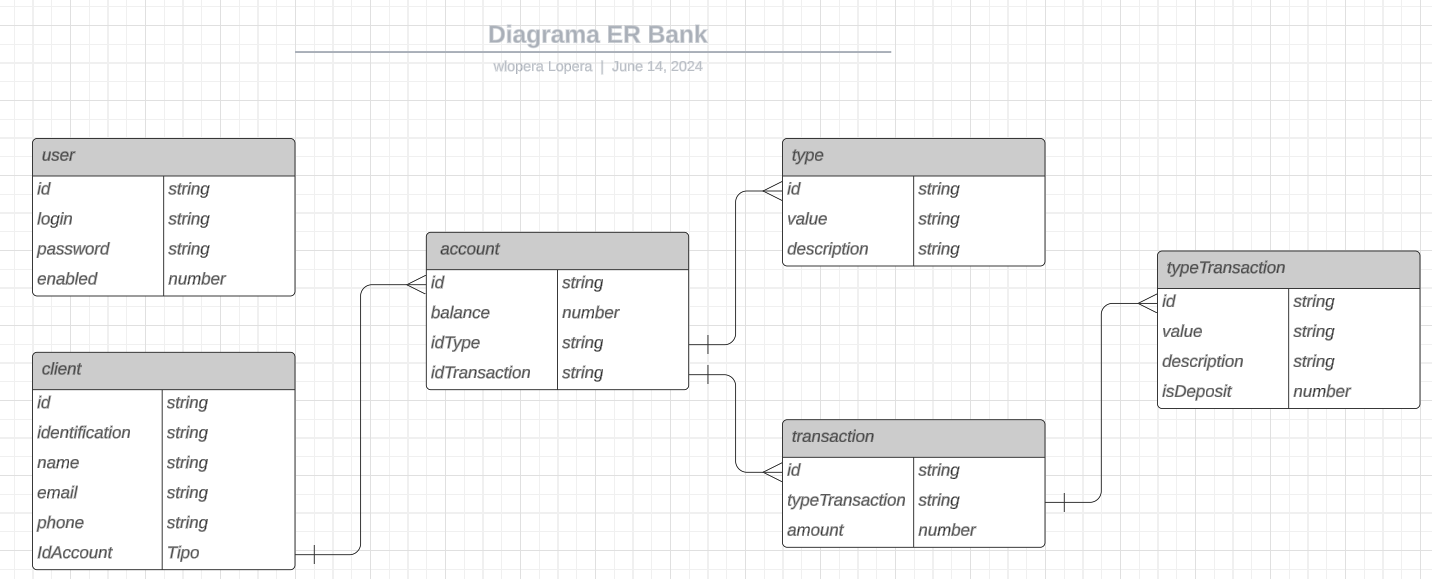
**PROYECTO BANCA PYTHON, REACT Y REACT NATIVE**

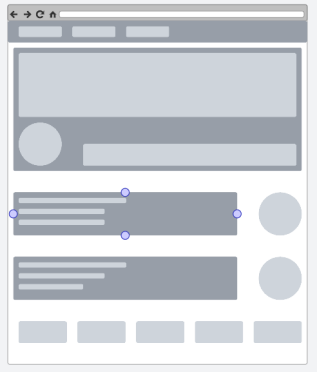
Proyecto de estudio de un Banco que permite conectarse como administrador y generar usuarios, cuentas y transacciones y como usuario permite conectarse, consultar cuentas y realizar transacciones (Pagos y transferencias bancarias).

* Python: APIs de servicios
  + - Usuarios
    - Clientes
    - Cuentas
    - Tipos
    - Transacciones
    - Tipo de transacciones

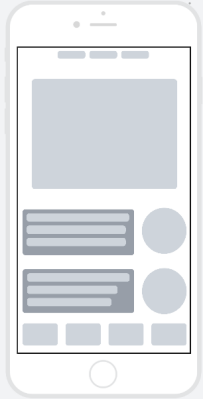


Nota: MondoDB

* React: Web del banco
  + - Admin: administrar usuarios y cuentas
    - Usuarios: Consulta y transacciones bancarias



* React Native: Consulta y transacciones bancarias solo para usuarios



**Python:**

* Desarrollo de API de usuarios que permiten ingresar a la banca. Inicialmente se va a permitir el acceso pero luego se debe generar token y validaciones de seguridad.

Librerías requeridas:

* + - FastApi (fastapi)
    - APIRouter (fastapi)
    - HTTPException (fastapi)
    - BaseModel (pydantic)
    - pymongo (MongoClient)
    - uvicorn Para levantar el servidor de python

pip install fastapi

<https://pypi.org/project/fastapi/>

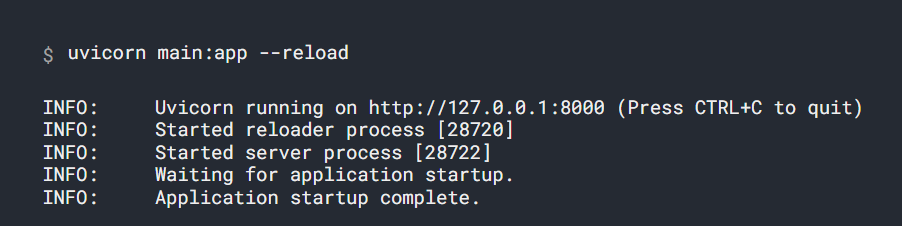
<https://fastapi.tiangolo.com/es/>

pip install "uvicorn[standard]"

<https://pypi.org/project/uvicorn/>

<https://pywombat.com/articles/introduccion-pydantic>

Para levantar servidor



pip install pymongo

<https://pypi.org/project/pymongo/>

Extensiones o plugin para VSCODE

* + - THUNDER CLINT Cliente para consumo de servicios API
    - Python
    - Python Extension Pack
* Probar Servidor de Python

from fastapi import FastAPI

app = FastAPI()

