# LLumo AI
# Python + MongoDB Assessment (FastAPI)

Name – Sahil Koshriya
Branch – Masters of Computer Application
Roll No. – 22223110

# Python + MongoDB Assessment (FastAPI)

Github Link :

https://github.com/SAHILKOSHRIYA/LLUMO_FastAPI_MongoDB_Assingment110.git

## Problem Statement

Managing employee data manually or through unstructured systems becomes inefficient when organizations grow. We need a system that allows storing and managing employee records in a database, with the ability to perform **CRUD operations**, **search**, and **aggregation** for insights like average salaries by department. Security is also important, so sensitive operations should be protected with **JWT authentication**.

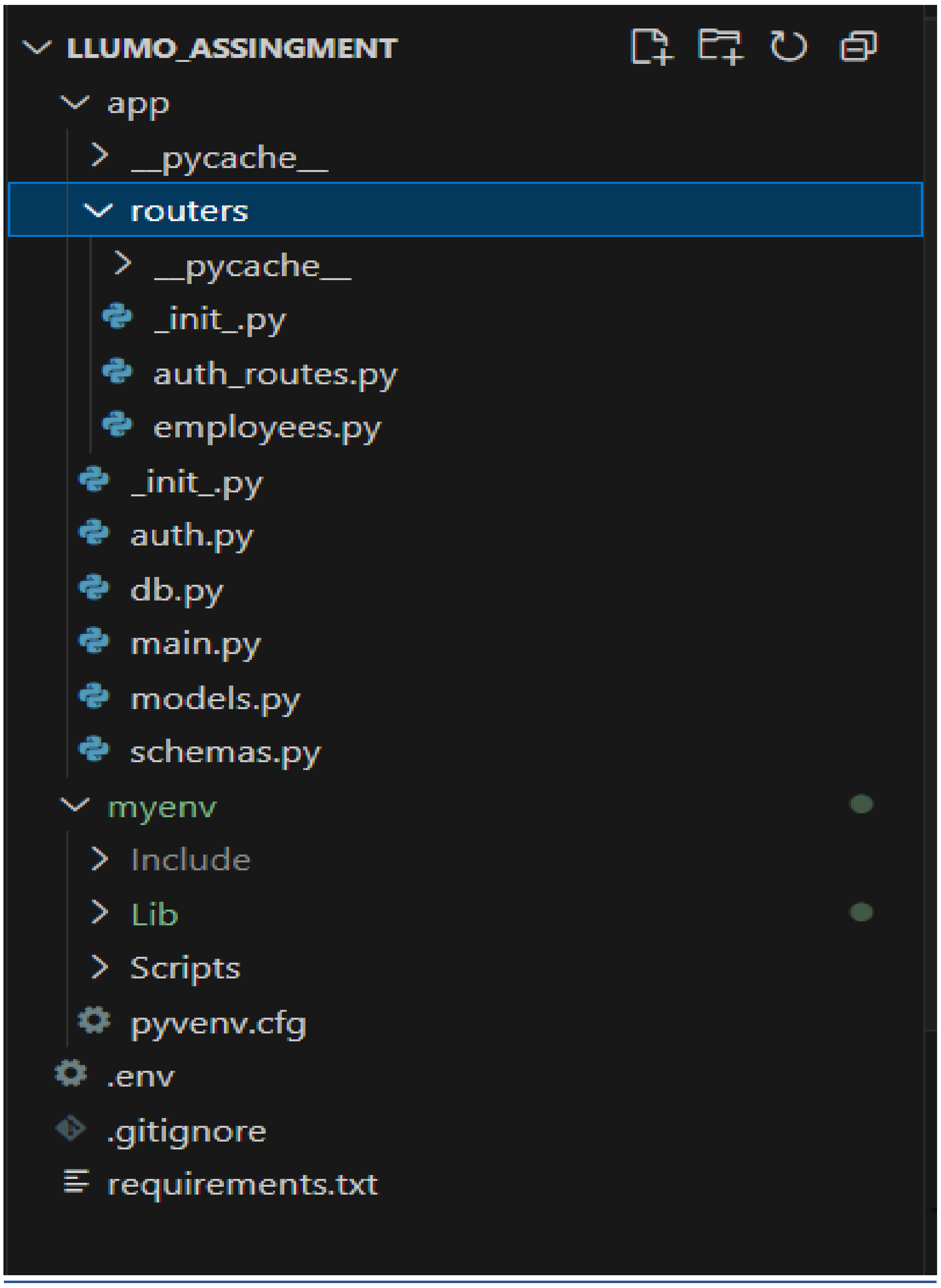
## Solution Statement

To solve this, I built a **RESTful API** using **FastAPI (or Django REST Framework)** with **MongoDB** as the backend database (`assessment_db`, collection: `employees`).

- The API supports **core CRUD operations**: create, read, update, and delete employee records, ensuring each `employee_id` is unique.
- Querying endpoints allow fetching employees by department (with sorting), searching by skills, and calculating average salary per department using MongoDB's aggregation pipeline.
- To secure sensitive routes like creating, updating, or deleting employees, I added **JWT authentication**. A valid token is required to access these protected endpoints.
- Additionally, enhancements like **pagination**, **MongoDB indexes on employee_id**, and **schema validation** improve performance and data consistency.

This setup provides a simple yet powerful system that can run locally, is easy to explain, and covers real-world requirements for an employee management service.

