OAuth Authentication in Django with social-auth

Martín Lamas Follow
May 28, 2018 · 5 min read

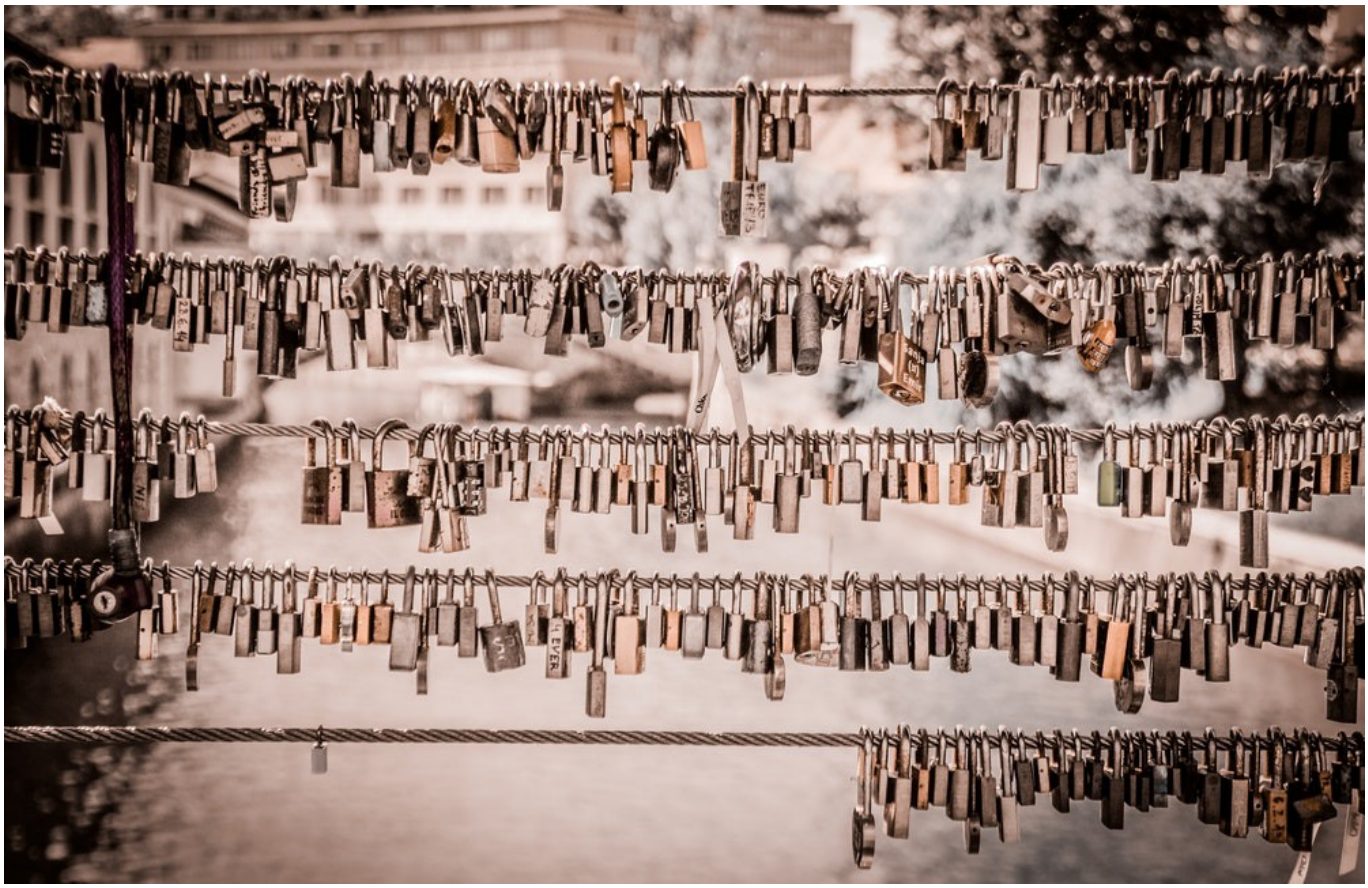


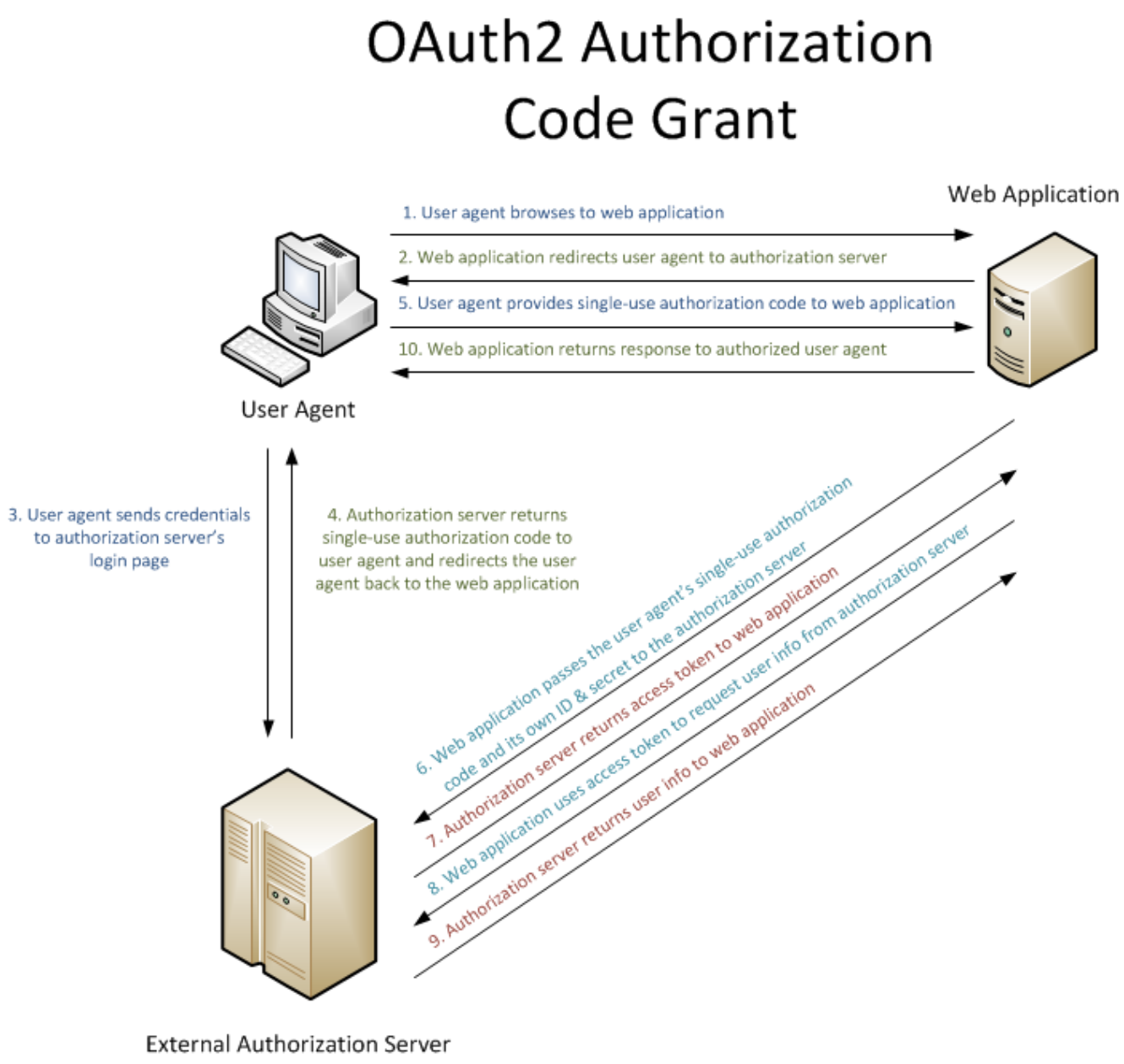
Photo by Rubén Bagüés on Unsplash

During the development of my last django project I had to provide user authentication with Google accounts. To achieve this, I used the social-app-django library that implements an authentication/registration mechanism which supports several auth providers and protocols like OAuth (version 1 and 2) or OpenId.

