



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Ciclo III

Desarrollo de Software



Capa Lógica: Conexión con la BD

19

Jeisson Andrés Vergara Vargas

Departamento de Ingeniería de Sistemas e Industrial

<http://colswu.unal.edu.co/~javergarav/>
javergarav@unal.edu.co

2020

©

Objetivo de Aprendizaje

Identificar y utilizar las operaciones provistas por el ORM SQLAlchemy.

Conceptos Básicos

Comunicación a través del ORM

Una de las **ventajas** del **ORM** es que **elimina** el uso de consultas con **SQL**. Como **reemplazo** provee una **serie de funciones** con las cuales se puede **administrar** la **Base de Datos**.



Add

Commit

Query

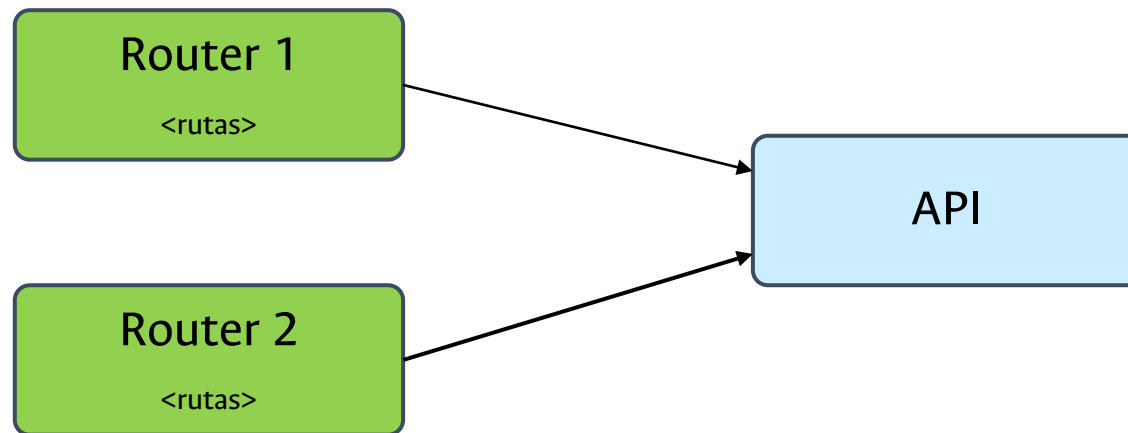
Refresh

La **función** de cada una será mas **evidente** en el **código**.

División en Routers

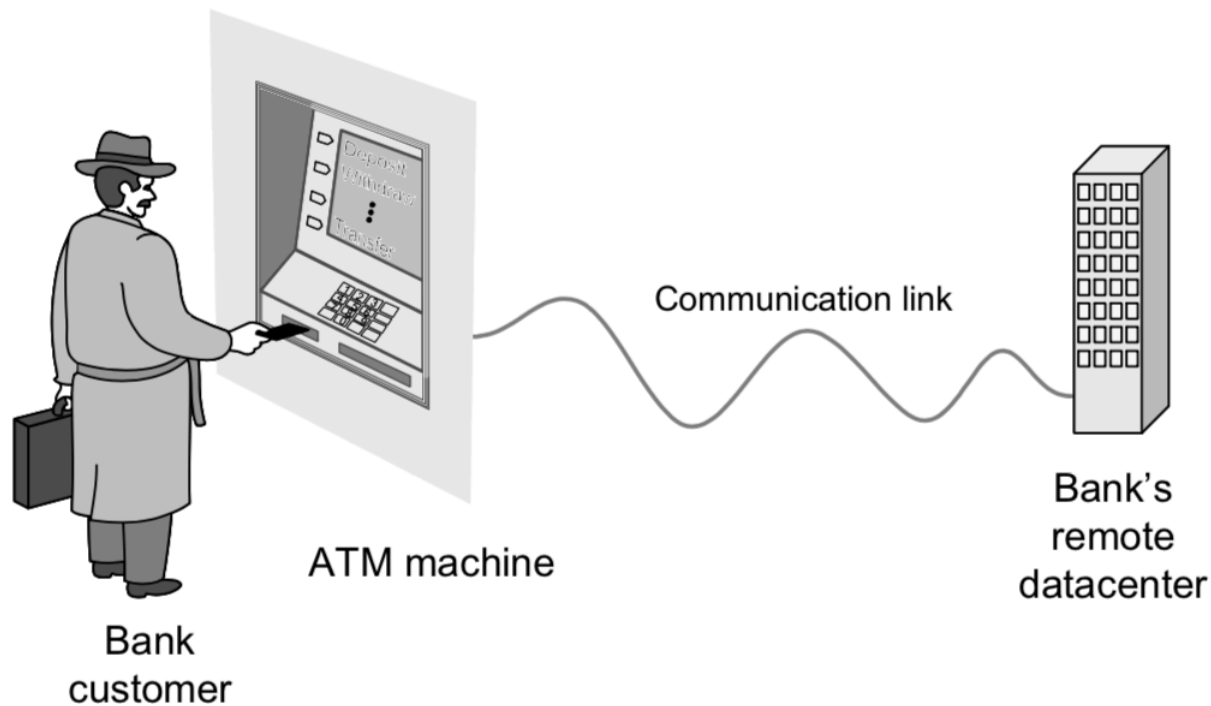
Todas las **rut**as se suelen definir en **main.py**, a medida que crece el **proyecto** esto es insostenible.