## Directory structure



```
LLUMO_ASSINGMENT
  app
    __pycache__
    routers
      __pycache__
      _init_.py
      auth_routes.py
      employees.py
    _init_.py
    auth.py
    db.py
    main.py
    models.py
    schemas.py
  myenv
    Include
    Lib
    Scripts
    pyvenv.cfg
  .env
  .gitignore
  requirements.txt
```

## 1. app/auth.py

```
from datetime import datetime, timedelta
from jose import JWTError, jwt
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from app.db import users_collection

SECRET_KEY =
"cfb362f8727388babb5e94ecddf2e3e1947b4725b09ade3fd3f845656158075f"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")

def hash_password(password: str) -> str:

    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)



def create_access_token(data: dict, expires_delta: timedelta | None = None):
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)



async def get_current_user(token: str = Depends(oauth2_scheme)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate token",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception
       user = await users_collection.find_one({"username": username})
    if user is None:
        raise credentials_exception
    return user
```

## 2. app/db.py

```
from motor.motor_asyncio import AsyncIOMotorClient
import os
from dotenv import load_dotenv

load_dotenv()

MONGO_URI = os.getenv("MONGO_URI")
DB_NAME = os.getenv("DB_NAME", "assessment_db")

client = AsyncIOMotorClient(MONGO_URI)
db = client[DB_NAME]


employees_collection = db["employees"]
users_collection = db["users"]
```

## 3. app\main.py

```
from fastapi import FastAPI
from app.routers import auth_routes, employees
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(title="Employee Assessment API")



app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(auth_routes.router)
app.include_router(employees.router)

@app.get("/")
async def root():
    return {"message": "Employee Assessment API."}
```

## 4. app\models.py

```python
from datetime import datetime, time
from typing import Dict, Any

def doc_to_employee(doc: Dict[str, Any]) -> Dict[str, Any]:
    """
     Converting MongoDB document to API-friendly dict.
    """
    if not doc:
        return None
    return {
        "employee_id": doc.get("employee_id"),
        "name": doc.get("name"),
        "department": doc.get("department"),
        "salary": doc.get("salary"),
        "joining_date": doc.get("joining_date"),
        "skills": doc.get("skills", []),
    }

def date_to_datetime(d):
    """
    Converting a date (datetime.date) to a timezone-naive datetime at 
midnight for storage.
    """
    if d is None:
        return None
    if isinstance(d, datetime):
        return d
    return datetime.combine(d, time.min)
```

## 5.app\schemas.py

```python
from pydantic import BaseModel, Field, validator
from typing import List, Optional
from datetime import date, datetime

class EmployeeBase(BaseModel):
    name: Optional[str] = None
    department: Optional[str] = None
    salary: Optional[float] = None
    joining_date: Optional[date] = None
    skills: Optional[List[str]] = None

    @validator("skills", pre=True, always=False)
    def empty_skills_to_list(cls, v):
        if v is None:
            return v
        return list(v)

class EmployeeCreate(EmployeeBase):
    employee_id: str = Field(..., min_length=1)
    name: str
    department: str
    salary: float
    joining_date: date
    skills: List[str]

class EmployeeUpdate(EmployeeBase):

    pass

class EmployeeOut(BaseModel):
    employee_id: str
    name: str
    department: str
    salary: float
    joining_date: datetime
    skills: List[str]



class UserCreate(BaseModel):
    username: str
    password: str

class UserLogin(BaseModel):
    username: str
    password: str


class Token(BaseModel):
    access_token: str
    token_type: str
```

## 6.routers\auth_routes.py

```
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from datetime import timedelta
from app.auth import hash_password, verify_password, create_access_token,
get_current_user
from app.db import users_collection
from app.schemas import UserCreate

router = APIRouter(prefix="/auth", tags=["auth"])

@router.post("/register")
async def register(user: UserCreate):

    existing = await users_collection.find_one({"username": user.username})
    if existing:
        raise HTTPException(status_code=400, detail="Username already
exists")

    hashed_pw = hash_password(user.password)
    new_user = {"username": user.username, "password": hashed_pw}
    await users_collection.insert_one(new_user)

    return {"message": "User registered successfully"}
@router.post("/login")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user = await users_collection.find_one({"username": form_data.username})
    if not user or not verify_password(form_data.password,
user["password"]):
        raise HTTPException(status_code=401, detail="Invalid username or
password")

    access_token = create_access_token(
        data={"sub": form_data.username},
expires_delta=timedelta(minutes=30)
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

## 7.routers\employees.py

```
from fastapi import Depends
from app.auth import get_current_user
from fastapi import APIRouter, HTTPException, status, Query
from typing import List, Optional
from app.db import employees_collection
from app.schemas import EmployeeCreate, EmployeeOut, EmployeeUpdate
from app.models import doc_to_employee, date_to_datetime
from pymongo import ReturnDocument
from bson.objectid import ObjectId
from datetime import datetime

router = APIRouter(prefix="/employees", tags=["employees"])



@router.post("", response_model=EmployeeOut,
status_code=status.HTTP_201_CREATED)
async def create_employee(emp: EmployeeCreate, current_user: dict =
Depends(get_current_user)):

    existing = await employees_collection.find_one({"employee_id":
emp.employee_id})
    if existing:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"employee_id '{emp.employee_id}' already exists."
        )

    doc = {
        "employee_id": emp.employee_id,
        "name": emp.name,
        "department": emp.department,
        "salary": emp.salary,

        "joining_date": date_to_datetime(emp.joining_date),
        "skills": emp.skills,
    }

    res = await employees_collection.insert_one(doc)
    created = await employees_collection.find_one({"_id": res.inserted_id})
    return doc_to_employee(created)


@router.get("/avg-salary")
async def avg_salary_by_department():
    pipeline = [
        {
            "$group": {
                "_id": "$department",
                "avg_salary": {"$avg": "$salary"}
            }
        },
        {
            "$project": {
                "_id": 0,
                "department": "$_id",
                "avg_salary": {"$round": ["$avg_salary", 2]}
            }
        },
        {
            "$sort": {"department": 1}
        }
    ]
    cursor = employees_collection.aggregate(pipeline)
    result = []
    async for doc in cursor:
        result.append(doc)
    return result