@app.get('/')

async def getUser():

    return {

        'id':"AQSWDE12345FRGTFG654646",

        'login':"wlopera",

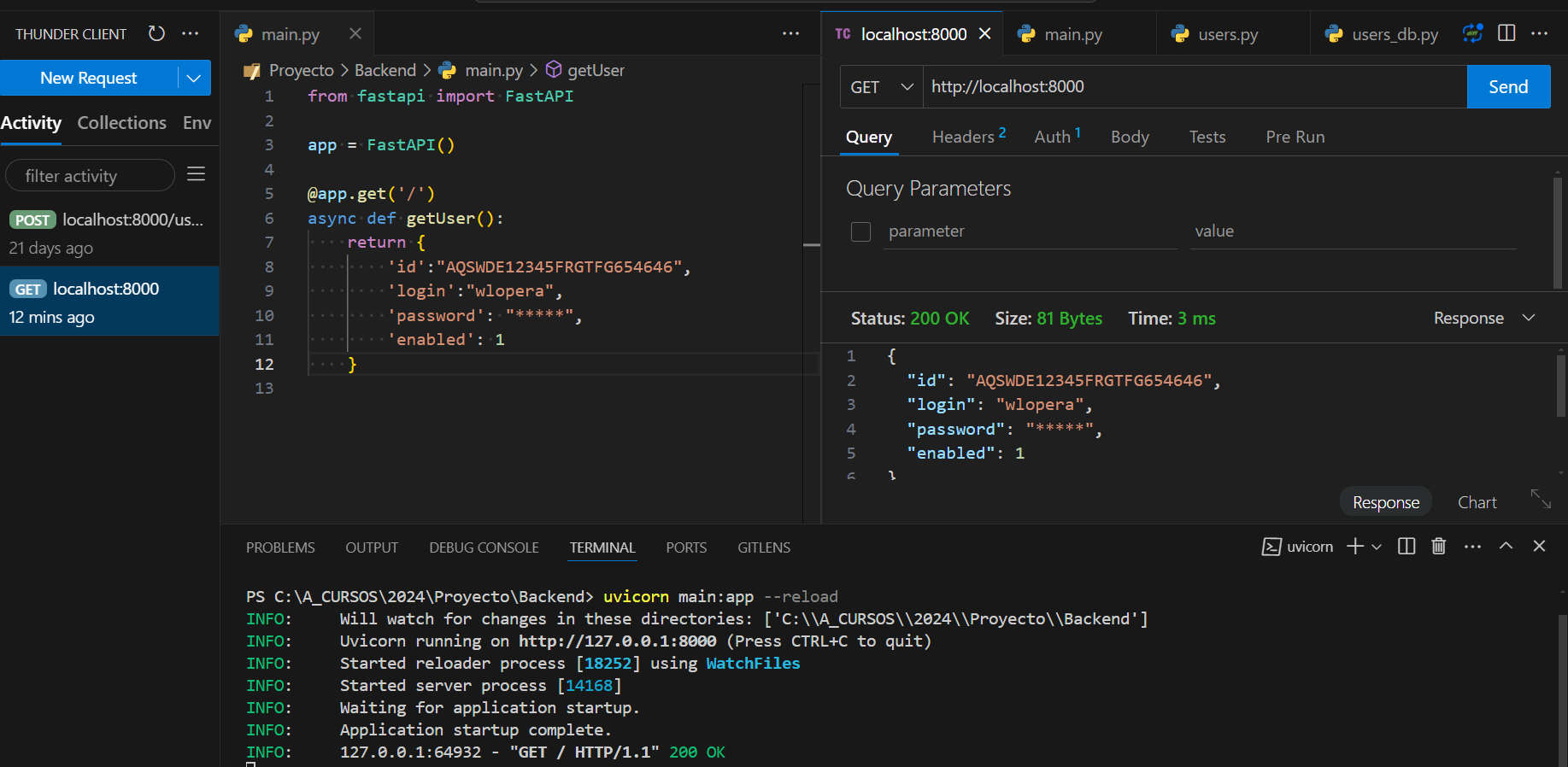
        'password': "\*\*\*\*\*",

        'enabled': 1

    }

Levantar y probar servicio

PS C:\A\_CURSOS\2024\Proyecto\Backend> uvicorn main:app --reload



Agregar route para manejo de varios APIS. Route API users

**users.py**

from fastapi import APIRouter

from pydantic import BaseModel

router = APIRouter(prefix="/users",

                   tags=["Usuarios"],

                   responses={404: {'message':"No Encontrado"}})

# Clase Usuario

class User(BaseModel):

    id: str

    login:str

    password:str

    enable: int

#Lista de Usuario

users\_list =[User(id="aqswe12345", login="lmessi",password="11111",enable=1),

             User(id="q1w2e3r4t5", login="cr7",password="22222",enable=1),

             User(id="f5g3h6j7hh", login="njunior",password="33333",enable=1),

             User(id="e3r4tf6ytf", login="lsuarez",password="44444",enable=0),

             User(id="98765fgdrs", login="jarango",password="55555",enable=1)]

# Consultar usuarios

@router.get("/")

async def users():

    return users\_list

**main.py**

from fastapi import FastAPI

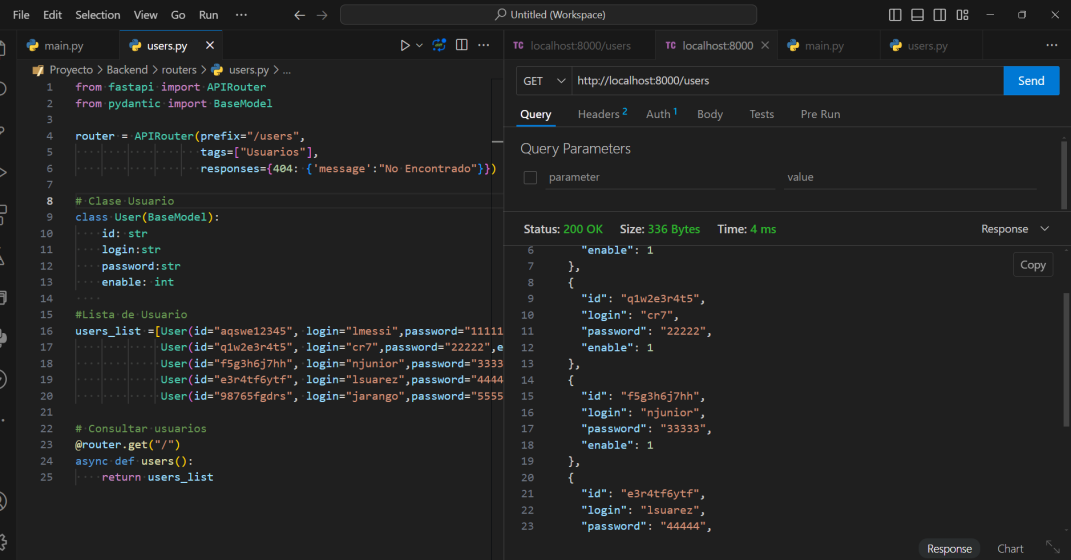
from routers import users

app = FastAPI()

