



safeSleep

Luigi Consiglio [0522501894]

Eljon Hida [0522501890]

Link repository: <https://github.com/Eljon99/SafeSleep>

Panoramica

Il progetto proposto è una web-application che permette di gestire un database contenente dati sul sonno e sulla salute di un gruppo di persone. Oltre alle consuete operazioni CRUD, saranno condotte analisi specifiche per estrarre informazioni significative. Per ottenere queste informazioni, verranno eseguite query che combinano dati provenienti da diverse tabelle.

Obiettivi

1. Implementazione delle operazioni CRUD (creazione, lettura, aggiornamento, eliminazione).
2. Uso di query per avere delle metriche e fare delle analisi più specifiche sui dati a disposizione.

Tecnologie

Il progetto userà le seguenti tecnologie:

- **MongoDB:** *è un database NoSQL orientato ai documenti, che utilizza documenti simili a JSON, con schemi dinamici. È ideale per gestire grandi volumi di dati non strutturati o semi-strutturati. MongoDB permette di memorizzare e recuperare dati in modo rapido e scalabile, rendendolo adatto per applicazioni moderne e flessibili. Per facilitare l'inserimento e la creazione del database a partire dal dataset, è stato utilizzato anche **MongoDB Compass**, uno strumento con interfaccia grafica che consente di eseguire query, aggregazioni e analisi dei dati.*
- **Python:** *è un linguaggio di programmazione ad alto livello, noto per la sua semplicità e leggibilità del codice. Viene utilizzato in una vasta gamma di applicazioni, tra cui lo sviluppo web, l'automazione, l'analisi dei dati e l'intelligenza artificiale.*
- **Flask:** *Flask è un micro-framework per Python, progettato per essere leggero e modulare. È spesso utilizzato per sviluppare applicazioni web e API.*
- **React:** *è una libreria JavaScript per la costruzione di interfacce utente. Viene utilizzata per creare applicazioni web dinamiche e responsive.*

- **PyCharm:** *PyCharm è un ambiente di sviluppo integrato (IDE) per il linguaggio Python, progettato per offrire strumenti avanzati di codifica, debugging e testing, ottimizzando la produttività degli sviluppatori grazie a funzionalità come il completamento del codice, il refactoring intelligente e l'integrazione con sistemi di controllo di versione.*

Scelta e gestione del Dataset

I. Scelta del Dataset

Il dataset scelto è: "Sleep_health_and_lifestyle_dataset.csv" preso da Kaggle: <https://www.kaggle.com/datasets/uom190346a/sleep-health-and-lifestyle-dataset>

Il dataset Sleep Health and Lifestyle comprende 400 righe e 13 colonne, che coprono un'ampia gamma di variabili relative al sonno e alle abitudini quotidiane. Include dettagli quali sesso, età, occupazione, durata del sonno, qualità del sonno, livello di attività fisica, livelli di stress, categoria BMI, pressione sanguigna, frequenza cardiaca, passi giornalieri e presenza o assenza di disturbi del sonno.

II. Gestione del Dataset

Abbiamo diviso il dataset principale Sleep_health_and_lifestyle_dataset.csv in:

- persona.csv
- diario_persona.csv

Abbiamo diviso la collezione MongoDB originale, chiamata 'sleep_data', in due nuove collezioni: 'persona' e 'diario_persona'. Vediamo i dettagli:

1. Definizione delle collezioni

- La collezione 'sleep_data' contiene i dati grezzi.

- 'persona' e 'diario_persona' sono le due nuove collezioni in cui i dati verranno suddivisi.
- 'contatori' tiene traccia dei contatori utilizzati per generare un identificativo incrementale ('Person ID').