@router.get("/search", response_model=List[EmployeeOut])
async def search_by_skill(skill: str = Query(..., description="Skill to
search for")):

    cursor = employees_collection.find({"skills": {"$regex": f"^{skill}$",
"$options": "i"}})
    docs = await cursor.to_list(length=1000)
    return [doc_to_employee(d) for d in docs]


@router.get("/{employee_id}", response_model=EmployeeOut)
async def get_employee(employee_id: str, current_user: dict =
Depends(get_current_user)):
    doc = await employees_collection.find_one({"employee_id": employee_id})
    if not doc:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Employee not found")
    return doc_to_employee(doc)

@router.put("/{employee_id}", response_model=EmployeeOut)
async def update_employee(employee_id: str, emp: EmployeeUpdate,
current_user: dict = Depends(get_current_user)):

    update_fields = {}
    if emp.name is not None:
        update_fields["name"] = emp.name
    if emp.department is not None:
        update_fields["department"] = emp.department
    if emp.salary is not None:
        update_fields["salary"] = emp.salary
    if emp.joining_date is not None:
        update_fields["joining_date"] = date_to_datetime(emp.joining_date)
    if emp.skills is not None:
        update_fields["skills"] = emp.skills

    if not update_fields:

        doc = await employees_collection.find_one({"employee_id":
employee_id})
        if not doc:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Employee not found")
        return doc_to_employee(doc)

    updated = await employees_collection.find_one_and_update(
        {"employee_id": employee_id},
        {"$set": update_fields},
        return_document=ReturnDocument.AFTER
    )

    if not updated:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Employee not found")
    return doc_to_employee(updated)

@router.delete("/{employee_id}")
async def delete_employee(employee_id: str, current_user: dict =
Depends(get_current_user)):
    result = await employees_collection.delete_one({"employee_id":
employee_id})
    if result.deleted_count == 0:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Employee not found")
    return {"status": "success", "message": f"Employee {employee_id}
deleted."}

@router.get("", response_model=List[EmployeeOut])
async def list_employees(department: Optional[str] = Query(None,
description="Filter by department")):
    query = {}
    if department:
        query["department"] = department

    cursor = employees_collection.find(query).sort("joining_date", -1)
    docs = await cursor.to_list(length=1000)
    return [doc_to_employee(d) for d in docs]
```

**Section 1: Setup**

Used:
**FastAPI**
**Database: MongoDB (Local) ,MongoDB Compass**
**Database name: assessment_db**
**Collection name: employees**

**Sample Document Structure**

```
{
  "employee_id": "E123",
  "name": "John Doe",
  "department": "Engineering",
  "salary": 75000,
  "joining_date":  "2023-01-15",
  "skills": ["Python", "MongoDB", "APIs"]
}
```

**Section 2: Core CRUD APIs**

**1. Create Employee**

Endpoint: POST /employees

Task: Insert a new employee record.

Validation: Ensure employee_id is unique.

**2. Get Employee by ID**

Endpoint: GET /employees/{employee_id}

Task: Fetch employee details by employee_id.

Check: Return 404 if not found.

**3. Update Employee**

Endpoint: PUT /employees/{employee_id}

Task: Update employee details.

Requirement: Allow partial updates (update only provided fields).

**4. Delete Employee**

Endpoint: DELETE /employees/{employee_id}

Task: Delete employee record.

Response: Success/failure message.

**Section 3: Querying & Aggregation**

**5. List Employees by Department**

Endpoint: GET /employees?department=Engineering

Task: Return employees in a department, sorted by joining_date (newest first).

**6. Average Salary by Department**

Endpoint: GET /employees/avg-salary

**Output: Used MongoDB aggregation to compute average salary grouped by department.**

Expected Output:

