

# Oauth

OAuth es un framework opensource que nos da la posibilidad de tener una autorización segura con el uso de un API. En su versión OAuth 2.0 sus principales mejoras son que ahora proporciona flujos de autorización para aplicaciones web, de escritorio, teléfonos móviles. Actualmente servicios como Google, Facebook, Github solo admiten el protocolo OAuth 2.0.

## Cómo funciona OAuth 2.0

Lo que hace es permitir que las aplicaciones obtengan acceso limitado a las cuentas de usuario de algunos servicios como Facebook, Google, Twitter y GitHub. Su funcionamiento básicamente consiste en delegar el permiso de autenticación del usuario al servicio que gestiona dichas cuentas, de modo que es el propio servicio el que nos da acceso a las aplicaciones de terceros.

## Flujo del protocolo OAuth

El flujo descrito a continuación es un flujo genérico que representa al protocolo OAuth.

1. La aplicación cliente solicita una autorización para acceder a los recursos de un usuario, en un servicio determinado.
2. Si el usuario autoriza esta solicitud, la aplicación recibe una **authorization grant**.
3. La aplicación cliente solicita un **access token** al **authorization server**, demostrando que es un cliente válido, y el permiso concedido anteriormente.
4. Si la identidad de la aplicación cliente se valida adecuadamente por el servicio, y la concesión de autorización es válida, el **authorization server** emite un **access token** a la aplicación cliente. Con esto la autorización se ha completado.
5. La aplicación cliente puede presentar el **access token** recibido y "solicitar un recurso" al **resource server**.
6. Si el **access token** es válido, el **resource server** hace entrega del recurso a la aplicación.

Tener en cuenta que con recurso nos referimos generalmente a acceder a información del usuario, pero si la API del servicio lo permite, también es posible ejecutar acciones a nombre del usuario.

