Google Login With Django

Dr. Tyler W Thomas
This Guide Created and Tested on 02/22/2026.
Please email if anything becomes out of date or broken.

What this does:
When you finish, you'll:

Open http://localhost:8000/
Click "Login with Google"
Get redirected back and see your Google email

How we get there:

Step 0 – Create a folder and (optionally) a virtualenv

0.1- First create a new google account for the application (safer to avoid using your real acct).

Open a terminal and run:
0.2 – Create and enter a project folder
mkdir google_login_demo
cd google_login_demo

0.3 – (Recommended) create a virtual environment
python -m venv venv

0.4 – Activate the virtualenv
On macOS / Linux:
source venv/bin/activate
On Windows (Command Prompt):
venv\Scripts\activate
On Windows (PowerShell):
venv\Scripts\Activate.ps1

Step 1 – Install Django and requests

pip install django requests

(note. It may prompt for a newer version of pip. I used the default for consistency. Newer versions may also work.)

Step 2 – Create a new Django project

Still inside the google_login_demo folder:

```
django-admin startproject google_login_demo
cd google_login_demo
```

You now have this structure:

```
google_login_demo/        # (project root with manage.py)
  manage.py
  google_login_demo/      # (inner package with settings, urls)
    __init__.py
    settings.py
    urls.py
    asgi.py
    wsgi.py
```

We will:

put our settings and URLs in google_login_demo/settings.py and google_login_demo/urls.py
create our views in google_login_demo/views.py (this file doesn't exist yet)

Step 3 – Create Google OAuth credentials

Do this once in your browser:

Go to the Google Cloud Console (search for "Google Cloud Console").

Create a project (or select an existing one).

In the left menu, go to "APIs & Services → OAuth consent screen":
(Note: you may have to click "get started") to go through the steps. I named the app "login_app"

Choose External user type (IMPORTANT NOTE: As of the date of this writing, Google Verification is NOT required for TEST accounts with External selected. Additionally, you can add up to 100 testing accounts. "Internal" requires complex organization modification and policy modification and I recommend avoiding it entirely)

Important note 2: You may wish to add email addresses as test accounts later so they can use the login. You can do that from the "Audience" category of "Google Auth Platform"

Fill in minimum required info (app name, support email, etc.)
Save and continue until it's done

In the left menu, go to "APIs & Services → Credentials".

Click "+ CREATE CREDENTIALS" → "OAuth client ID".

Choose Application type: Web application.

Under Authorized redirect URIs, add:

http://localhost:8000/oauth2callback/

Click Create.

You'll see a Client ID and Client secret. Keep that browser tab open or copy them somewhere; we'll paste them into settings.py. Please save them.

Step 4 – Edit Django settings

Open google_login_demo/settings.py in an editor and make these changes:

Set allowed hosts for local dev (near the top, after DEBUG):

```
DEBUG = True

ALLOWED_HOSTS = ["localhost", "127.0.0.1"]
```

2. Add Google OAuth settings at the bottom (use your own values):

```
# Google OAuth settings for this demo
GOOGLE_CLIENT_ID = "YOUR_GOOGLE_CLIENT_ID_HERE"
GOOGLE_CLIENT_SECRET = "YOUR_GOOGLE_CLIENT_SECRET_HERE"
GOOGLE_REDIRECT_URI = "http://localhost:8000/oauth2callback/"
```

GOOGLE_REDIRECT_URI must exactly match what you set in Google Cloud.

Do not remove anything else; the default INSTALLED_APPS, MIDDLEWARE, etc., are fine.
Step 5 – Create the views