```
[
  {“department”: “Engineering”, “avg_salary”: 80000},
  {“department”: “HR”, “avg_salary”: 60000}
]
```

**7. Search Employees by Skill**

Endpoint: GET /employees/search?skill=Python

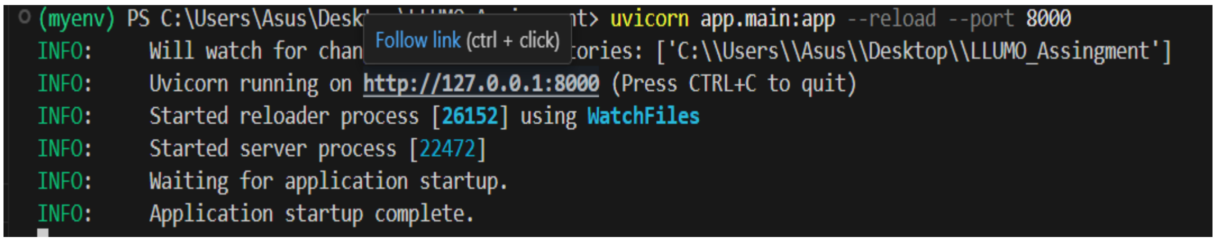
**Output: Returned employees who have the given skill in their skills array. Section 4:**

**Bonus Challenges (Optional)**

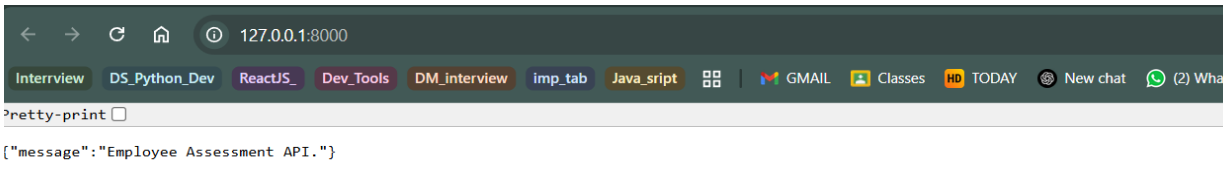
- Implement pagination for employee listing.
- Add MongoDB index on employee_id.
- Implement schema validation (e.g., using MongoDB JSON Schema).
- Add JWT authentication for protected routes.

**OUTPUT :Snapshots**

**1.Starting Server logs:**



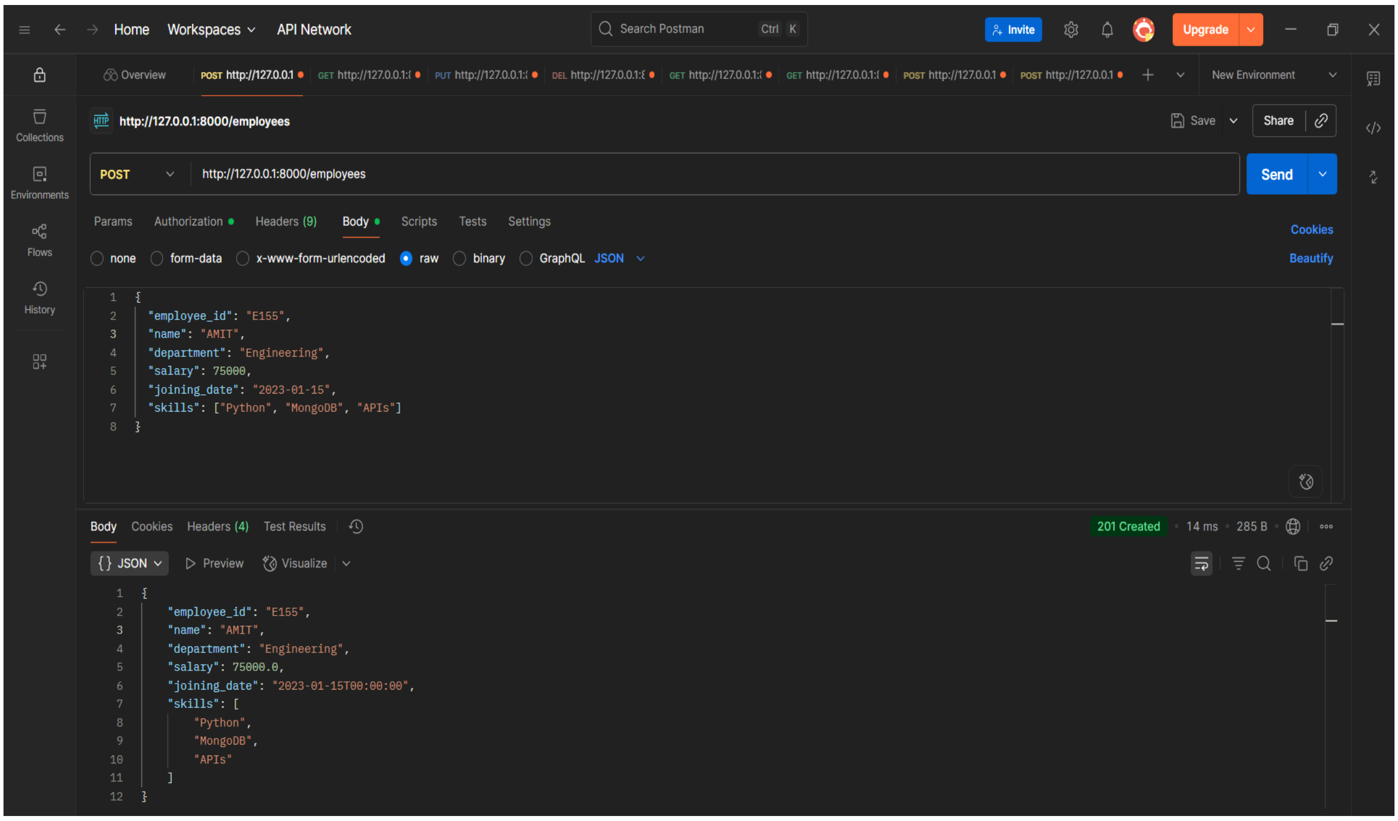
```
o (myenv) PS C:\Users\Asus\Desk[...]nt> uvicorn app.main:app --reload --port 8000
  INFO:     Will watch for chan[...]tories: ['C:\\Users\\Asus\\Desktop\\LLUMO_Assingment']
  INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
  INFO:     Started reloader process [26152] using WatchFiles
  INFO:     Started server process [22472]
  INFO:     Waiting for application startup.
  INFO:     Application startup complete.
```



