Cine Metrics

Luigi Emanuele Sica
Antonio Toppi
Luigi Allocca

Introduzione

- Data Base NoSQL (MongoDB)
- Dataset: Kaggle
- Principali funzionalità:
  - Operazioni CRUD
  - Ricerche personalizzate
  - Analisi statistiche

Scenari

Nome Scenari:

Aggiunta di un elemento

Ricerca documento

Visualizzazione statistiche

Scenario: Aggiunta di un elemento

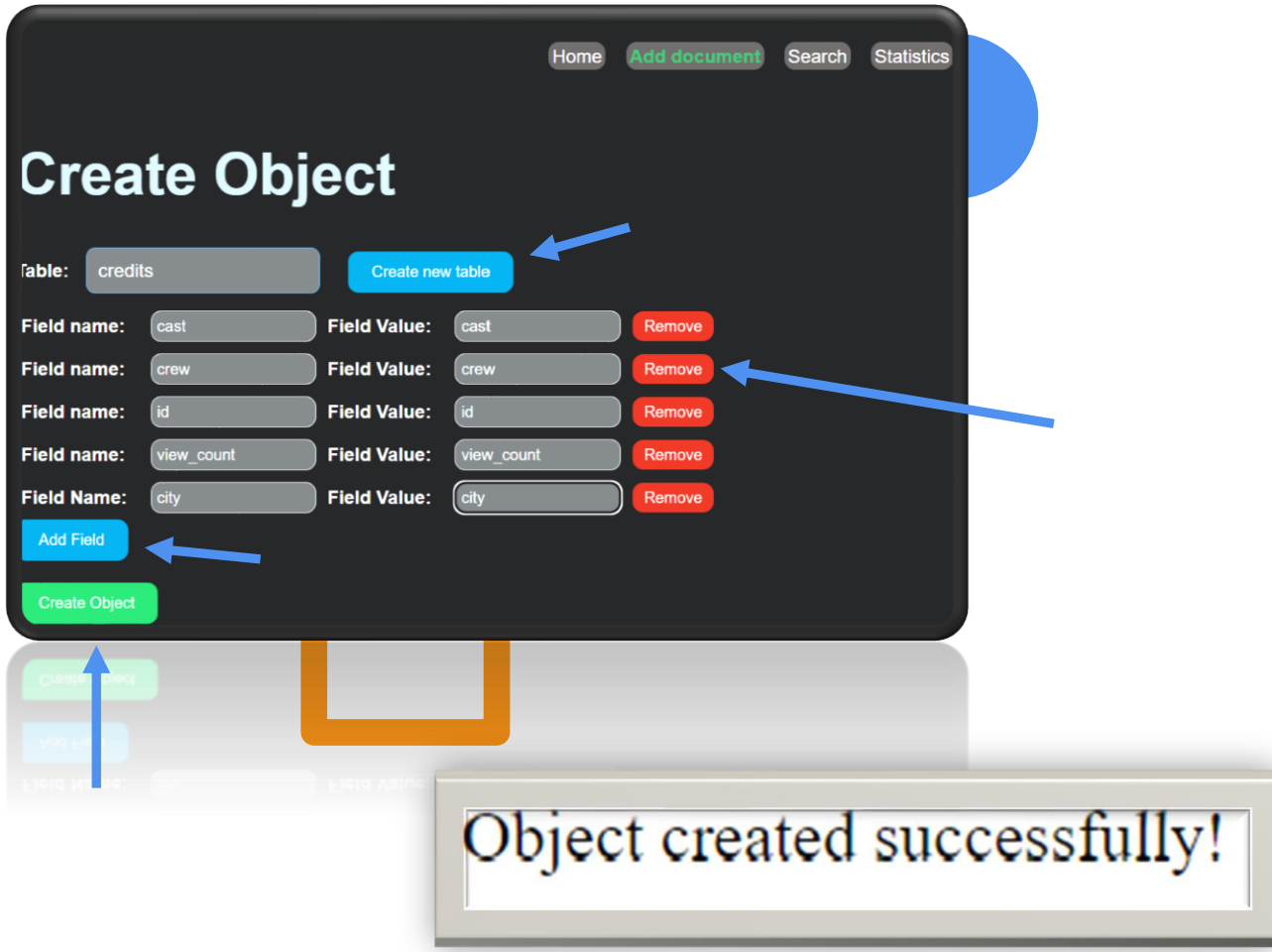
- Il Cliente (Mario) accede alla pagina web.
- Il sistema di Flask aprirà la schermata principale
- Mario tramite la barra di navigazione si muove nella sezione "Add document".