2. Suddivisione dei dati

Il codice procede con la suddivisione delle collezioni

```
# Processare ogni documento nella collezione originale
for document in original_collection.find():
    # Assume che ogni documento abbia un campo _id unico

    # Ottieni il prossimo Person ID autoincrementale
    next_person_id = get_next_sequence_value("person_id")

    # Documento per la collezione persona
    person_document = {
        'Person ID': next_person_id,
        'Gender': document['Gender'],
        'Age': document['Age'],
        'Occupation': document['Occupation'],
        'Physical Activity Level': document['Physical Activity Level'],
        'BMI Category': document['BMI Category']
    }

    # Documento per la collezione diario_persona
    diary_document = {
        'Person ID': next_person_id,
        'Sleep Duration': document['Sleep Duration'],
        'Quality of Sleep': document['Quality of Sleep'],
        'Stress Level': document['Stress Level'],
        'Blood Pressure': document['Blood Pressure'],
        'Heart Rate': document['Heart Rate'],
        'Daily Steps': document['Daily Steps'],
        'Sleep Disorder': document['Sleep Disorder']
    }
```

In sintesi, il codice è progettato per organizzare i dati da una singola collezione in due collezioni separate, mantenendo un riferimento coerente tramite un identificatore incrementale ('Person ID').

Dizionario dei dati

Tabella Persona

Colonna	Descrizione
Person ID	Identificatore per ogni individuo
Gender	Sesso della persona (Maschio/Femmina/Altro)
Age	Età della persona in anni
Occupation	Occupazione/Professione della persona
Physical Activity Level	Il numero di minuti in cui la persona svolge attività fisica quotidiana
BMI Category	Indice di massa corporeo della persona (ad esempio, sottopeso, normale, sovrappeso)

Tabella Registro Persona

Colonna	Descrizione
Person ID	Identificatore della persona che associa ogni registro a quest'ultima
Sleep Duration	Numero di ore in cui la persona dorme al giorno
Quality of Sleep	Valutazione soggettiva della qualità del sonno, che va da 1 a 10
Stress Level	Valutazione soggettiva del livello di stress sperimentato dalla persona, che va da 1 a 10
Blood Pressure	Misurazione della pressione sanguigna della persona, indicata come pressione sistolica rispetto alla pressione diastolica
Heart Rate	Frequenza cardiaca a riposo della persona in battiti al minuto (bpm)
Daily Steps	Numero di passi che la persona compie al giorno
Sleep Disorder	Presenza o assenza di un disturbo del sonno nella persona (nessuno, insonnia, apnea notturna)

Configurazione del progetto

In questa sezione verranno illustrati i passaggi necessari per configurare e far funzionare il sistema implementato sul proprio ambiente di sviluppo.

Per configurare il progetto:

1. Clona il repository:
 - `git clone https://github.com/Eljon99/SafeSleep.git`
2. Caricare il dataset su MongoDB Compass per permettere l'accesso al database (assicurarsi che il nome che viene associato alla collection sia uguale a quello associato in `initDb.py`).
3. Se stai usando un IDE diverso da PyCharm installa e attiva il virtual environment di Python:
 - `python -m venv venv`
4. Installa le dipendenze per il backend:
 - `pip install -r requirements.txt`
5. Installa le dipendenze per il frontend e builda il progetto se stai usando un IDE diverso da PyCharm:
 - `cd ProgettoDB2/main/safe-sleep`
 - `npm install`
 - `npm run build`
6. Se stai usando PyCharm crea due configurazioni nuove, una per React (command: start) e una per Flask (Environment Variables: `FLASK_APP=app.py`).
7. Effettuare il run del progetto avviando le due configurazioni.

Operazioni CRUD

Le operazioni CRUD sono state implementate su entrambe le collezioni, la CREATE e READ sono state usate per entrambe, mentre, DELETE è stata usata per la collezione 'persona' e UPDATE per la collezione 'registro_persona'.