```
{"message":"Employee Assessment API."}
```

## 2.Postman Testing – Output

POST http://127.0.0.1:8000/employees



http://127.0.0.1:8000/employees

POST http://127.0.0.1:8000/employees

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

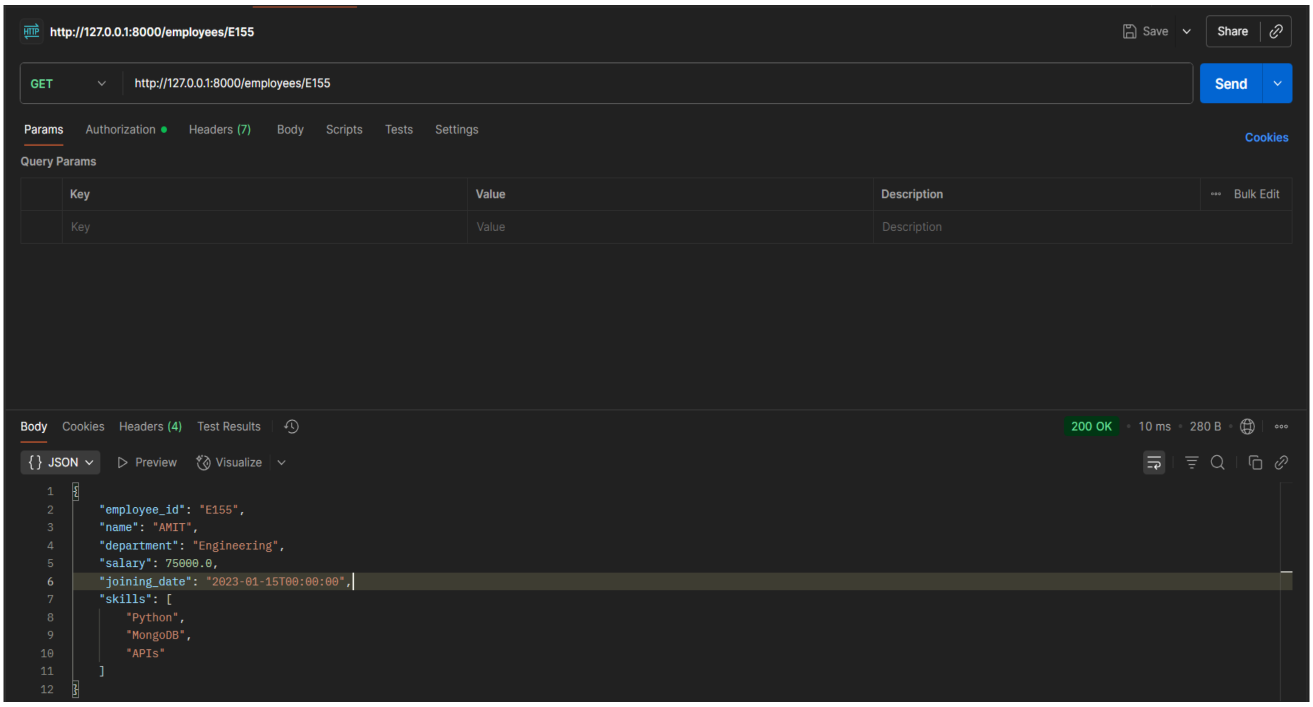
```
{
  "employee_id": "E155",
  "name": "AMIT",
  "department": "Engineering",
  "salary": 75000,
  "joining_date": "2023-01-15",
  "skills": ["Python", "MongoDB", "APIs"]
}
```

Body Cookies Headers (4) Test Results — 201 Created · 14 ms · 285 B

{} JSON

```
{
    "employee_id": "E155",
    "name": "AMIT",
    "department": "Engineering",
    "salary": 75000.0,
    "joining_date": "2023-01-15T00:00:00",
    "skills": [
        "Python",
        "MongoDB",
        "APIs"
    ]
}
```

GET http://127.0.0.1:8000/employees/E155



http://127.0.0.1:8000/employees/E155

GET http://127.0.0.1:8000/employees/E155

Query Params

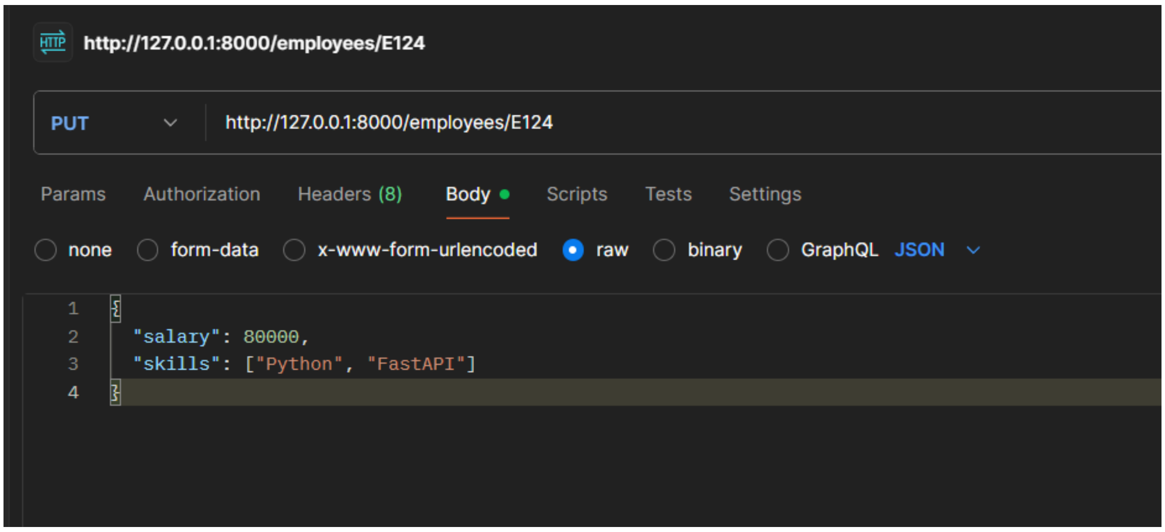
| Key | Value | Description |
|---|---|---|
| Key | Value | Description |

Body Cookies Headers (4) Test Results — 200 OK · 10 ms · 280 B

{} JSON

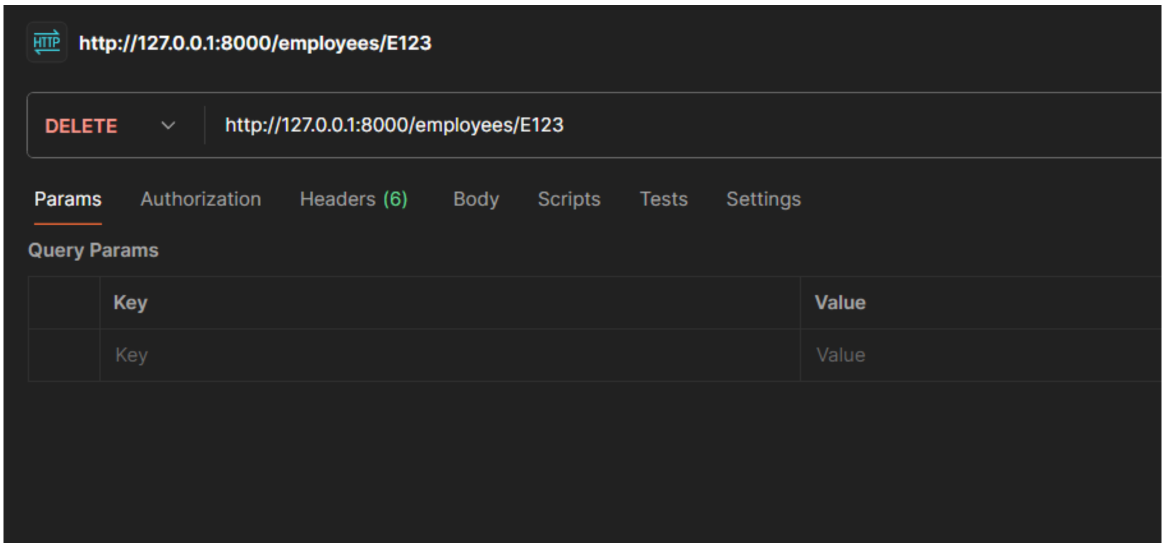
```
{
    "employee_id": "E155",
    "name": "AMIT",
    "department": "Engineering",
    "salary": 75000.0,
    "joining_date": "2023-01-15T00:00:00",
    "skills": [
        "Python",
        "MongoDB",
        "APIs"
    ]
}
```

PUT http://127.0.0.1:8000/employees/E124



