

Google ve X.com OAuth 2.0 Entegrasyon Rehberi

gistify.pro için Kapsamlı Uygulama Kılavuzu

React Frontend + Python (FastAPI) Backend

İçindekiler

1. Genel Bakış ve Mimari
2. Google OAuth 2.0 Entegrasyonu
 - 2.1 Google Cloud Console Kurulumu
 - 2.2 Backend (Python/FastAPI) Implementasyonu
 - 2.3 Frontend (React) Implementasyonu
3. X.com (Twitter) OAuth 2.0 Entegrasyonu
 - 3.1 X Developer Portal Kurulumu
 - 3.2 Backend (Python/FastAPI) Implementasyonu
 - 3.3 Frontend (React) Implementasyonu
4. Güvenlik ve En İyi Uygulamalar
5. Sorun Giderme

1. Genel Bakış ve Mimari

Bu rehber, gistify.pro uygulamanıza Google ve X.com (Twitter) sosyal giriş özelliklerini eklemek için adım adım talimatlar sunmaktadır. OAuth 2.0 protokolü kullanılarak güvenli kimlik doğrulama sağlanacaktır.

1.1 OAuth 2.0 Akışı

Authorization Code Flow with PKCE kullanılacaktır:

1. Kullanıcı "Google/X ile Giriş Yap" butonuna tıklar
2. Frontend, kullanıcıyı OAuth sağlayıcısına yönlendirir
3. Kullanıcı izin verir ve authorization code ile geri döner
4. Backend, code'u access token ile değiştirir
5. Backend, kullanıcı bilgilerini alır ve JWT oluşturur

1.2 Gerekli Bağımlılıklar

Backend (Python)

```
pip install fastapi uvicorn python-jose[cryptography] httpx
```

```
pip install python-dotenv pydantic sqlalchemy
pip install google-auth google-auth-oauthlib
```

Frontend (React)

```
npm install @react-oauth/google axios
npm install react-router-dom
```

2. Google OAuth 2.0 Entegrasyonu

2.1 Google Cloud Console Kurulumu

1. Google Cloud Console'a gidin: console.cloud.google.com
2. Yeni bir proje oluşturun veya mevcut projeyi seçin
3. APIs & Services > OAuth consent screen'e gidin
4. User Type olarak "External" seçin ve formu doldurun
5. Credentials > Create Credentials > OAuth client ID seçin
6. Application type: "Web application" seçin
7. Authorized JavaScript origins ekleyin:
 - http://localhost:3000 (development)
 - https://gistify.pro (production)
8. Authorized redirect URIs ekleyin:
 - http://localhost:8000/api/auth/google/callback
 - https://gistify.pro/api/auth/google/callback

☐ Client ID ve Client Secret'ı güvenli bir yerde saklayın. Bunları asla frontend kodunda kullanmayın!

2.2 Backend Implementasyonu (FastAPI)

config.py - Yapılandırma Dosyası

```
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    # Google OAuth
    GOOGLE_CLIENT_ID: str
    GOOGLE_CLIENT_SECRET: str
    GOOGLE_REDIRECT_URI: str = "http://localhost:8000/api/auth/google/callback"

    # JWT
    JWT_SECRET_KEY: str
    JWT_ALGORITHM: str = "HS256"
    JWT_EXPIRE_MINUTES: int = 60 * 24 # 24 saat

    # App
    FRONTEND_URL: str = "http://localhost:3000"

    class Config:
        env_file = ".env"

settings = Settings()
```