Scenario: Aggiunta di un elemento

- Mario, che si trova nella schermata dell'inserimento, seleziona che tipo di documento vuole inserire
- Il sistema automaticamente mostrerà i campi che compongono il documento
- Mario compilerà i campi e avrà la possibilità di aggiungerne di nuovi.

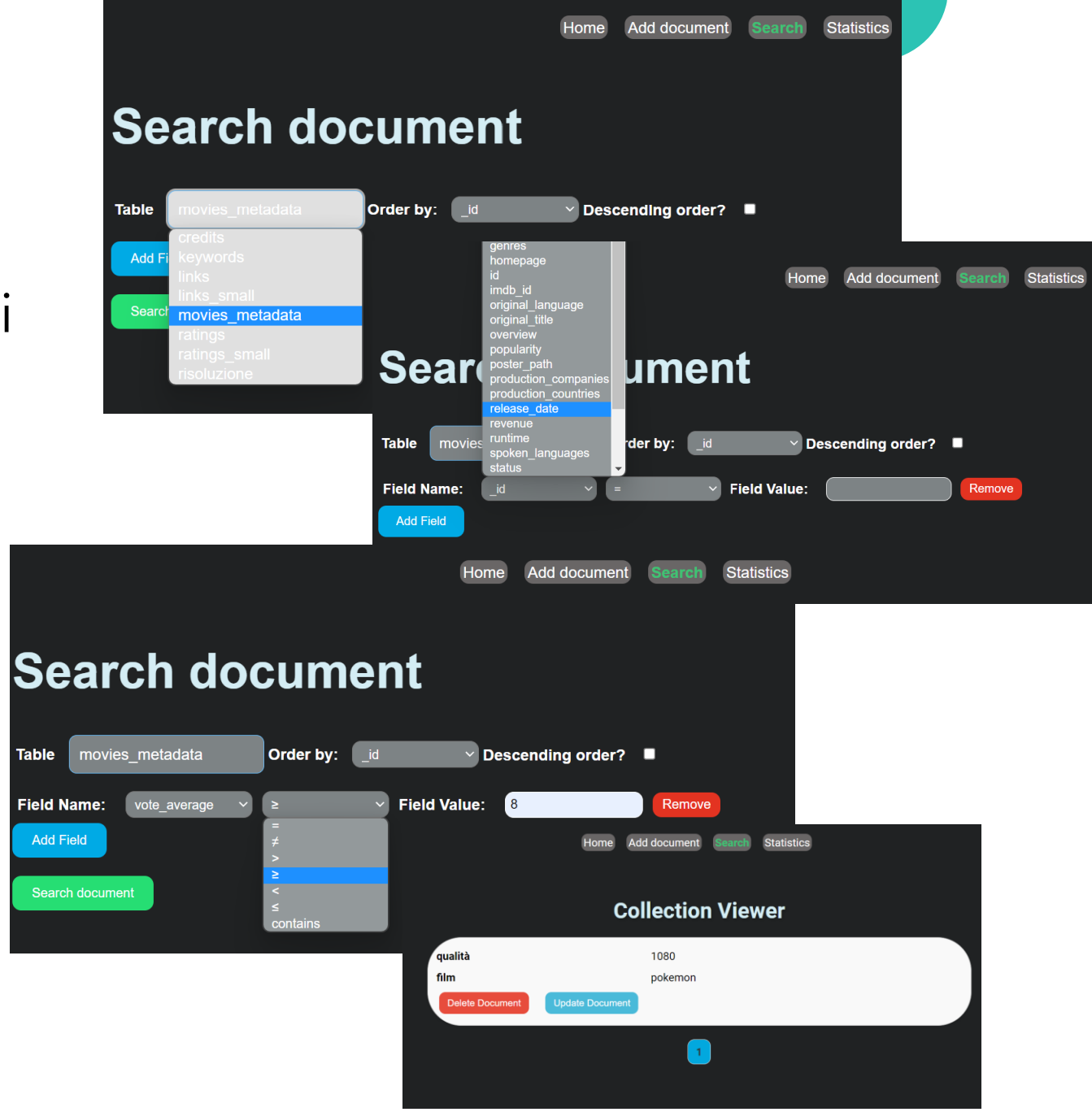
Scenario: Aggiunta di un elemento

- Mario dopo aver compilato i campi, seleziona il pulsante "Create Object".
- Il sistema provvederà ad inserire l'elemento e reindirizzerà l'utente ad una schermata di conferma.