# Routers

app.include\_router(users.router)

* Salida



**Uso de MongoDB**

* Instalar MongoDB y Herramientas de MongoDB:

<https://www.mongodb.com/docs/manual/installation>

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-windows>

* Agregar MongoDb al path
* Crear carpetas para db y log:

- C:\A\_CURSOS\2024\Proyecto\Backend\data\db

- C:\A\_CURSOS\2024\Proyecto\Backend\data\log

* Crear archivo para levantar MongoDB:

- C:\A\_CURSOS\2024\Proyecto\Backend\data\mongod.cfg

storage:

dbPath: C:\A\_CURSOS\2024\Proyecto\Backend\data\db

# Descomentar si quiero crear archivo de log:

# systemLog:

# destination: file

# path: C:\A\_CURSOS\2024\Proyecto\Backend\data\logs\mongod.log

# logAppend: true

net:

  bindIp: 127.0.0.1

  port: 27017

* Levantar MongoDB

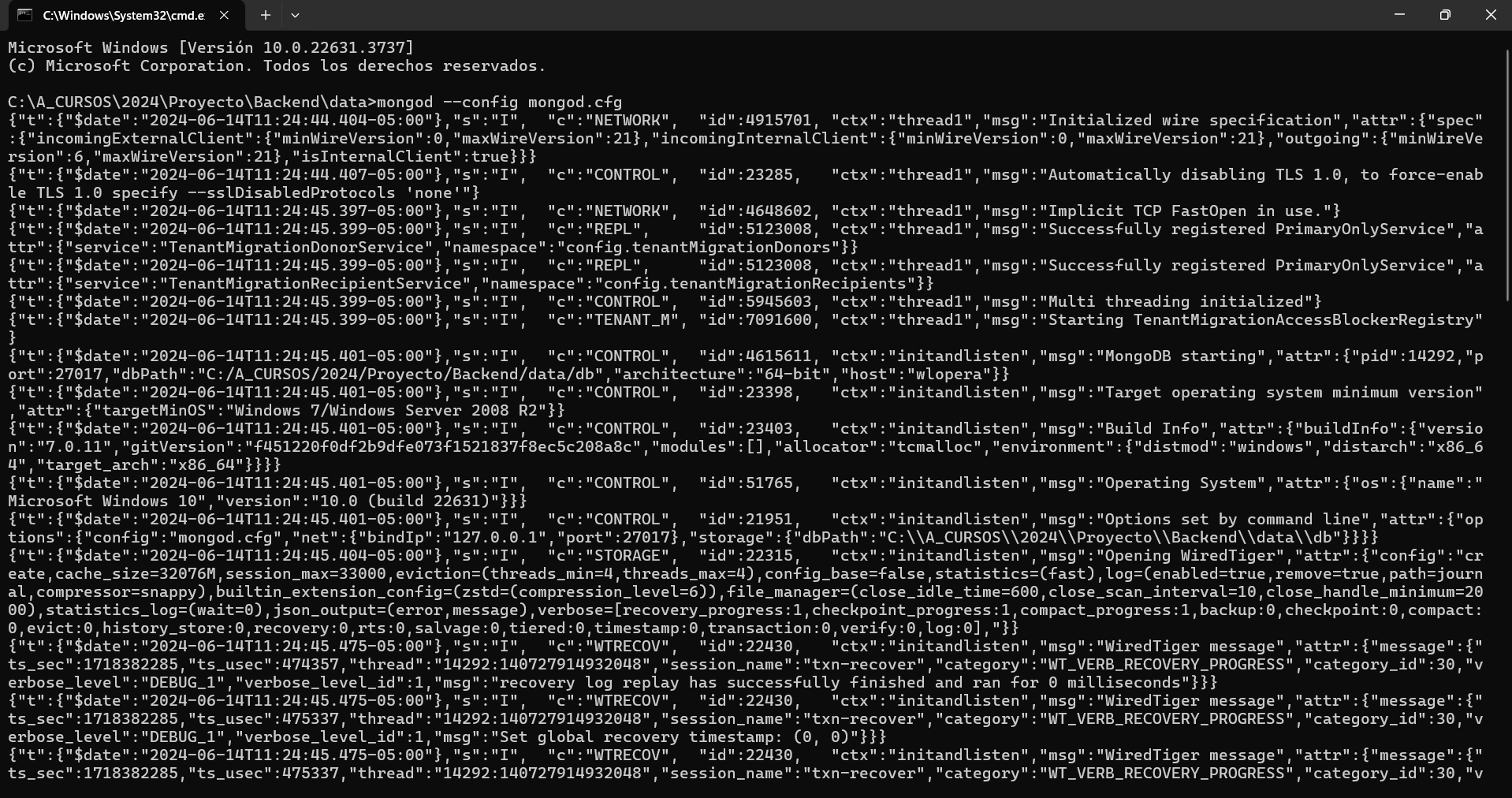
C:\A\_CURSOS\2024\Proyecto\Backend\data> **mongod --config mongod.cfg**