auth/google.py - Google OAuth İşlemleri

```
import httpx
```

```

from fastapi import APIRouter, HTTPException
from fastapi.responses import RedirectResponse
from urllib.parse import urlencode
from config import settings
from auth.jwt_handler import create_access_token

router = APIRouter(prefix="/api/auth/google", tags=["Google Auth"])

GOOGLE_AUTH_URL = "https://accounts.google.com/o/oauth2/v2/auth"
GOOGLE_TOKEN_URL = "https://oauth2.googleapis.com/token"
GOOGLE_USERINFO_URL = "https://www.googleapis.com/oauth2/v2/userinfo"

@router.get("/login")
async def google_login():
    """Google OAuth login başlatır"""
    params = {
        "client_id": settings.GOOGLE_CLIENT_ID,
        "redirect_uri": settings.GOOGLE_REDIRECT_URI,
        "response_type": "code",
        "scope": "openid email profile",
        "access_type": "offline",
        "prompt": "consent"
    }
    auth_url = f"{GOOGLE_AUTH_URL}?{urlencode(params)}"
    return RedirectResponse(url=auth_url)

@router.get("/callback")
async def google_callback(code: str = None, error: str = None):
    """Google OAuth callback işler"""
    if error:
        raise HTTPException(status_code=400, detail=f"OAuth error: {error}")

    if not code:
        raise HTTPException(status_code=400, detail="Authorization code missing")

    # Code'u access token ile değiştir
    async with httpx.AsyncClient() as client:
        token_response = await client.post(
            GOOGLE_TOKEN_URL,
            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 token_response.status_code != 200:
        raise HTTPException(status_code=400, detail="Token exchange failed")

    tokens = token_response.json()
    access_token = tokens.get("access_token")

    # Kullanıcı bilgilerini al
    userinfo_response = await client.get(
        GOOGLE_USERINFO_URL,
        headers={"Authorization": f"Bearer {access_token}"}
    )

    if userinfo_response.status_code != 200:
        raise HTTPException(status_code=400, detail="Failed to get user info")

    user_info = userinfo_response.json()

```

```
# Kullanıcıyı veritabanında bul veya oluştur
# user = await get_or_create_user(user_info)

# JWT token oluştur
jwt_token = create_access_token({
    "sub": user_info["id"],
    "email": user_info["email"],
    "name": user_info.get("name", ""),
    "picture": user_info.get("picture", ""),
    "provider": "google"
})

# Frontend'e redirect et
redirect_url = f"{settings.FRONTEND_URL}/auth/callback?token={jwt_token}"
return RedirectResponse(url=redirect_url)
```

auth/jwt_handler.py - JWT İşlemleri

```
from datetime import datetime, timedelta
from jose import jwt, JWTError
from config import settings

def create_access_token(data: dict) -> str:
    """JWT access token oluşturur"""
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=settings.JWT_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, settings.JWT_SECRET_KEY,
                      algorithm=settings.JWT_ALGORITHM)

def verify_token(token: str) -> dict:
    """JWT token doğrular"""
    try:
        payload = jwt.decode(token, settings.JWT_SECRET_KEY,
                              algorithms=[settings.JWT_ALGORITHM])
        return payload
    except JWTError:
        return None
```

2.3 Frontend Implementasyonu (React)

App.jsx - Ana Uygulama

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { GoogleOAuthProvider } from '@react-oauth/google';
import Login from './pages/Login';
import AuthCallback from './pages/AuthCallback';
import Dashboard from './pages/Dashboard';

const GOOGLE_CLIENT_ID = import.meta.env.VITE_GOOGLE_CLIENT_ID;

function App() {
    return (
        <GoogleOAuthProvider clientId={GOOGLE_CLIENT_ID}>
            <BrowserRouter>
                <Routes>
                    <Route path="/login" element={<Login />} />
                    <Route path="/auth/callback" element={<AuthCallback />} />
                    <Route path="/dashboard" element={<Dashboard />} />
                </Routes>
            </BrowserRouter>
        </GoogleOAuthProvider>
    );
}
```

```

    });
  }

  export default App;

```

pages/Login.jsx - Giriş Sayfası

```

import React from 'react';
import { GoogleLogin, useGoogleLogin } from '@react-oauth/google';

const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000';

function Login() {
  // Yöntem 1: Backend redirect ile
  const handleGoogleLoginRedirect = () => {
    window.location.href = `${API_URL}/api/auth/google/login`;
  };

