

Servidor Falso de DOTERS



Para poder continuar con nuestro desarrollo vamos a tener un servidor FLASK en python el cual va a simular todas las respuestas:

Este proyecto implementa un **servidor Flask en Python** que simula el comportamiento de la **API de Doters**, incluyendo los flujos de autenticación, registro, generación de puntos, redención y callbacks de partners como **Rappi**.

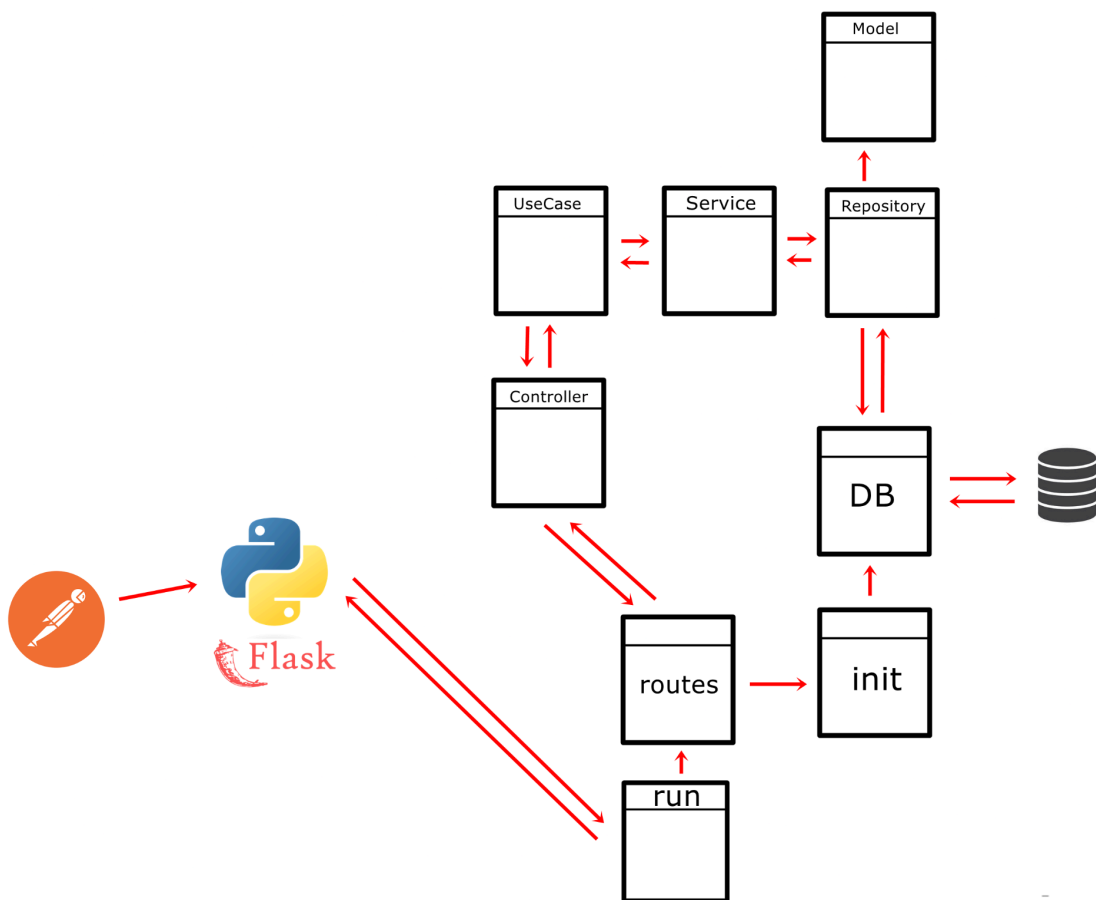
Su propósito es servir como entorno de pruebas (*mock server*) para integraciones sin conexión con servicios productivos.

Arquitectura:

Capa	Responsabilidad	Descripción
Routing	app/routes.py	Registra blueprints o handlers por dominio.
INIT	app/__init__.py	Inicializa la APP, ROUTER y DB.
Helpers	app\helpers\response.py	Genera una respuesta estándar JSON.
Controller	app/controllers/*	Capa de orquestación HTTP.
UseCase	app/UseCases/*	Orquesta servicios para resolver la lógica de negocio.
Services	app/services/*	Es el código de lógica de negocio.
Repositorios	app/repositories/*	Realiza llamados a DB usando Modelos. Acceso a datos (SQLite) usando DbContext.

Modelos	app/models/*	Son las entidades del dominio de negocio.
Infraestructura	app\Database\context.py	Singleton de SQLite, crea la conexión a DB y expone la base de datos.