To provide user authentication Google supports several protocols including OAuth 2.

OAuth is an open protocol that provides secure authorization for web, mobile and desktop applications in a simple and standard way. The protocol relies on a trusted third party to establish the authentication process. It grants client access to a resource delegating the authorization process to an external authorization server with the approval of the resource owner.

In this diagram you can see how it works:



Attribution: stack overflow

In this post, we'll build a simple blog application using the OAuth protocol to provide user authentication with Google accounts.

Let's go!

1. Installing the library

As a first step, we must install and enable the social-auth-app-django library in our django project. We can install this library using either pip or pipenv according to our environment/personal preferences.

If we choose pip:

```
pip install social-auth-app-django
```

or if we use pipenv:

```
pipenv install social-auth-app-django
```

Once the library has been installed, we must add the app to the `INSTALLED_APPS` list in the project settings file using the `social_django` identifier:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
    'social_django',
]
```

settings.py hosted with ❤ by **GitHub** **view raw**

2. Adding the Google OAuth2 authentication backend

As mentioned earlier, we're going to use Google accounts to establish the authentication, so we'll need to add the Google OAuth2 backend to the list of authentication backends. Also, we have to include explicitly the default model authentication backend in order to continue using the django admin site using local accounts:

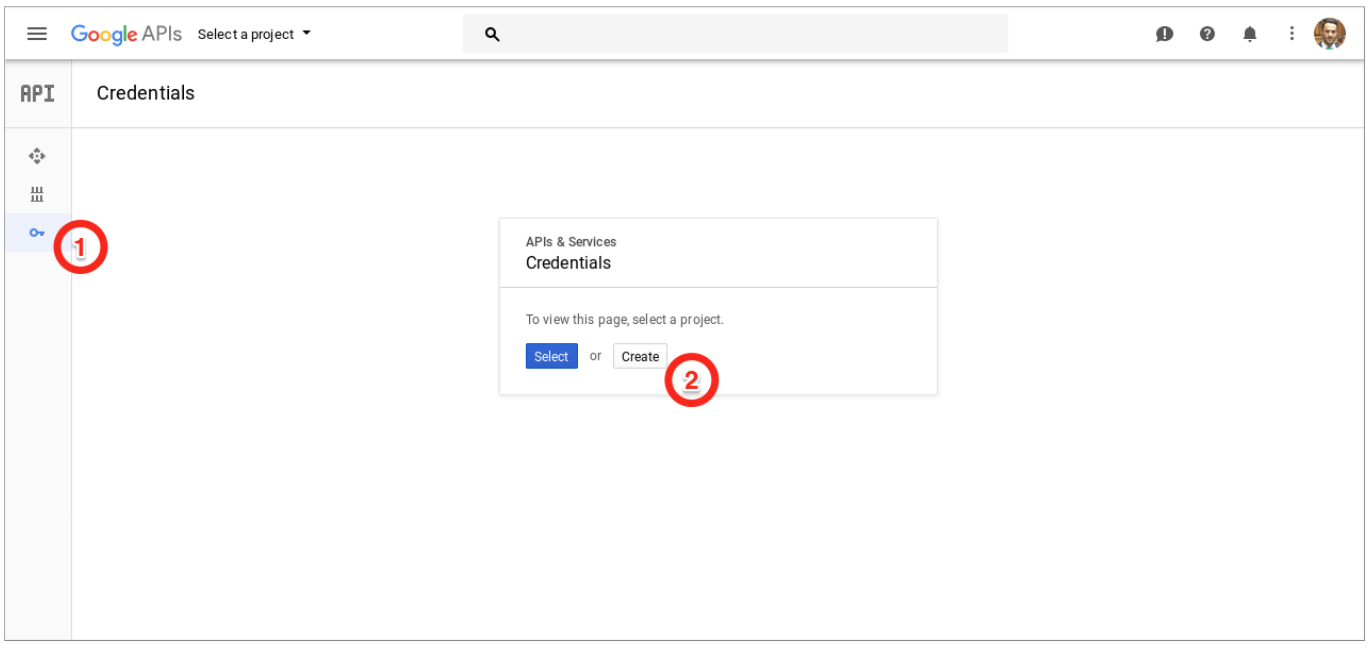
```
AUTHENTICATION_BACKENDS = (
