

Guião de apoio 9 Aplicações OAuth 2 em Flask

1. Introdução ao tema

Neste guião exploraremos o protocolo OAuth. Será implementada uma aplicação através da *framework* de desenvolvimento WEB *Flask* com autenticação e autorização OAuth.

2. Introdução ao OAuth 2

O protocolo OAuth utiliza HTTPS e permite a aplicações clientes o acesso a recursos protegidos de utilizadores, sem que seja necessário a obtenção das credenciais do utilizador dono dos recursos. No entanto, é necessário que obtenha a autorização prévia do dono dos recursos. É necessário haver uma terceira-parte, o servidor de autorização, que fará a ponte entre os dois intervenientes, nomeadamente:

- 1) aceitar o pedido da aplicação cliente para acesso aos recursos;
- 2) pedir a autorização ao dono do recurso, obtendo deste o código de autorização (*authorization code*);
- 3) criar o token de acesso com base no código de autorização e a entrega deste à aplicação cliente.

A aplicação cliente acede aos recursos utilizando o token de acesso. Para que todo o processo de OAuth 2 funcione, é necessário que a aplicação seja registada numa API de OAuth (p. ex., Github, Spotify, Google, Facebook, Twitter) para a obtenção de um *Client_ID* e um *Secret_ID* e a indicação do URI para onde será redireccionado o código de autorização.

A listagem a seguir é um exemplo de como conectar uma aplicação cliente em OAuth, registada na API da GitHub. Exemplos semelhantes para outros servidores de autorização podem ser encontrados em <http://requests-oauthlib.readthedocs.io/en/latest/index.html>.

Listagem 1 - Exemplo com OAuth 2

```
from requests_oauthlib import OAuth2Session
import os
os.environ['OAUTHLIB_INSECURE_TRANSPORT'] = '1'

# Credenciais da app cliente registada no Github (https://github.com/settings/applications/new)
client_id = '<o id obtido da github>'
client_secret = '<o secret obtido da github>'

# URIs do github para obtencao do authorization_code, do token, de callback e do recurso protegido
authorization_base_url = 'https://github.com/login/oauth/authorize'
token_url = 'https://github.com/login/oauth/access_token'
redirect_uri = 'http://localhost'
protected_resource = 'https://api.github.com/user'

github = OAuth2Session(client_id, redirect_uri=redirect_uri)
# Pedido do authorization_code ao servidor de autorização (e dono do recurso a aceder)
authorization_url, state = github.authorization_url(authorization_base_url)
print ('Aceder ao link (via browser) para obter a autorizacao,', authorization_url)

# Obter o authorization_code do servidor vindo no URL de redirecionamento
url_response = input('insira o URL devolvido no browser e cole aqui:')

# Obtencao do token
github.fetch_token(token_url, client_secret=client_secret, authorization_response=url_response)

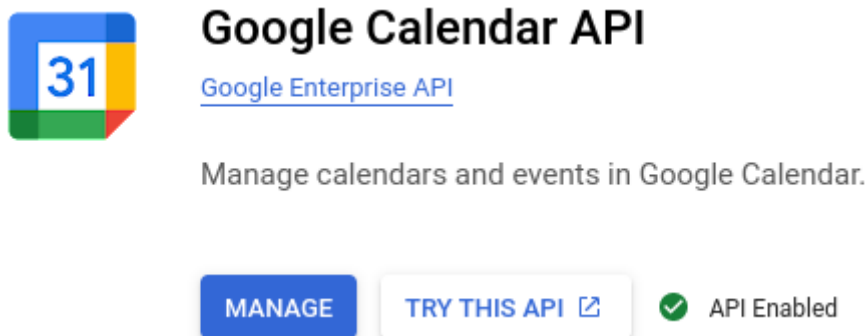
# Acesso a um recurso protegido
r = github.get(protected_resource)
print (r.content.decode())
```

3. Exercícios

1. Registe no GitHub (<https://github.com/settings/applications/new>) uma aplicação para obtenção de um `client_id` e `secret_id`. No registo da aplicação redirecione esta para o `http://localhost`.

2. Copie o programa apresentado na Listagem 1, configure-o com as credenciais que obteve e execute-o. Apoiando-se na documentação sobre o módulo `requests_oauthlib`, deverá entender o papel de algumas funções disponíveis nas classes disponibilizadas pelo módulo (em destaque na listagem). Para além disso, deverá perceber o funcionamento do programa e identificar os passos do protocolo (indicados na introdução).

3. Registe no Google (<https://console.cloud.google.com/projectcreate>) um Projecto (nome: AD 2023), caso ainda não o tenha feito para o projeto da disciplina. Active a API cujas funcionalidades quer utilizar na sua aplicação. Para o efeito, active a API Calendário (<https://console.cloud.google.com/apis/library/calendar-json.googleapis.com>).




Crie agora uma Credencial do tipo OAuth client ID (<https://console.cloud.google.com/apis/credentials/oauthclient?previousPage=%2Fapis%2Fcredentials>) para uma aplicação do tipo “Web Application” com o nome “AD PL09 Client”. Nas configurações da aplicação, acrescente o `http://localhost` como URI de redireccionamento.

Deverá agora criar uma Aplicação usando o menu de Consent Screen (<https://console.cloud.google.com/apis/credentials/consent>).

Dê um nome à sua App “AD 2023 App”, adicione as funcionalidades (scopes) que pretende – Calendar API), e por razões de segurança como as aplicações Google são inicializadas em modo de *Testing*, deverá adicionar todos os endereços de email dos clientes autorizados a utilizar a sua aplicação




Your sensitive scopes

Sensitive scopes are scopes that request access to private user data.

