

04_03_ASSI_mongodb

Descripción General

Este proyecto es una aplicación backend construida con FastAPI y MongoDB, que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre una colección de libros. Es ideal para aprender cómo conectar una API moderna en Python con una base de datos NoSQL como MongoDB y cómo organizar un proyecto con buenas prácticas.

Tecnologías Usadas

- FastAPI: Framework moderno y rápido para construir APIs en Python.
- MongoDB: Base de datos NoSQL orientada a documentos.
- Motor: Cliente async para conectarse a MongoDB desde Python.
- Pydantic: Validación de datos mediante modelos.
- Python-dotenv: Manejo de variables de entorno (como la URI de conexión).

Estructura del Proyecto

main.py

Contiene toda la lógica de la API:

Contiene toda la lógica de la API:

- Conexión y desconexión de MongoDB (lifespan).
- Endpoints:
 - * POST /create-book: Crear un libro.
 - * GET /list-books: Listar todos los libros.
 - * GET /get-book/{isbn}: Buscar un libro por ISBN.
 - * PUT /update-book/{isbn}: Actualizar un libro.
 - * DELETE /delete-book/{isbn}: Eliminar un libro.

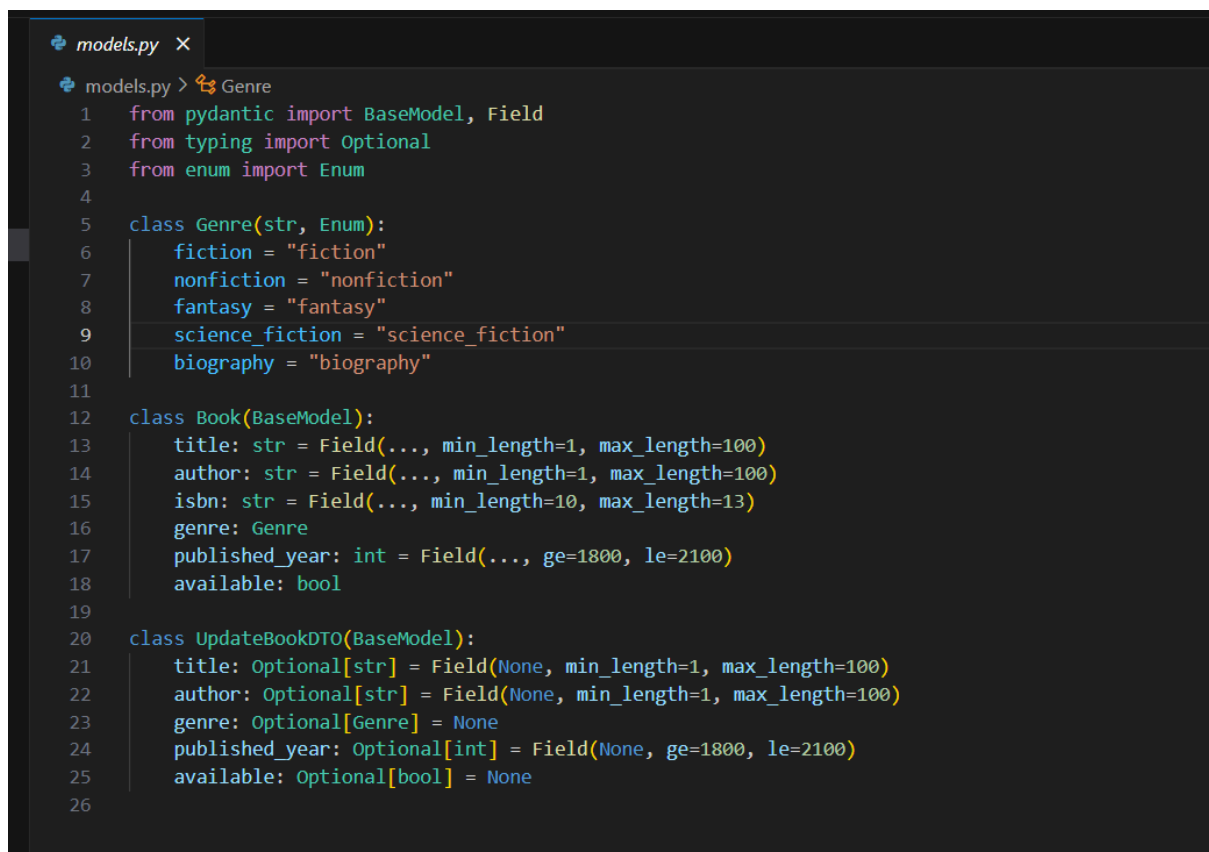
```
main.py
main.py > ...
1  from fastapi import FastAPI, HTTPException
2  from models import Book, UpdateBookDTO
3  from motor.motor_asyncio import AsyncIOMotorClient
4  from contextlib import asynccontextmanager
5  from typing import List
6  import os
7  from dotenv import load_dotenv
8
9  load_dotenv()
10
11  mongo_url = os.getenv("MONGO_URL")
12
13  @asynccontextmanager
14  async def lifespan(app: FastAPI):
15      await startup_db_client(app)
16      yield
17      await shutdown_db_client(app)
18
19  async def startup_db_client(app):
20      app.mongodb_client = AsyncIOMotorClient(mongo_url)
21      app.mongodb = app.mongodb_client.get_database("library")
22
23      # Crear índices para búsquedas eficientes
24      #await app.mongodb["books"].create_index("title")
25      #await app.mongodb["books"].create_index("author")
26      #await app.mongodb["books"].create_index("isbn", unique=True)
27
```