Para solventar esto surgen los **router**s, un herramienta que permite definir **rut**as en un **archivo diferente** y luego añadirlas al api, esto **facilita la modularidad**.



Ejemplo

Software para un «ATM»



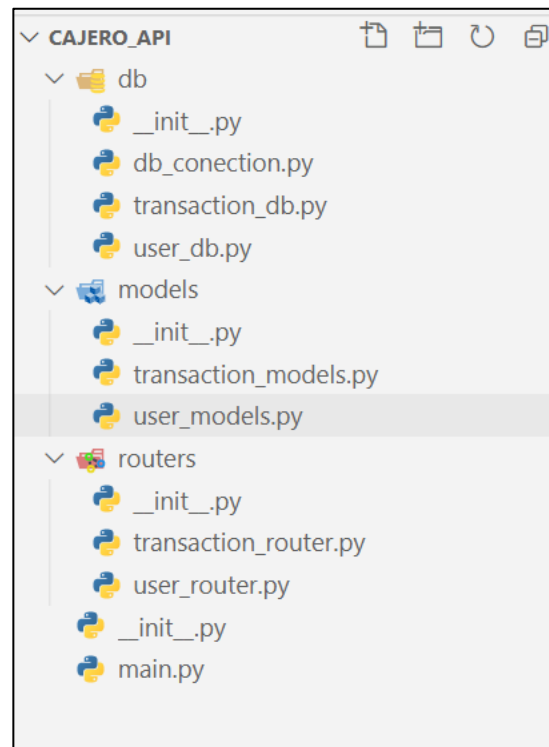
Ejemplo

En esta **sesión** se **realizarán** las siguientes actividades:

1. Definir entidades de datos **adicionales**, con **Pydantic**.
2. Crear las **rutas** para **usuarios** y **transacciones** en **archivos separados** y con las **funcionalidades** del **ORM**.
3. Ejecutar y **Probar** la Aplicación.

Estructura Actual

Hasta el **momento** se tiene la siguiente **estructura**:



La **capa** de **db** ya está **implementada**, lo demás esta **vacío**.

Creación de Entidades de Datos

Creación de Entidades de Datos

De manera **similar** al **primer acercamiento**, se definen unas **entidades** de datos **adicionales**, que representarán diversos estados de los objetos.

- **UserIn**
- **UserOut**
- **TransactionIn**
- **TransactionOut**

En el caso de las **entidades con Out**, es necesario **indicar** que estos provienen de un **ORM**.

Entidades: User

En el archivo `user_models.py` definir el siguiente código:

```
from pydantic import BaseModel

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

class UserOut(BaseModel):
    username    : str
    balance     : int

class Config:
    orm_mode = True
```

Entidades: Transaction

En el archivo `user_transaction.py` definir el siguiente código:

```
from pydantic import BaseModel
from datetime import datetime

class TransactionIn(BaseModel):
    username    : str
    value       : int

class TransactionOut(BaseModel):
    id          : int
    username    : str
    date        : datetime
    value       : int
    actual_balance : int

class Config:
    orm_mode = True
```

Implementando Rutas

Implementando Rutas

A continuación se implementarán las **rutas** que ofrece la **API**, para esto tener en cuenta los siguientes aspectos:

- Las **rutas** para **user** y **transaction** se implementarán en **archivos diferentes** usando **routers**.
- Se debe notar la **inyección** de la **dependencia db**.
- Notar el uso de las **funcionalidades proveídas** por el **ORM** para manipular los **Datos**.
- La **unión** de los **Routers**.