ó

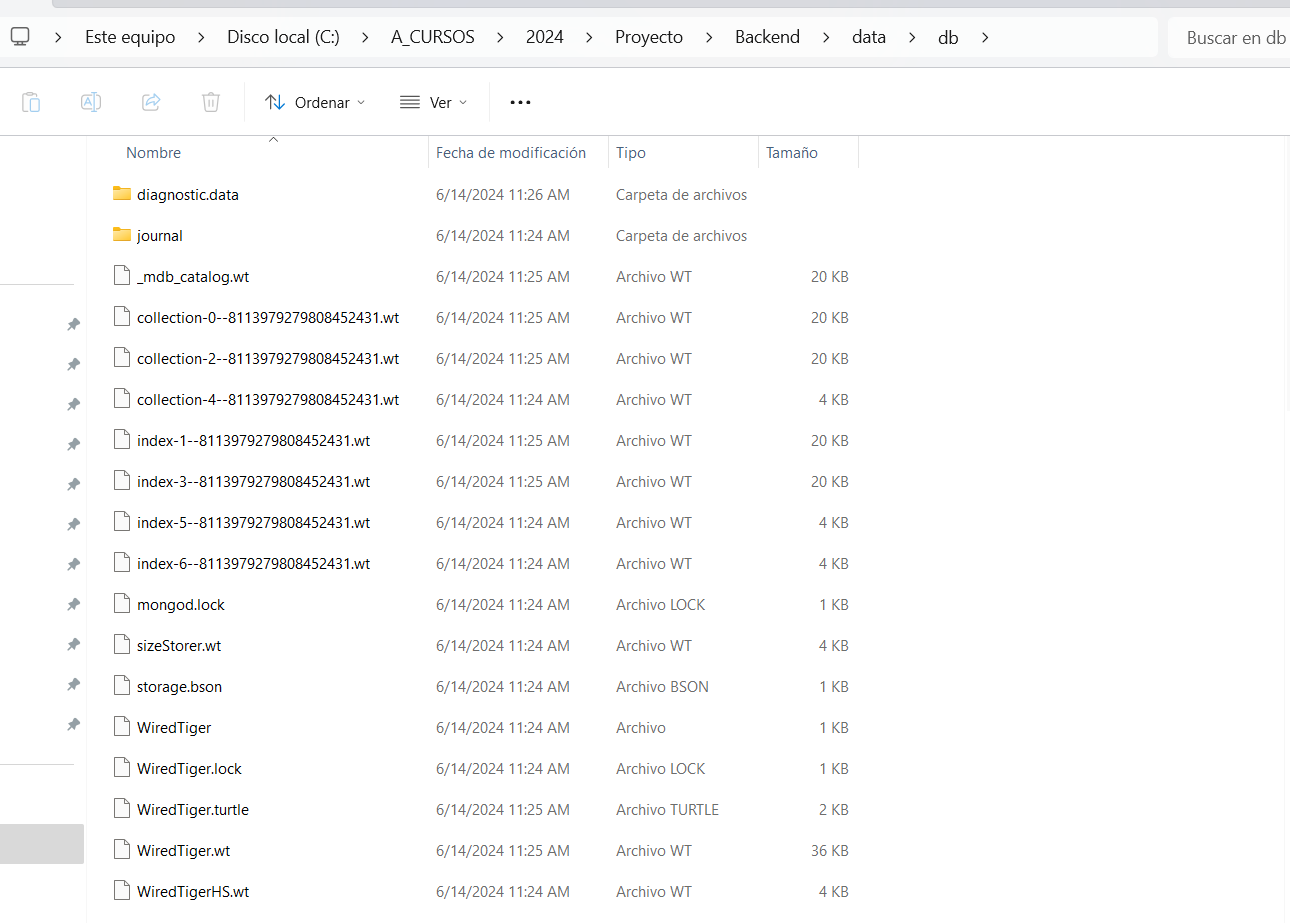
C:\A\_CURSOS\2024\Proyecto\Backend\data>

**mongod --dbpath "C:\A\_CURSOS\2024\Proyecto\Backend\data\db"**

Conexión: mongod://localhost [url:port]



Se crear el archivo de mongo en la ruta definida en el archivo config mongod.cfg



* Instalar Plugin MongoDB en Vscode

Conectar a MongoDB desde el plugin:

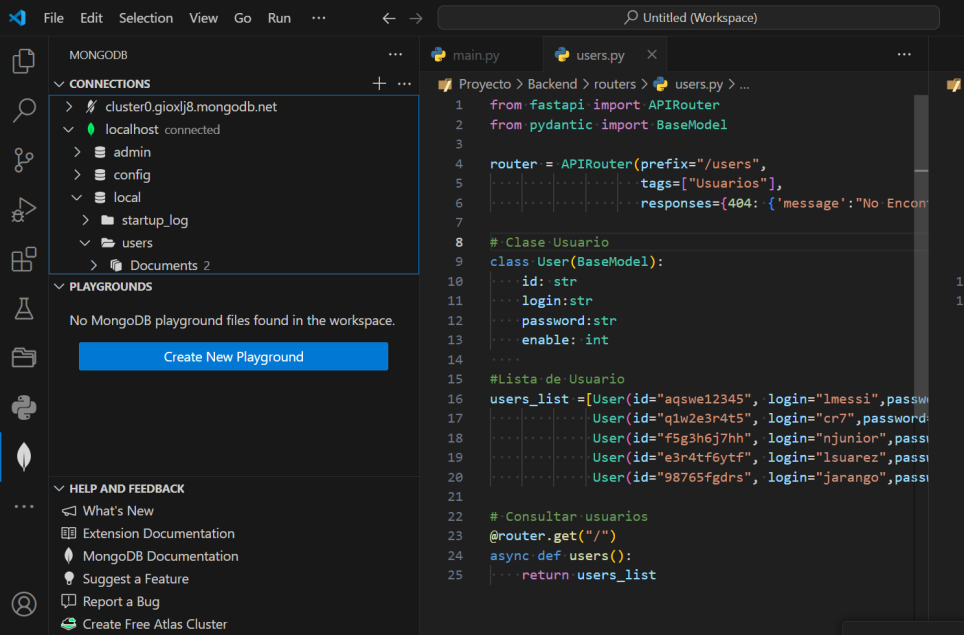
> mongodb://localhost:27017

localhost:

admin

config

local



* Conexio a MongoDB Atlas

- Crear Projecto en Mongo Atlas

- Crear DB an Mongo Atlas.

