpenAI
  - `AnthropicMockClient`: simula risposte di un LLM Anthropic
- Logging
  - Logging delle chiamate
  - Logging degli errori
  - Configurazione dei log levels
- Testing: unit test per ogni componente

Release 0.2.0

- Breaking change: aggiungi un nuovo parametro obbligatorio in tutte le chiamate ai LLM
- Usa Pydantic per fare validazione input/output

Containerizzazione

- Fai in modo che la libreria sia testabile tramite docker

Deliverable

Un repository GitHub con:

- Codice sorgente
- Versioni taggate (0.1.0, 0.2.0)
- Test suite
- Dockerfile e docker-compose.yml
- README con istruzioni di setup e testing

Parte 2

Scenario

Il team R&D ha sviluppato diversi moduli Python indipendenti per NLP (text cleaning, entity extraction, sentiment analysis, text generation). Ogni modulo ha una propria interfaccia.

Esempio di moduli esistenti:

```
class TextCleaner:
   def clean(self, text: str) -> str:
       # rimuove caratteri speciali
       return text.strip()

class EntityExtractor:
   def extract_entities(self, text: str) -> List[str]:
       # estrae entità dal testo
       return ['entity1', 'entity2']

class SentimentAnalyzer:
   def analyze(self, text: str) -> float:
       # calcola sentiment score (-1 to 1)
       return 0.8

class TextGenerator:
   def generate(self, prompt: str, max_length: int) -> str:
       # genera testo dato un prompt
       return f"Generated text based on: {prompt}"
```

Implementa un'architettura che permetta di:

1. Definire un'interfaccia comune per i moduli
2. Concatenare i moduli in pipeline tramite configurazione
3. Validare input/output tra moduli

La soluzione deve permettere di definire pipeline in formato YAML:

```
pipeline:
  - name: cleaner
    type: TextCleaner
    # ...
  - name: generator
    type: TextGenerator
    params:
        max_length: 100
```

Deliverable

- Codice sorgente
- Esempio di configurazione YAML funzionante
- Descrizione della soluzione implementata