    'social_core.backends.google.GoogleOAuth2',
    'django.contrib.auth.backends.ModelBackend',
)
```

settings.py hosted with ❤ by **GitHub** **view raw**

Adding Google OAuth2 authentication backend

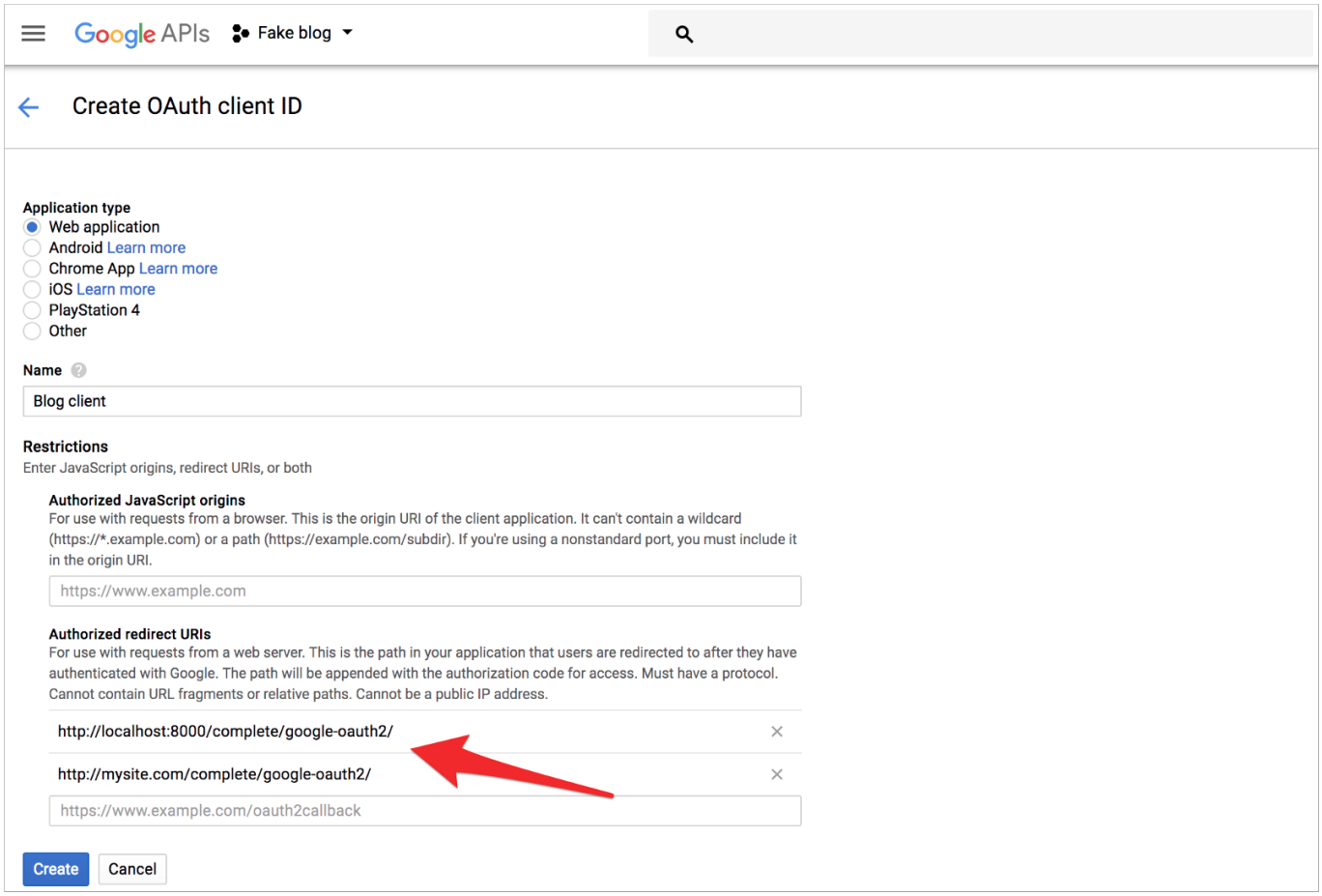
3. Configuring the Google Authentication API

Once we added the backend, we must configure it to be used in the application. To perform this task, we go to the Google Developers Console and, once there, we navigate to the *Credentials* section in the left menu. A dialog will be displayed to either select or create a project. At this time we still don't have any project, so we'll choose the second option to create it.



Now we need to perform the following tasks:

1. Give a descriptive name to the project. It will help us to keep things clean when managing several applications.
2. Create the credentials selecting *OAuth client ID* in the *Create Credentials* menu.
3. Select *Web application* and fill the required data. The *Authorized redirect URIs* are used by the Google Authentication Backend in order to redirect the users to the application again once the authentication is performed.
4. Click on *Create* button and a key and a secret will be generated. We'll need these parameters in the next step.



Setting redirect URIs

Google includes authentication with OAuth 2 through its Google + API, so we need to enable it. Go to *APIs & Services* and click on *Enable APIs and Services* button. Then, search

Enable APIs and services

4. Configuring the project in the backend

Now it's time to configure the authentication backend in our project. We must edit the settings file and initialize the Google OAuth2 key and the secret using the ones obtained in the credentials generation process:

```
1  SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = 'INSERT_PROVIDED_KEY_HERE'
2  SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = 'INSERT_PROVIDED_SECRET_HERE'
```

settings.py hosted with ❤ by **GitHub** **view raw**

Google OAuth2 key and secret configuration

We also need to add the *LOGIN_URL*, *LOGIN_REDIRECT_URL* and *LOGOUT_REDIRECT_URL* keys to the configuration file. The social-app-django library uses the *LOGIN_URL* key to redirect the user to the Google authentication page. *LOGIN_REDIRECT_URL* and *LOGOUT_REDIRECT_URL* set the URLs where the user will be redirected after the login and logout events.

```
1  LOGIN_URL = '/auth/login/google-oauth2/'
2
3  LOGIN_REDIRECT_URL = '/'
4  LOGOUT_REDIRECT_URL = '/'
```

settings.py hosted with ❤ by **GitHub** **view raw**

Configuring login and logout redirect URLs

To finish the configuration process we need to add some routes:

```
urlpatterns = [
  ...,
  path('', include('social_django.urls', namespace='social')),
  path('logout/', logout, {'next_page': settings.LOGOUT_REDIRECT_URL},
    name='logout'),
  ...,
]
```

urls.py hosted with ❤ by **GitHub** **view raw**

Configuring routes

We also need to add the following key to the settings file:

```
SOCIAL_AUTH_URL_NAMESPACE = 'social'
```

settings.py hosted with ❤ by **GitHub** **view raw**

At this point the configuration should be ready. Now, it's time to put some logic in the views and templates!

5. Adding logic in views and templates

As we said earlier, we are building a simple blog application. The default behaviour here is to only show the posts when the user is authenticated. We put the logic to control this behaviour in the view:

```
from django.shortcuts import render
from .models import Post


def index(request):
    context = {
        'posts': Post.objects.order_by('-date')
        if request.user.is_authenticated else []
    }

    return render(request, 'blog/index.html', context)
```

views.py hosted with ❤ by **GitHub** **view raw**

As you can see, the view only sends the posts list to the template when the user is authenticated.

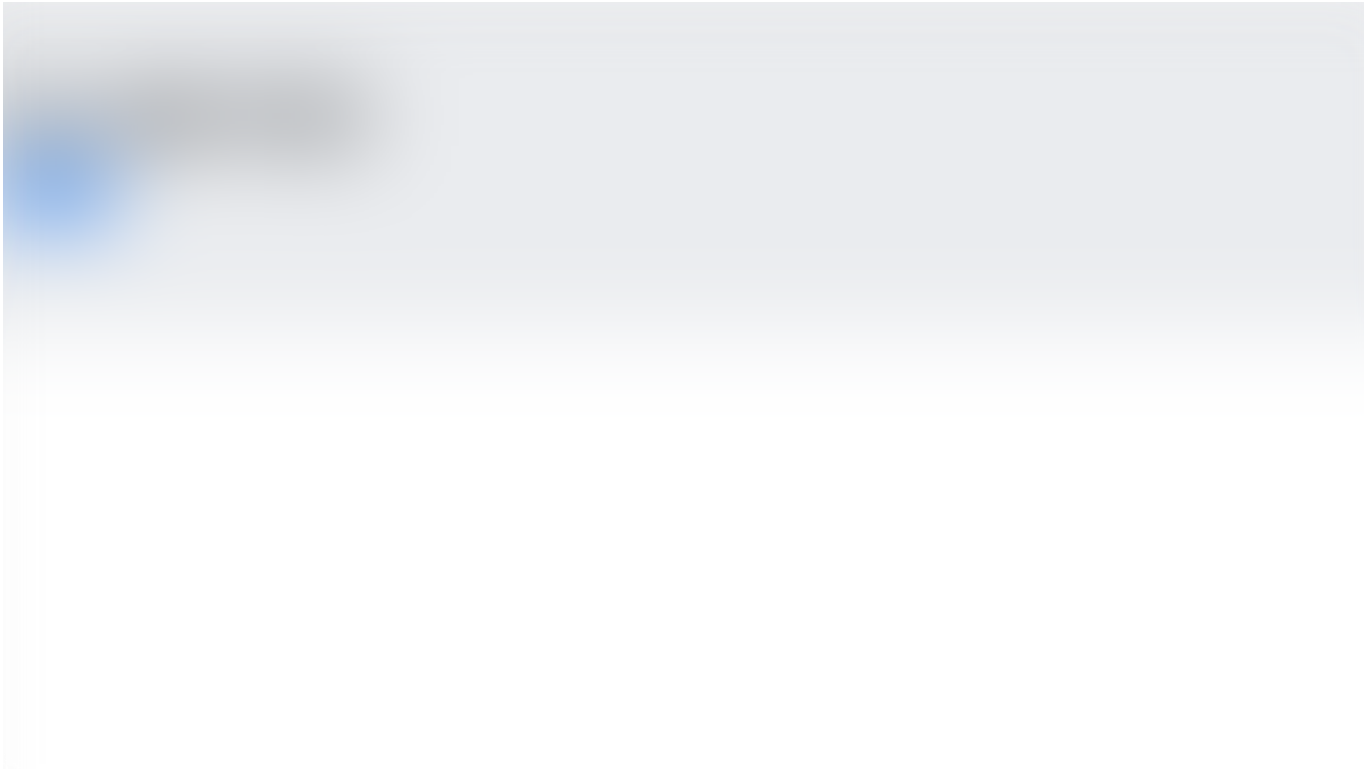
In the template we put the code to render the posts and a link to log out but, if the user is not authenticated, we show a link to log in instead.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">

  <title>My fake blog!</title>

  <link
    rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
    integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
    crossorigin="anonymous">
</head>
<body>
  <div class="jumbotron">
    <h1>My fake blog!</h1>
    {% if user.is_authenticated %}
      <p>Logged as {{ user.username }}</p>
      <a class="btn btn-primary" href="{% url 'logout' %}">Logout</a>
    {% else %}
      <a class="btn btn-primary" href="{% url 'social:begin' 'google-oauth2' %}">
        Login
      </a>
    {% endif %}
  </div>

  <div class="container-fluid">
  {% for post in posts %}
    <div>
      <h2>{{ post.title }}</h2>
      <p>{{ post.content }}</p>
      <p>Posted by {{ post.user.username }} | {{ post.date }}</p>
    </div>
  {% endfor %}
  </div>
```

