

# PROYECTO DE SOFTWARE

Cursada 2024

# TEMARIO

- Identificación, Autenticación y Autorización
- Estándares:
  - OAuth
  - OpenID Connect

# ¿API?

- Interfaz de programación de aplicaciones (IPA) o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- Ejemplo: librerías del sistema operativo

# ¿QUÉ NOS PERMITEN LAS APIS?

- En general:
  - Intercambiar datos con un tercero.
  - Aprovechar el software y/o capacidad de procesamiento y almacenamiento de terceros para utilizarlo en nuestro sistema pero sin necesariamente incluirlo en nuestro desarrollo sino sólo invocándolo.
  - Los cambios en lo que está en la capa de atrás de la API no nos afectan.
  - Por ej. si la API define una función **listarDatos**, desde nuestro desarrollo no nos afecta que la implementación de esa función cambie de usar un while a un for siempre que devuelva lo que esperamos.

**¿IDENTIFICACIÓN? ¿AUTORIZACIÓN? ¿AUNTENTICACIÓN?**

# CONCEPTOS

- **Identificación**: es una secuencia de caracteres que **identifica** unívocamente al **usuario**: nombre de usuario.
- **Autenticación**: es la **verificación** que realiza el sistema sobre la **identificación**.  
Se puede realizar a través de:
  - **Algo que se conoce**: clave de acceso
  - **Algo que se posee**: tokens / tarjeta
  - **Algo que se es**: huella digital, iris, retina, voz
- **Autorización**: son los **permisos asociados** al usuario autenticado.

## EN EL TRABAJO DE LA CÁTEDRA:

- ¿Identificador? -> **Id de usuario**
- ¿Con qué nos autenticamos? -> **contraseña**
- ¿Quién nos autentica? -> **nuestra aplicación flask**

## **EN EL TRABAJO DE LA CÁTEDRA:**

**¿Quién nos autoriza?: nuestra app flask**

**¿Qué autorización podemos tener?: dependerá del rol del usuario**



## **EN EL TRABAJO DE LA CÁTEDRA:**

### **¿CÓMO LO CONTROLAMOS?**

- ¿Identificador? -> Id de Usuario
- ¿Con qué nos autenticamos? -> contraseña
- ¿Quién nos autentica? -> nuestra aplicación flask
- ¿Quién nos autoriza? -> nuestra aplicación flask
- ¿Qué autorización podemos tener? -> dependerá del rol del usuario

## ESCENARIO

- Usuario introduce identificador y contraseña en el browser.
- Aplicación flask valida e inicia sesión o no.
- Devuelve al usuario la vista que se corresponde con la autorización definida.

# **LIMITACIONES DEL ESQUEMA**

## **Varios sistemas:**

- Múltiples pares usuario/contraseña, uno por cada sistema.
- Se dificulta la interacción entre los sistemas, no hay confianza.
- En cada sistema nuevo se debe redesarrollar la autenticación y todo lo relacionado, por ejemplo: recuperación de contraseña.

## ALTERNATIVAS DISPONIBLES

- SAML
- OAuth
- OpenID Connect

# JWT: JSON WEB TOKEN

- JSON Web Token (JWT) es un estándar abierto.
- Define una forma para transmitir información de forma segura entre las partes como un objeto JSON.
- Utiliza firma digital para garantizar confiabilidad en la información transmitida.
- Está definido en la **RFC 7519**
- **JWT Handbook**

# ESTRUCTURA DEL TOKEN

**header.payload.signature**

- Por ejemplo, tiene la siguiente forma:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjYWRtaW4iOnRy

TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

# EL ENCABEZADO

Ejemplo:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

- Es un objeto JSON.
- Incluye algoritmo de firma ("**alg**") y tipo del token ("**typ**").
- "alg" es obligatorio.
- Si el token no requiere encriptación: **alg: None**

# LOS DATOS

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

- Los datos que nos interesan van en el **payload**.
- También es un objeto JSON.
- No hay elementos obligatorios, aunque sí algunos con un significado específico.
- Por ejemplo: **"sub"**, que es un valor único que identifica una de las partes.
- **+ Info**



## LA FIRMA

- Permite establecer la autenticidad del JWT: es decir que **los datos contenidos en el JWT no han sido alterados**.