  // Yöntem 2: @react-oauth/google ile (Authorization Code Flow)
  const googleLogin = useGoogleLogin({
    onSuccess: async (codeResponse) => {
      // Code'u backend'e gönder
      const response = await fetch(`${API_URL}/api/auth/google/token`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ code: codeResponse.code })
      });
      const data = await response.json();
      localStorage.setItem('token', data.token);
      window.location.href = '/dashboard';
    },
    flow: 'auth-code',
  });

  return (
    <div className="login-container">
      <h1>gistify.pro'ya Hoş Geldiniz</h1>

      {/* Yöntem 1: Redirect Button */}
      <button onClick={handleGoogleLoginRedirect} className="google-btn">
        
        Google ile Giriş Yap
      </button>

      {/* Yöntem 2: Custom Button with Hook */}
      <button onClick={() => googleLogin()} className="google-btn">
        Google ile Devam Et
      </button>

      {/* Yöntem 3: Built-in Google Button */}
      <GoogleLogin
        onSuccess={credentialResponse => {
          console.log(credentialResponse);
        }}
        onError={() => console.log('Login Failed')}
        theme="outline"
        size="large"
        text="signin_with"
      />
    </div>
  );
}

export default Login;

```

pages/AuthCallback.jsx - Callback Handler

```
import React, { useEffect } from 'react';
import { useNavigate, useSearchParams } from 'react-router-dom';

function AuthCallback() {
  const navigate = useNavigate();
  const [searchParams] = useSearchParams();

  useEffect(() => {
    const token = searchParams.get('token');
    const error = searchParams.get('error');

    if (error) {
      console.error('Auth error:', error);
      navigate('/login?error=' + error);
      return;
    }

    if (token) {
      localStorage.setItem('token', token);
      navigate('/dashboard');
    } else {
      navigate('/login');
    }
  }, [searchParams, navigate]);

  return (
    <div className="loading">
      <p>Giriş yapılıyor...</p>
    </div>
  );
}

export default AuthCallback;
```

3. X.com (Twitter) OAuth 2.0 Entegrasyonu

3.1 X Developer Portal Kurulumu

1. X Developer Portal'a gidin: developer.twitter.com
2. Yeni bir proje ve uygulama oluşturun
3. User authentication settings'e gidin
4. OAuth 2.0'ı etkinleştirin
5. App permissions: Read (veya ihtiyaca göre)
6. Type of App: Web App, Automated App or Bot
7. Callback URI / Redirect URL ekleyin:
 - <http://localhost:8000/api/auth/twitter/callback>
 - <https://gistify.pro/api/auth/twitter/callback>
8. Website URL: <https://gistify.pro>

❏ X API Free tier ile sadece Read erişimi alabilirsiniz. Tweet post etmek için Basic (\$100/ay) veya üzeri plan gereklidir.

3.2 Backend Implementasyonu (FastAPI)

config.py - X.com Yapılandırması Ekleme

```
class Settings(BaseSettings):
    # ... Google settings ...

    # X.com (Twitter) OAuth
    TWITTER_CLIENT_ID: str
    TWITTER_CLIENT_SECRET: str
    TWITTER_REDIRECT_URI: str = "http://localhost:8000/api/auth/twitter/callback"

    # ... diğer ayarlar ...
```

auth/twitter.py - X.com OAuth İşlemleri

```
import httpx
import secrets
import hashlib
import base64
from fastapi import APIRouter, HTTPException, Request
from fastapi.responses import RedirectResponse
from urllib.parse import urlencode
from config import settings
from auth.jwt_handler import create_access_token

router = APIRouter(prefix="/api/auth/twitter", tags=["Twitter Auth"])

TWITTER_AUTH_URL = "https://twitter.com/i/oauth2/authorize"
TWITTER_TOKEN_URL = "https://api.twitter.com/2/oauth2/token"
TWITTER_USERINFO_URL = "https://api.twitter.com/2/users/me"

