Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Corso di Basi di Dati II

PokéMongo

Docenti
Tortora Genoveffa
Di Biasi Luigi

Studenti
Fortino Rosalia
Matricola: 0522502023
Gabriele Gaudiosi
Matricola: 0522502053

Anno Accademico 2024/2025

Indice

Capitolo 1

Introduzione

1.1 Obiettivo generale

L'obiettivo del progetto è la realizzazione di un sistema informativo basato su un database NoSQL che gestisca dati relativi ai Pokémon e ai loro allenatori. Il sistema punta a garantire operazioni CRUD complete su due entità principali e a offrire altre funzionalità di query. L'accesso alle funzionalità avverrà tramite un'interfaccia utente semplice e interattiva, pensata per rendere la gestione delle informazioni intuitiva e immediata.

1.2 Approccio NoSQL

Abbiamo scelto un approccio NoSQL per sfruttare la flessibilità e la scalabilità offerte da MongoDB, in particolare per la gestione di dati semi-strutturati e collegati in modo naturale. Le caratteristiche di disponibilità e tolleranza alla partizione proprie di MongoDB lo rendono adatto a contesti distribuiti, in linea con i principi del teorema CAP.

1.3 Dataset utilizzato

Il sistema è costruito a partire dal dataset Pokémon Trainers, disponibile su Kaggle (`https://www.kaggle.com/datasets/lrcusack/pokemontrainers`). Il dataset include informazioni su allenatori, i loro Pokémon, livelli, tipi e altre caratteristiche rilevanti.

1.4 Tecnologie utilizzate

- Backend: Sviluppato in Python con Flask e PyMongo per la gestione flessibile delle richieste HTTP e l'integrazione con MongoDB;
- Database: MongoDB come soluzione NoSQL document-oriented;
- Frontend: Interfaccia realizzata in HTML e JavaScript per un'esperienza utente intuitiva.

Capitolo 2

Descrizione del Miniworld

2.1 Entità

Il miniworld del progetto ruota attorno a due entità fondamentali:

- **Trainer**: rappresenta l'allenatore, ovvero l'utente principale del sistema. Ogni trainer può essere associato a uno o più Pokémon;
- **Pokémon**: rappresenta ciascun esemplare posseduto da un trainer. Ogni Pokémon è associato a un solo trainer e ne eredita l'identificativo.

2.2 Relazioni tra entità

Le entità Trainer e Pokémon sono collegate da una relazione di tipo uno-a-molti (1:N):

- Un singolo trainer può possedere più Pokémon;
- Ogni Pokémon è associato a un solo trainer.

Questa relazione è implementata attraverso il campo `trainerID`, presente in entrambe le collection:

- Nella collection dei trainer come identificativo primario;
- Nella collection dei Pokémon come chiave esterna.

2.3 Attributi

Tabella 2.1: Attributi dell'entità Trainer

Attributo	**Descrizione**
trainerID	Identificatore univoco dell'allenatore
trainername	Nome dell'allenatore

Tabella 2.2: Attributi dell'entità Pokémon

Attributo	**Descrizione**
trainerID	Riferimento all'allenatore proprietario
place	Luogo di provenienza del Pokémon
pokename	Nome del Pokémon
pokelevel	Livello del Pokémon
type1	Tipo primario
type2	Tipo secondario (se presente)
hp	Punti salute attuali
maxhp	Punti salute massimi
attack	Valore di attacco
defense	Valore di difesa
spatk	Valore di attacco speciale
spdef	Valore di difesa speciale
speed	Valore di velocità

Capitolo 3

Contesto Applicativo

3.1 Scenari di utilizzo

Il sistema informativo può essere utilizzato in vari scenari:

- Strumento gestionale per amministrare i dati di allenatori e Pokémon;
- Piattaforma di consultazione per analisi e statistiche;
- Base per applicazioni ludiche o sociali legate a gamification, community e tornei.

La struttura flessibile del modello NoSQL lo rende particolarmente adatto a ciascuno di questi contesti d'uso.