Scenario: Ricerca di un documento

- Mario dalla barra di navigazione si sposta nella sezione "Search".
- Il sistema lo reindirizza nella pagina appropriata.
- Mario seleziona la tabella da cercare.
- Mario definisce le caratteristiche delle ricerca di suo interesse
- Cliccherà sulla funzionalità "Search document".
- Il sistema lo indirizza alla pagina contente i risultati della ricerca.



Scenario: Visualizzazione statistiche

- Tale scenario richiede l'accesso come admin.
- Una volta convalidato l'accesso dal sistema, l'admin viene indirizzato alla sezione "Statistics"
- IN questa pagina l'admin può visualizzare un istogramma e una mappa
- L'istogramma mostra i 10 generi che hanno generato un incasso medio più alto
- L'admin può selezionare il genere di film per ottenere gli incassi medi e confrontarli con i 10 generi presenti sull'istogramma.
- Dalla mappa invece può visionare i generi più popolari per ogni stato e vedere quanti film sono sati prodotti su ognuno di essi.

Requisiti Funzionali

Requisiti Funzionali

Utente

- RFU 1 - Il sistema deve fornire una pagina principale. **Priorità: Media**
- RFU 2 - Deve essere possibile effettuare operazioni di creazione dei nuovi documenti. **Priorità: Alta**
- RFU 3 - Deve essere possibile effettuare operazioni di ricerca sui documenti. **Priorità: Alta**
- RFU 4 - Deve essere possibile effettuare operazioni di aggiornamento dei documenti. **Priorità: Alta**
- RFU 5 - Il sistema deve mostrare l'avvenuto successo della operazioni. **Priorità: Media**
- RFU 6 - Il sistema deve mostrare un in caso di problemi con le transazioni. **Priorità: Alta**

Admin

- RFA 1 - Il sistema deve fornire all'admin la possibilità di effettuare l'accesso dedicato. **Priorità: Alta**
- RFA 2 - Deve essere possibile effettuare operazioni di rimozione dei documenti. **Priorità: Alta**
- RFA 3 - Il sistema deve mostrare statistiche sui film che hanno avuto incassi medi più alti. **Priorità: Alta**
- RFA 4 - Il sistema deve mostrare statistiche sui generi più popolari nei vari paesi. **Priorità: Alta**
- RFA 5 - Il sistema deve il numero di film per paese. **Priorità: bassa**

Requisiti Non Funzionali

Requisiti non funzionali: **Usabilità**

1

RNF 1 - Il sistema deve fornire un'interfaccia grafica.

Priorità: Media.

2

RNF 2 – Il sistema deve fornire effetti visivi all'utente per una migliore navigazione.

Priorità: Media.

3

RNF 3 – Il sito deve essere responsive: Smartphone, Tablet, Laptop, Televisore.

Priorità: Media.

4

RNF 4 – Il sito deve fornire una iterazione con i dati.

Priorità: Alta.

Requisiti non funzionali: **Prestazioni**

1

RNF 5 – Le funzionalità del sistema effettuano una transazione in meno di 2000ms.

Priorità: Alta.

2

RNF 6 - Il sistema servirà fino a 100 utenti simultaneamente.

Priorità: Alta.

Requisiti non funzionali

Sicurezza:

1

RNF 7 – Gli utenti non devono poter effettuare operazioni consentite solo agli admin.

Priorità: Alta.

Affidabilità:

2

RNF 8 - Il sistema risponde con un messaggio di errore in caso di fallimenti della transazione.

Priorità: Media.

Sopportabilità:

3

RNF 9 - Il sistema deve essere facilmente comprensibile da nuovi sviluppatori.

Priorità: Media.

Manutenibilità:

4

RNF 10 – Le funzionalità del sistema devono essere debolmente accoppiate.

Priorità: Bassa.

Target environnement

Cine Metrics, sarà web-based, quindi accessibile da qualsiasi dispositivo dotato di browser che sia connesso a Internet. Per implementarlo verrà utilizzato un Web Server Flask che interagirà con MongoDB.