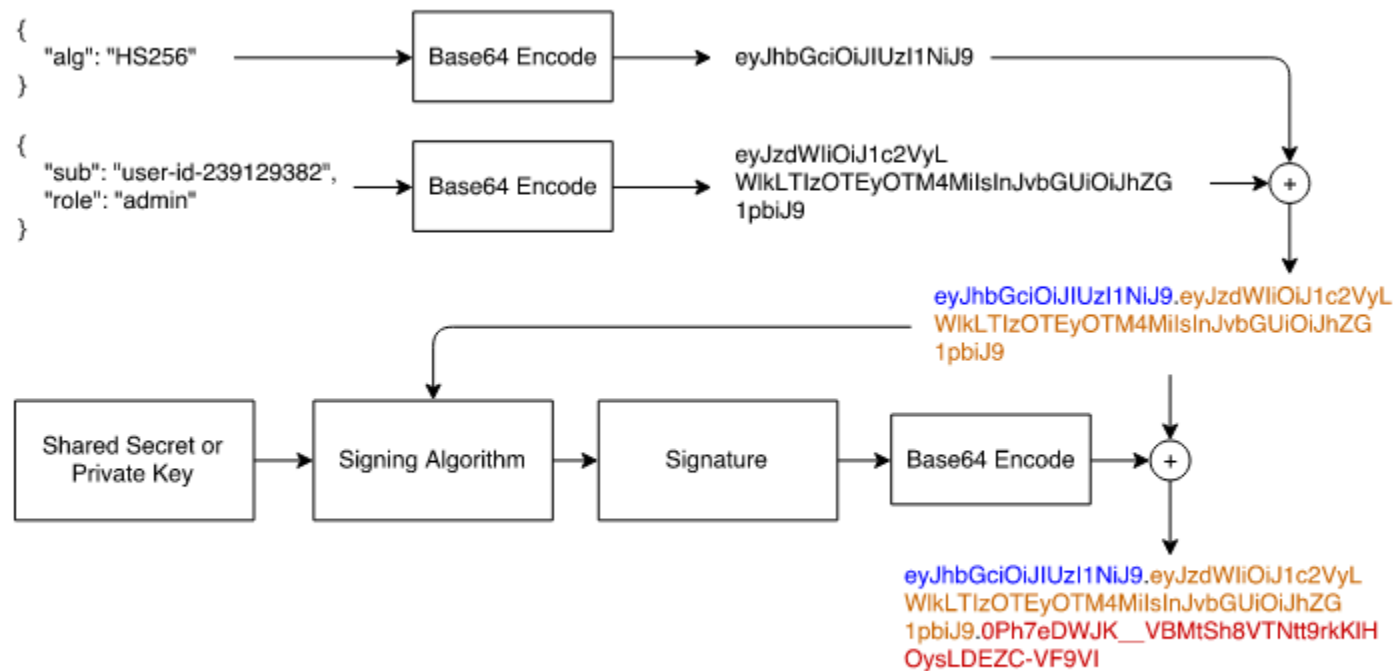


Imagen sacada de "JWT Handbook", <https://auth0.com/resources/ebooks/jwt-handbook>

## ¿USAMOS JWT?

- ¿Para qué necesitábamos esto?
- Proporciona una manera de transmitir información del cliente al servidor de forma segura y sin estado.

## UN EJEMPLO

<https://realpython.com/token-based-authentication-with-flask/>

# SAML

- Site AML



- Security Assertion Markup Language (SAML).
- Basado en XML.
- Este protocolo sirve de base para algunos sistemas propietarios de single-sign-on, pero no es utilizado por los grandes proveedores de servicios en Internet.

# OAuth

- El protocolo OAuth, es un protocolo de autorización, más exactamente, de delegación de acceso.
- Es decir, permite definir cómo un tercero va a acceder a los recursos propios.