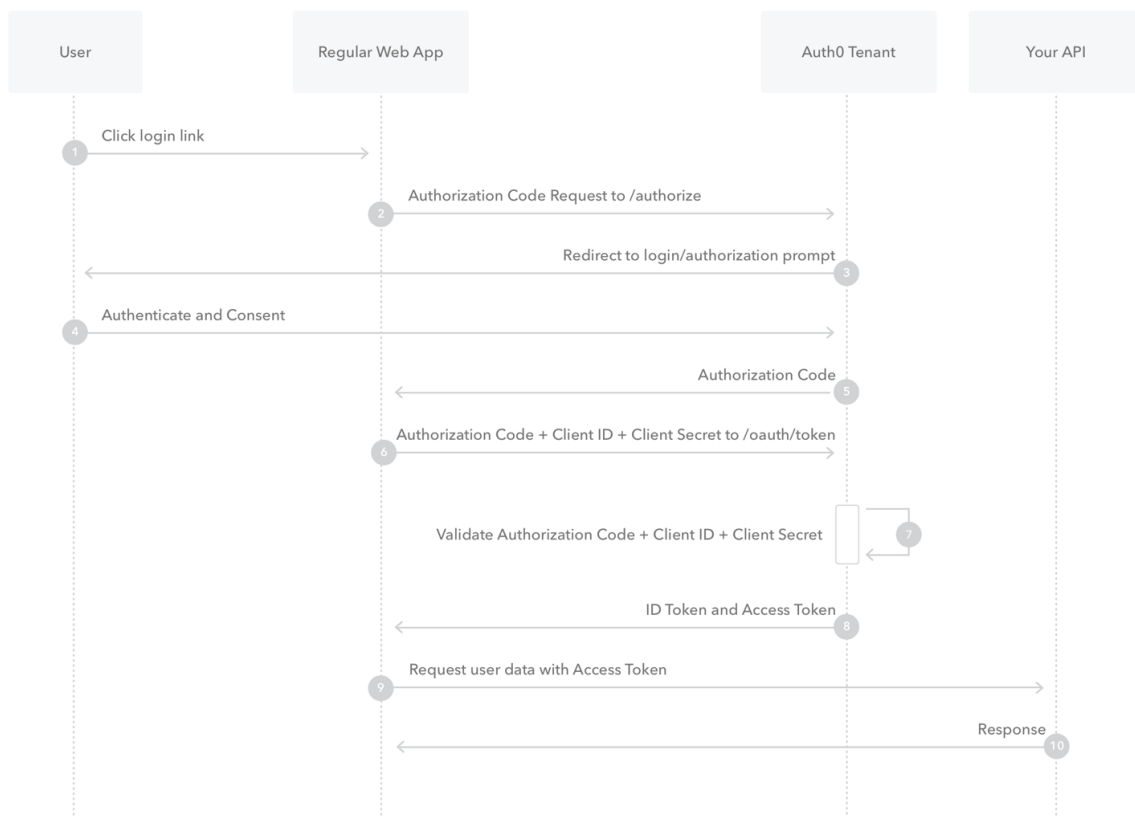
## Flujo de protocolo abstracto



Como nosotros usamos Auth0 el flujo cambia un poco para adaptarse al funcionamiento de este.

1. El usuario hace clic en Iniciar sesión dentro de la aplicación web normal.
2. El SDK de Auth0 redirige al usuario al servidor de autorización de Auth0 (/authorize endpoint).
3. El servidor de autorización Auth0 redirige al usuario a la solicitud de inicio de sesión y autorización. El usuario podrá elegir entre los diferentes proveedores de identidad configurados en la app Auth0 (Google, GitHub, etc o con la cuenta del propio Auth0).
4. El usuario se autentica utilizando una de las opciones de inicio de sesión y puede ver una página de consentimiento que enumera los permisos que Auth0 otorgará a la aplicación web normal.
5. El servidor de autorización Auth0 redirige al usuario a la aplicación con un **authorization Code**, que sirve para un uso.

6. El SDK de Auth0 envía este código al servidor de autorización de Auth0 (/oauth/token endpoint) junto con el **Client ID** y el **Client secret** de la aplicación.
7. Su servidor de autorización Auth0 verifica el código, el **Client ID** y el **Client secret** del cliente.
8. El servidor de autorización Auth0 responde con un **ID Token** y un **access token** (y, opcionalmente, un token de actualización).
9. Nuestra aplicación puede usar el **access token** para llamar a una API para acceder a información sobre el usuario.
10. La API responde con los datos solicitados.



## Módulo de autenticación social

Python Social Auth tiene como objetivo ser un mecanismo de autenticación y autorización social fácil de configurar para proyectos de Python que admiten protocolos como OAuth 1 y 2 (que es el que usa Auth0), y otros. Nos proporciona el registro de usuario y el inicio de sesión utilizando las credenciales de diversas redes sociales, como Github, Facebook, Google, Twitter, y un largo etc.. Entre los varios frameworks que soporta está Django. Esto es

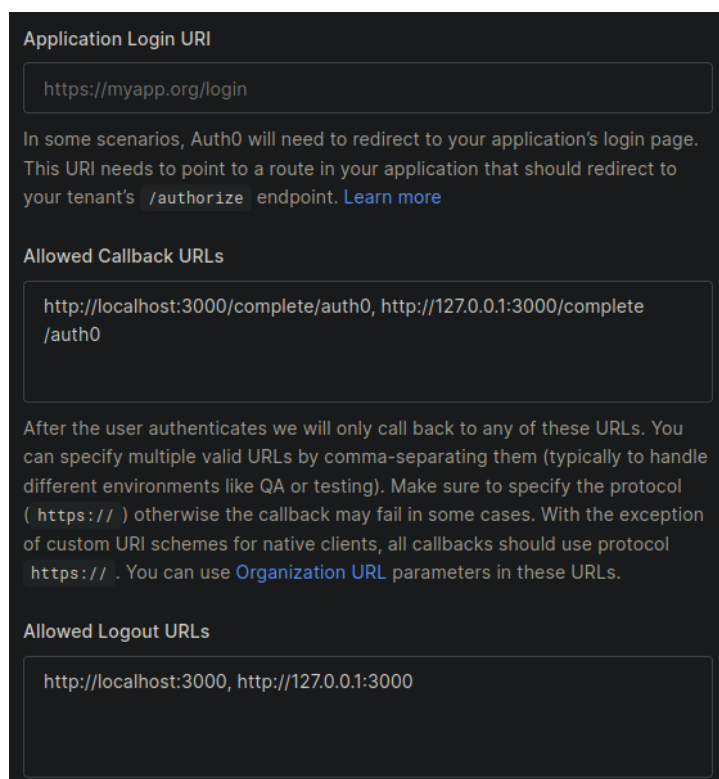
esperado considerando que Python Social Auth es realmente un derivado de Django Social Auth.

