10/09/2025

Presentazione PB

Gruppo 20 - A.A. 2024/25

7Commits

Capitolato C1- Artificial QI

Obiettivo Principale:

Realizzare un'unica applicazione che consenta di:

- Archiviare dataset di domande/risposte attese;
- Eseguire test automatizzati via API su LLM esterni;
- Valutare automaticamente la correttezza e verosimiglianza delle risposte;
- Generare e presentare risultati dettagliati.

Contesto:

- Necessità di valutare le risposte ottenute da un LLM rispetto a quelle attese.

ZUCCHETTI

7Commits

Frontend

Streamlit

Architettura:

- Componenti UI;
- Componenti di comunicazione;
- Componenti gestione dello stato.

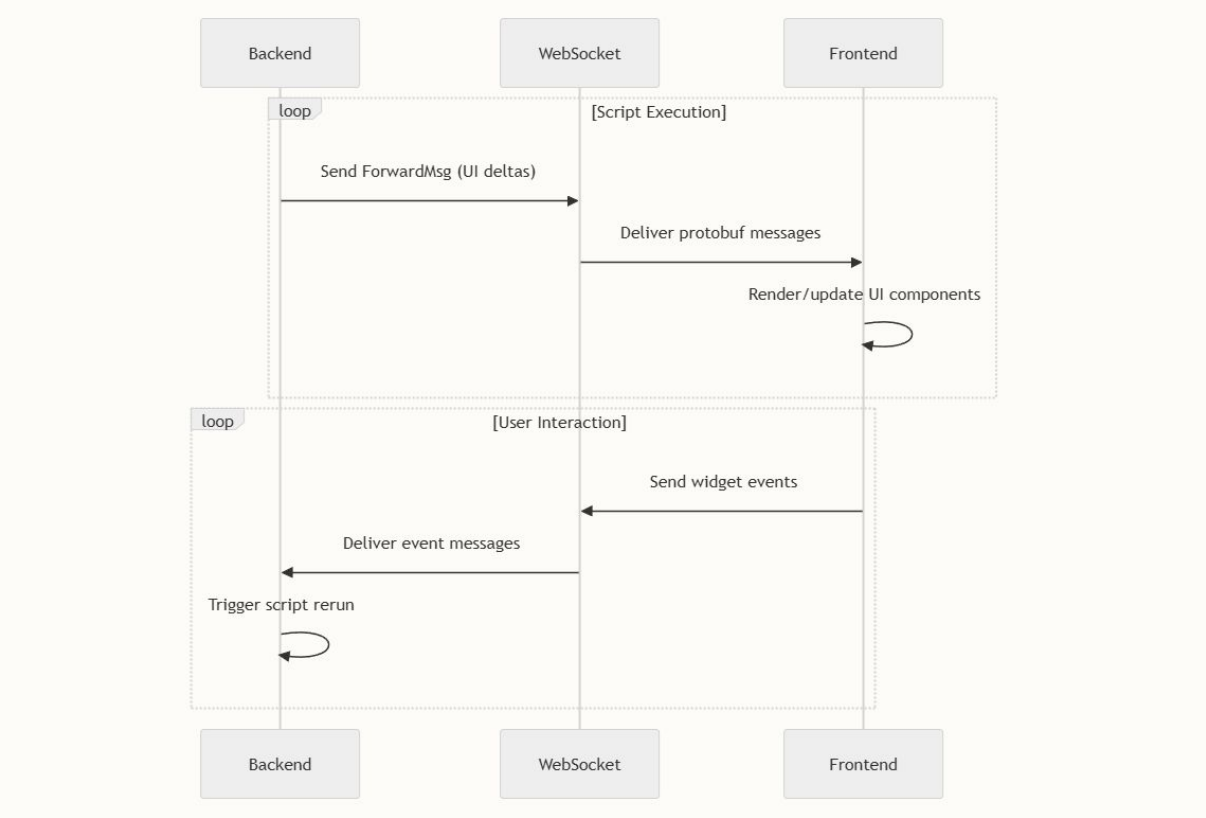
Backend

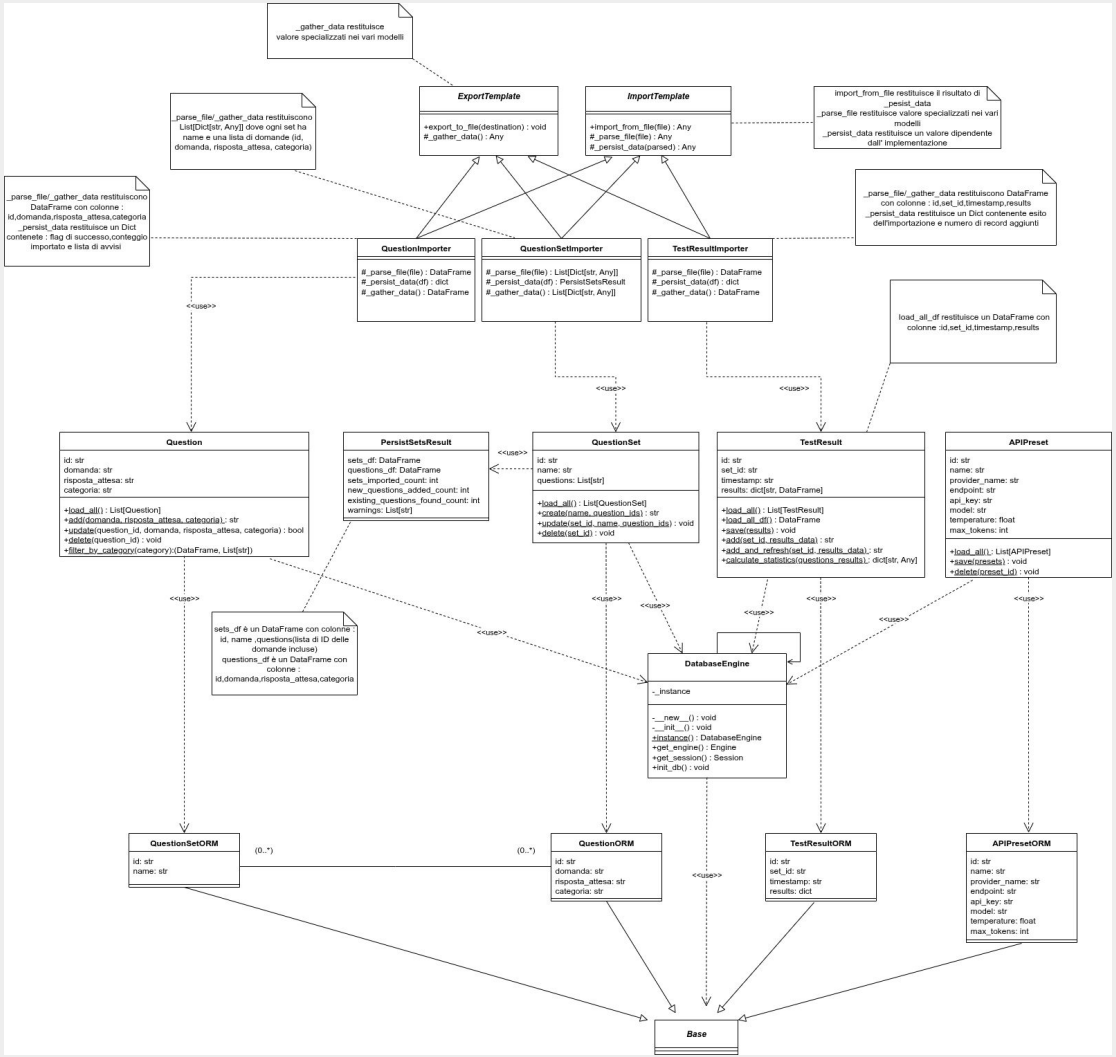
Python

Architettura:

- Componenti interfaccia utente;
- Logica di controllo;
- Strato dati;
- Integrazione con servizi esterni.