## **OAUTH - FUNCIONAMIENTO**

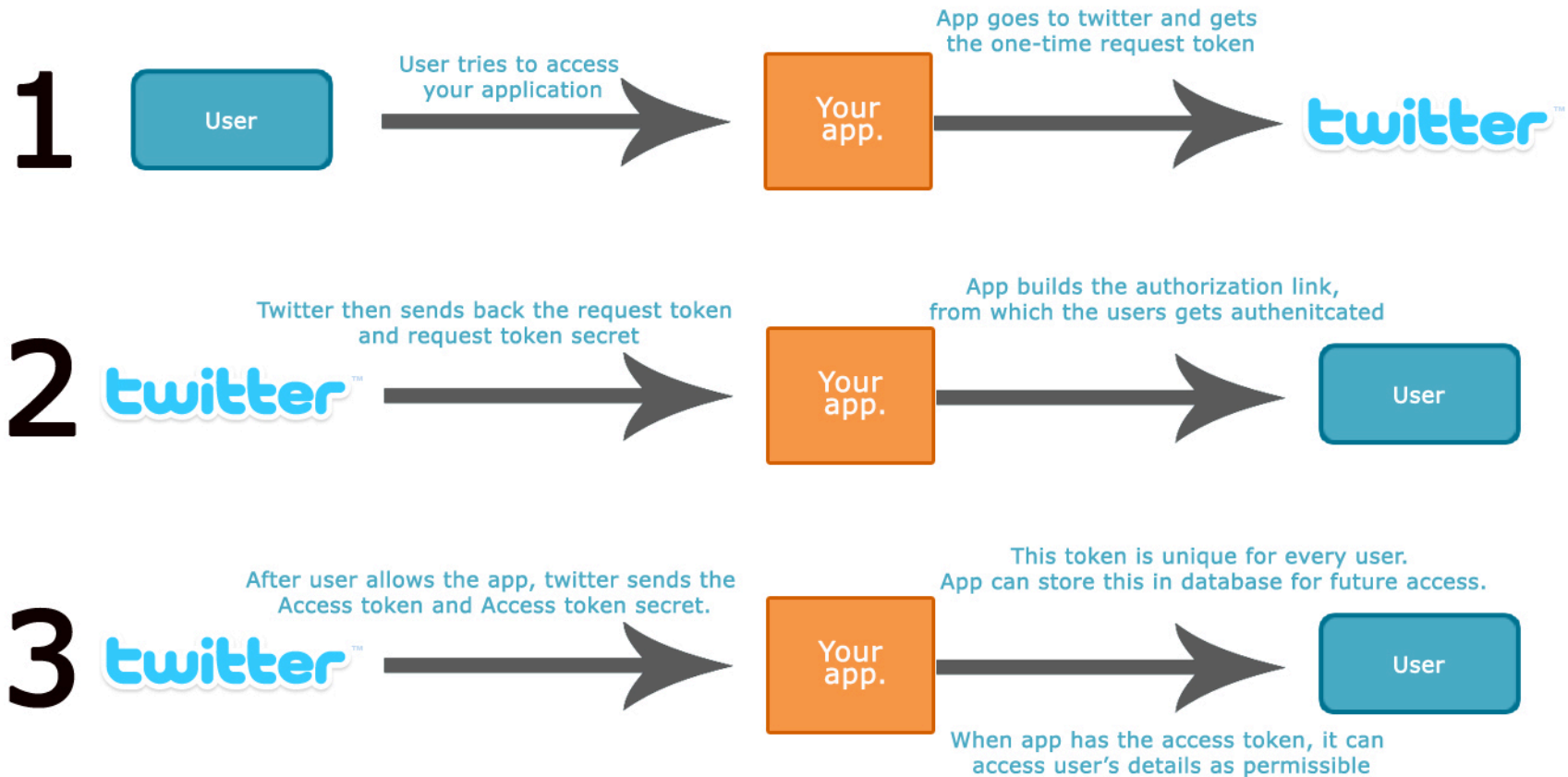
- El usuario dispone de una serie de recursos propios en un servidor (el “proveedor”).
- Un servidor externo (el “consumidor”) desea acceder a un subconjunto de esos recursos.
- El consumidor redirige al usuario hacia el proveedor.
- El usuario se autentica en el proveedor (si no lo estaba previamente).
- El proveedor pregunta al usuario si autoriza al consumidor a que utilice esos determinados recursos.
- El usuario autoriza al consumidor a utilizar esos recursos.
- El servidor externo (consumidor) consigue acceso a esos recursos.

# **OAUTH – ¿QUIÉNES PARTICIPAN?**

- Amazon
- Dailymotion
- Twitter
- Twitch
- Google
- Facebook
- Instagram
- GitHub
- LinkedIn
- PayPal
- y muchos más...



