

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 usuarios.

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:

- Conexión y desconexión de MongoDB (lifespan)
- Endpoints:
 - POST /create-user: Crear un usuario.
 - GET /read-all-users: Listar todos los usuarios.
 - GET /read-user/{email}: Buscar usuario por correo.
 - PUT /update-user/{email}: Actualizar usuario.
 - DELETE /delete-user/{email}: Eliminar usuario.

```
main.py M x  models.py  README.md  requirements.txt
main.py > ...
1  # File: main.py
2  # Project: FastAPI CRUD with MongoDB
3  # import fast api
4  from fastapi import FastAPI, HTTPException
5
6  from models import User # import the user model defined by us
7
8  # imports for the MongoDB database connection
9  from motor.motor_asyncio import AsyncIOMotorClient
10
11 # import for fast api lifespan
12 from contextlib import asynccontextmanager
13
14 from typing import List # Supports for type hints
15
16 from pydantic import BaseModel # Most widely used data validation library for python
17
18 # define a lifespan method for fastapi
19 @asynccontextmanager
20 async def lifespan(app: FastAPI):
21     # Start the database connection
22     await startup_db_client(app)
23     yield
```

models.py

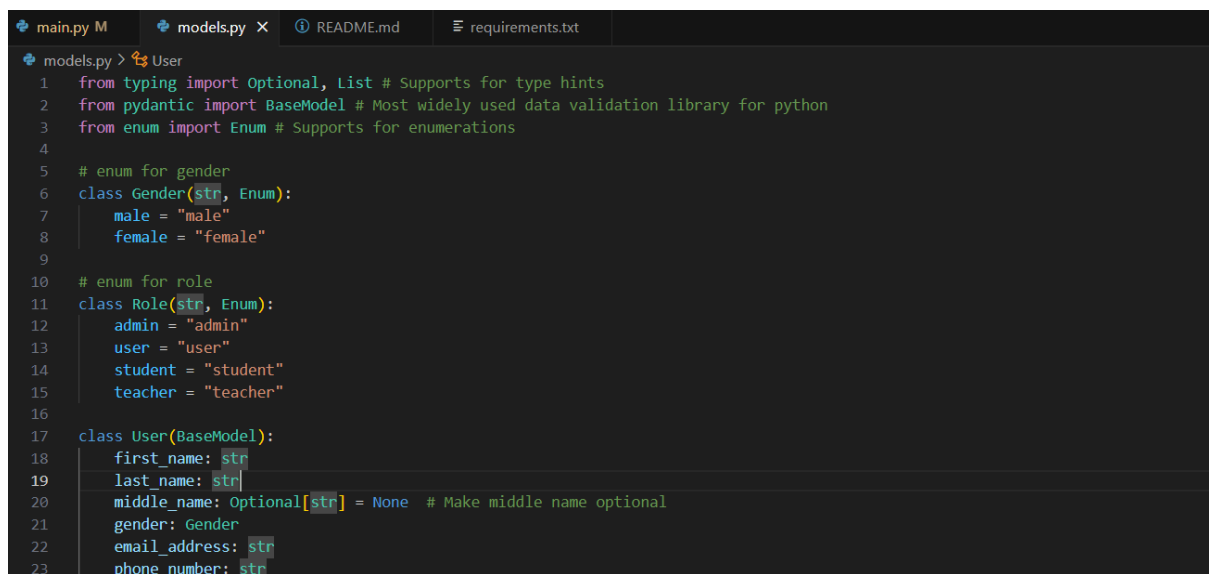
Define el modelo User con validación usando Pydantic. También incluye dos Enum:

- Gender: "male" o "female"
- Role: "admin", "user", "student", "teacher"

Esto permite garantizar que los valores enviados por el cliente sean válidos.