Comunicazione tra Frontend e Backend




_gather_data restituisce
valore specializzati nei vari modelli
ExportTemplate
+export_to_file(destination) : void
#_gather_data() : Any
ImportTemplate
+import_from_file(file) : Any
#_parse_file(file) : Any
#_persist_data(parsed) : Any
import_from_file restituisce il risultato di _pesist_data
_parse_file restituisce valore specializzati nei vari modelli
_persist_data restituisce un valore dipendente dall' implementazione
_parse_file/_gather_data restituiscono List[Dict[str, Any]] dove ogni set ha name e una lista di domande (id, domanda, risposta_attesa, categoria)
_parse_file/_gather_data restituiscono DataFrame con colonne : id,domanda,risposta_attesa,categoria
_persist_data restituisce un Dict contenente : flag di successo,conteggio importato e lista di avvisi
_parse_file/_gather_data restituiscono DataFrame con colonne : id,set_id,timestamp,results
_persist_data restituisce un Dict contenente esito dell'importazione e numero di record aggiunti
QuestionImporter
#_parse_file(file) : DataFrame
#_persist_data(df) : dict
#_gather_data() : DataFrame
QuestionSetImporter
#_parse_file(file) : List[Dict[str, Any]]
#_persist_data(df) : PersistSetsResult
#_gather_data() : List[Dict[str, Any]]
TestResultImporter
#_parse_file(file) : DataFrame
#_persist_data(df) : dict
#_gather_data() : DataFrame
load_all_df restituisce un DataFrame con colonne :id,set_id,timestamp,results
<<use>>
Question
id: str
domanda: str
risposta_attesa: str
categoria: str
+load_all() : List[Question]
+add(domanda, risposta_attesa, categoria) : str
+update(question_id, domanda, risposta_attesa, categoria) : bool
+delete(question_id) : void
+filter_by_category(category):(DataFrame, List[str])
PersistSetsResult
sets_df: DataFrame
questions_df: DataFrame
sets_imported_count: int
new_questions_added_count: int
existing_questions_found_count: int
warnings: List[str]
QuestionSet
id: str
name: str
questions: List[str]
+load_all() : List[QuestionSet]
+create(name, question_ids) : str
+update(set_id, name, question_ids) : void
+delete(set_id) : void
TestResult
id: str
set_id: str
timestamp: str
results: dict[str, DataFrame]
+load_all() : List[TestResult]
+load_all_df() : DataFrame
+save(results) : void
+add(set_id, results_data) : str
+add_and_refresh(set_id, results_data) : str
+calculate_statistics(questions_results) : dict[str, Any]
APIPreset
id: str
name: str
provider_name: str
endpoint: str
api_key: str
model: str
temperature: float
max_tokens: int
+load_all() : List[APIPreset]
+save(presets) : void
+delete(preset_id) : void
sets_df è un DataFrame con colonne id, name ,questions(lista di ID delle domande incluse)
questions_df è un DataFrame con colonne :
id,domanda,risposta_attesa,categoria
DatabaseEngine
-_instance
-__new__() : void
-__init__() : void
+instance() : DatabaseEngine
+get_engine() : Engine
+get_session() : Session
+init_db() : void
QuestionSetORM
id: str
name: str
(0..*)
(0..*)
QuestionORM
id: str
domanda: str
risposta_attesa: str
categoria: str
TestResultORM
id: str
set_id: str
timestamp: str
results: dict
APIPresetORM
id: str
name: str
provider_name: str
endpoint: str
api_key: str
model: str
temperature: float
max_tokens: int
Base

Design pattern architetturale: MVC

Separazione tra dati, interfaccia e logica di controllo:

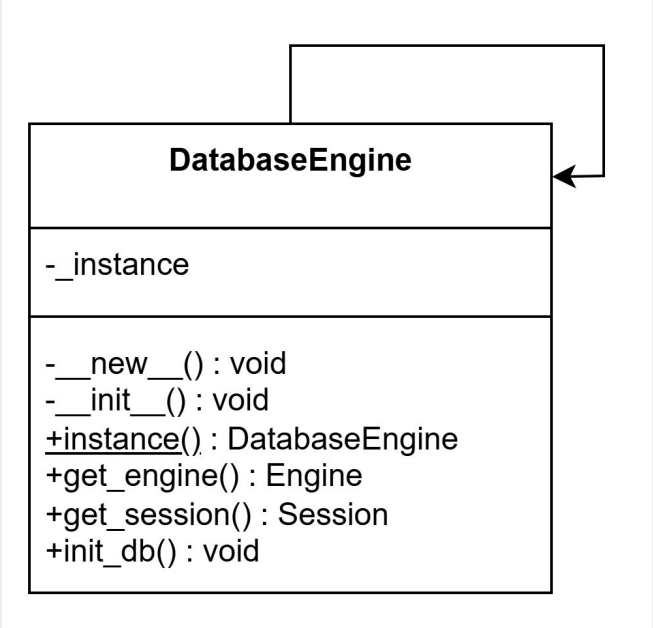
- **Model** → gestisce e memorizza i dati;
- **View** → presenta l'interfaccia e le interazioni;
- **Controller** → coordina il flusso e la comunicazione tra modello (dati) e vista (interfaccia utente).