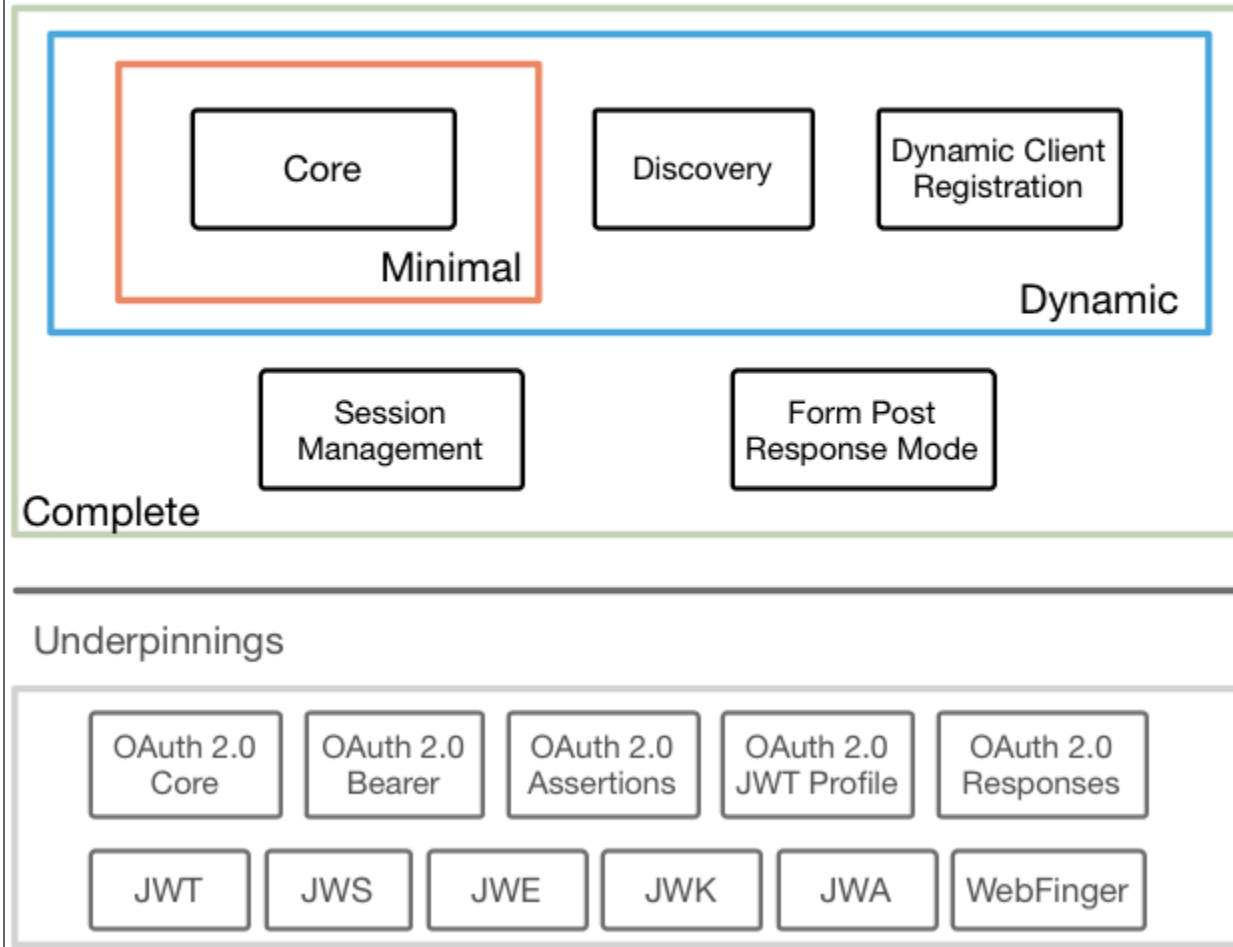
# OAUTH - FUNCIONAMIENTO



## ¿QUÉ ES OPENID CONNECT?

- **OpenID Connect** es una capa de identificación construida sobre OAuth 2.0.
- Permite al cliente verificar la identidad del usuario final basándose en la autenticación realizada por el servidor de autorización,
- Facilita además obtener información básica del perfil del usuario final.
- OpenID Connect permite cliente de todo tipo web, mobile, y clientes JavaScript clients.
- Opcionalmente se puede utilizar encriptación, discovery de proveedores OpenID, o manejo de sesión.

**OPENID CONNECT**



## ¿Y NOSOTROS?

- Necesitamos 1ero. que todo poder usar la API y para eso necesitamos las keys (tokens) para el consumidor OAuth, de la API para eso debemos registrar la aplicación.
- Obtener una extensión OAuth para Python.
- Hay varias opciones de **OAuth para Python**
- Usarlo :D

# **OAUTH EN PYTHON: SEGUIMOS CON FLASK**

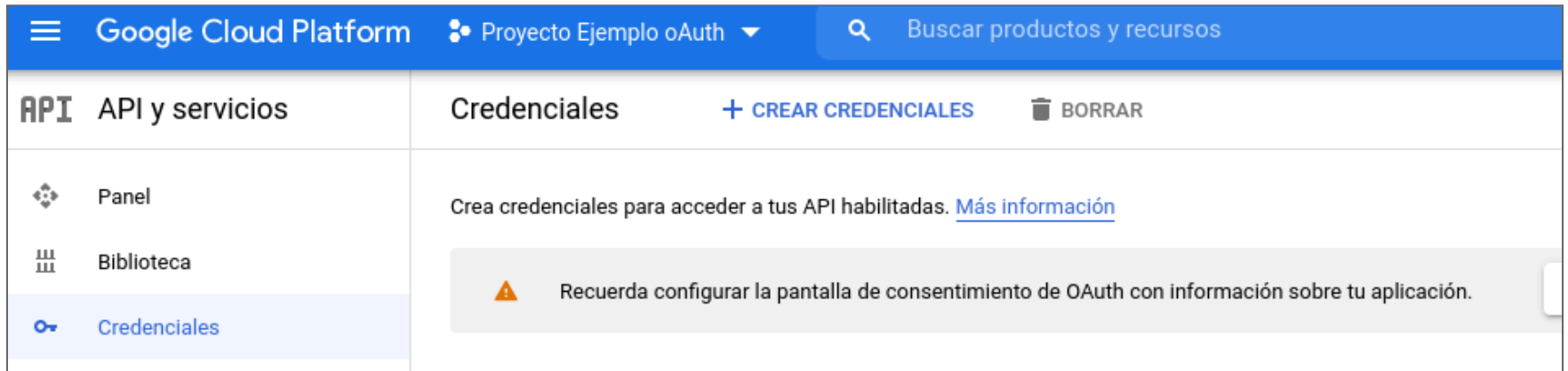
- **AuthLib**
- "Reemplaza" a **Flask-OAuthlib** y **Flask-OAuth** para **el cliente**.
- Permite crear **proveedores del servicio**.
- Algunos Ejemplos:
- **<https://github.com/authlib/demo-oauth-client/blob/master/flask-google-login/app.py>**
- **<https://realpython.com/flask-google-login/#oauthlib>**
- **<https://github.com/lepture/flask-oauthlib/blob/master/example/twitter.py>**