Create a new file:
google_login_demo/google_login_demo/views.py

Put this in it:

```
import secrets
from urllib.parse import urlencode

import requests
from django.conf import settings
from django.http import HttpResponse, HttpResponseBadRequest
from django.shortcuts import redirect


def home(request):
    """
    Simple page showing:
      - 'Login with Google' if not logged in
      - email + logout link if logged in
    """
```

```
    user = request.session.get("user")
    if not user:
        return HttpResponse('<a href="/login/">Login with Google</a>')
    return HttpResponse(
        f"Hello {user['email']} "
        f'(<a href="/logout/">logout</a>)'
    )


def google_login(request):
    """
    Redirects the user to Google's OAuth 2.0 authorization endpoint.
    """
    # Random string to protect against CSRF
    state = secrets.token_urlsafe(16)
    request.session["oauth_state"] = state

    params = {
        "client_id": settings.GOOGLE_CLIENT_ID,
        "redirect_uri": settings.GOOGLE_REDIRECT_URI,
        "response_type": "code",
        "scope": "openid email profile",
        "state": state,
    }

    url = "https://accounts.google.com/o/oauth2/v2/auth?" + urlencode(params)
    return redirect(url)


def google_callback(request):
    """
    Handles Google's redirect back:
      - checks 'state'
      - exchanges 'code' for an access token
      - fetches user info
      - stores email in the session
    """
    if request.GET.get("state") != request.session.get("oauth_state"):
        return HttpResponseBadRequest("Invalid state")

    code = request.GET.get("code")
    if not code:
        return HttpResponseBadRequest("Missing code")

    # Exchange code for tokens
    token_res = requests.post(
        "https://oauth2.googleapis.com/token",
        data={
            "code": code,
```

```
            "client_id": settings.GOOGLE_CLIENT_ID,
            "client_secret": settings.GOOGLE_CLIENT_SECRET,
            "redirect_uri": settings.GOOGLE_REDIRECT_URI,
            "grant_type": "authorization_code",
        },
    )
    if not token_res.ok:
        return HttpResponseBadRequest("Token request failed")

    access_token = token_res.json().get("access_token")
    if not access_token:
        return HttpResponseBadRequest("No access token")

    # Fetch basic user info
    userinfo_res = requests.get(
        "https://www.googleapis.com/oauth2/v3/userinfo",
        headers={"Authorization": f"Bearer {access_token}"},
    )
    if not userinfo_res.ok:
        return HttpResponseBadRequest("Userinfo request failed")

    userinfo = userinfo_res.json()

    # Store just the email in the session for this demo
    request.session["user"] = {"email": userinfo.get("email")}
    request.session.pop("oauth_state", None)

    return redirect("/")


def logout_view(request):
    """Clear the session and go back to home."""
    request.session.pop("user", None)
    request.session.pop("oauth_state", None)
    return redirect("/")
```

Step 6 – Wire up URLs

Open google_login_demo/google_login_demo/urls.py and replace its contents with:

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", views.home, name="home"),
    path("login/", views.google_login, name="login"),
    path("oauth2callback/", views.google_callback, name="oauth2callback"),
```

```
    path("logout/", views.logout_view, name="logout"),
]
```

Step 7 – Run database migrations

From the folder that contains manage.py (you should already be there: the inner google_login_demo folder):

python manage.py migrate

This sets up the database tables Django needs (including sessions).
Step 8 – Run the development server

python manage.py runserver

You should see something like:

Starting development server at http://127.0.0.1:8000/

You will probably also receive a nice “WARNING. This is a development server” or something like that. Ignore it (unless you actually plan to use this as a production server)

Step 9 – Test the Google login flow

Open A NEW BROWSER WINDOW OR TAB (otherwise it might use an old cookie and give an error) and go to http://localhost:8000/
Note: Do not use 127.0.0.1. Use localhost. We setup the google oauth credential and script to use localhost and they see that as something different from 127.0.0.1 even though in practice they should both point to the same device.

You should see a simple page with a “Login with Google” link.

Click it:
You’ll be redirected to Google’s login/consent screen.
Log in and accept.

Google redirects you back to:

http://localhost:8000/oauth2callback/?code=...&state=...

Then you’ll finally be redirected to / and see something like:

Hello your_email@example.com (logout)

Click logout; you’ll go back to the login link.

If it closes out and needs to be restarted, it can be done by running this command in the same directory as the manage.py file:
python manage.py runserver

Part 2 - Everyday use (Running it later after a system restart)

Any time later if it does not start when running python manage.py runserver (e.g., next day), do not recreate the project. Just:

Open PowerShell.

Go to your project root (the one that contains venv and the google_login_demo folder):

cd path\to\GoogleLoginDemo

Activate the virtual environment:

.\venv\Scripts\Activate.ps1

You should see (venv) at the start of the prompt.

Go into the Django project folder (where manage.py lives):

cd google_login_demo

Run the server:

python manage.py runserver

Visit http://localhost:8000/ in your browser.

If Django is “not found” at step 5, that means you either:

didn’t activate the venv, or
installed Django outside the venv

In that case, after activating the venv, just reinstall inside it once:

pip install django requests

Then python manage.py runserver should work.

from the use of this guide or the accompanying code. This project is not affiliated with, endorsed by, or officially supported by Google, the Django Software Foundation, or any other organization mentioned.