# ✍️ How to Add "Sign in with Google" to Your Signup Page (FastAPI Version)

OAuth 2.0 is the standard that powers 'Sign in with Google'. It allows your app to access user data like email, name, and profile picture without ever handling their Google passwords.

## 🧑‍🤝‍🧑 Key Players:

1. Your Application (Client): Your FastAPI backend service.  
2. Google (Authorization Server/Resource Server): Manages authentication and user data.  
3. User (Resource Owner): The person signing up.

## 🔄 Flow Overview:

1️⃣ Redirect to Google — App redirects the user to Google's login & consent page.

2️⃣ User Grants Permission — User logs in and authorizes access.

3️⃣ Authorization Code — Google redirects back to your app with an authorization code.

4️⃣ Exchange for Access Token — Your FastAPI server exchanges this code for an access token.

5️⃣ Access User Data — Use the token to fetch user info from Google APIs.

6️⃣ Create a User Account — Register or log in the user in your app using Google data.

7️⃣ Store User Info — Save the essential data (email, name, avatar) and create a session.

## ☁️ A. Google Cloud Console Setup:

1️⃣ Go to: Google Cloud Console

2️⃣ Create a New Project (if needed).

3️⃣ Enable Google Identity Platform API.

4️⃣ Configure OAuth 2.0 Credentials:

- Go to APIs & Services → Credentials → + CREATE CREDENTIALS → OAuth client ID.

- Select Web Application.

- Set Authorized JavaScript origins and Redirect URIs.

- After creation, note your Client ID and Client Secret.

⚠️ Never expose your Client Secret in frontend code!

## 🧑‍💻 B. FastAPI Backend Implementation:

1️⃣ Install dependencies:  
  
pip install fastapi uvicorn python-jose requests python-dotenv

2️⃣ Example Token Verification Function:

from fastapi import FastAPI, HTTPException, Request  
from pydantic import BaseModel  
import requests  
from jose import jwt  
  
app = FastAPI()  
  
GOOGLE\_CLIENT\_ID = "YOUR\_CLIENT\_ID"  
  
class TokenData(BaseModel):  
 token: str  
  
def verify\_google\_token(token: str):  
 try:  
 response = requests.get(f"https://oauth2.googleapis.com/tokeninfo?id\_token={token}")  
 if response.status\_code != 200:  
 return None  
 payload = response.json()  
 if payload["aud"] != GOOGLE\_CLIENT\_ID:  
 return None  
 return {  
 "userid": payload["sub"],  
 "email": payload["email"],  
 "name": payload.get("name"),  
 "picture": payload.get("picture")  
 }  
 except Exception as e:  
 return None  
  
@app.post("/auth/google/callback")  
async def google\_callback(data: TokenData):  
 user\_info = verify\_google\_token(data.token)  
 if user\_info:  
 # Logic to check/create user and handle session  
 return {"message": "User authenticated", "user": user\_info}  
 raise HTTPException(status\_code=401, detail="Invalid Google token")

## 💡 User Management:

1️⃣ Check existing users by email.  
2️⃣ Create new users if none exist.  
3️⃣ Set up session or JWT.

## 🔒 Security Considerations:

- Never expose Client Secret in frontend.  
- Always verify tokens on the back-end.  
- Use HTTPS.  
- Proper CORS configuration.

## 🧠 User Experience:

- Make the Google button clear and visible.  
- Graceful error handling.  
- Always provide a fallback signup option.

## 📃 Privacy & Testing:

- Request only necessary permissions.  
- Update your privacy policy.  
- Allow unlinking Google accounts.  
- Test across environments with multiple Google accounts.

## 🗃️ Example Database Schema (SQL):

CREATE TABLE users (  
 id SERIAL PRIMARY KEY,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 name VARCHAR(255),  
 google\_id VARCHAR(255),  
 profile\_picture VARCHAR(255),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

## 🧯 Troubleshooting:

- 🚨 redirect\_uri\_mismatch — Double-check URIs.  
- 🔑 JWT verification failures — Validate Client ID and token integrity.  
- 🌐 CORS issues — Properly set headers.  
- 🔍 Missing user data — Check API scopes.