## EJEMPLO – OAUTH + GOOGLE

- Necesitamos 1ero. que todo poder usar la API y para eso necesitamos las keys (tokens) para el consumidor OAuth, de la API para eso debemos registrar la aplicación.
- Obtener una extensión OAuth para python
- Usarlo :D

# EJEMPLO - OAUTH + GOOGLE

- Para obtener una API key y una Secret Key debemos registrar la aplicación que las utilizará en <https://console.developers.google.com/apis/credentials>
- Primero creamos un proyecto



The screenshot shows the Google Cloud Platform console interface. At the top, there's a blue header with the Google Cloud Platform logo, the project name 'Proyecto Ejemplo OAuth', and a search bar. Below the header, the left sidebar shows 'API y servicios' with options for 'Panel', 'Biblioteca', and 'Credenciales' (which is selected). The main content area is titled 'Credenciales' and features a '+ CREAR CREDENCIALES' button and a 'BORRAR' button. Below this, there's a message: 'Crea credenciales para acceder a tus API habilitadas. [Más información](#)'. At the bottom, a warning banner states: 'Recuerda configurar la pantalla de consentimiento de OAuth con información sobre tu aplicación.'



# EJEMPLO - OAUTH + GOOGLE

## Seleccionar un proyecto



PROYECTO NUEVO

🔍 Buscar en proyectos y carpetas

RECIENTES

DESTACADOS

TODOS

Nombre

ID



Proyecto Ejemplo oAuth ?

proyecto-ejemplo-oauth

# EJEMPLO - OAUTH + GOOGLE

## Credenciales

[+ CREAR CREDENCIALES](#) BORRAR

Crea credenciales para acceder a tus API habilitadas. [Más información](#)



Recuerda configurar la pantalla de consentimiento de OAuth con información sobre tu aplicación.

[CONFIGURAR PANTALLA DE CONSENTIMIENTO](#)

## Claves de API

<input type="checkbox"/>	Nombre	Fecha de creación ↓	Restricciones	Clave	Acciones
No hay claves de API para mostrar					

## ID de clientes OAuth 2.0

<input type="checkbox"/>	Nombre	Fecha de creación ↓	Tipo	ID de cliente	Acciones
No hay clientes de OAuth para mostrar					

## Cuentas de servicio

[Administrar cuentas de servicio](#)

<input type="checkbox"/>	Correo electrónico	Nombre ↑	Acciones
No hay cuentas de servicio para mostrar			

# EJEMPLO – OAUTH + GOOGLE



## Crear ID de cliente de OAuth

Un ID de cliente se usa con el fin de identificar una sola app para los servidores de OAuth de Google. Si la app se ejecuta en varias plataformas, cada una necesitará su propio ID de cliente. Consulta [Configura OAuth 2.0](#) para obtener más información. [Obtén más información](#) sobre los tipos de clientes de OAuth.



Para crear un ID de cliente de OAuth, primero debes configurar la pantalla de consentimiento

[CONFIGURAR PANTALLA DE CONSENTIMIENTO](#)

# EJEMPLO - OAUTH + GOOGLE

## Dominios autorizados

Cuando un dominio se usa en la pantalla de consentimiento o en la configuración del cliente de OAuth, debe contar con un registro previo aquí. Si debes verificar la app, ve [Google Search Console](#) para comprobar si tus dominios están autorizados. [Más información](#) sobre el límite de dominios autorizados.

[+ AGREGAR UN DOMINIO](#)

# EJEMPLO – OAUTH + GOOGLE



## Crear ID de cliente de OAuth

Un ID de cliente se usa con el fin de identificar una sola app para los servidores de OAuth de Google. Si la app se ejecuta en varias plataformas, cada una necesitará su propio ID de cliente. Consulta [Configura OAuth 2.0](#) para obtener más información. [Obtén más información](#) sobre los tipos de clientes de OAuth.

Tipo de aplicación \*

Aplicación web



Nombre \*

Flask Local

El nombre de tu cliente de OAuth 2.0. Este nombre solo se usa para identificar al cliente en la consola y no se mostrará a los usuarios finales.



Los dominios de los URI que agregues a continuación se incorporarán automáticamente a tu [pantalla de consentimiento de OAuth](#) como [dominios autorizados](#).