Implementando Rutas: User

Rutas: User

Parte 1: En el archivo `user_router.py`, importar todo lo necesario:

```
from typing import List
```

```
from fastapi import Depends, APIRouter, HTTPException  
from sqlalchemy.orm import Session
```

```
from db.db_connection import get_db
```

```
from db.user_db import UserInDB  
from db.transaction_db import TransactionInDB
```

```
from models.user_models import UserIn, UserOut  
from models.transaction_models import TransactionIn, TransactionOut
```

Rutas: User

Parte 2: En el archivo `user_router.py`, definir el siguiente código:

```
router = APIRouter()

@router.post("/user/auth/")
async def auth_user(user_in: UserIn, db: Session = Depends(get_db)):
    user_in_db = db.query(UserInDB).get(user_in.username)

    if user_in_db == None:
        raise HTTPException(status_code=404,
                             detail="El usuario no existe")

    if user_in_db.password != user_in.password:
        raise HTTPException(status_code=403,
                             detail="Error de autenticacion")

    return {"Autenticado": True}
```

Rutas: User

Parte 3: En el archivo `user_router.py`, definir el siguiente código:

```
@router.get("/user/balance/{username}", response_model=UserOut)
async def get_balance(username: str, db: Session = Depends(get_db)):

    user_in_db = db.query(UserInDB).get(username)

    if user_in_db == None:
        raise HTTPException(status_code=404,
                             detail="El usuario no existe")

    return user_in_db
```

Implementando Rutas: Transaction

Rutas: Transaction

Parte 1: En el archivo `transaction_router.py`, importar todo lo necesario:

```
from typing import List
```

```
from fastapi import Depends, APIRouter, HTTPException  
from sqlalchemy.orm import Session
```

```
from db.db_connection import get_db
```

```
from db.user_db import UserInDB  
from db.transaction_db import TransactionInDB
```

```
from models.user_models import UserIn, UserOut  
from models.transaction_models import TransactionIn, TransactionOut
```

Rutas: Transaction

Parte 2: En el archivo `transaction_router.py`, definir:

```
router = APIRouter()

@router.put("/user/transaction/", response_model=TransactionOut)
async def make_transaction(transaction_in: TransactionIn,
                           db: Session = Depends(get_db)):

    user_in_db = db.query(UserInDB).get(transaction_in.username)

    if user_in_db == None:
        raise HTTPException(status_code=404,
                             detail="El usuario no existe")

    if user_in_db.balance < transaction_in.value:
        raise HTTPException(status_code=400,
                             detail="No se tienen los fondos suficientes")
```

Rutas: Transaction

Parte 3: En el archivo `transaction_router.py`, definir:

```
user_in_db.balance = user_in_db.balance - transaction_in.value
db.commit()
db.refresh(user_in_db)

transaction_in_db = TransactionInDB(**transaction_in.dict(),
                                     actual_balance = user_in_db.balance)

db.add(transaction_in_db)
db.commit()
db.refresh(transaction_in_db)

return transaction_in_db
```

Nota: es importante garantizar la **identación**.

Uniando Rutas

Definiendo Main.py

Parte 3: En el archivo **main.py**, unir los routers:

```
from fastapi import Depends, FastAPI

from routers.user_router import router as router_users
from routers.transaction_router import router as router_transactions

api = FastAPI()

api.include_router(router_users)
api.include_router(router_transactions)
```

Instalación de Paquetes Necesarios

Paquetes Necesarios

Para la correcta **ejecución** de la **API**, son necesario los **siguientes paquetes**:

- `pip install fastapi`
- `pip install uvicorn`
- `pip install SQLAlchemy`
- `pip install pydantic`
- `pip install psycopg2`