```
{
  "salary": 80000,
  "skills": ["Python", "FastAPI"]
}
```

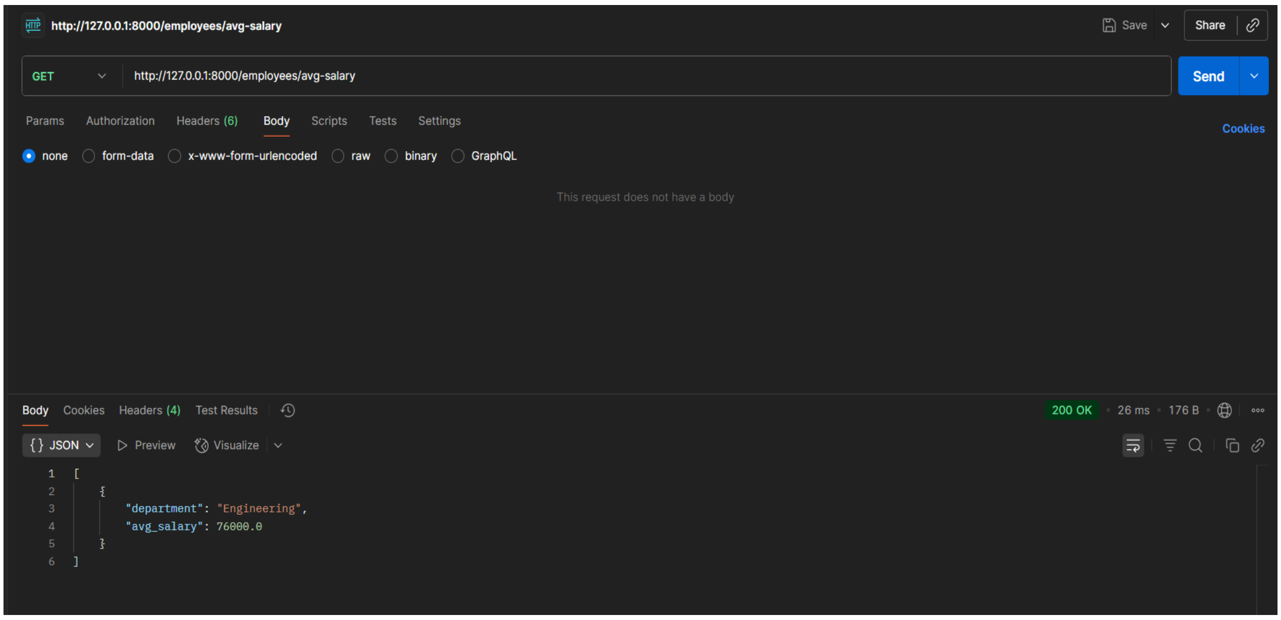
Delete http://127.0.0.1:8000/employees/E123


http://127.0.0.1:8000/employees/E123
DELETE
http://127.0.0.1:8000/employees/E123
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Query Params

| Key | Value |
|---|---|
| Key | Value |

GET :http://127.0.0.1:8000/employees/avg-salary

Aggregation:



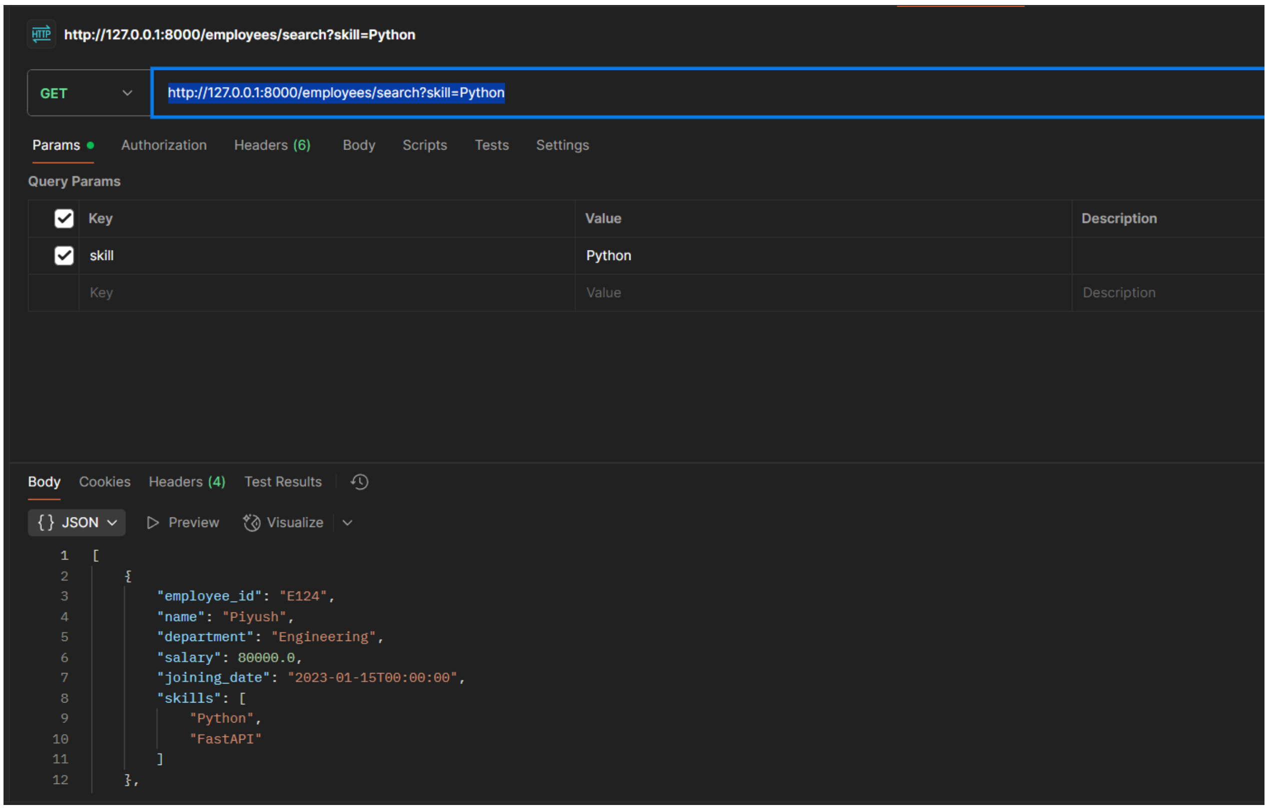
GET http://127.0.0.1:8000/employees/avg-salary

This request does not have a body

200 OK • 26 ms • 176 B