Ejemplo de JSON de usuario:

```
{  
  
  "first_name": "Juan",  
  
  "last_name": "Pérez",  
  
  "gender": "male",  
  
  "email_address": "juan@example.com",  
  
  "phone_number": "123456789",  
  
  "roles": ["student", "user"]  
}
```

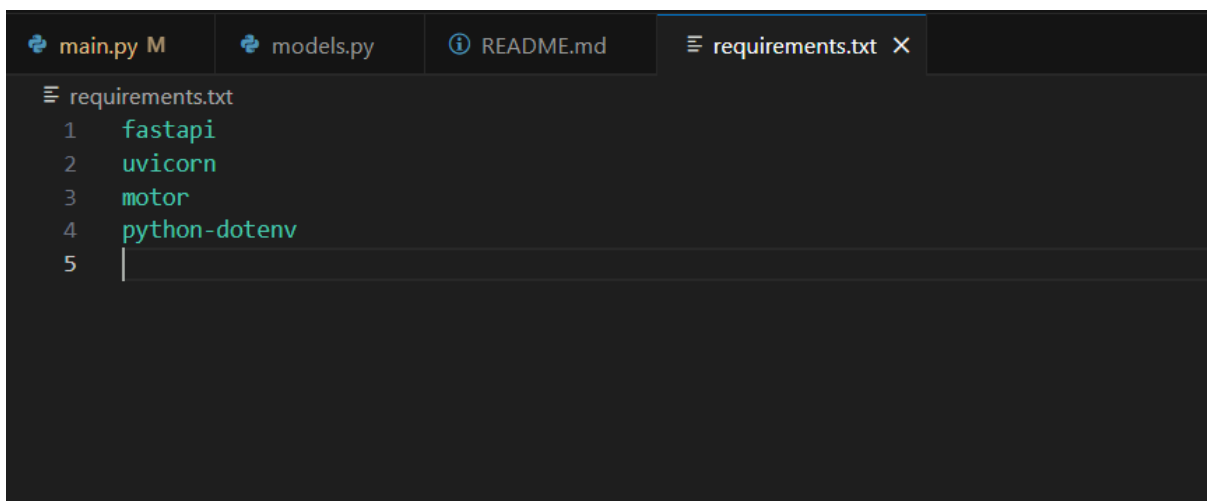


```
models.py > User  
1 from typing import Optional, List # Supports for type hints  
2 from pydantic import BaseModel # Most widely used data validation library for python  
3 from enum import Enum # Supports for enumerations  
4  
5 # enum for gender  
6 class Gender(str, Enum):  
7     male = "male"  
8     female = "female"  
9  
10 # enum for role  
11 class Role(str, Enum):  
12     admin = "admin"  
13     user = "user"  
14     student = "student"  
15     teacher = "teacher"  
16  
17 class User(BaseModel):  
18     first_name: str  
19     last_name: str  
20     middle_name: Optional[str] = None # Make middle name optional  
21     gender: Gender  
22     email_address: str  
23     phone_number: str
```

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.

```
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	⌵
POST	/api/v1/create-user	Insert User	⌵
GET	/api/v1/read-all-users	Read Users	⌵
GET	/api/v1/read-user/{email_address}	Read User By Email	⌵
PUT	/api/v1/update-user/{email_address}	Update User	⌵
DELETE	/api/v1/delete-user/{email_address}	Delete User By Email	⌵

En el caso de querer usar el endpoint UPDATE USER:

PUT

/api/v1/update-user/{email_address}

Update User

⌵

Parameters

Cancel

Name	Description
email_address * required string (path)	<input type="text" value="email_address"/>

Request body required

application/json

```
{  "other_names": [    "string"  ],  "age": 0}
```

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

PUT

/api/v1/update-user/{email_address}

Update User

⌵

Parameters

Cancel

Name	Description
email_address * required string (path)	<input type="text" value="sebas9999@gmail.com"/>

Request body required

application/json

```
{  "other_names": [    "string"  ],  "age": 0}
```

Execute

Le damos a “Execute”:

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/api/v1/update-user/sebas9999%40gmail.com' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "other_names": [
      "string"
    ],
    "age": 0
  }'
```

Request URL

http://127.0.0.1:8000/api/v1/update-user/sebas9999%40gmail.com

Y la respuesta:

200 Successful Response

Media type

application/json

Controls Accept header.

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