Es un proyecto que hoy en día está separado en varios módulos para hacerlo más flexible. Con Auth0, lo que nos interesa a nosotros es social-auth-app-django.

## Configuración en Auth0

Si bien esta aplicación puede utilizar la configuración actual en Auth0, o bien seguir la documentación del servicio para desplegar una API Proxy para OAuth2, se desarrollarán algunas características importantes a considerar.

- URIs



The screenshot shows the Auth0 configuration interface. It has a dark background with light text. The 'Application Login URI' section has a text input field containing 'https://myapp.org/login'. Below it is a paragraph explaining that this URI points to the application's login page. The 'Allowed Callback URLs' section has a text input field containing 'http://localhost:3000/complete/auth0, http://127.0.0.1:3000/complete/auth0'. Below it is a paragraph explaining that these are the URLs where Auth0 will redirect after authentication. The 'Allowed Logout URLs' section has a text input field containing 'http://localhost:3000, http://127.0.0.1:3000'.

Application Login URI

In some scenarios, Auth0 will need to redirect to your application's login page. This URI needs to point to a route in your application that should redirect to your tenant's `/authorize` endpoint. [Learn more](#)

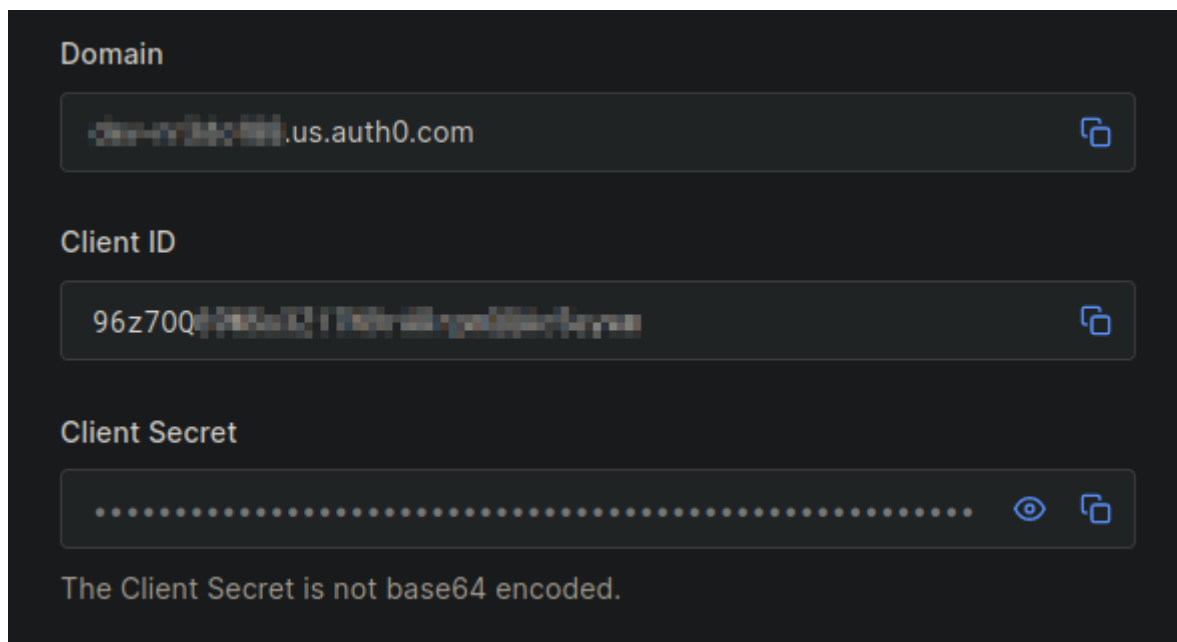
Allowed Callback URLs

After the user authenticates we will only call back to any of these URLs. You can specify multiple valid URLs by comma-separating them (typically to handle different environments like QA or testing). Make sure to specify the protocol ( `https://` ) otherwise the callback may fail in some cases. With the exception of custom URI schemes for native clients, all callbacks should use protocol `https://` . You can use [Organization URL](#) parameters in these URLs.