models.py

Define el modelo Book con validación usando Pydantic. Incluye un Enum para el género literario.

Ejemplo de JSON de libro:

```
{  
  "title": "1984",  
  "author": "George Orwell",  
  "isbn": "1234567890",  
  "genre": "fiction",  
  "published_year": 1949,  
  "available": true  
}
```

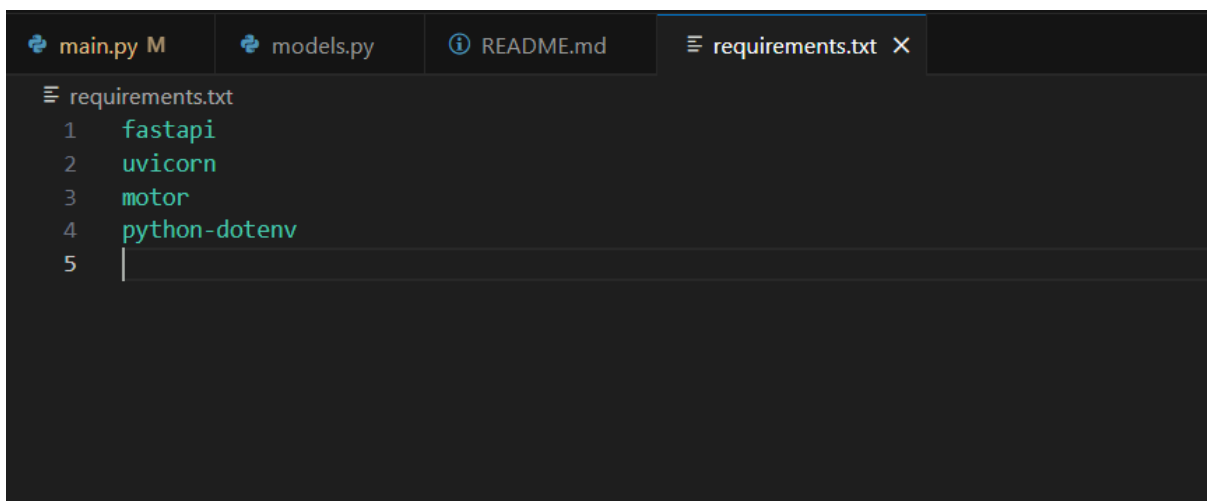


```
models.py ×  
models.py > Genre  
1  from pydantic import BaseModel, Field  
2  from typing import Optional  
3  from enum import Enum  
4  
5  class Genre(str, Enum):  
6      fiction = "fiction"  
7      nonfiction = "nonfiction"  
8      fantasy = "fantasy"  
9      science_fiction = "science_fiction"  
10     biography = "biography"  
11  
12  class Book(BaseModel):  
13      title: str = Field(..., min_length=1, max_length=100)  
14      author: str = Field(..., min_length=1, max_length=100)  
15      isbn: str = Field(..., min_length=10, max_length=13)  
16      genre: Genre  
17      published_year: int = Field(..., ge=1800, le=2100)  
18      available: bool  
19  
20  class UpdateBookDTO(BaseModel):  
21      title: Optional[str] = Field(None, min_length=1, max_length=100)  
22      author: Optional[str] = Field(None, min_length=1, max_length=100)  
23      genre: Optional[Genre] = None  
24      published_year: Optional[int] = Field(None, ge=1800, le=2100)  
25      available: Optional[bool] = None  
26
```