# PKCE için state ve code_verifier saklamak için basit cache
# Production'da Redis kullanın
auth_states = {}

def generate_pkce():
    """PKCE code_verifier ve code_challenge oluşturur"""
    code_verifier = secrets.token_urlsafe(32)
    code_challenge = base64.urlsafe_b64encode(
        hashlib.sha256(code_verifier.encode()).digest()
    ).decode().rstrip("=")
    return code_verifier, code_challenge

@router.get("/login")
async def twitter_login():
    """X.com OAuth login başlatır"""
    state = secrets.token_urlsafe(16)
    code_verifier, code_challenge = generate_pkce()

    # State ve verifier'ı sakla
    auth_states[state] = code_verifier

    params = {
        "response_type": "code",
        "client_id": settings.TWITTER_CLIENT_ID,
        "redirect_uri": settings.TWITTER_REDIRECT_URI,
        "scope": "tweet.read users.read offline.access",
        "state": state,
        "code_challenge": code_challenge,
        "code_challenge_method": "S256"
    }
    auth_url = f"{TWITTER_AUTH_URL}?{urlencode(params)}"
    return RedirectResponse(url=auth_url)

@router.get("/callback")
```

```

async def twitter_callback(code: str = None, state: str = None, error: str = None):
    """X.com OAuth callback işler"""
    if error:
        raise HTTPException(status_code=400, detail=f"OAuth error: {error}")

    if not code or not state:
        raise HTTPException(status_code=400, detail="Missing code or state")

    # State doğrula ve code_verifier'ı al
    code_verifier = auth_states.pop(state, None)
    if not code_verifier:
        raise HTTPException(status_code=400, detail="Invalid state")

    # Basic Auth header oluştur
    credentials = f"{settings.TWITTER_CLIENT_ID}:{settings.TWITTER_CLIENT_SECRET}"
    basic_auth = base64.b64encode(credentials.encode()).decode()

    async with httpx.AsyncClient() as client:
        # Code'u access token ile değiştir
        token_response = await client.post(
            TWITTER_TOKEN_URL,
            headers={
                "Content-Type": "application/x-www-form-urlencoded",
                "Authorization": f"Basic {basic_auth}"
            },
            data={
                "code": code,
                "grant_type": "authorization_code",
                "redirect_uri": settings.TWITTER_REDIRECT_URI,
                "code_verifier": code_verifier
            }
        )

        if token_response.status_code != 200:
            raise HTTPException(
                status_code=400,
                detail=f"Token exchange failed: {token_response.text}"
            )

        tokens = token_response.json()
        access_token = tokens.get("access_token")

        # Kullanıcı bilgilerini al
        userinfo_response = await client.get(
            TWITTER_USERINFO_URL,
            headers={"Authorization": f"Bearer {access_token}"},
            params={"user.fields": "id,name,username,profile_image_url"}
        )

        if userinfo_response.status_code != 200:
            raise HTTPException(status_code=400, detail="Failed to get user info")

        user_data = userinfo_response.json()["data"]

        # JWT token oluştur
        jwt_token = create_access_token({
            "sub": user_data["id"],
            "name": user_data["name"],
            "username": user_data["username"],
            "picture": user_data.get("profile_image_url", ""),
            "provider": "twitter"
        })

    # Frontend'e redirect et

```

```
redirect_url = f"{Settings.FRONTEND_URL}/auth/callback?token={jwt_token}"
return RedirectResponse(url=redirect_url)
```

3.3 Frontend Implementasyonu (React)

components/TwitterLoginButton.jsx

```
import React from 'react';

const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000';

function TwitterLoginButton() {
  const handleTwitterLogin = () => {
    window.location.href = `${API_URL}/api/auth/twitter/login`;
  };

  return (
    <button onClick={handleTwitterLogin} className="twitter-btn">
      <svg viewBox="0 0 24 24" width="20" height="20" fill="currentColor">
        <path d="M18.244 2.25h3.308l-7.227 8.26 8.502 11.24H16.17l-5.214-6.817L4.99
21.75H1.68l7.73-8.835L1.254 2.25H8.08l4.713 6.231zm-1.161 17.52h1.833L7.084
4.126H5.117z"/>
      </svg>
      X ile Giriş Yap
    </button>
  );
}

export default TwitterLoginButton;
```

pages/Login.jsx - Güncellenmiş

```
import React from 'react';
import { GoogleLogin } from '@react-oauth/google';
import TwitterLoginButton from '../components/TwitterLoginButton';

const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000';

function Login() {
  const handleGoogleLogin = () => {
    window.location.href = `${API_URL}/api/auth/google/login`;
  };

  return (
    <div className="login-container">
      <h1>gistify.pro</h1>
      <p>Hesabınıza giriş yapın</p>

