MongoDB & FastAPI Examples

Ahmad Hamo

17-5-2025

FastAPI with MongoDB (1/2)

```python
from fastapi import FastAPI, HTTPException
from pymongo import MongoClient
from pydantic import BaseModel
from typing import Optional
from bson import ObjectId

app = FastAPI()

# MongoDB Connection
client = MongoClient("mongodb://localhost:27017/")

db = client["demo"]
movies = db["movies"]
```

Movie API

```
🗁 Movie API
├── GET /movies
├── GET /movies/{id}
├── POST /movies
│   └── Example Body (JSON)
├── POST /movies/bulk
│   └── Example Body (JSON Array)
├── PUT /movies/{id}
│   └── Example Body (Full Update)
├── PATCH /movies/{id}
│   └── Example Body (Partial Update)
└── DELETE /movies/{id}
```

FastAPI with MongoDB (2/2)

```python
# Pydantic Models
class Movie(BaseModel):
    title: str
    year: int
    director: str
    genres: list[str]
    rating: float

class UpdateMovie(BaseModel):
    title: Optional[str] = None
    year: Optional[int] = None
    director: Optional[str] = None
    genres: Optional[list[str]] = None
    rating: Optional[float] = None
```

GET Endpoints (Example 1)

 # Get All Movies (Paginated)

```python
@app.get("/movies")
def get_movies(skip: int=0, limit: int=5):
    try:
        # Fetching movies and converting ObjectId to string
        movie_list = [
            {**movie, "_id": str(movie["_id"])}
            for movie in movies.find().skip(skip).limit(limit)
        ]
        return {"movies": movie_list}
    except Exception as e:
        return {"error": str(e)}
```

Postman Request:

```
GET http://localhost:8000/movies?skip=0&limit=5
```

GET Endpoints (Example 2)

 # Get Movie by ID

```python
@app.get("/movies/{movie_id}")
def get_movie(movie_id: str):
    if not ObjectId.is_valid(movie_id):
        raise HTTPException(400, "Invalid ID format")

    movie = movies.find_one({"_id":ObjectId(movie_id)})
    if not movie:
        raise HTTPException(404, "Movie not found")
    movie["_id"] = str(movie["_id"])
    return movie
```

Postman Request:

```
GET
http://localhost:8000/movies/507f1f77bcf86cd799439011
```

POST Endpoints (Example 1)

 # Create New Movie

```
@app.post("/movies")
def create_movie(movie: Movie):
    movie_data = movie.dict()
    result = movies.insert_one(movie_data)
    return {"id": str(result.inserted_id)}
```

Postman Request:

```
POST http://localhost:8000/movies
Headers: Content-Type: application/json
Body:
{
    "title": "Inception",
    "year": 2010,
    "director": "Christopher Nolan",
    "genres": ["Action", "Sci-Fi"],
    "rating": 8.8
}
```

POST Endpoints (Example 2)

 # Bulk Insert Movies

```
@app.post("/movies/bulk")
def bulk_create_movies(movie_list: list[Movie]):
    movie_data = [movie.dict() for movie in movie_list]
    result = movies.insert_many(movie_data)
    return {"inserted_ids": [str(id) for id in result.inserted_ids]}
```

Postman Request:

```
POST http://localhost:8000/movies/bulk
Headers: Content-Type: application/json
Body:
[
    {
        "title": "The Dark Knight",
        "year": 2008,
        "director": "Christopher Nolan",
        "genres": ["Action", "Crime"],
        "rating": 9.0
    },
    {
       "title": "Interstellar",
       "year": 2014,
        "director": "Christopher Nolan",
        "genres": ["Adventure", "Sci-Fi"],
        "rating": 8.6
    }
]
```

PUT (Examples)

 # Full Update (PUT)

```
    @app.put("/movies/{movie_id}")
    def update_movie(movie_id: str, movie: Movie):
        if not ObjectId.is_valid(movie_id):
            raise HTTPException(400, "Invalid ID format")

        result = movies.update_one(
            {"_id": ObjectId(movie_id)},
            {"$set": movie.dict()}
        )

        if result.matched_count == 0:
            raise HTTPException(404,
                        "Movie not found")

        return {"message": "Movie updated"}
}
```

Postman Request:

```
PUT
http://localhost:8000/movies/507f1f77bcf86cd79
9439011
Headers: Content-Type: application/json
Body:
{
    "title": "Inception (Updated)",
    "year": 2010,
    "director": "Christopher Nolan",
    "genres": ["Action", "Sci-Fi",
"Thriller"],
    "rating": 9.1
```

PATCH Endpoint (Example)

#Partial Update (PATCH)

```
    @app.patch("/movies/{movie_id}")
    def partial_update_movie(movie_id: str, movie: UpdateMovie):
        if not ObjectId.is_valid(movie_id):
            raise HTTPException(400, "Invalid ID format")

        update_data = {k: v for k, v in movie.dict().items() if v is not None}

        result = movies.update_one(
            {"_id": ObjectId(movie_id)},
            {"$set": update_data}
        )

        if result.matched_count == 0:
            raise HTTPException(404, "Movie not found")

        return {"message": "Movie partially updated"}
```

Postman Request:

```
PATCH http://localhost:8000/movies/507f1f77bcf86cd799439011
Headers: Content-Type: application/json
Body:
{
    "rating": 9.3,
    "genres": ["Action", "Sci-Fi", "Thriller", "Mystery"]
}
```

DELETE Endpoints (Example 1)

 # Delete by ID

```
@app.delete("/movies/{movie_id}")
def delete_movie(movie_id: str):
    if not ObjectId.is_valid(movie_id):
        raise HTTPException(400, "Invalid ID format")

    result = movies.delete_one({"_id": ObjectId(movie_id)})

    if result.deleted_count == 0:
        raise HTTPException(404, "Movie not found")

    return {"message": "Movie deleted"}
```

Postman Request:

```
DELETE http://localhost:8000/movies/507f1f77bcf86cd799439011
```

DELETE Endpoints (Example 2)

 # Delete by Query (Multiple)

```
@app.delete("/movies")
def delete_movies_by_query(year: int):
    result = movies.delete_many({"year": year})
    return {"deleted_count":
                result.deleted_count}
```

Postman Request:

```
DELETE http://localhost:8000/movies?year=1999
```

Key Takeaways

1. **GET**: Retrieve data (single or paginated lists)
2. **POST**: Create new documents (single or bulk)
3. **PUT/PATCH**: Full vs partial updates
4. **DELETE**: Remove documents (single or by query)
5. **Always validate**:
   - ObjectID format
   - Existence of documents
   - Required fields in POST/PUT

more complex examples (aggregations ….)? later