Allowed Logout URLs

Para que Auth0 reconozca el parámetro ***redirect\_uri***, deben configurarse las URIs válidas como se indica arriba. Para la integración con Auth0, solo es necesario indicar las llamadas Callback URLs (a dónde se hará un REDIRECT 302 al finalizar de forma exitosa el proceso de autenticación) y Logout URLs (a donde se hará el REDIRECT 302 una vez finalizado el cierre de sesión). Como la aplicación puede ejecutarse de forma local usando el hostname ***localhost*** o la IP reservada 127.0.0.1, deben configurarse ambas para evitar inconvenientes.

- Credenciales de la aplicación en Auth0



The screenshot shows the Auth0 application settings interface. It has a dark theme. There are three main input fields: 'Domain' with the value 'dev-123456.us.auth0.com', 'Client ID' with the value '96z70Q...', and 'Client Secret' which is masked with dots. Each field has a copy icon to its right. Below the 'Client Secret' field, there is a message: 'The Client Secret is not base64 encoded.'

La aplicación proxy en Auth0 cuenta con las credenciales requeridas por la aplicación en Django al momento de establecer la autenticidad y autorización de la misma. Estas credenciales son cargadas por el Social Backend al momento de establecer la conexión con Auth0.

También establece un dominio asociado a la aplicación, el cuál deberá ser indicado en la configuración de Django. Si bien estos datos deben suministrarse en settings.py, se declaran variables de entorno para poder cambiar fácilmente los datos en caso de querer usar otra aplicación proxy, y no exponer información sensible en el repo. A modo de ejemplo, debe crearse localmente un archivo .env que defina las siguientes variables:

`<dominio-de-app>`

- Social Connections con proveedores de identidad

De forma análoga debe establecerse la identidad de la aplicación Auth0 como del proveedor de Social Login. Ejemplo con GitHub:

**Name**

github

If you are triggering a login manually, this is the identifier you would use on the connection parameter.

**Client ID**

1cd9301177494140a089013052e78f5a

[How to obtain a Client ID?](#)

**Client Secret**

.....


For security purposes, we don't show your existing Client Secret.

Estos parámetros serán registrados en la sección correspondiente de OAuth Apps del proveedor de identidad.

**Client ID**

1cd9301177494140a089013052e78f5a

**Client secrets** [Generate a new client secret](#)



Client secret

\*\*\*\*\*a769cb4b

Added 11 days ago by rcastro33

Last used within the last week


You cannot delete the only client secret. Generate a new client secret first.

Delete

Para cada Social Connection deberán configurarse estos parámetros, permitiendo así al usuario más medios válidos de autenticación.

**Social Connections** [+ Create Connection](#)

Configure social connections like Facebook, Twitter, Github and others so that you can let your users login with them. [Learn more](#)



github  
GitHub

1 Application enabled

...

# Requerimientos

- Python 3.8
  - Windows: <https://www.python.org/downloads/>
  - Linux: `$ sudo apt install python3.8`
- Pip
  - Windows:  
`$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`  
`$ py get-pip.py`
  - Linux: `$ sudo apt install python3-pip`
- Pipenv o virtualenv:
  - `$ pip install pipenv`
  - `$ pip install virtualenv`

# Instalación y ejecución

Para ejecutar la aplicación, se debe clonar el repositorio desde Github en la carpeta deseada

```
$ git clone https://github.com/rcastro33/django-oauth.git  
$ cd /django-oauth
```

Crear un entorno virtual de Python 3.8 y activar el entorno (ejemplo con pipenv)

```
$ pipenv shell  
$ cd /django-oauth
```

Instalar dependencias, e inicializar la Base de Datos

```
$ pip install -r requirements.txt  
$ py manage.py migrate
```

Ejecutar el servidor (se ha configurado para ejecutar en el puerto 3000 de localhost)

```
$ py manage.py runserver 3000
```

## Snapshot Login



DjanGoRide Login

Ejemplo de Login con Django

**INICIAR SESIÓN CON OAUTH2**