# EJEMPLO - OAUTH + GOOGLE

## Orígenes autorizados de JavaScript ⓘ

Para usar con solicitudes de un navegador

URI \*

[+ AGREGAR URI](#)

## URI de redireccionamiento autorizados ⓘ

Para usar con solicitudes de un servidor web

URI \*

[+ AGREGAR URI](#)

# EJEMPLO – OAUTH + GOOGLE

## Se creó el cliente de OAuth

Puedes acceder al ID de cliente y el secreto desde “Credenciales” en API y servicios



El acceso OAuth está restringido a los [usuarios de prueba](#) que aparecen en la [pantalla de consentimiento de OAuth](#)

## EJEMPLO OAUTH

- Vamos a necesitar un certificado ssl para poder probar.
- Flask nos brinda una solución para desarrollo: `ssl_context="adhoc"`
- Instalar:
- Authlib, requests y pyOpenSSL.

```
poetry add AuthLib
poetry add requests
poetry add pyOpenSSL

poetry run flask run --cert adhoc
```



# EJEMPLO OAUTH

```
from src.web import create_app
from pathlib import Path

static_folder =
Path(__file__).parent.absolute().joinpath("public")

app = create_app(env="development", static_folder=static_folder)

def main():
    app.run()

if __name__ == "__main__":
    main()
```

# EJEMPLO OAUTH

```
## config.py
...

class DevelopmentConfig(Config):
    """Development configuration."""

    DB_SERVER = environ.get("DB_SERVER", "localhost")
    DB_DATABASE = environ.get("DB_DATABASE", "proyecto_db")
    DB_USER = environ.get("DB_USER", "proyecto_db")
    DB_PASSWORD = environ.get("DB_PASSWORD", "proyecto_db")
    DB_PORT = environ.get("DB_PORT", "5432")
    SQLALCHEMY_DATABASE_URI = f"postgresql+psycopg2://{DB_USER}:"
    {DB_PASSWORD}@{DB_SERVER}:{DB_PORT}/{DB_DATABASE}"
    SQLALCHEMY_TRACK_MODIFICATIONS = True
    GOOGLE_CLIENT_ID = os.getenv('GOOGLE_CLIENT_ID')
    GOOGLE_CLIENT_SECRET = os.getenv('GOOGLE_CLIENT_SECRET')

...
```

# EJEMPLO OAUTH

```
# __init__.py
from flask import Flask, url_for, render_template, session,
redirect
from os import environ, urandom
from src.web.config import config
from authlib.integrations.flask_client import OAuth

def create_app(env="development", static_folder="static"):
    app = Flask(__name__, static_folder=static_folder)

    app.secret_key = environ.get("SECRET_KEY") or urandom(24)
    app.config.from_object(config[env])

    CONF_URL = 'https://accounts.google.com/.well-known/openid-
configuration'
    oauth = OAuth(app)
    oauth.register(
        name='google',
        server_metadata_url=CONF_URL,
        client_kwargs={
            'scope': 'openid email profile'
        }
    )

    @app.route('/')
    def homepage():
        user = session.get('user')
```

```
        return render_template('index.html', user=user)

@app.route('/login')
def login():
    redirect_uri = url_for('auth', _external=True)
    return oauth.google.authorize_redirect(redirect_uri)

@app.route('/login/callback')
def auth():
    token = oauth.google.authorize_access_token()
    session['user'] = token['userinfo']
    return redirect('/')

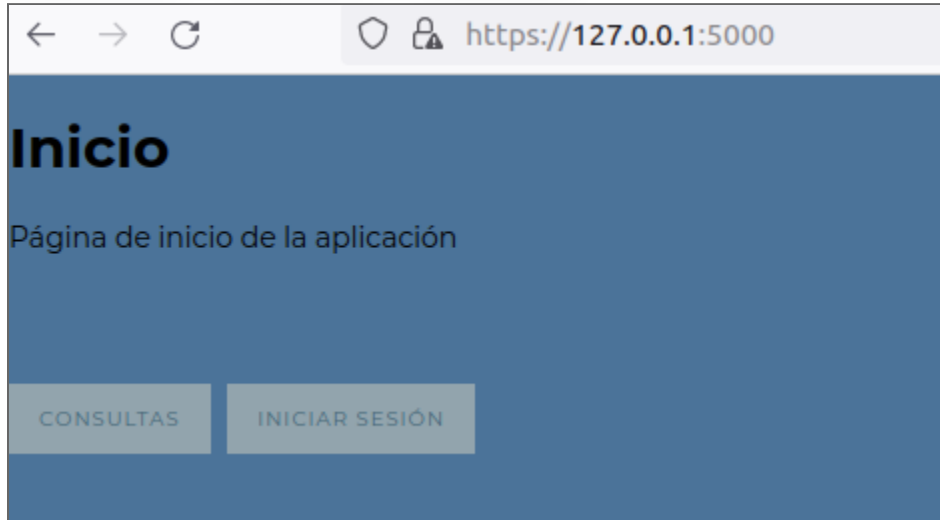
@app.route('/logout')
def logout():
    session.pop('user', None)
    return redirect('/')

return app
```