* Conectarse a Mongo Atlas desde VSCODE

\_> mongodb+srv://admin:<password>@cluster0.gioxlj8.mongodb.net/

\_>mongodb+srv://admin:admin@cluster0.gioxlj8.mongodb.net/

* Conectarse a Mongo Atlas desde Pyhton - 3.12+:

\_> mongodb+srv://admin:<password>@cluster0.gioxlj8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

\_> mongodb+srv://admin:admin@cluster0.gioxlj8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

* Codigo de Ejemplo:

from pymongo.mongo\_client import MongoClient

from pymongo.server\_api import ServerApi

uri = "mongodb+srv://admin:<password>@cluster0.gioxlj8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"

# Create a new client and connect to the server

client = MongoClient(uri, server\_api=ServerApi('1'))

# Send a ping to confirm a successful connection

try:

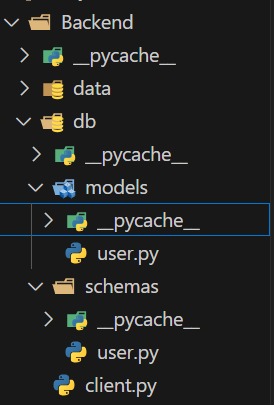
client.admin.command('ping')

print("Pinged your deployment. You successfully connected to MongoDB!")

except Exception as e:

print(e)

* Crear API de Usuarios conectado a MongoDB users\_db.py



Esquemas:

def user\_schemas(user) -> dict:

    return {

        "id": str(user["\_id"]),

        "login": user["login"],

        "password": user["password"],

        "enable": user["enable"]

    }

def users\_schemas(users) -> list:

    return [user\_schemas(user) for user in users]

Modelo:

### Clase usuario de negocio ###

from pydantic import BaseModel   # Permite crear una Entidad

from typing import Optional

class User(BaseModel):

    id: Optional[str] = None # Implica que es opcional

    login: str

    password: str

    enable: int

Cliente:

from pymongo import MongoClient

# Conexion a DB local (localhost)

db\_client = MongoClient().local

API

### USER DB API ###

from fastapi import APIRouter

from db.models.user import User

from db.schemas.user import user\_schemas, users\_schemas

from db.client import db\_client

from bson import ObjectId

router = APIRouter(prefix="/users",

                   tags=["Usuarios"],

                   responses={404: {'message':"No Encontrado"}})

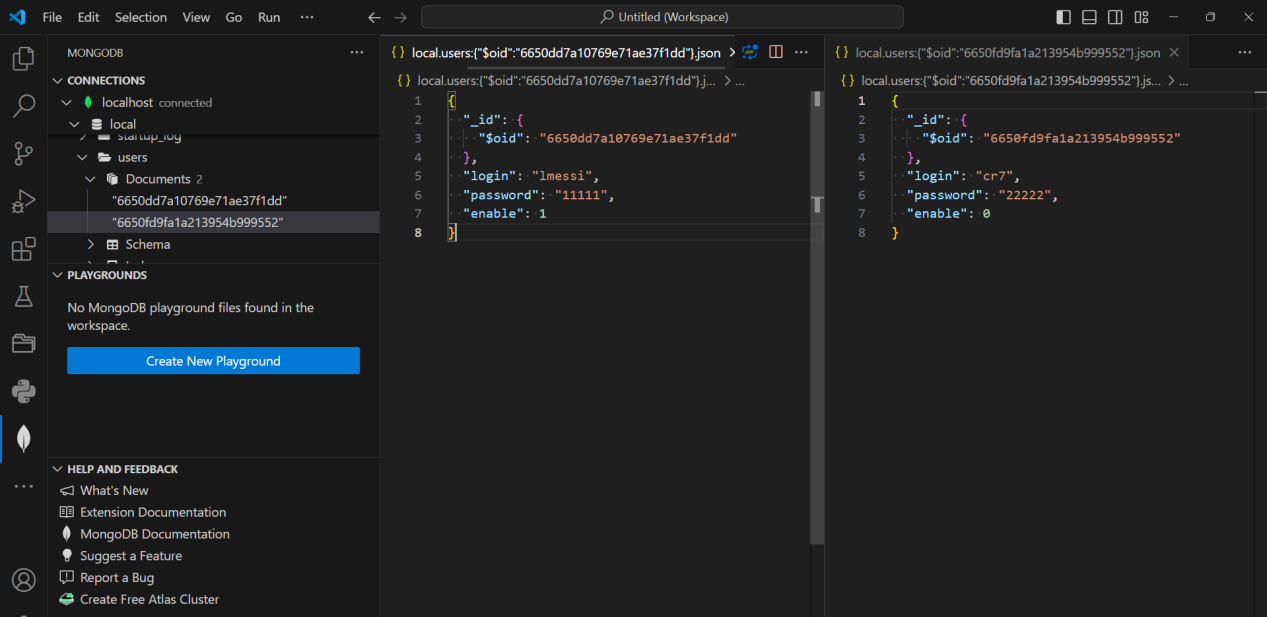
# API Consulta de usuarios

@router.get("/", response\_model=list[User])

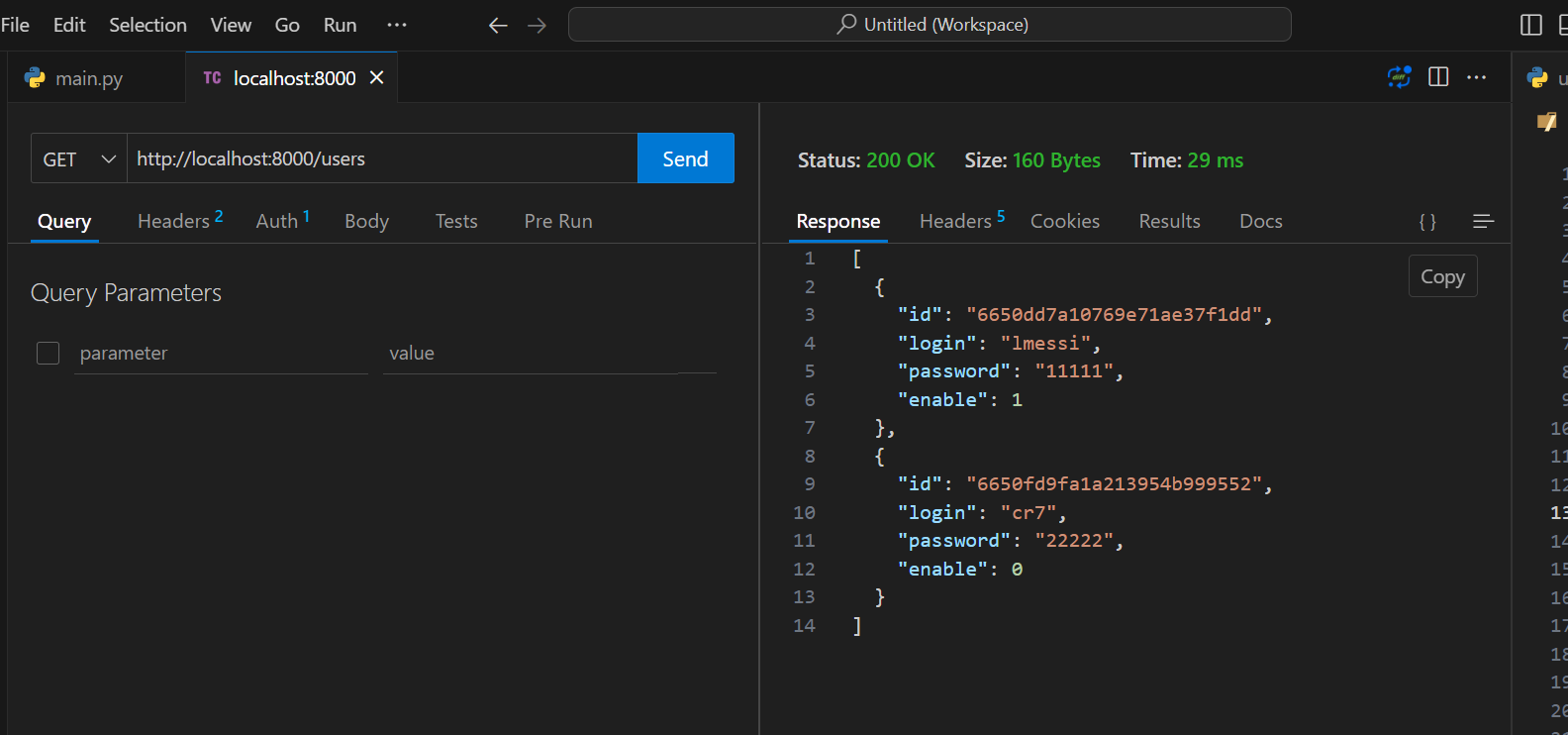
async def users():

