**This project is a REST API built using FAST API and MongoDB, which is used to demonstrate CRUD (create, read, update, delete) operations for managing the items.**

The project structure is as follows:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fastapi-mongodb-crud/

│

├── app/ # Main application package

│ ├── \_\_init\_\_.py # Package initializer

│ ├── crud.py # CRUD operation functions

│ ├── database.py # MongoDB connection settings

│ ├── main.py # App entry point

│ ├── models.py # Pydantic models

│ └── routes.py # API route definitions

│

├── venv/ # Python virtual environment

│ ├── Include/

│ ├── Lib/

│ └── Scripts/

│

├── .env # Environment variables (e.g., MongoDB URI)

├── .gitignore # Files to ignore in version control

├── requirements.txt # Python dependencies

└── README.md # Project overview and usage

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Setting up the python environment**

mkdir fastapi-mongodb-crud

cd fastapi-mongodb-crud

python -m venv venv

source venv/bin/activate

1. **Installing required packages:**

pip install fastapi uvicorn motor pymongo pydantic

1. **Creating a requirements.txt:**

fastapi

uvicorn

motor

pymongo

pydantic

1. **Create the file structure**

mkdir app

touch app/main.py app/database.py app/models.py app/crud.py app/routes.py

1. **Connecting to MongoDB (database.py)**

import motor.motor\_asyncio

import os

from dotenv import load\_dotenv

load\_dotenv()

MONGO\_URL = os.getenv("MONGO\_URL")

print("Connecting to MongoDB at:", MONGO\_URL)

client = motor.motor\_asyncio.AsyncIOMotorClient(MONGO\_URL)

db = client["mydatabase"]

collection = db["users"]

1. **Defining pydantic models (moels.py)**

from pydantic import BaseModel, EmailStr

class User(BaseModel):

name: str

email: EmailStr

1. **Create CRUD logic:**

from app.database import collection

async def create\_user(user):

print(" Received user:", user) # Debug print

try:

result = await collection.insert\_one(user)

print("Inserted user with ID:", result.inserted\_id)

return {"message": "User created", "id": str(result.inserted\_id)}

except Exception as e:

print(" Error inserting user:", e)

raise e

async def get\_users():

try:

print(" Fetching all users...")

users\_cursor = collection.find()

users = await users\_cursor.to\_list(length=100)

for user in users:

user["\_id"] = str(user["\_id"])

print(" Users fetched:", users)

return users

except Exception as e:

import traceback

traceback.print\_exc()

raise e

async def get\_user\_by\_email(email):

try:

user = await collection.find\_one({"email": email})

if user:

user["\_id"] = str(user["\_id"])

return user

except Exception as e:

import traceback

traceback.print\_exc()

raise e

async def update\_user(email, new\_data):

try:

result = await collection.update\_one({"email": email}, {"$set": new\_data})

if result.modified\_count == 1:

return {"message": "User updated successfully"}

elif result.matched\_count == 1:

return {"message": "No new data to update

else:

return {"message": "User not found"}

except Exception as e:

import traceback

traceback.print\_exc()

raise e

async def delete\_user(email):

try:

result = await collection.delete\_one({"email": email})

if result.deleted\_count == 1:

return {"message": "User deleted successfully"}

else:

return {"message": "User not found"}

except Exception as e:

import traceback

traceback.print\_exc()

raise e

1. **Setup the routes (routes.py):**

from fastapi import APIRouter

from app.models import User

from app import crud

router = APIRouter()

@router.get("/")

async def root():

return {"message": "Welcome to the FastAPI + MongoDB CRUD app"}

@router.post("/user")

async def add\_user(user: User):

print("Received user:", user)

return await crud.create\_user(user.dict())

@router.get("/users")

async def list\_users():

return await crud.get\_users()

@router.get("/user/{email}")

async def get\_user(email: str):

return await crud.get\_user\_by\_email(email)

@router.put("/user/{email}")

async def update\_user(email: str, user: User):

return await crud.update\_user(email, user.dict())

@router.delete("/user/{email}")

async def delete\_user(email: str):

return await crud.delete\_user(email)

1. **Launching the app (main.py):**

from fastapi import FastAPI

from app.routes import router

app = FastAPI()

app.include\_router(router)

1. **Run the app using :**

uvicorn app.main:app –reload

1. **Visit:** [**http://localhost:8000/docs**](http://localhost:8000/docs) **to test the API via Swagger UI.**
2. **Example Test Cases for FastAPI MongoDB CRUD App:**

**🔹 1. Test Root Route**

**Request:**

sql

Copy code

GET /

**Expected Response:**

json

Copy code

{

"message": "Welcome to the FastAPI + MongoDB CRUD app"

}

**🔹 2. Create a User**

**Request:**

pgsql

Copy code

POST /user

Content-Type: application/json

**Request Body:**

json

Copy code

{

"name": "Tanisha",

"email": "tanisha@example.com"

}

**Expected Response:**

json

Copy code

{

"message": "User created",

"id": "60abc123456789..."

}

**🔹 3. Get All Users**

**Request:**

bash

Copy code

GET /users

**Expected Response:**

json

Copy code

[

{

"\_id": "60abc123456789...",

"name": "Tanisha",

"email": "tanisha@example.com"

}

]

**🔹 4. Get User by Email**

**Request:**

sql

Copy code

GET /user/tanisha@example.com

**Expected Response:**

json

Copy code

{

"\_id": "60abc123456789...",

"name": "Tanisha",

"email": "tanisha@example.com"

}

**🔹 5. Update Existing User**

**Request:**

pgsql

Copy code

PUT /user/tanisha@example.com

Content-Type: application/json

**Request Body:**

json

Copy code

{

"name": "Tanisha Savitha",

"email": "tanisha@example.com"

}

**Expected Response (if updated):**

json

Copy code

{

"message": "User updated successfully"

}

**OR (if same data was sent):**

json

Copy code

{

"message": "No new data to update"

}

**🔹 6. Delete User by Email**

**Request:**

sql

Copy code

DELETE /user/tanisha@example.com

**Expected Response (if user exists):**

json

Copy code

{

"message": "User deleted successfully"

}

**OR (if user doesn’t exist):**

json

Copy code

{

"message": "User not found"

}