Sviluppo del sistema:

- CSS, HTML e JavaScript per la gestione del frontend
- Python come linguaggio di scrittura per il backend.
- MongoDB per gestione dei dati.
- Flask come web server.

Preprocessing

```
adult = "False"
belongs_to_collection = """
            {'id': 10194, 'name': 'Toy Story Collection',
            'poster_path': '/7G9915LfUQ2lVfwMEEhDsn3kT4B.jpg',
            'backdrop_path': '/9FBwqcd9IRruEDUrTdcaafOMKUq.jpg'}"""
budget = "30000000"
genres = """
            [{'id': 16, 'name': 'Animation'},
            {'id': 35, 'name': 'Comedy'},
            {'id': 10751, 'name': 'Family'}]
        """
homepage = "http://toystory.disney.com/toy-story"
id = "862"
imdb_id = "tt0114709"
original_language = "en"
original_title = "Toy Story"
overview = "Led by Woody, Andy's toys live happily in his room until Andy's..."
popularity = "21.946943"
poster_path = "/rhIRbceoE9lR4veEXuwCC2wARtG.jpg"
production_companies = "[{'name': 'Pixar Animation Studios', 'id': 3}]"
production_countries = "[{'iso_3166_1': 'US', 'name': 'United States of America'}]"
release_date = "1995-10-30"
revenue = "373554033"
runtime = "81.0"
spoken_languages = "[{'iso_639_1': 'en', 'name': 'English'}]"
status = "Released"
tagline = ""
title = "Toy Story"
video = "False"
vote_average = "7.7"
vote_count = "5415"
```

Preprocessing

```python
def csv_to_json(csv_file, file_name, mongo):
    data = []
    i = 1
    with open(csv_file, 'r') as file:
        csv_data = csv.DictReader(file)
        for row in csv_data:
            k: dict = row
            new_k = dict()
            for key, value in k.items():
                if value in ["id", "..."]:
                    new_k[key] = value
                    continue
                try:
                    new_k[key] = eval(value)
                    continue
                except:
                    pass
                new_k[key] = value
            data.append(new_k)
            if len(data) == 100000:
                i += 1
                try:
                    mongo[file_name[:-4]].insert_many(data, ordered=False)
                except:
                    exit(1)
                data.clear()
    print(data[0])
    mongo[file_name[:-4]].insert_many(data, ordered=False)
    print("parsed " + file_name)
```

Preprocessing

Prima

```
adult = "False"
belongs_to_collection = """
            {'id': 10194, 'name': 'Toy Story Collection',
            'poster_path': '/7G9915LfUQ2lVfwMEEhDsn3kT4B.jpg',
            'backdrop_path': '/9FBwqcd9IRruEDUrTdcaafOMKUq.jpg'}"""
budget = "30000000"
genres = """
            [{'id': 16, 'name': 'Animation'},
            {'id': 35, 'name': 'Comedy'},
            {'id': 10751, 'name': 'Family'}]
        """
homepage = "http://toystory.disney.com/toy-story"
id = "862"
imdb_id = "tt0114709"
original_language = "en"
original_title = "Toy Story"
overview = "Led by Woody, Andy's toys live happily in his room until Andy's..."
popularity = "21.946943"
poster_path = "/rhIRbceoE9lR4veEXuwCC2wARtG.jpg"
production_companies = "[{'name': 'Pixar Animation Studios', 'id': 3}]"
production_countries = "[{'iso_3166_1': 'US', 'name': 'United States of America'}]"
release_date = "1995-10-30"
revenue = "373554033"
runtime = "81.0"
spoken_languages = "[{'iso_639_1': 'en', 'name': 'English'}]"
status = "Released"
tagline = ""
title = "Toy Story"
video = "False"
vote_average = "7.7"
vote_count = "5415"
```

Dopo

```
_id: ObjectId('647b2d51dcf470613b40b6dc')
adult: false
▸ belongs_to_collection: Object
budget: 30000000
▸ genres: Array
homepage: "http://toystory.disney.com/toy-story"
id: 862
imdb_id: "tt0114709"
original_language: "en"
original_title: "Toy Story"
overview: "Led by Woody, Andy's toys live happily in his room until Andy's birthd…"
popularity: 21.946943
poster_path: "/rhIRbceoE9lR4veEXuwCC2wARtG.jpg"
▸ production_companies: Array
▸ production_countries: Array
release_date: 1955
revenue: 373554033
runtime: 81
▸ spoken_languages: Array
status: "Released"
tagline: ""
title: "Toy Story"
video: false
vote_average: 7.7
vote_count: 5415
```

```
▾ belongs_to_collection: Object
    id: 10194
    name: "Toy Story Collection"
    poster_path: "/7G9915LfUQ2lVfwMEEhDsn3kT4B.jpg"
    backdrop_path: "/9FBwqcd9IRruEDUrTdcaafOMKUq.jpg"
budget: 30000000
```