    return users\_schemas(db\_client.users.find())

Data cargada como dummy en la DB de mongo:



* Salida:



Consultas de MongoDB en Python. Algunas páginas de ayuda

<https://www.w3schools.com/python/python_mongodb_find.asp>

<https://www.w3schools.com/python/python_mongodb_query.asp>

user-db.py

...

class UserLogin(BaseModel):

    login: str

password: str

...

# API Consulta de usuario por login

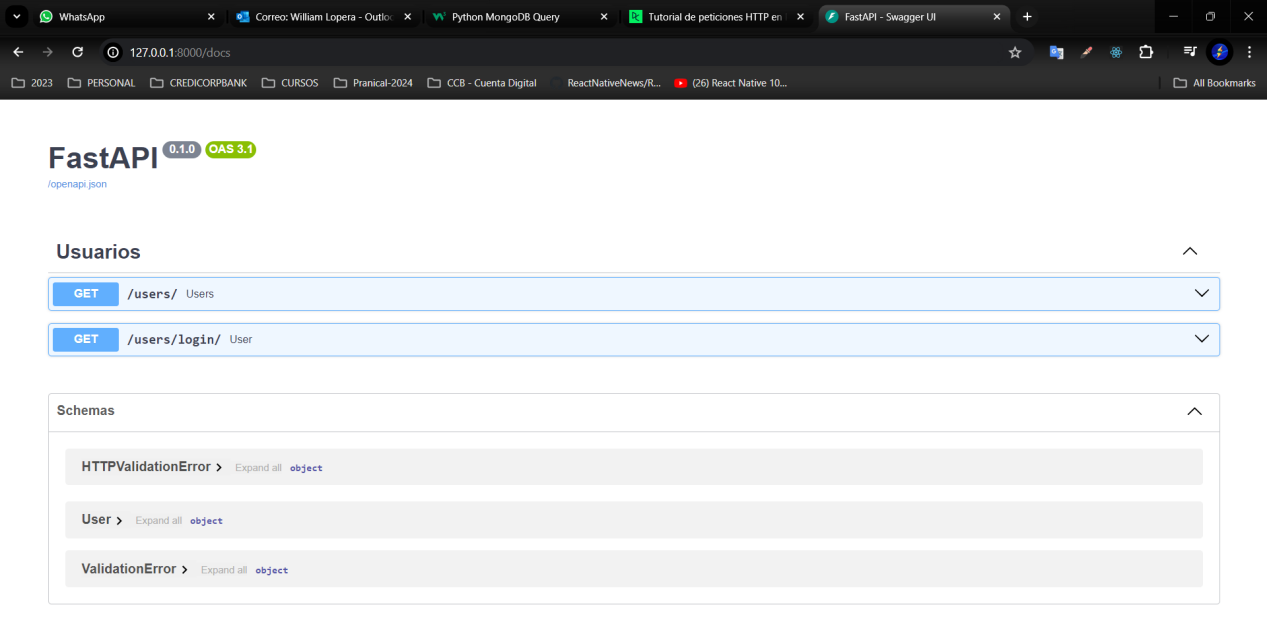
@router.get("/login/", response\_model=User)