index.html hosted with ❤ by **GitHub** **view raw**

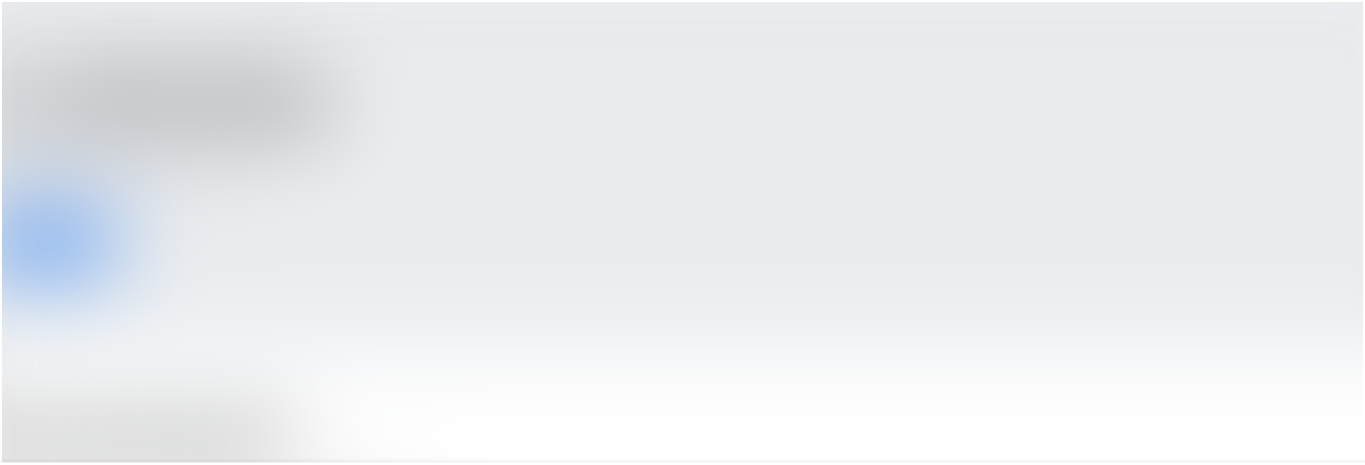
Template code

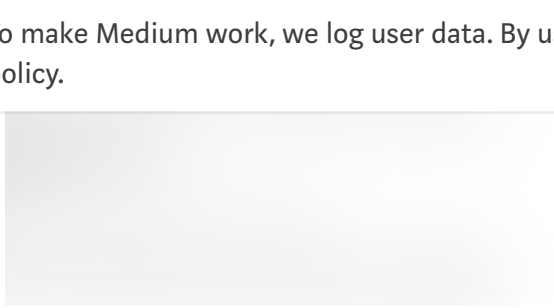
When the user access to our blog unauthenticated, the page looks like this:



Page view when the user is not authenticated

And when the user clicks on the login link and performs the authentication process, the posts list is shown and the page looks like this:





Page view when the user is authenticated

Conclusions

In this post we have seen how the social-app-django library allows to implement OAuth authentication in our django projects in a simple way. In the sample blog application, we've used the library to add support to use Google accounts to authenticate the application users. Likewise, we could have used other backends to add authentication using twitter, github, facebook, etc.

Finally, I would like to stress that this library also works with a variety of python frameworks, allowing us to implement authentication as easily as we did with django.

Thanks to David Barral, Asís García, Ceci García García, Lucas Andión, Roman Coedo, and Marcos Abel.

Oauth Django Python Authentication Google Api