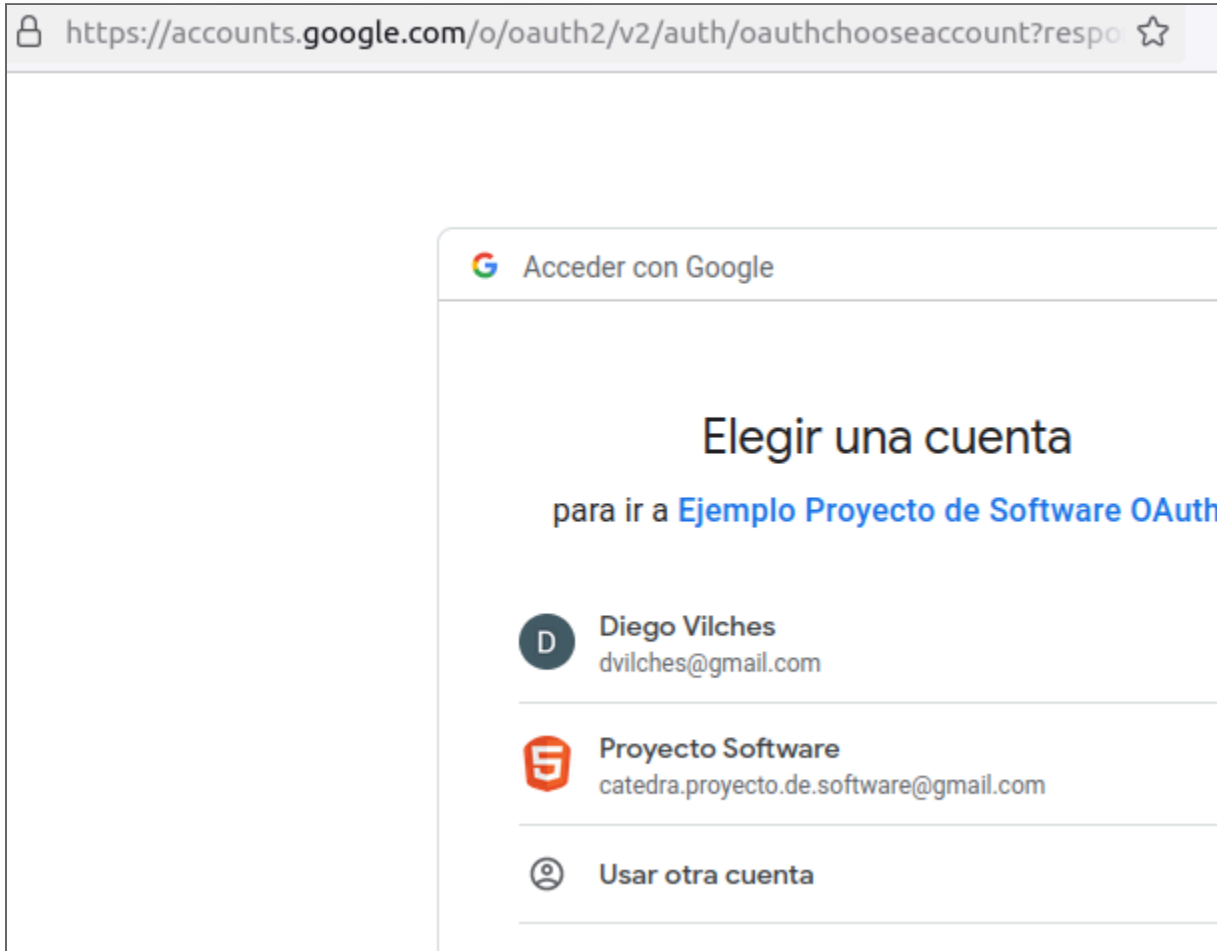
# EJEMPLO OAUTH

```
{% extends "layout.html" %}
{% block title %}Ejemplo Proyecto{% endblock %}
{% block head %}
    {{ super() }}
{% endblock %}
{% block content %}
{% if user %}
<pre>
{{ user.email }}
</pre>
<a href="/logout">logout</a>
{% else %}
<a href="/login">login</a>
{% endif %}
{% endblock %}
```

# EJEMPLO - OAUTH + GOOGLE



# EJEMPLO - OAUTH + GOOGLE



# EJEMPLO - OAUTH + GOOGLE





## EJEMPLO 2 - OAUTH + GOOGLE

- Fuente: Real Python
- Vamos a necesitar un certificado ssl para poder probar.
- Flask nos brinda una solución para desarrollo: `ssl_context="adhoc"`

```
# run.py
from app import create_app

if __name__ == "__main__":
    app = create_app()
    app.run(ssl_context="adhoc")
```