Di seguito sono mostrate le operazioni di CREATE per entrambe le collezioni:

Create Persona:

```
#CREAZIONE DI UNA PERSONA
@api/persona  ⬆ Eljon Hida +1 *
@app.route(rule: '/api/persona', methods=['POST'])
def add_persona():
    try:
        # Ottiene i dati della persona dal corpo della richiesta
        dataPersona = request.json

        required_fields = ['Gender', 'Age', 'Occupation', 'Physical Activity Level', 'BMI Category']
        for field in required_fields:
            if field not in dataPersona or not dataPersona[field]:
                return jsonify({'error': f'Dato mancante: {field}'}), 400

        # Ottiene il prossimo Person ID autoincrementale
        next_person_id = get_next_sequence_value("person_id")

        # Aggiunge Person ID ai dati della persona
        dataPersona['Person ID'] = next_person_id

        # Inserisce i dati della persona nella collezione
        result = collectionP.insert_one(dataPersona)
        new_person = collectionP.find_one(filter: {'_id': result.inserted_id}, *args: {'_id': 0})

        return jsonify(new_person), 201
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```


Create Registro Persona:

```
#CREAZIONE DI UN DIARIO
@api/diario  luicons01 *
@app.route(rule: '/api/diario', methods=['POST'])
def add_diario():
    try:
        # Ottiene i dati del diario dal corpo della richiesta
        dataDiario = request.json

        required_fields = ['Person ID', 'Sleep Duration', 'Quality of Sleep', 'Stress Level', '']
        for field in required_fields:
            if field not in dataDiario or not dataDiario[field]:
                return jsonify({'error': f'Dato mancante: {field}'}), 400

        # Verifica se il Person ID esiste nella collezione persona
        try:
            person_id = int(dataDiario['Person ID'])
        except ValueError:
            return jsonify({'error': 'Person ID non valido'}), 400

        person_exists = collectionP.find_one({'Person ID': person_id})
        if not person_exists:
            return jsonify({'error': 'Person ID non valido'}), 400

        # Inserisce i dati del diario nella collezione
        result = collectionD.insert_one(dataDiario)
        new_diario = collectionD.find_one(filter: {'_id': result.inserted_id}, *args: {'_id': 0})
```

In queste due immagini vengono mostrate le operazioni di create che permettono di creare un'istanza di persona e una di un registro che si riferisce a quella persona.

Non è per forza necessario creare un registro dopo aver creato una persona.

Inoltre vi è un controllo alla creazione di un nuovo registro che si assicura che la persona a cui verrà associato il registro sia già presente nella collezione 'persona' (attraverso il Person ID).

Get Persona:

```
#LETTURA DELLE PERSONE
@api/persona  luicons01 *
@app.route(rule: '/api/persona', methods=['GET'])
def get_persone():
    try:
        # Esegue una query per ottenere tutti i documenti, escludendo il campo _id e ordinando per Person ID
        records = list(collectionP.find(*args: {}, {'_id': 0}).sort(key_or_list: '_id', pymongo.ASCENDING))

        # Restituisce i record come risposta JSON
        return jsonify(records), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Get Registro Persona:

```
#LETTURA DEI DIARI
@api/diario  Eljon99 +1 *
@app.route(rule: '/api/diario', methods=['GET'])
def get_diario():
    try:
        # Esegue una query per ottenere tutti i documenti, escludendo il campo _id e ordinando per Person ID
        records = list(collectionD.find(*args: {}, {'_id': 0}).sort(key_or_list: 'Person ID', pymongo.ASCENDING))

        return jsonify(records), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Nelle immagini sono mostrate le operazioni di get per le due collezioni. Permettono di avere la collezione di tutte le persone e i registri associati ad ognuna di esse.

Update Registro Persona:

```
#MODIFICA DI UN DIARIO
~/api/diario/{person_id}  luicons01 *
@app.route(rule: '/api/diario/<int:person_id>', methods=['PUT'])
def update_diario(person_id):
    try:
        updated_data = request.json
        print("Dati ricevuti per l'aggiornamento:", updated_data) # Log dei dati ricevuti

        # Verifica se il Person ID esiste nella collezione diario
        existing_diario = collectionD.find_one({'Person ID': person_id})
        if not existing_diario:
            return jsonify({'error': 'Person ID non trovato'}), 404