3.2 Operazioni

Il sistema supporta un insieme di operazioni fondamentali per la gestione e l'interrogazione del database:

- Operazioni CRUD: per entrambe le entità (Trainer e Pokémon) sono previste funzionalità CRUD, che permettono di aggiungere nuovi dati, modificarli, o eliminarli in modo controllato e coerente;
- Query avanzate: visualizzazione di tutti i Pokémon di un specifico allenatore e ricerca degli allenatori con Pokémon di livello superiore a un valore specificato.

Capitolo 4

Soluzione Proposta

4.1 Modellazione delle collection

Il sistema è basato su due collection distinte:

```
Trainers
{
  "_id": ObjectId,
  "trainerID": Integer,
  "trainername": String
}
```

```
Pokemon
{
  "_id": ObjectId,
  "trainerID": Integer,
  "place": String,
  "pokename": String,
  "pokelevel": Integer,
  "type1": String,
  "type2": String,
  "hp": Integer,
  "maxhp": Integer,
  "attack": Integer,
  "defense": Integer
}
```

4.2 Relazione tra le collection

La relazione tra le due entità è mantenuta tramite il campo `trainerID`, presente sia nei documenti `Pokemon` (come chiave esterna) che in `Trainers` (come chiave primaria). Questa relazione 1:N consente di eseguire operazioni di join tramite l'operatore `$lookup`.

4.3 Strategie di modellazione NoSQL

È stato scelto un approccio con documenti separati, piuttosto che annidare i Pokémon direttamente nel documento dell'allenatore. Questa scelta:

- Garantisce maggiore flessibilità;
- Evita problemi di dimensione massima dei documenti MongoDB;
- Semplifica aggiornamenti parziali su singoli Pokémon.

4.4 Operazioni

Il backend, sviluppato in Flask, espone endpoint RESTful per operazioni CRUD su trainer e Pokémon e per visualizzazioni aggregate tramite `$lookup`. Le operazioni sono protette da retry logic per garantire resilienza e continuità in ambienti distribuiti.

4.5 Interfaccia utente

L'interfaccia utente, sviluppata con HTML e JavaScript, offre un accesso intuitivo a tutte le funzionalità attraverso tre sezioni principali (gestione Pokémon, gestione trainer e visualizzazione join), con un design user-friendly per utenti di ogni tipo.

Capitolo 5

Metodologia Utilizzata

5.1 Fasi dello sviluppo

- Analisi del dataset: studio del formato e del contenuto del dataset Pokémon Trainers;
- Modellazione NoSQL: progettazione delle collection MongoDB tenendo conto della natura dei dati e delle operazioni richieste;
- Progettazione backend: realizzazione degli endpoint REST in Flask, con gestione di errori, retry automatico e operazioni transazionali;
- Realizzazione frontend: sviluppo dell'interfaccia HTML e JS per visualizzare, inserire e modificare i dati tramite API.

5.2 Dettaglio delle query implementate

Il backend espone una serie di endpoint RESTful che eseguono query MongoDB tramite PyMongo, includendo operazioni CRUD e aggregazioni avanzate con `$lookup`.

Di seguito sono riportate le principali operazioni effettuate sul database da parte dell'applicazione. Per ciascuna operazione vengono mostrate esclusivamente le query fondamentali, estratte dalle funzioni corrispondenti, omettendo la logica di contorno e il codice di gestione HTTP.