## EJEMPLO 2 - OAUTH + GOOGLE

- Recordar definir las rutas

```
# __init__.py

def create_app(environment="development"):
    app.secret_key = environ.get("SECRET_KEY") or urandom(24)

    # OAuth 2 client setup
    client = WebApplicationClient(GOOGLE_CLIENT_ID)

    login_manager = LoginManager()
    login_manager.init_app(app)

    app.add_url_rule(
        "/autenticacion", "auth_authenticate", auth.authenticate,
        methods=["POST"]
    )
    app.add_url_rule(
        "/login/callback", "auth_callback", auth.callback,
        methods=["GET"]
    )
    app.add_url_rule(
        "/login/bienvenido", "auth_bienvenido", auth.bienvenido,
```

```
methods=["GET"]  
)
```

## EJEMPLO 2 - OAUTH + GOOGLE

- Utilizar los valores client id y client secret generados en la consola de google.
- En este caso lo pasamos desde el entorno

```
# auth.py

GOOGLE_CLIENT_ID = environ.get("GOOGLE_CLIENT_ID", None)
GOOGLE_CLIENT_SECRET = environ.get("GOOGLE_CLIENT_SECRET", None)
GOOGLE_DISCOVERY_URL = (
    "https://accounts.google.com/.well-known/openid-configuration"
)

def get_google_provider_cfg():
    return requests.get(GOOGLE_DISCOVERY_URL).json()

client = WebApplicationClient(GOOGLE_CLIENT_ID)
```

## EJEMPLO 2 - OAUTH + GOOGLE

- Definimos los métodos que "atienden" las nuevas rutas

```
# auth.py

def login():
    # Find out what URL to hit for Google login
    google_provider_cfg = get_google_provider_cfg()
    authorization_endpoint =
google_provider_cfg["authorization_endpoint"]

    # Use library to construct the request for Google login and
    provide
    # scopes that let you retrieve user's profile from Google
    request_uri = client.prepare_request_uri(
        authorization_endpoint,
        redirect_uri="https://127.0.0.1:5000/login/callback",
        scope=["openid", "email", "profile"],
    )
    return redirect(request_uri)
```

## EJEMPLO 2 - OAUTH + GOOGLE

- Definimos los métodos que "atienden" las nuevas rutas

```
# auth.py

def callback():
    # Get authorization code Google sent back to you
    code = request.args.get("code")

    google_provider_cfg = get_google_provider_cfg()
    token_endpoint = google_provider_cfg["token_endpoint"]
    token_url, headers, body = client.prepare_token_request(
        token_endpoint,
        authorization_response=request.url,
        redirect_url=request.base_url,
        code=code
    )
    token_response = requests.post(
        token_url,
        headers=headers,
        data=body,
        auth=(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET),
    )

    # Parse the tokens!

    client.parse_request_body_response(json.dumps(token_response.json()))

    userinfo_endpoint = google_provider_cfg["userinfo_endpoint"]
    uri, headers, body = client.add_token(userinfo_endpoint)
```

```
userinfo_response = requests.get(uri, headers=headers,  
data=body)  
if userinfo_response.json().get("email_verified"):  
    unique_id = userinfo_response.json()["sub"]  
    users_email = userinfo_response.json()["email"]  
    picture = userinfo_response.json()["picture"]  
    users_name = userinfo_response.json()["given_name"]  
else:  
    return "User email not available or not verified by  
Google.", 400  
  
    # Create a user in your db with the information provided  
    # by Google  
    user = User(  
        id=unique_id, name=users_name, email=users_email,  
profile_pic=picture  
    )  
  
    # Doesn't exist? Add it to the database.  
    print (userinfo_response.json())  
    if not User.get(unique_id):  
        User.create(unique_id, users_name, users_email, picture)  
  
    # Begin user session by logging the user in  
    login_user(user)  
  
    # Send user back to homepage  
    return redirect(url_for("auth_bienvenido"))
```

## EJEMPLO 2 - OAUTH + GOOGLE

- Definimos los métodos que "atienden" las nuevas rutas

```
# auth.py  
  
def bienvenido():  
    return render_template("auth/bienvenido.html")
```



# REFERENCIAS

- OpenID:
  - <http://openid.net/>
  - <http://openidexplained.com/>
- OAuth:
  - <http://oauth.net/>
  - <https://oauth.net/code/python/>
  - <http://wiki.oauth.net/>
  - <http://es.scribd.com/doc/91623356/Entendiendo-oAuth>
  - <http://oauthbible.com/>
- OpenID Connect:
  - <http://openid.net/connect/>
  - <https://www.returngis.net/2019/04/oauth-2-0-openid-connect-y-json-web-tokens-jwt-que-es-que/>
- Flask
  - <https://realpython.com/flask-google-login/>
  - <https://github.com/realpython/materials/tree/master/flask-google-login>

- <https://docs.authlib.org/en/latest/client/flask.html>
- <https://github.com/authlib/demo-oauth-client/tree/master/flask-google-login>
- <https://medium.com/thedevproject/how-to-setup-https-using-flask-for-local-development-b17ea906b231>

Speaker notes