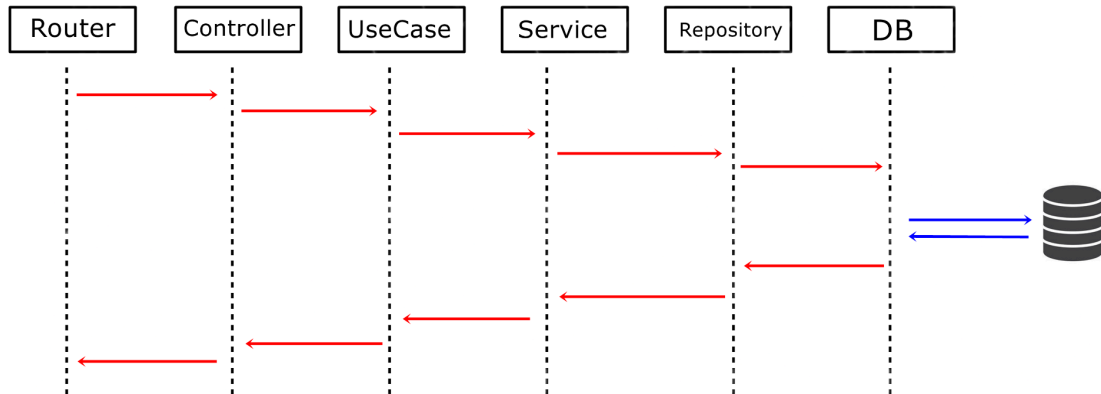
Diagrama de alto nivel



ROUTER > CONTROLLER > UseCase > Service > Repository > DB

Flask enruta las peticiones hacia los controladores, los cuales delegan la lógica de negocio en los *UseCases*, que a su vez coordinan los servicios y repositorios para acceder a la base de datos SQLite.

Diagrama de secuencia



Funcionalidad 0 “REGISTER”

Es posible persistir un usuario y ciertos datos personales mediante el siguiente cURL de ejemplo:

```
curl --location 'http://127.0.0.1:5000/v2/user/signup' \
--header 'Content-Type: application/json' \
--data-raw '{
  "language": "es-MX",
  "loyaltyDetails": {
    "enrollingSponsor": "1",
    "enrollingChannel": "AUTO"
  },
  "phoneNumber": "5590234567",
  "email": "rappi_qa@rappi.com",
  "password": "kmzwa8awaa",
  "activationUrl": "doters.com",
  "subscribeToNotifications": true,
  "personalDetails": {
    "name": {
      "firstName": "Andres Felipe",
      "lastName": "Hernandez",
      "title": null
    }
  },
  "dateOfBirth": "1991-01-13T00:00:00.000Z",
  "gender": "XX",
  "residentCountry": "MX"
}'
```

Esos datos quedarán almacenados sin cifrar en la base de datos de SQLite3 que podemos ver en la ruta:

DB [DOTERS.db](#)

Funcionalidad 1 “REGISTRO DE URL CALLBACK”

Para poder realizar el proceso de login es necesario establecer previamente una dirección de callback/rebote la cual se encargará de recibir los datos que nuestro servidor genera:

access_token, id_token,refresh_token,scope...

Y para establecer la dirección de rebote lo hacemos mediante el siguiente cURL:

```
curl --location 'http://127.0.0.1:5000/sso/v2/member/callback/register' \
--header 'Content-Type: application/json' \
--data '{
  "memberId": 1,
  "callbackUrl": "http://127.0.0.1:777/crazy"
}'
```

Y si es exitoso vamos a recibir el siguiente mensaje:

The screenshot displays a REST client interface with the following details:

- URL:** `http://127.0.0.1:5000/sso/v2/member/callback/register`
- Method:** `POST`
- Body (raw):**

```
{
  "memberId": 1,
  "callbackUrl": "http://127.0.0.1:777/crazy"
}
```
- Status:** `200 OK` (10 ms, 313 B)
- Response (JSON):**

```
{
  "action": "CONTINUE",
  "data": {
    "callback_url": "http://127.0.0.1:777/crazy",
    "id": 3,
    "member_id": 1
  },
  "type": "SUCCESS"
}
```

Funcionalidad 2 “LOGIN”

Para poder obtener los tokens de la aplicación es necesario abrir el navegador y entrar a la siguiente dirección

```
http://127.0.0.1:5000/?
  clientId={clientId}
  &clientSecret={clientSecret}
  &language={language}
  &redirectUri={redirectUri}
  &state={state}
  &go_to_page={gotoPage}
  &utms={utmsParams}
```

De momento podemos acceder con el usuario que viene por defecto en el cURL de registro “POSTMAN” (Funcionalidad 0):

http://127.0.0.1:5000/?clientId=1&clientSecret=kmzwa8awaa&language=es-MX&redirectUri=http://127.0.0.1:4000/callback&state=xyz&go_to_page=login

Bienvenido

Usuario

Contraseña

Iniciar sesión

Al momento de ingresar la contraseña y el correo si las credenciales son correctas se enviará el JSON de respuesta a la URL de CALLBACK registrada (Funcionalidad 1)

Lo que se hace es redirigir y mandar los tokens como parámetro a la dirección de callback por un método POST.