        # Aggiorna i dati del diario nella collezione
        result = collectionD.update_one(
            filter: {'Person ID': person_id},
            update: {'$set': updated_data}
        )

        updated_diario = collectionD.find_one(filter: {'Person ID': person_id}, *args: {'_id': 0})
        return jsonify(updated_diario), 200
```

Questa immagine mostra l'update di un registro persona che permette di modificarne tutti i campi tranne il Person ID.

Delete Persona:

```
#ELIMINAZIONE DI UNA PERSONA
% /api/persona/{person_id}  ➤ Eljon Hida *
@app.route(rule: '/api/persona/<int:person_id>', methods=['DELETE'])
def delete_persona(person_id):
    try:
        # Cerca e elimina la persona con il Person ID specificato
        result_persona = collectionP.delete_one({"Person ID": person_id})

        if result_persona.deleted_count == 1:
            # Se la persona è stata eliminata con successo, elimina anche il diario/registro associato
            result_diario = collectionD.delete_many({"Person ID": person_id})

            return jsonify({
                'message': 'Persona e diario/registro associato eliminati con successo',
                'deleted_diaries_count': result_diario.deleted_count
            }), 200
        else:
            # Se non è stata trovata nessuna persona con il Person ID specificato
            return jsonify({'message': 'Persona non trovata'}), 404
    except Exception as e:
        # Gestisce eventuali errori e restituisce una risposta di errore
        return jsonify({'error': str(e)}), 500
```

L'immagine mostra l'operazione di delete di una persona. Questa operazione non elimina solo la persona selezionata ma anche il registro associato ad essa nel caso ve ne fosse uno.

Quality of Sleep - Age Correlation:

```
🔗 /api/sleep-age-correlation  👤 luicons01 *  
@app.route(rule: '/api/sleep-age-correlation', methods=['GET'])  
def get_sleep_age_correlation():  
    try:  
        pipeline = [  
            {  
                "$lookup": {  
                    "from": "diario_persona",  
                    "localField": "Person ID",  
                    "foreignField": "Person ID",  
                    "as": "diary_info"  
                }  
            },  
            {  
                "$unwind": "$diary_info"  
            },  
            {  
                "$lookup": {  
                    "from": "persona",  
                    "localField": "Person ID",  
                    "foreignField": "Person ID",  
                    "as": "person_info"  
                }  
            },  
        ],
```

```

    {
        "$unwind": "$person_info"
    },
    {
        "$project": {
            "_id": 0,
            "Person ID": 1,
            "Age": "$person_info.Age", # Aggiungi 'Age' dalla collezione 'persona'
            "Quality of Sleep": "$diary_info.Quality of Sleep"
        }
    }
]
results = list(collectionP.aggregate(pipeline))
return jsonify(results), 200
except Exception as e:
    return jsonify({'error': str(e)}), 500

```

L'obiettivo principale di questa pipeline è unire i dati tra due collezioni, *persona* e *diario_persona*, per correlare alcune informazioni, in particolare l'età e la qualità del sonno, di ogni persona.

La pipeline unisce i dati provenienti da due collezioni MongoDB ('persona' e 'diario_persona') basate sul campo comune 'Person ID'.

Il processo include l'uso di '\$lookup' per combinare i dati delle due collezioni, seguito da '\$unwind' per "srotolare" gli array risultanti in documenti separati. Infine, la query seleziona e restituisce solo i campi 'Person ID', 'Age' e 'Quality of Sleep', offrendo una panoramica diretta della relazione tra l'età e la qualità del sonno per ogni utente.

Deduzioni:

La qualità del sonno sembra essere influenzata dall'età. Per le persone più giovani, come i 27-28enni, la qualità del sonno tende a variare tra 4 e 7. Con l'aumentare dell'età, si nota una tendenza verso una qualità del sonno più alta, soprattutto tra i 35 e i 40 anni, dove la qualità del sonno tende a stabilizzarsi attorno a valori alti (7 o 8). Tuttavia, verso la fine della lista, per le persone più anziane (sopra i 50 anni), alcuni individui mostrano una qualità del sonno superiore, arrivando a 9.

Physical Activity Level Distribution:


```
@app.route(rule: '/api/activity-level-distribution', methods=['GET'])
def get_activity_level_distribution():
    try:
        pipeline = [
            {
                "$bucket": {
                    "groupBy": "$Physical Activity Level",
                    "boundaries": [30, 46, 61, 76, 91], # Definisci i limiti dei range
                    "default": "Other", # Valori fuori dai range specificati
                    "output": {
                        "count": {"$sum": 1}
                    }
                }
            }
        ]
        results = list(collectionP.aggregate(pipeline))

        # Formatta i risultati per renderli compatibili con il frontend
        formatted_results = [
            {"name": f"{result['_id']} - {result['_id'] + 15}", "value": result["count"]}
            for result in results if result["_id"] != "Other"
        ]

        return jsonify(formatted_results), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

L'obiettivo di questa pipeline è quello di calcolare e restituire la distribuzione dei livelli di attività fisica (*minuti impegnati nell'allenamento quotidiano*) tra gli utenti.

Utilizza una pipeline di aggregazione MongoDB con l'operatore '\$bucket' per raggruppare i dati in base a intervalli definiti di livelli di attività fisica ('Physical Activity Level').



I documenti vengono classificati in gruppi con limiti specificati (30-45, 46-60, 61-75, 76-90, 91-120) e viene conteggiato il numero di utenti per ciascun intervallo.

I risultati vengono formattati per mostrare i range di attività e il conteggio degli utenti in ogni range, escludendo i dati che non rientrano nei limiti definiti.

Deduzioni:

La maggior parte degli utenti si trova nel range di attività fisica tra 30 e 45, con 150 utenti che rientrano in questo intervallo. I gruppi successivi hanno un numero decrescente di utenti. I dati mostrano che la maggior parte degli utenti ha un livello di attività fisica moderato o basso, con una diminuzione progressiva del numero di utenti nei range di attività fisica più elevati.

BMI Category - Stress Level Correlation:

```
@app.route(rule: '/api/bmi-stress-correlation', methods=['GET'])
def get_bmi_stress_correlation():
    try:
        pipeline = [
            {
                "$lookup": {
                    "from": "diario_persona",
                    "localField": "Person ID",
                    "foreignField": "Person ID",
                    "as": "diary_info"
                }
            },
            {
                "$unwind": "$diary_info"
            },
            {
                "$lookup": {
                    "from": "persona",
                    "localField": "Person ID",
                    "foreignField": "Person ID",
                    "as": "person_info"
                }
            },
        ],
```

```

    {
        "$unwind": "$person_info"
    },
    {
        "$group": {
            "_id": "$person_info.BMI Category",
            "Average Stress Level": {"$avg": "$diary_info.Stress Level"}
        }
    },
    {
        # Arrotonda il valore di Average Stress Level a due decimali
        "$project": {
            "_id": 0,
            "BMI Category": "$_id",
            "Average Stress Level": {
                "$round": ["$Average Stress Level", 2]
            }
        }
    }
]
results = list(collectionP.aggregate(pipeline))
return jsonify(results), 200
except Exception as e:
    return jsonify({'error': str(e)}), 500

```

La query analizza i dati per trovare la relazione tra la categoria BMI (Indice di Massa Corporea) e il livello di stress medio degli utenti.

Combina i dati dalla collezione 'diario_persona' con quelli della collezione 'persona' utilizzando il campo 'Person ID'.

Si parte con la trasformazione e separazione dei documenti uniti per facilitare il calcolo, dopodiché si raggruppano i dati per categoria di BMI e si calcola il livello medio di stress per ciascun gruppo.

Fatto ciò, si arrotonda il livello medio di stress a due decimali e si restituiscono i risultati con la categoria di BMI e il livello di stress medio.

Deduzioni:

I livelli medi di stress sono più alti nelle categorie "Obese" e "Overweight" rispetto alle categorie "Normal Weight" e "Normal".