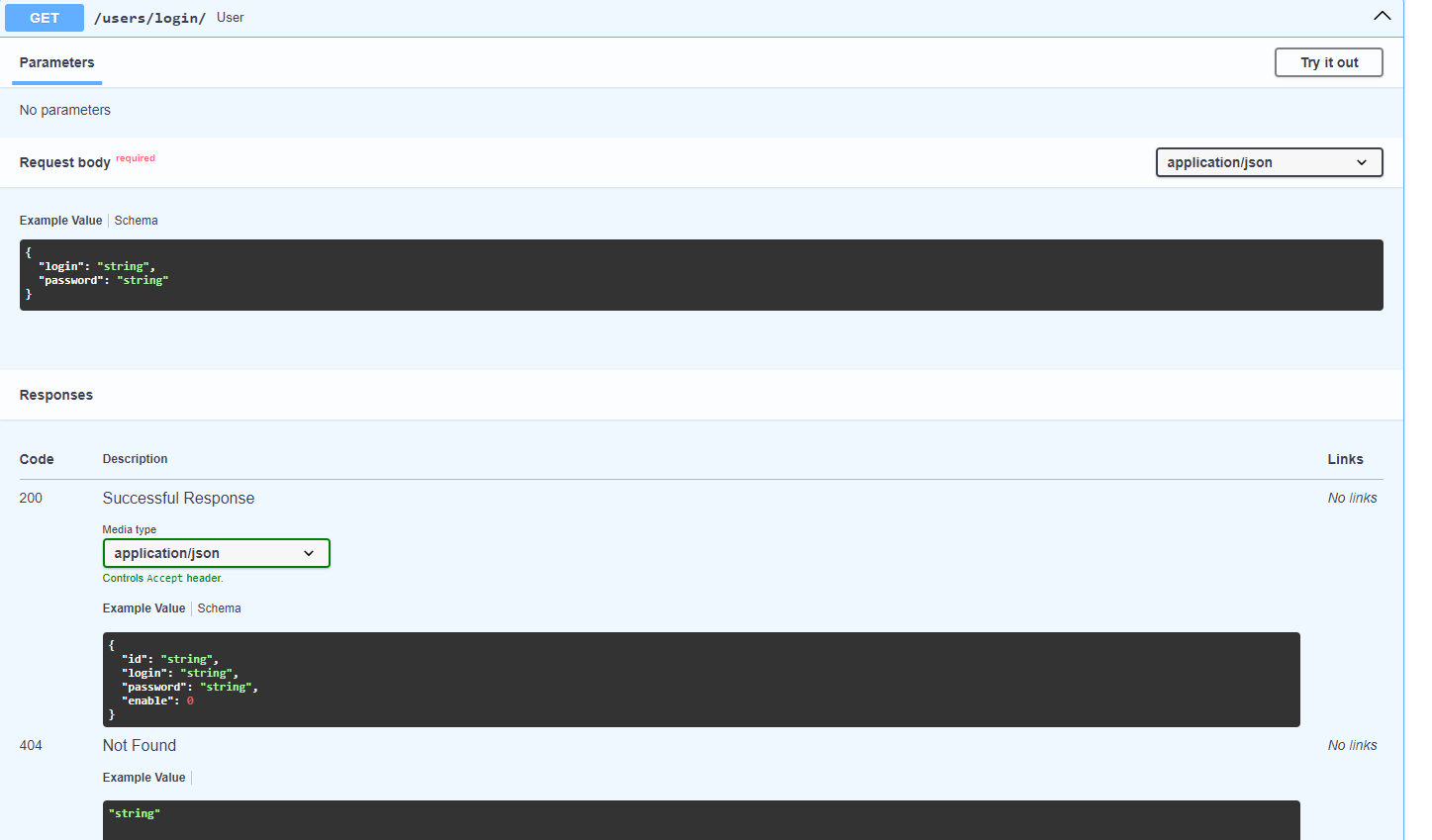
async def user(userLogin:UserLogin):

return user\_schemas(db\_client.users.find\_one({'login':userLogin.login}))

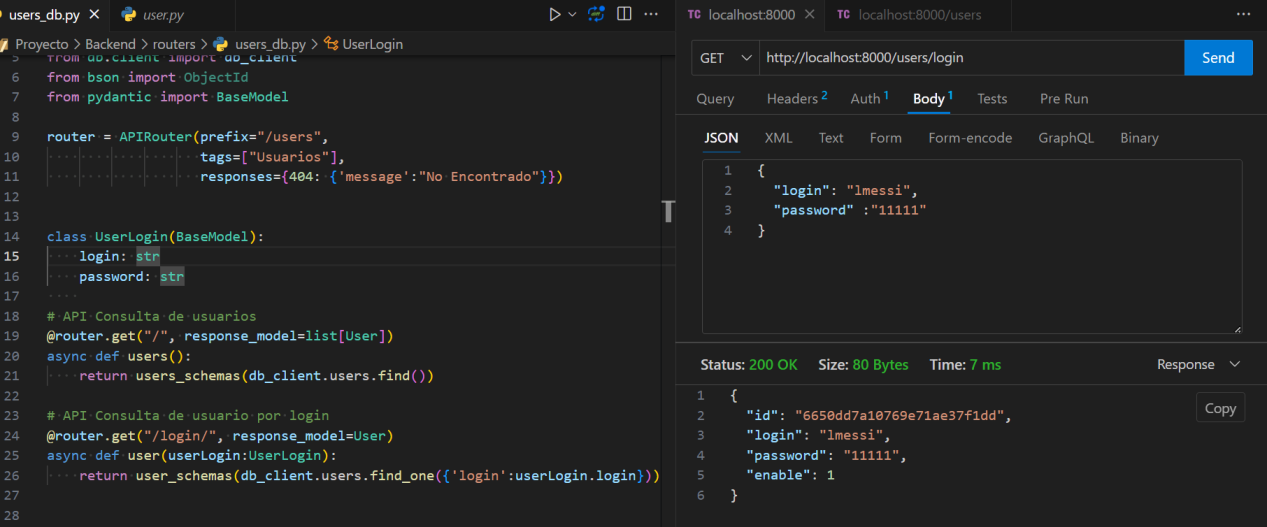
...

* Ver API vía web por Swagger





* Salida



**Código:**

* Clase Usuario

### Clase usuario de negocio ###

from pydantic import BaseModel, Field   # Permite crear una Entidad

from typing import Optional

from bson import ObjectId

class User\_db(BaseModel):

    \_id: Optional[ObjectId] = None

    login: str

    password: Optional[str] = None

    enabled: int

class User\_business(BaseModel):

    id: str

    login: str

    password: Optional[str] = None

enabled: int

* Esquema usuario

# Esquema de la DB User

def user\_schema(user, field) -> dict:

    return {

        field: str(user[field]),

        "login": user["login"],

        "password": user["password"],

        "enabled": user["enabled"]

    }

def users\_schema(users, field) -> list:

    return [user\_schema(user, field) for user in users]

* Route usuarios

### USER DB API ###

from fastapi import APIRouter, HTTPException, status

from db.models.user import User\_db, User\_business

from db.schemas.user import user\_schema, users\_schema

from db.client import db\_client

from bson import ObjectId

from pydantic import BaseModel

router = APIRouter(prefix="/users",

                   tags=["Usuarios"],

                   responses={404: {'message':"No Encontrado"}})

class UserLogin(BaseModel):

    login: str

    password: str

# API Consulta de usuarios

@router.get("/", response\_model=list[User\_business])

async def users():

    users\_db = users\_schema(db\_client.users.find(), "\_id")

    users\_business = [convert\_to\_user\_business(user) for user in users\_db]

    return users\_business

# API Consulta de usuario por login

@router.get("/login/", response\_model=User\_business)

async def user(userLogin:UserLogin):

    try:

        user\_db = db\_client.users.find\_one({"login": userLogin.login})

        user\_bussiness = convert\_to\_user\_business(user\_db)

        return user\_bussiness

    except:

        raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND, detail={"error": "No se encontro usuario"})