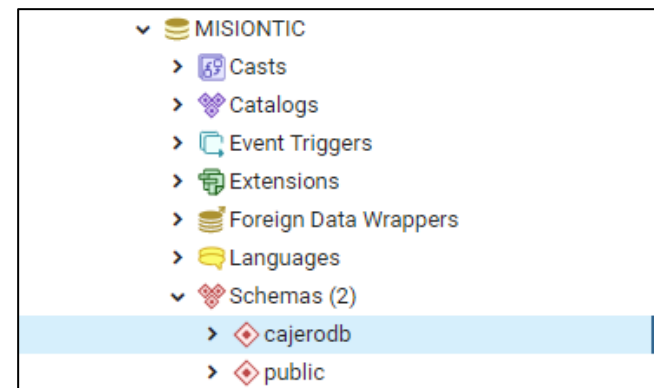
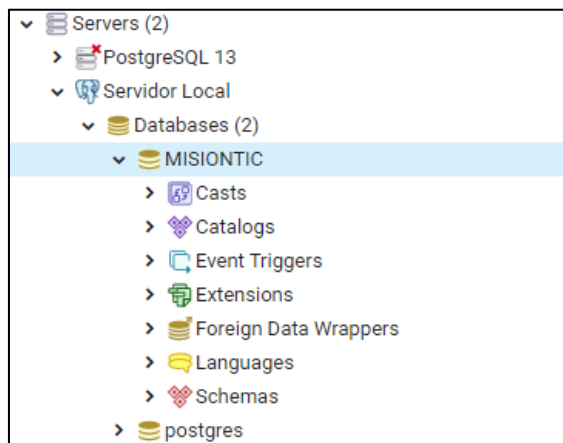
Instalar cualquier otro **paquete** en caso de **ser requerido**.

Base de Datos y Esquema

Base de Datos

Recordando la **sesión pasada**, se debe tener un **servidor** que cuente con:

- Una **base de datos** llamada “**MISIONTIC**”
- La anterior base de datos debe tener un **esquema vacío** llamado “**cajerodb**”



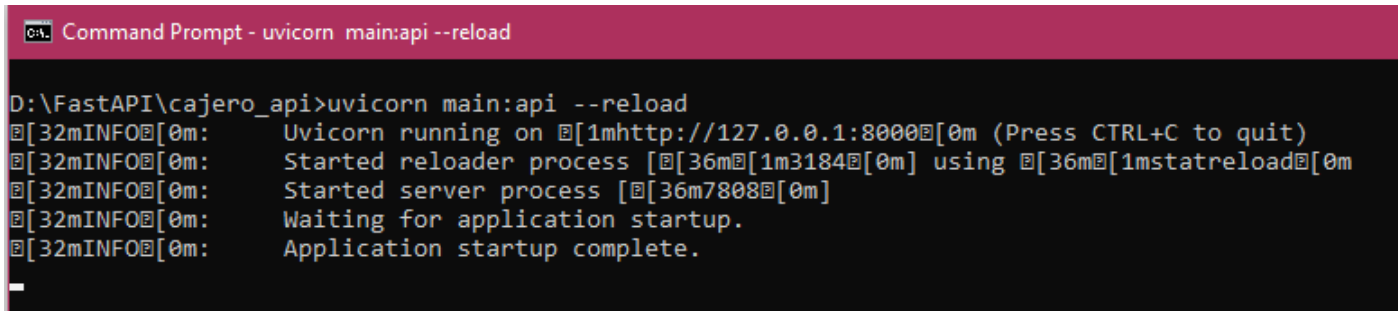
Ejecución

Exécution API-REST

Para **ejecutar** la **api**, **abrir una consola** y ubicarse en la **raíz del proyecto** (en la carpeta `cajero_api`) y ejecutar el siguiente **comando**:

`uvicorn main:api --reload`

Resultado:



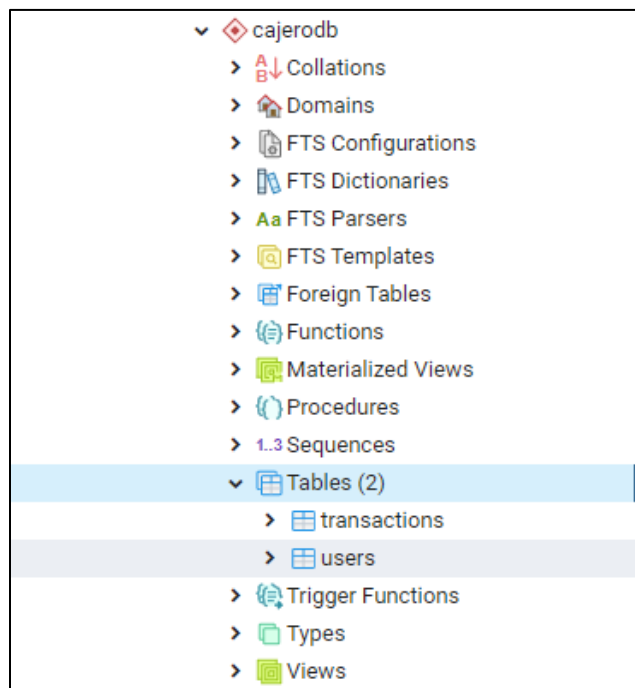
```
Command Prompt - uvicorn main:api --reload

D:\FastAPI\cajero_api>uvicorn main:api --reload
[32mINFO[0m:      Uvicorn running on [1mhttp://127.0.0.1:8000[0m (Press CTRL+C to quit)
[32mINFO[0m:      Started reloader process [36m[1m3184[0m] using [36m[1mstatreload[0m
[32mINFO[0m:      Started server process [36m[1m7808[0m]
[32mINFO[0m:      Waiting for application startup.
[32mINFO[0m:      Application startup complete.
```

Migraciones Automáticas

Migraciones Automáticas

Una vez es **ejecutada** la aplicación, se puede observar cómo se **crean** las correspondientes **tablas** en la base de datos:



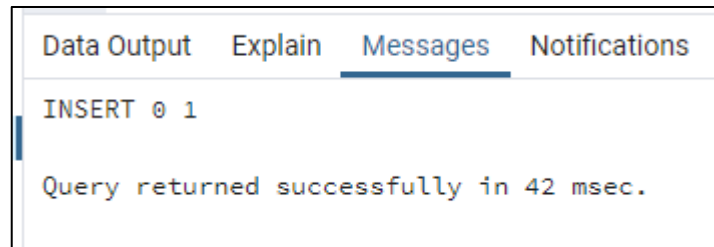
Prueba de la Aplicación

Interacción con la Base de Datos

Para iniciar, se debe **poblar** la **base de datos**, ingresar algunos usuarios, ejecutando la **siguiente** Query en **pgAdmin**:

```
INSERT INTO cajeroadb.users(username, password, balance)  
VALUES ('camilo24', 'root', 12000);
```

El **resultado** será:



Nota: ingresar más usuarios.

Probando la Petición: auth_user

Ingresa la siguiente **información** que construirá la **petición**:

Método: POST

URL: http://127.0.0.1:8000/user/auth/

Body (JSON): {

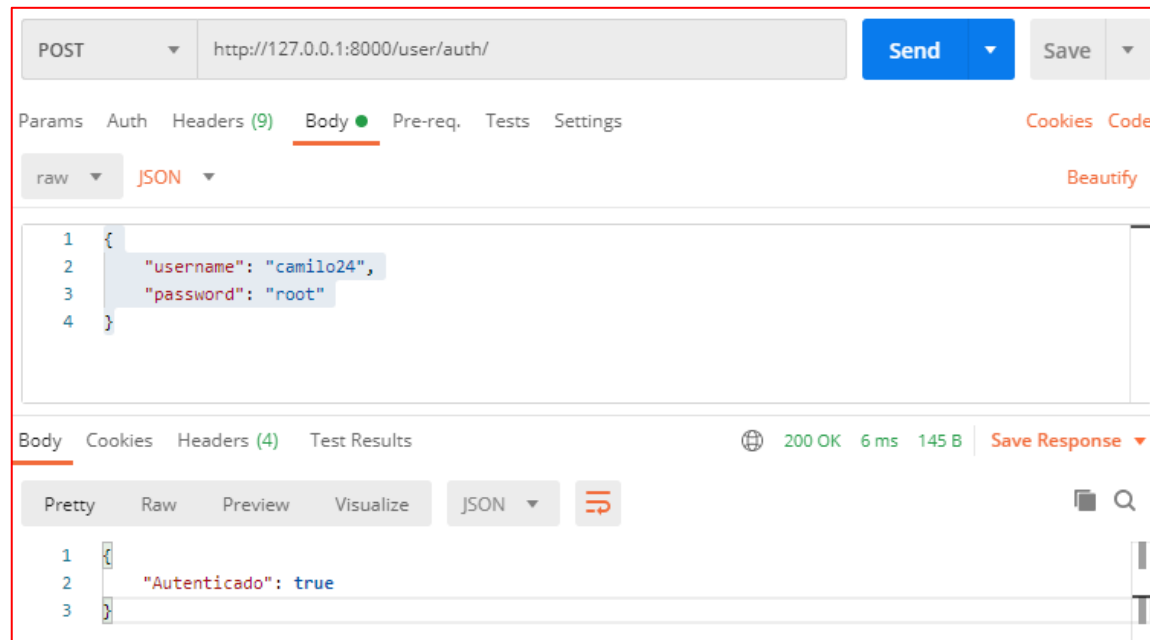
 "username": "camilo24",

 "password": "root"

}

Probando la Petición: auth_user

En **POSTMAN**:



Probando la Petición: get_balance

Ingresar la siguiente **información** que construirá la **petición**:

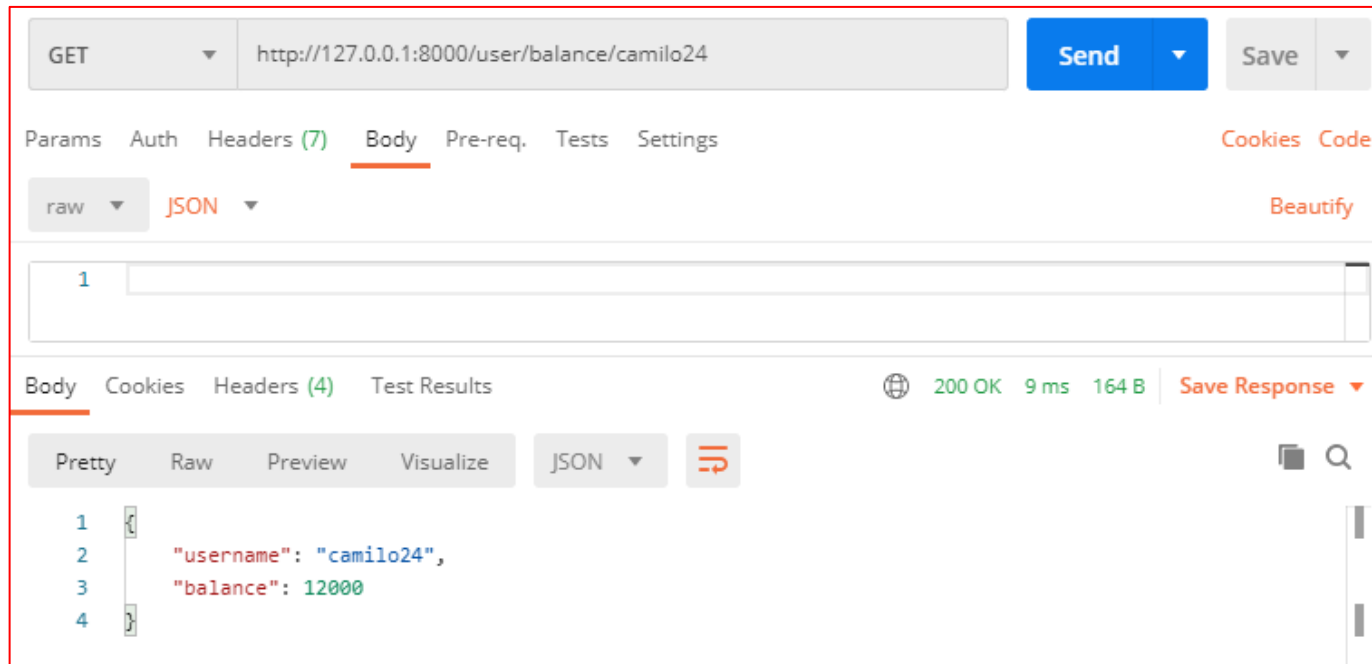
Método: GET

URL: `http://127.0.0.1:8000/user/balance/camilo24`

Body (JSON): `{}`

Probando la Petición: get_balance

En **POSTMAN**:



Probando la Petición: make_transaction

Ingresa la siguiente **información** que construirá la **petición**:

Método: PUT

URL: http://127.0.0.1:8000/user/transaction/

Body (JSON): {

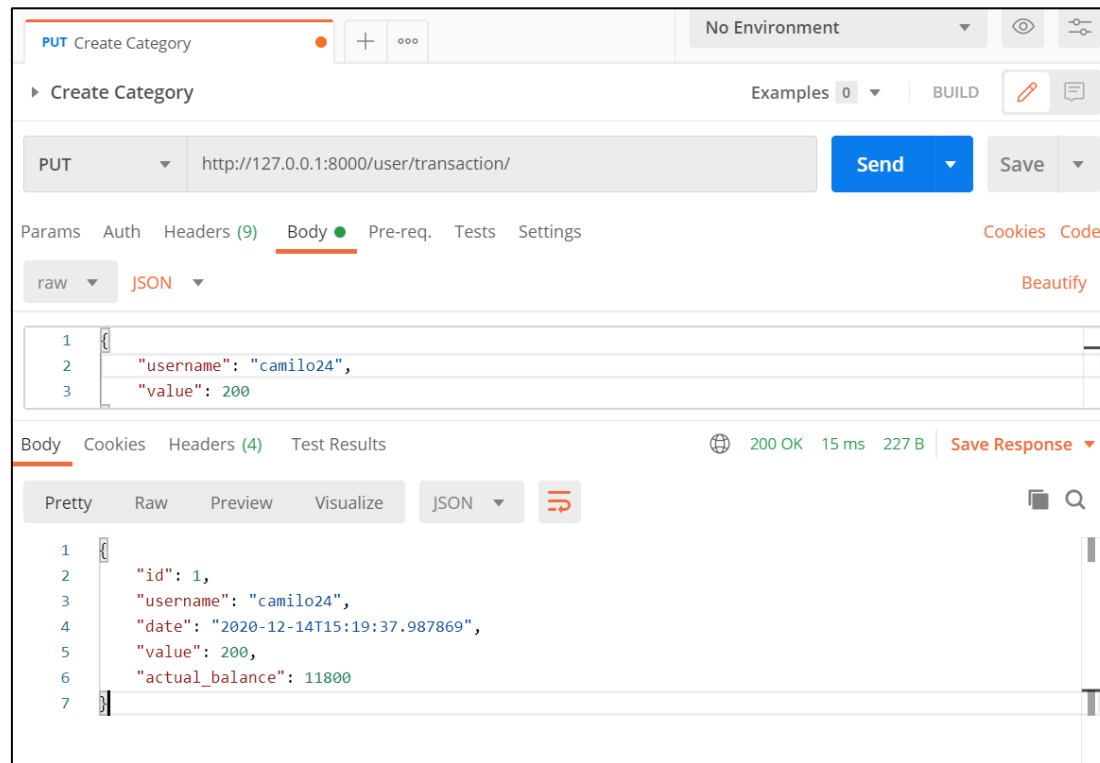
"username": "camilo24",

"value": 200

}

Probando la Petición: make_transaction

En **POSTMAN**:



Revisando Cambios en la Base de Datos

Ver **información** de **Usuario**:

```
SELECT username, password, balance
FROM cajerodb.users
WHERE username='camilo24';
```

El **resultado** será:

Data Output				Explain	Messages	Notifications
	username [PK] character varying	password character varying	balance integer			
1	camilo24	root	11800			

Notar que el saldo cambió.

Revisando Cambios en la Base de Datos

Ver **transacciones**:

```
SELECT id, username, date, value, actual_balance  
FROM cajerodb.transactions;
```

El **resultado** será:

Data Output Explain Messages Notifications						
	id [PK] integer	username character varying	date timestamp without time zone	value integer	actual_balance integer	
1	1	camilo24	2020-12-14 15:19:37.987869	200	11800	

Notar que la transacción fue realizada.

Referencias

- [FASTAPI] Comunidad FastAPI. (2020, noviembre). FastAPI. FastAPI.
<https://fastapi.tiangolo.com/>