Y esta es una respuesta de ejemplo:

http://127.0.0.1:4000/callback?access_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJkb3RlcnMtyXBwiiwY2pZW50X2lkIjoizG9rOZxzLWFwcCslbmV4cCl6MTc2MDc0MTcyNSwiaWF0IjoxNzM5YWNzM0MTT1LCJpc3MiOiJkb3RlcnMtbW9jaWsiLnJybGUioiJzZW1iZXII.CjZy29wZS6lm9wZ5pZCbWmZsaW5XfY2VzcjBlkFpbCBwcm9maWxllicwicz3ViljoicmFwcGlfcFAcmFwcGkuY29tliwidGr9rZ5fdXNllloiYWJWJnZjXNllwiidWkljoisMSJ9.XxOSza9gc6oX8Dm2XLIDzKqbbChTZVN_2hOKPLGIThg&expires_in=86400&id_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJkb3RlcnMtyXBwiiwY29tliwidGr9rZ5fdXNllloiYWJWJnZjXNllwiidWkljoisMSJ9.XxOSza9gc6oX8Dm2XLIDzKqbbChTZVN_2hOKPLGIThg&refresh_token=0WsteyOWU14Ew-tvf7KEK0Q3SSr6ak1PJMWVv9N8abg&scope=openid+offline_access+email+profile&token_type=bearer&state=12345

Parámetro	Descripción
access_token	Es un token de acceso que representa la autorización del usuario para que el cliente (por ejemplo, Rappi) pueda acceder a ciertos recursos en nombre de ese usuario, Se usa para hacer peticiones a las APIs protegidas. Por ejemplo, consultar puntos, historial de transacciones, etc.
expires_in	Indica la duración en segundos del access_token antes de que expire, Permite al cliente saber cuánto tiempo puede usar el token antes de solicitar uno nuevo o usar el refresh_token.
id_token	<p>Es un JWT (JSON Web Token) que contiene la identidad del usuario autenticado.</p> <p>Informa al cliente quién es el usuario que inició sesión, incluyendo campos como:</p> <p>sub: identificador único del usuario. email, uid, name, role. aud: el cliente al que va dirigido el token. iat, exp: tiempos de emisión y expiración. iss: quién emitió el token (servidor DOTERS).</p>
refresh_token	<p>Es un token de actualización de larga duración, Permite al cliente obtener un nuevo access_token sin que el usuario tenga que volver a iniciar sesión.</p> <p>Usualmente se intercambia en un endpoint como /token con el grant_type=refresh_token.</p>
scope	<p>Es una lista de permisos (separados por espacios) que define qué recursos puede acceder el access_token.</p> <p>Le dice al servidor qué nivel de acceso tiene el cliente. Ejemplo de scopes típicos:</p> <p>openid: obligatorio en OpenID Connect. email: acceso al correo del usuario. profile: acceso al nombre o datos personales. offline_access: permite usar refresh_token.</p>
token_type	Especifica el tipo de token que se está entregando, en OAuth2 el valor estándar es siempre "Bearer". Indica al cliente cómo debe enviar el token en las peticiones HTTP.
state	<p>Permite al cliente mantener la relación entre la solicitud y la respuesta (para saber a qué sesión o acción pertenece).</p> <p>También previene ataques CSRF.</p> <p>El servidor lo devuelve exactamente igual a como lo recibió.</p>

Como crear un nuevo ROUTER FULL

- Crear model en caso de que no exista:
app/models
- Crear la tabla de DB en caso de que no exista:
app/Database/context.py
- Crear el repositorio:
app/repositories
- Crear el servicio:
app/services
- Crear el caso de uso:
app/UseCases
- Crear el controlador:
app/controllers
- Registrar en el router:
app/routes.py

Ejemplo para crear la feature login

Paso 0:

Verificar la existencia de la tabla usuario que contenga email y pass en DB:
app\Database\context.py

Paso 1:

Crear el repositorio que va a extraer email y pass de la tabla User:
app\repositories\login_repository.py




Paso 2:

Crear el servicio que se encargará de hacer la llamada al el repositorio:
app/services/login_service.py

Paso 3:

Crear el caso de uso el cual ejecutará el request para dar el resultado:

app/UseCases/login_use_case.py

> UseCases >  login_use_case.py >  LoginUseCase >  execute

```
class LoginUseCase:
```

```
    def __init__(self, service: LoginService):  
        self.service = service
```

```
>    def execute(self, payload: dict) -> bool: ...
```