API ↑	Scope	User-facing description	
Google	.../auth	See, create, change, and delete	
Calendar	/calendar	events on Google calendars you	
API	.events	own	
	.owned		

Tem agora a possibilidade de descarregar as credenciais (`client_secret.json`) que criou para permitir o cliente python aceder à sua Aplicação.

OAuth 2.0 Client IDs

<input type="checkbox"/>	Name	Creation date ↓	Type	Client ID	Actions
<input type="checkbox"/>	AD PL09 Client	Apr 17, 2023	Web application	168417782737-3uks...	  

Em vez de variáveis `client_id` e `client_secret`, leia directamente do ficheiro json as credenciais no seu código, assim como mostrado na listagem abaixo.

Listagem 2 - Exemplo com OAuth 2

```
from google_auth_oauthlib.flow import InstalledAppFlow

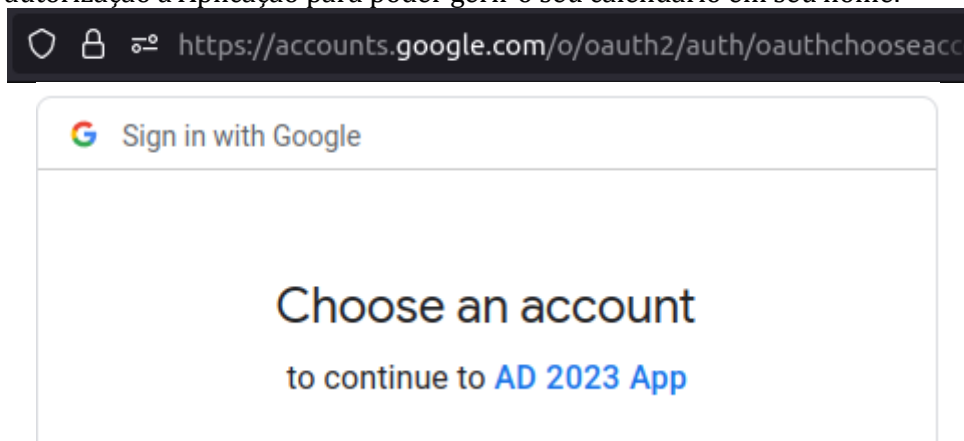
# Configurar o OAuth2 flow para obter consentimento do utilizador
flow = InstalledAppFlow.from_client_secrets_file(
    'client_secret.json',
    scopes=['https://www.googleapis.com/auth/calendar.events'],
    redirect_uri='http://localhost:5000'
)
auth_url, state = flow.authorization_url(prompt='consent')
print('auth_url', auth_url) # Abrir link no Browser

# Redirecionar o utilizador para o URL de autorização e obter o código
# de autorização. Após o utilizador conceder o consentimento, o Google
# redireciona-o para o URI de redirecionamento, com um código de
# autorização (code) como parâmetro na string de consulta.
#                               Exemplo                               de                               URI:
http://localhost:8000/?code=4/0AbUR2VPbIBd5AXL1Rrw&scope=....

# Extrair o código da string de consulta, e trocá-lo por um token de
# acesso.
url_response = input('insira o code do URL devolvido no browser:')
flow.fetch_token(code=url_response)

# Gravar as credenciais em token.json para usar recurso protegido.
with open('token.json', 'w') as token_file:
    token_file.write(flow.credentials.to_json())
```

Acendendo no seu browser ao link apresentado na variável `auth_url`, poderá fazer login com a sua conta google e dar autorização à Aplicação para poder gerir o seu calendário em seu nome.



4. Acrescente no programa anterior as linhas necessárias para convertê-lo para uma aplicação Flask (ver PL07). Para testar se está a funcionar, pode criar um método para o recurso raiz que retorna apenas um 'Hello world!'.

5. Crie um recurso (p. ex., `/login`) com o Flask para que o utilizador possa iniciar o processo de autorização com o OAuth 2 (ver TP09). Este recurso obterá a URL de autorização e **redirecionará** o utilizador para a mesma. O utilizador poderá então autorizar a aplicação a aceder ao recurso protegido.

6. Crie um recurso (p. ex., `/callback`) com o Flask para ser utilizado como endereço de redirecionamento do protocolo de autorização. Este recurso receberá, na URL do request, o código de autorização e estado necessários para pedir o token de acesso automaticamente. Note que a variável `redirect_uri` e a URI de redirecionamento configurada na página do Google precisam mudar para `https://localhost:5000/callback`. Após obter o token de acesso, este recurso deverá

redirecionar o utilizador para um outro recurso (ver o próximo exercício) que acederá ao recurso protegido.

7. Crie um recurso (p. ex., `/profile`) com o Flask para que a aplicação cliente utilize o token de acesso obtido no exercício anterior (`token.json`) para aceder ao recurso protegido no lugar do utilizador. Este recurso protegido pode ser por exemplo adicionar um novo evento ao calendário do utilizador no dia 10 de maio 2023, das 10h00 as 12h00 chamado “Teste 123”.

8. Modifique a aplicação Flask para que ela e o cliente se comuniquem de TLS com autenticação mútua. Para tal, copie e utilize no Flask as chaves e os certificados gerados na PL08.

4. Bibliografia e outro material de apoio

- Flask User’s guide:
<https://flask.palletsprojects.com/>
- Flask API:
<https://flask.palletsprojects.com/en/2.1.x/api/>
- Módulo requests:
<http://docs.python-requests.org/en/master/>
- Protocolo OAuth
<https://oauth.net>
- Módulo requests-oauthlib:
<https://pypi.python.org/pypi/requests-oauthlib>
<http://requests-oauthlib.readthedocs.io/en/latest/index.html>