# API activar o inactivar usuario

@router.put("/enabled/",  response\_model=User\_business)

async def setEnabled(user\_business:User\_business):

    try:

        update\_data = {

            'enabled': user\_business.enabled

        }

        db\_client.users.update\_one({"\_id": ObjectId(user\_business.id)},  {"$set": update\_data})

        return user\_business

    except:

        return "No se ha actualizado el estado del usuario"

# API modificar password

@router.put("/password/",  response\_model=User\_business)

async def setPassword(user\_business:User\_business):

    try:

        update\_data = {

            'password': user\_business.password

        }

        db\_client.users.update\_one({"\_id": ObjectId(user\_business.id)},  {"$set": update\_data})

        return user\_business

    except:

        return "No se ha actualizado el password"

# Crear usuario

@router.post("/add/", response\_model= User\_business, status\_code=status.HTTP\_201\_CREATED )

async def add\_user(user:User\_db):

    if type(search\_user("login", user.login)) == User\_db:

         raise HTTPException(status\_code=status.HTTP\_404\_NOT\_FOUND, detail="El usuario ya existe")

    else:

        # Crear usuario DB y convertir a Json

        user\_db = User\_db(login=user.login, password =user.password, enabled=user.enabled)

        # Esquema de usuario - generar usuario y obtener el id generado en mongoDB

        id = db\_client.users.insert\_one(dict(user)).inserted\_id

        # Consultar el usuario generado, directamente en mongoDB (\_id es como lo genera mongoDB)

        new\_user = db\_client.users.find\_one({"\_id": id})

        # Generar el objeto User de la respuesta del schema

        return convert\_to\_user\_business(new\_user)

# Funcion para consultar un usuario por campo generico

def search\_user(field: str, key):

    try:

        user\_db = db\_client.users.find\_one({field: key})

        return user\_db

    except:

        return {"error": "No se encontro usuario"}

# Función para convertir de User\_db a User\_business

def convert\_to\_user\_business(user\_db):

    return User\_business(

        id=str(user\_db['\_id']),

        login=user\_db['login'],

        # Comentar para no enviar al frontend

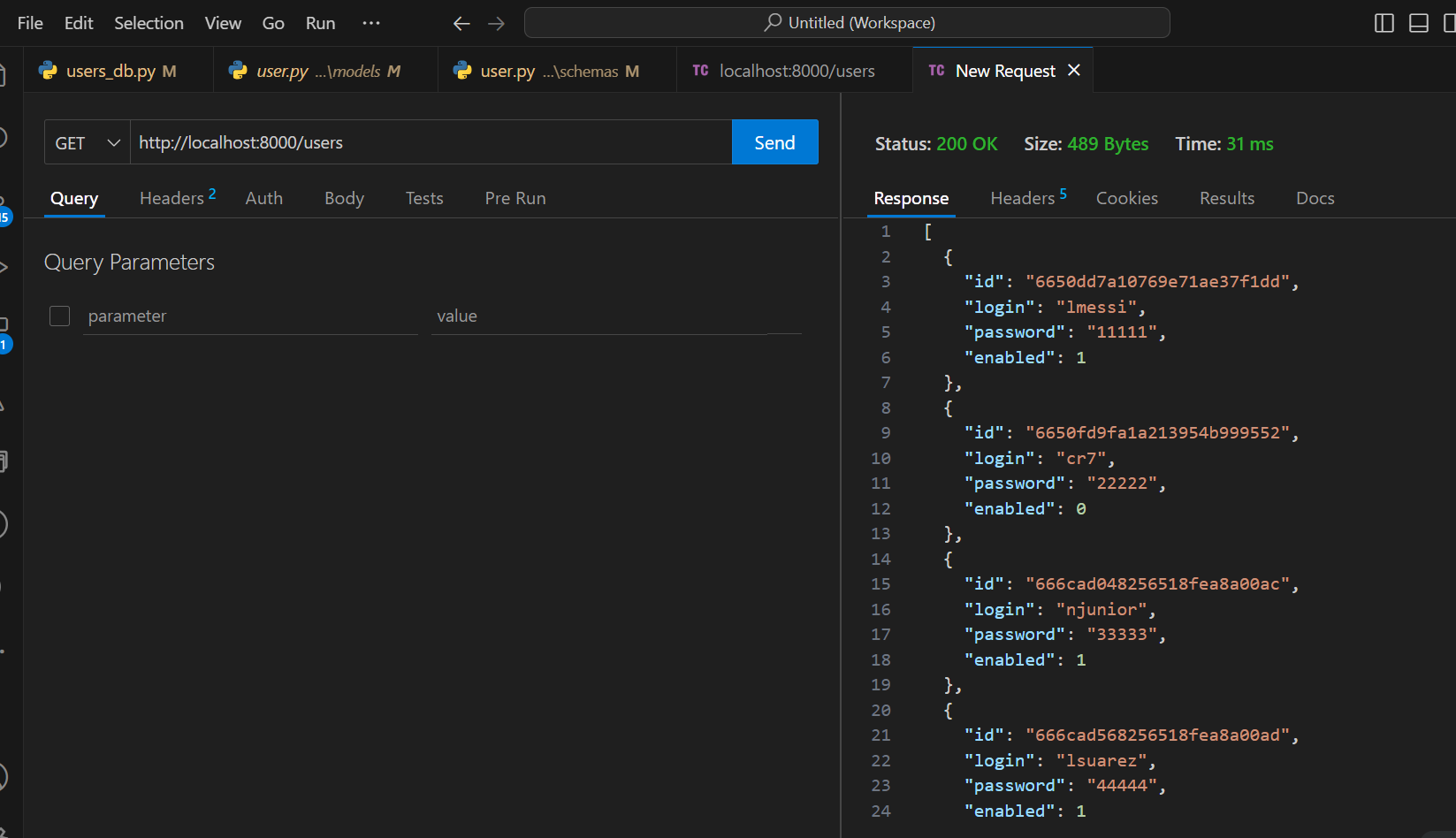
        password=user\_db['password'],

        enabled=user\_db['enabled']

    )

**Salida**

* Consultar Usuarios



* Crear Usuario