```
[
    {
        "department": "Engineering",
        "avg_salary": 76000.0
    }
]
```

GET http://127.0.0.1:8000/employees/search?skill=Python



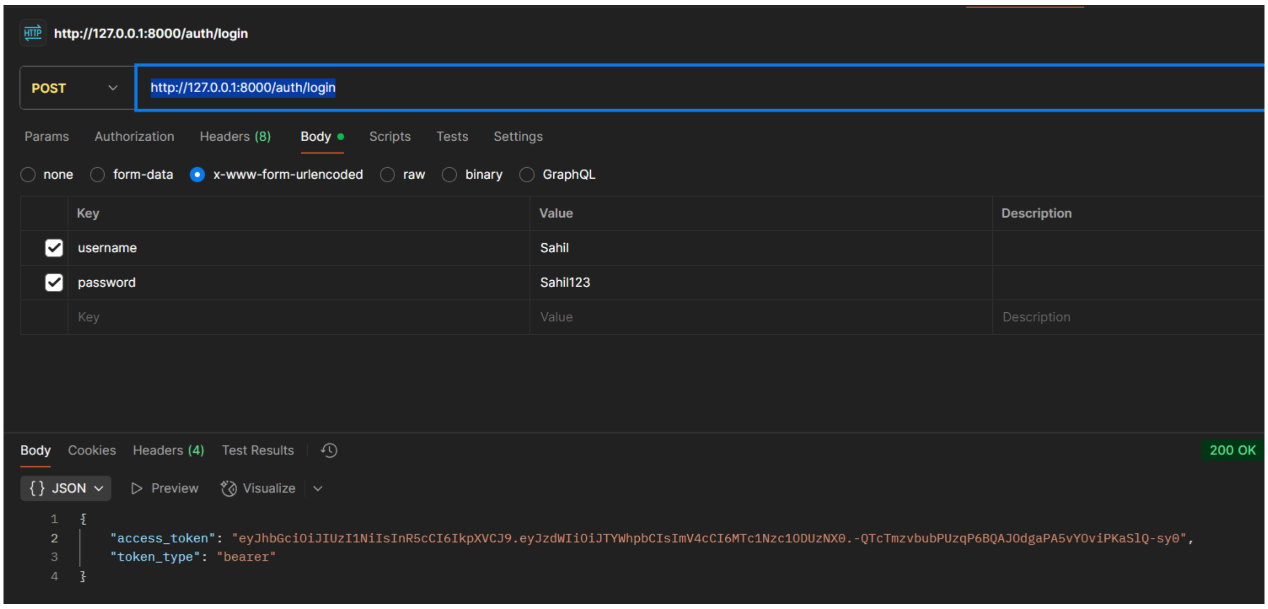
GET http://127.0.0.1:8000/employees/search?skill=Python

Query Params

| Key | Value | Description |
|---|---|---|
| ✓ skill | Python | |