5.2.1 CRUD su Trainer

Create trainer

```
1 trainers_col.insert_one(data, session=session)
```

Read Trainer

```
1 trainers_col.find(session=session)
```

Update Trainer

```
trainers_col.update_one(
    {"_id": ObjectId(trainer_id)},
    {"$set": data},
    session=session
)
```

Delete Trainer

```
trainers_col.delete_one(
    {"_id": ObjectId(trainer_id)},
    session=session
)

pokemon_col.delete_many(
    {"trainerID": trainer_id},
    session=session
)
```

5.2.2 CRUD su Pokémon

Create Pokémon

```
pokemon_col.insert_one(data, session=session)
```

Read Pokémon

```
pokemon_col.find(session=session)
```

Update Pokémon

```
pokemon_col.update_one(
    {"_id": ObjectId(pokemon_id)},
    {"$set": data},
    session=session
)
```

Delete Pokémon

```
pokemon_col.delete_one(
    {"_id": ObjectId(pokemon_id)},
    session=session
)
```

5.2.3 JOIN

Elenco dei Pokémon di un Trainer

```
pipeline = [
    {"$match": {"trainerID": trainer_id}},
    {"$lookup": {
        "from": "Pokemon",
        "localField": "trainerID",
        "foreignField": "trainerID",
        "as": "pokemon"
    }},
    {"$project": {
        "_id": 0,
        "trainerID": 1,
        "trainername": 1,
        "pokemon": 1
    }}
]
trainers_col.aggregate(pipeline, session=session)
```

Elenco dei Trainer con Pokémon sopra un certo livello

```
pipeline = [
    {"$match": {"pokelevel": {"$gt": min_level}}},
    {"$group": {
        "_id": "$trainerID",
        "pokemon_count": {"$sum": 1},
        "pokemon_list": {
            "$push": {
                "name": "$pokename",
                "level": "$pokelevel",
                "type1": "$type1",
                "type2": "$type2"
            }
        }
    }},
    {"$lookup": {
        "from": "Trainers",
        "localField": "_id",
        "foreignField": "trainerID",
        "as": "trainer_info"
    }},
    {"$unwind": "$trainer_info"},
    {"$project": {
        "_id": 0,
        "trainer": {
            "trainerID": "$_id",
            "name": "$trainer_info.trainername"
        },
        "pokemon": "$pokemon_list"
```

```
    }},
    {"$sort": {"trainer.trainerID": 1}}
]
pokemon_col.aggregate(pipeline, session=session)
```

5.3 Proprietà BASE e coerenza dei dati

Il sistema è progettato secondo il modello BASE (Basically Available, Soft state, Eventually consistent). A differenza del modello ACID tipico dei database relazionali, qui la coerenza è gestita:

- A livello applicativo;
- Tramite transazioni MongoDB;
- Con controlli espliciti lato backend.

5.4 Considerazioni sul Teorema CAP

Il sistema è stato progettato nel rispetto del Teorema CAP, che afferma che in un sistema distribuito è possibile garantire al massimo due tra: Consistenza (C), Disponibilità (A) e Tolleranza alla Partizione (P). Abbiamo scelto di dare priorità a Consistenza e Tolleranza alla Partizione, accettando una minore Disponibilità in scenari di rete degradati.

5.4.1 Partition Tolerance

La Tolleranza alla Partizione è una caratteristica intrinseca di MongoDB in configurazione replica set, e viene garantita dalle seguenti scelte progettuali:

- Connessione a replica set locale:

```
    mongodb://localhost:27017,localhost:27018,localhost:27019/pokemon_db
        ?replicaSet=rs0
```

- Timeout configurati per evitare blocchi lunghi:

```
    serverSelectionTimeoutMS=5000, connectTimeoutMS=2000
```

- Retry automatico per operazioni fallite (lettura e scrittura), tramite il decoratore `@retry`;
- Write Concern impostato su `majority` per garantire che una scrittura venga replicata con successo;
- Read Preference su `secondaryPreferred` per bilanciare traffico e resilienza.

5.4.2 Consistency

Abbiamo esplicitamente dato priorità alla Consistenza dei dati, soprattutto nelle operazioni critiche:

- Scritture con:

```
1    WriteConcern(w='majority', wtimeout=5000)
```

  Le modifiche vengono accettate solo se replicate sulla maggioranza dei nodi;

- Letture consistenti con:

```
1    ReadConcern(level='majority')
```

  Garantisce che i dati letti siano confermati dalla maggioranza e non provvisori;

- Uso di transazioni MongoDB per operazioni composte.