requirements.txt

Lista de dependencias necesarias para correr el proyecto:

- fastapi
- uvicorn
- motor
- pydantic
- python-dotenv

A screenshot of a code editor interface with a dark theme. At the top, there are four tabs: 'main.py M', 'models.py', 'README.md', and 'requirements.txt X'. The 'requirements.txt' tab is active. Below the tabs, the file 'requirements.txt' is open, showing a list of dependencies: 'fastapi', 'uvicorn', 'motor', and 'python-dotenv', each on a new line. Line numbers 1 through 5 are visible on the left side of the editor. The text is in a light green color on a dark background.

```
requirements.txt
1 fastapi
2 uvicorn
3 motor
4 python-dotenv
5 |
```

¿Cómo Probarlo?

1. Clona el repositorio.
2. Instala las dependencias con este comando: `pip install -r requirements.txt`
3. Crea un archivo `.env` con la URL de conexión:
`MONGO_URL=mongodb://localhost:27017`
4. Corre el servidor: `uvicorn main:app --reload`
5. Visita <http://localhost:8000/docs> para usar Swagger UI.

Interfaz de <http://localhost:8000/docs>

FastAPI 0.1.0 OAS 3.1
/openapi.json

default			^
GET	/	Read Root	v
POST	/api/v1/create-book	Create Book	v
GET	/api/v1/list-books	List Books	v
GET	/api/v1/get-book/{isbn}	Get Book	v
PUT	/api/v1/update-book/{isbn}	Update Book	v
DELETE	/api/v1/delete-book/{isbn}	Delete Book	v
GET	/api/v1/books/stats	Book Statistics	v

En el caso de querer usar el endpoint DELETE BOOK:

PUT	/api/v1/update-book/{isbn}	Update Book	v
DELETE	/api/v1/delete-book/{isbn}	Delete Book	^
Parameters			Try it out
Name	Description		
isbn * required			
string	isbn		
(path)			

Le damos a “Try it out” y le metemos el ISBN.

DELETE	/api/v1/delete-book/{isbn}	Delete Book	^
Parameters			Cancel
Name	Description		
isbn * required			
string	1234567899876		
(path)			
Execute			

Le damos a “Execute”:

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8080/api/v1/delete-book/1234567899876' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8080/api/v1/delete-book/1234567899876
```

Y la respuesta:

200

Successful Response

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "additionalProp1": {}
}
```

Conclusión

Este proyecto es una excelente base para entender cómo:

- Validar datos con Pydantic
- Usar MongoDB de forma asíncrona
- Construir un backend RESTful en Python con FastAPI
- Aplicar principios de diseño limpio en proyectos reales