Design pattern creazionale: Singleton

Garantisce un’unica istanza con accesso globale:

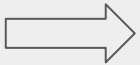
- Controllo centralizzato delle risorse;
- Riduzione dell’uso di memoria;
- Consistenza dei dati;
- Lazy initialization.

Design pattern creazionale: Singleton



Singleton: Testing (1/2)

Test unit sull'unicità del Database Engine



```python
def test_get_engine_uses_config_and_create_engine(monkeypatch):
    DatabaseEngine.reset_instance()  # assicura un singleton pulito
    db = DatabaseEngine.instance()
    fake_cfg = {'user': 'u', 'password': 'p', 'host': 'h', 'database': 'db'}
    monkeypatch.setattr(DatabaseEngine, '_load_config', lambda self: fake_cfg)
    called = {}

    def fake_ensure(self, cfg):
        called['ensure'] = cfg
    monkeypatch.setattr(DatabaseEngine, '_ensure_database', fake_ensure)
    fake_engine = SimpleNamespace(dispose=lambda: None)

    def fake_create_engine(url, pool_pre_ping=True, pool_recycle=3600):
        called['url'] = url
        return fake_engine
    monkeypatch.setattr(database, 'create_engine', fake_create_engine)

    engine = db.get_engine()
    assert engine is fake_engine
    assert called['ensure'] == fake_cfg
    assert 'mysql+pymysql://u:p@h:3306/db' in called['url']
    # la seconda chiamata riutilizza lo stesso engine
    assert db.get_engine() is fake_engine
```

Singleton: Testing (2/2)

Test unit sull'inserimento delle domande

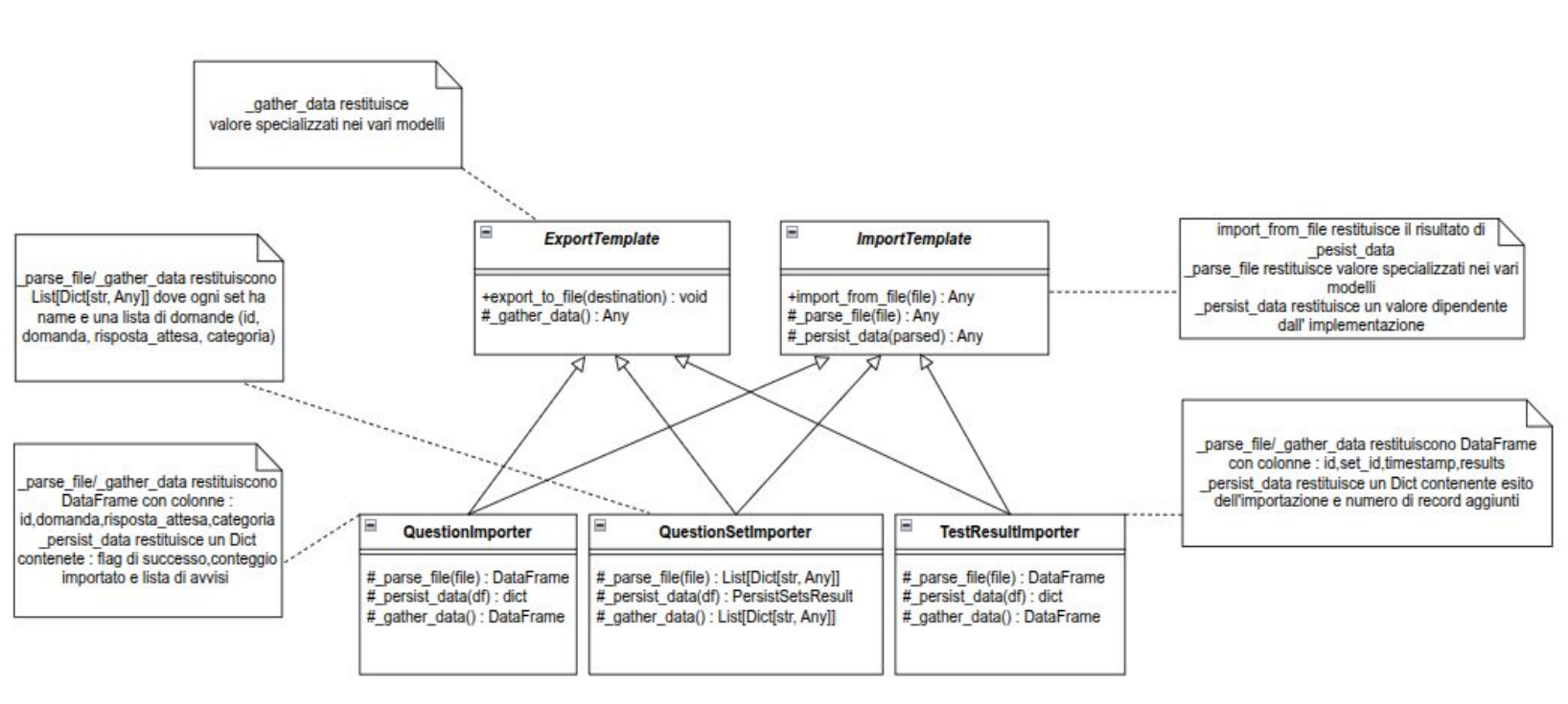
```python
def test_add_and_update_question(in_memory_db):
    qid = Question.add('d1', 'r1', 'c1')
    with DatabaseEngine.instance().get_session() as session:
        q = session.get(QuestionORM, qid)
        assert q.domanda == 'd1'
    assert Question.update(qid, domanda='d2', categoria='c2') is True
    with DatabaseEngine.instance().get_session() as session:
        q = session.get(QuestionORM, qid)
        assert q.domanda == 'd2'
        assert q.categoria == 'c2'
    assert Question.update('missing', domanda='x') is False
```

