

Lecture Slides: PyMongo and FastAPI with Movies Collection

Instructor: [Ahmad Hamo]

Date: [5-5-2025]

What is PyMongo?

- Official MongoDB driver for Python.
- Enables interaction with MongoDB databases.
- Supports CRUD operations, indexing, and aggregation.

Why PyMongo?

- Simple Python syntax.
- Full MongoDB feature support.

Installing PyMongo

```
pip install pymongo
```

For better performance (optional):

```
pip install pymongo[srv]
```

Connecting to MongoDB

```
from pymongo import MongoClient

# Connect to MongoDB (local or Atlas)
client =
MongoClient("mongodb://localhost:27017/")

# Access the "demo" database
db = client["demo"]

# Get the "movies" collection
movies = db["movies"]
```

Inserting a Movie

```
new_movie = {
    "title": "Inception",
    "year": 2010,
    "directors": ["Christopher Nolan"],
    "genres": ["Action", "Sci-Fi", "Thriller"],
    "imdb": {"rating": 8.8, "votes": 2000000}
}

# Insert one movie
movie_id =
movies.insert_one(new_movie).inserted_id

print(f"Inserted movie with ID: {movie_id}")
```

Querying Movies

```
# Find one movie
movie = movies.find_one({"title": "Inception"})
print(movie)

# Find all Sci-Fi movies
scifi_movies = movies.find({"genres": "Sci-Fi"})
for movie in scifi_movies.limit(5):
    print(movie["title"])

# Find highly-rated movies
top_movies = movies.find({"imdb.rating": {"$gt": 8.5}})
for movie in top_movies:
    print(f"{movie['title']} - {movie['imdb']['rating']}")
```

Updating Movies

```
# Update one movie
movies.update_one(
    {"title": "Inception"},
    {"$set": {"imdb.rating": 8.9}}
)

# Add a new genre
movies.update_one(
    {"title": "Inception"},
    {"$push": {"genres": "Mystery"}}
)
```

Deleting Movies

```
# Delete one movie
movies.delete_one({"title": "Inception"})

# Delete all movies before 1950
movies.delete_many({"year": {"$lt": 1950}})
```

FastAPI + PyMongo Movies API

```

from fastapi import FastAPI
from pymongo import MongoClient

app = FastAPI()

client = MongoClient("mongodb://localhost:27017/")
db = client["demo"]
movies = db["movies"]

@app.get("/movies")
def get_movies(genre: str = None, min_rating: float = 0):
    query = {}
    if genre:
        query["genres"] = genre
    if min_rating:
        query["imdb.rating"] = {"$gt": min_rating}

    result = movies.find(query, {"_id": 0}).limit(10)
    return {"movies": list(result)}

@app.post("/movies")
def add_movie(movie: dict):
    movies.insert_one(movie)
    return {"message": "Movie added successfully!"}

```

Examples API Requests

1. Get Sci-Fi movies with rating > 8:

GET /movies?genre=Sci-Fi&min_rating=8

2. Add a new movie:

POST /movies

Body: {

```

    "title": "Interstellar",
    "year": 2014,
    "directors": ["Christopher Nolan"],
    "genres": ["Sci-Fi", "Adventure"],
    "imdb": {"rating": 8.6, "votes": 1500000}
}
```

References:

- [Official PyMongo Tutorial](#)
- [FastAPI Documentation](#)

CORS and Middleware with FastAPI

What is CORS?

Cross-Origin Resource Sharing (CORS) is a security mechanism that restricts web applications from making requests to a different domain (origin) than the one that served the web page.

Why is CORS Needed?

- Browsers enforce the **same-origin policy**, blocking requests to different domains by default.
- If your **frontend** (e.g., `http://localhost:3000`) tries to call your **FastAPI backend** (`http://localhost:8000`), the browser blocks it unless CORS is properly configured.

CORS Errors You Might See

Access to fetch at 'http://localhost:8000/movies' from origin 'http://localhost:3000' has been blocked by CORS policy.

No 'Access-Control-Allow-Origin' header is present on the requested resource.

How CORS Works in FastAPI

FastAPI provides built-in support for CORS via `CORSMiddleware`.

Key CORS Headers

Header	Purpose
<code>Access-Control-Allow-Origin</code>	Specifies which domains can access the API (* for all)
<code>Access-Control-Allow-Methods</code>	Lists allowed HTTP methods (GET, POST, etc.)
<code>Access-Control-Allow-Headers</code>	Defines allowed request headers (Content-Type, Authorization, etc.)

Example: CORS Middleware

```
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

# CORS Setup
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Allows all origins (for development)
    allow_methods=["*"], # Allows all methods (GET, POST, etc.)
    allow_headers=["*"], # Allows all headers
)
```

What This Does:

- ✓ Allows requests from **any domain** ("*")
- ✓ Permits **all HTTP methods** (GET, POST, PUT, etc.)
- ✓ Accepts **all headers** (Content-Type, Authorization, etc.)

⚠ Warning:

- allow_origins=["*"] is **unsafe for production** (only use in development).

- In production, specify exact domains:

```
python allow_origins=["https://yourfrontend.com", "http://localhost:3000"]
```

What is Middleware in FastAPI?

Middleware is a layer that processes **requests before they reach your route handlers** and **responses before they go back to the client**.

Common Uses of Middleware

- **CORS Handling** (as shown above)
- **Authentication** (checking JWT tokens)
- **Logging** (tracking request/response data)
- **Rate Limiting** (preventing API abuse)

How Middleware Works

1. A request comes in.
2. Middleware processes it (e.g., checks CORS headers).
3. If allowed, the request reaches your route (`@app.get("/movies")`).
4. The response goes back through middleware before being sent to the client.

Example: Custom Logging Middleware

```
from fastapi import Request
import time

@app.middleware("http")
async def log_requests(request: Request, call_next):
    start_time = time.time()

    # Process the request
    response = await call_next(request)

    # Log request details
    process_time = time.time() - start_time
    print(f"Request: {request.method} {request.url} | Time: {process_time:.2f}s")

    return response
```

What This Does:

- Logs every request (GET /movies, POST /movies, etc.)
- Measures **response time**
- Useful for debugging and monitoring

Summary

Concept	Purpose	Example
CORS	Allows cross-origin requests	CORSMiddleware
Middleware	Intercepts requests/responses	Logging, Auth, CORS
allow_origins	Whitelists domains	["http://localhost:3000"]
allow_methods	Permits HTTP methods	["GET", "POST"]
allow_headers	Allows request headers	["Content-Type"]

Further Reading

- [FastAPI CORS Docs](#)
- [MDN CORS Guide](#)