CRUD - Create

```python
@app.route('/create-document', methods=['POST'])
def create_object():
    # Riceve i dati del form per creare un nuovo documento nel database
    field_names = request.form.getlist('field_name[]')
    field_values = request.form.getlist('field_value[]')
    table = request.form.get("table")

    # Crea un nuovo oggetto dinamicamente usando i dati ricevuti dal form
    new_object = dict()
    for name, value in zip(field_names, field_values):
        if name == "_id":
            value = ObjectId(value)
        try:
            value = json.loads(value)
        except:
            pass
        new_object[name] = value

    # Esempio: Accesso all'oggetto creato dinamicamente
    for name, value in new_object.items():
        print(f"{name}: {value}")
    new_object[VIEW_COUNT_FIELD] = 0
    # Inserisce il nuovo oggetto nel database nella collezione specificata
    result = db[table].insert_one(new_object)
```

CRUD - Read

```
# Crea un nuovo oggetto di query dinamicamente usando i parametri ricevuti dalla query string
query = {}
for name, op, value in zip(field_names, field_operation, field_values):
    if name == "_id":
        value = ObjectId(value)
    try:
        value = json.loads(value)
    except:
        pass

    if op == "containsIgnoreCase":
        # Create a case-insensitive regex pattern
        pattern = re.compile(re.escape(value), re.IGNORECASE)
        condition = {"$regex": pattern}
    else:
        condition = {op: value}

    if name in query:
        query[name][op] = value
    else:
        query[name] = condition
```

```
def render_query(collection_name: str, query: Mapping[str, Any], page: int, order_field: str, order_desc: bool,
                 is_admin: bool):
    order_type = pymongo.DESCENDING if order_desc else pymongo.ASCENDING

    # Calcola il numero di documenti da saltare in base al numero di pagina e al numero di documenti per pagina
    skip = (page - 1) * ITEMS_PER_PAGE
    collection = db[collection_name]
    # Recupera i documenti dalla collezione con la paginazione
    items = collection.find(query).skip(skip).limit(ITEMS_PER_PAGE).sort(order_field, order_type)
```

CRUD - Update

```python
@app.route('/apply-update/<collection_name>/<object_id>', methods=["POST"])
def apply_update(collection_name, object_id):
    field_names = request.form.getlist('field_name[]')
    field_values = request.form.getlist('field_value[]')
    new_object = dict()
    for name, value in zip(field_names, field_values):
        if name == "_id":
            value = ObjectId(value)
        try:
            value = json.loads(value)
        except:
            pass
        new_object[name] = value
    print(new_object)
    result = db[collection_name].update_one({'_id': ObjectId(object_id)}, {'$set': new_object})
    if result.acknowledged:
        return "Object updated successfully!"
    else:
        return "Error in update"
```

CRUD - Delete

```python
@app.route('/delete/<collection_name>/<document_id>', methods=['POST'])
def delete_document(collection_name, document_id):
    # Riceve il nome della collezione e l'ID del documento da eliminare
    print(collection_name, document_id)
    collection = db[collection_name]
    document_id = ObjectId(document_id)

    # Elimina il documento dalla collezione
    result = collection.delete_one({'_id': document_id})

    # Restituisce il numero di documenti eliminati come JSON
    return jsonify({'deleted_count': result.deleted_count})
```

Statistiche

```
def media_revenue():
    collection = db['movies_metadata']
    genre_list = collection.distinct('genres.name')
    genre_list.sort()
    pipeline = [
        {
            "$unwind": "$genres"
        },
        {
            "$match": {
                "genres.name": {"$ne": None}
            }
        },
        {
            "$group": {
                "_id": "$genres.name",
                "averageRevenue": {"$avg": "$revenue"}
            }
        },
        {
            "$sort": {"averageRevenue": -1}
        },
        {
            "$limit": 10
        }
    ]
    genres_revenue_10 = list(collection.aggregate(pipeline))
    return genres_revenue_10
```


Fine