Design pattern comportamentale: Template Method

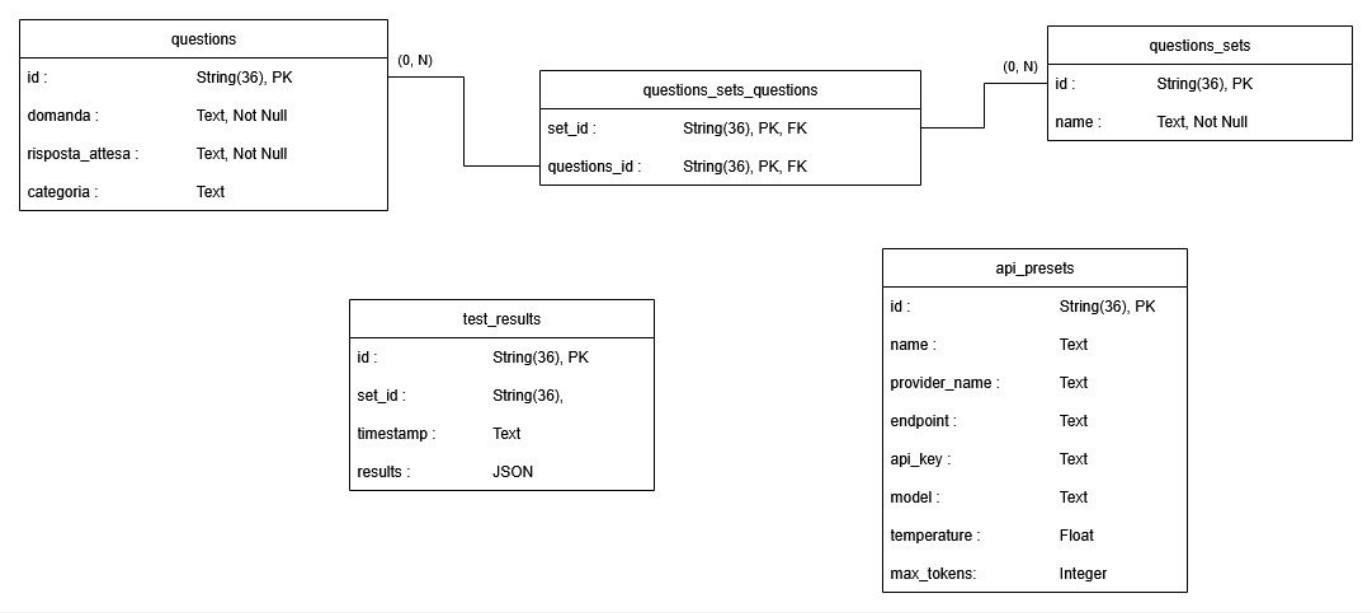
Definisce lo scheletro di un algoritmo, delegando i passi specifici alle sottoclassi.

- Eliminazione della duplicazione del codice;
- Flessibilità ed estensibilità;
- Separazione chiara dei compiti.

Design pattern comportamentale: Template Method



Database



Grazie

- Marco Cola
- Ruize Lin
- Stefano Dal Poz
- Giada Rossi
- Mattia Piva
- Giulia Hu