```
[
    {
        "employee_id": "E124",
        "name": "Piyush",
        "department": "Engineering",
        "salary": 80000.0,
        "joining_date": "2023-01-15T00:00:00",
        "skills": [
            "Python",
            "FastAPI"
        ]
    },
```

**Post: http://127.0.0.1:8000/auth/login**


http://127.0.0.1:8000/auth/login

POST http://127.0.0.1:8000/auth/login

Body: x-www-form-urlencoded

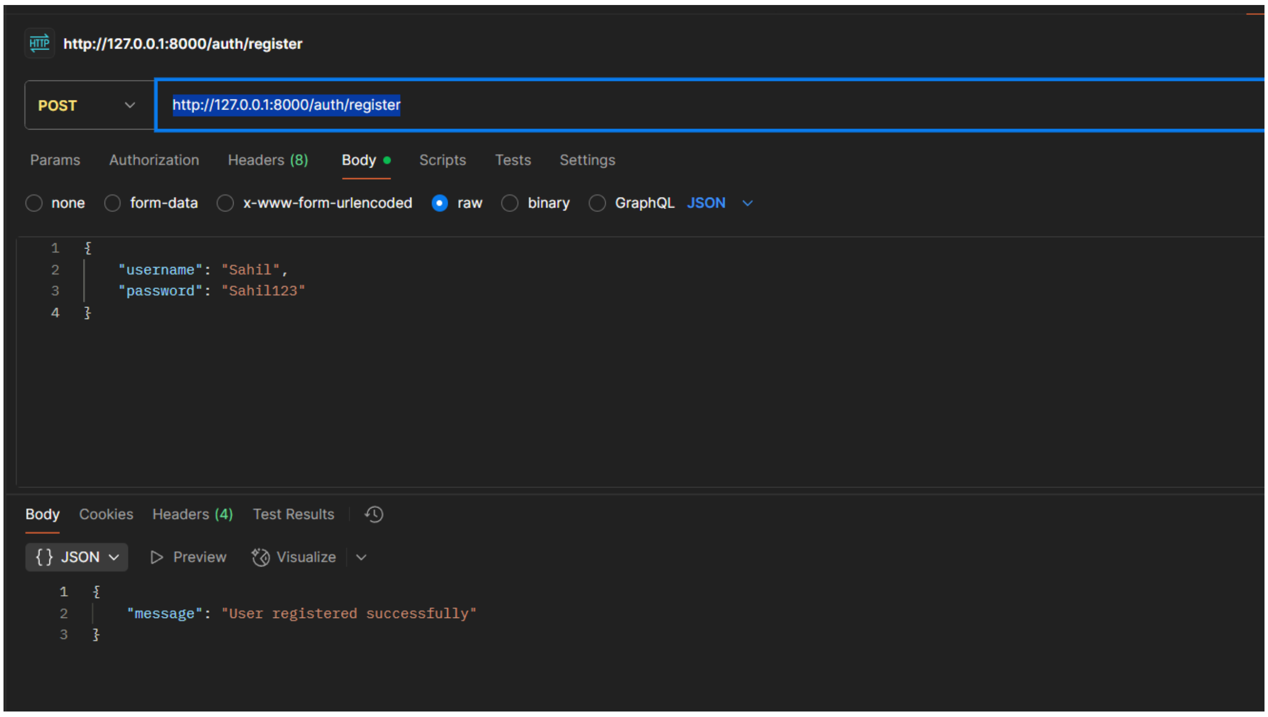
| Key | Value | Description |
|---|---|---|
| ✓ username | Sahil | |
| ✓ password | Sahil123 | |

200 OK

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJTYWhpbCIsImV4cCI6MTc1Nzc1ODUzNX0.-QTcTmzvbubPUzqP6BQAJOdgaPA5vYOviPKaSlQ-sy0",
    "token_type": "bearer"
}
```

**Post http://127.0.0.1:8000/auth/register**

**New user registration**


http://127.0.0.1:8000/auth/register

POST http://127.0.0.1:8000/auth/register

```
{
    "username": "Sahil",
    "password": "Sahil123"
}
```

```
{
    "message": "User registered successfully"
}
```

## 3. Version Control : GitHub

**Commands**

**1.git init**

**2.git remote add origin** **https://github.com/SAHILKOSHRIYA/LLUMO_FastAPI_MongoDB_Assingment110.git**

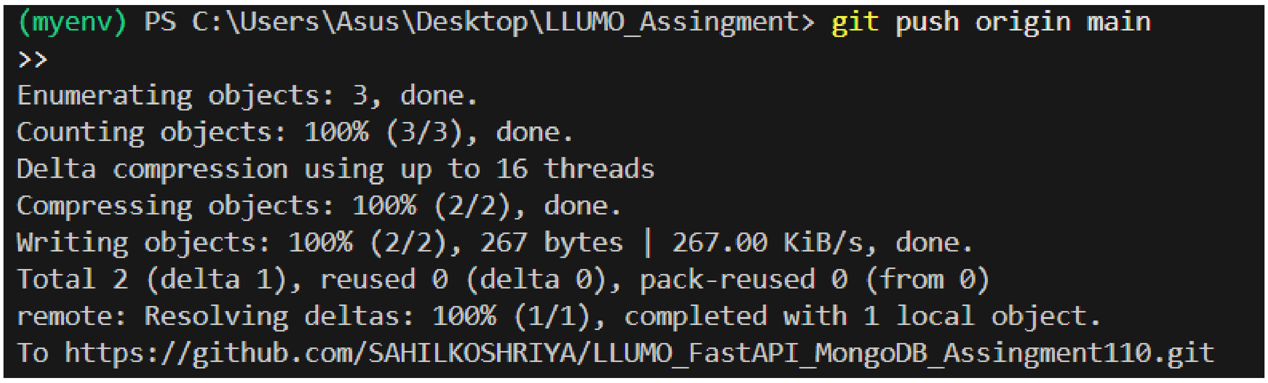
**3.git branch -M main**

**4. git add .**

**5. git commit -m "Initial commit - FastAPI MongoDB Employee CRUD with JWT"**

**6. git push -u origin main**

**Snapshot:**



```
(myenv) PS C:\Users\Asus\Desktop\LLUMO_Assingment> git push origin main
>>
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 267 bytes | 267.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/SAHILKOSHRIYA/LLUMO_FastAPI_MongoDB_Assingment110.git
```

## Conclusion

This project demonstrates how to build a secure and scalable REST API using **FastAPI** with **MongoDB** as the database. It covers the complete lifecycle of employee management through **CRUD operations**, advanced querying like **department-based filtering, skill search, and salary aggregation**, and ensures data integrity with unique employee IDs and validations.

To enhance real-world applicability, the solution also integrates **JWT authentication** for protected routes, enabling secure access control.

Overall, this implementation provides a solid foundation for developing modern backend systems that are fast, secure, and production-ready.

**THANK YOU!**