      <div className="social-buttons">
        <button onClick={handleGoogleLogin} className="google-btn">
          
          Google ile Giriş Yap
        </button>

        <TwitterLoginButton />
      </div>

      <div className="divider">
        <span>veya</span>
      </div>

      </* Email/password form buraya eklenebilir */>
    </div>
  );
}
```

```
);
}

export default Login;
```

4. Güvenlik ve En İyi Uygulamalar

4.1 Environment Variables

.env dosyanızı asla Git'e commit etmeyin. .gitignore'a ekleyin:

```
.env
.env.local
.env.production
```

Backend .env Örneği

```
# Google OAuth
GOOGLE_CLIENT_ID=your-google-client-id.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=your-google-client-secret
GOOGLE_REDIRECT_URI=http://localhost:8000/api/auth/google/callback

# Twitter/X OAuth
TWITTER_CLIENT_ID=your-twitter-client-id
TWITTER_CLIENT_SECRET=your-twitter-client-secret
TWITTER_REDIRECT_URI=http://localhost:8000/api/auth/twitter/callback

# JWT
JWT_SECRET_KEY=your-super-secret-key-min-32-characters
JWT_ALGORITHM=HS256

# App
FRONTEND_URL=http://localhost:3000
```

Frontend .env Örneği

```
VITE_GOOGLE_CLIENT_ID=your-google-client-id.apps.googleusercontent.com
VITE_API_URL=http://localhost:8000
```

4.2 CORS Yapılandırması

main.py - CORS Ayarları

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from auth.google import router as google_router
from auth.twitter import router as twitter_router

app = FastAPI()

# CORS ayarları
origins = [
    "http://localhost:3000",
    "https://gistify.pro",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
# Router'ları ekle
app.include_router(google_router)
app.include_router(twitter_router)
```

4.3 Token Güvenliği

- JWT token'larını httpOnly cookie olarak saklayın (XSS koruması)
- Refresh token mekanizması implementasyon için düşünün
- Token expire sürelerini makul tutun (örn: 24 saat)
- HTTPS kullanın (production'da zorunlu)

Secure Cookie Implementasyonu

```
from fastapi.responses import RedirectResponse

@router.get("/callback")
async def google_callback(code: str = None):
    # ... token ve user işlemleri ...

    jwt_token = create_access_token(user_data)

    response = RedirectResponse(url=f"{settings.FRONTEND_URL}/dashboard")
    response.set_cookie(
        key="access_token",
        value=jwt_token,
        httponly=True,
        secure=True, # Production'da True
        samesite="lax",
        max_age=86400 # 24 saat
    )
    return response
```

5. Sorun Giderme

5.1 Yaygın Hatalar

Google OAuth Hataları

- Error 400: redirect_uri_mismatch → Redirect URI'yi Google Console'da tam olarak aynı şekilde ekleyin
- Error 401: invalid_client → Client ID/Secret'ı kontrol edin
- Error 403: access_denied → OAuth consent screen'i yayınlayın (Test users ekleyin)

X.com OAuth Hataları

- Error 401: Unauthorized → Client credentials'ı kontrol edin
- Error 403: Forbidden → API erişim seviyenizi kontrol edin
- Invalid state → State caching mekanizmasını kontrol edin (Redis önerilir)
- PKCE hataları → code_verifier ve code_challenge uyumunu kontrol edin

5.2 Debug İpuçları

- Backend loglarını detaylı tutun
- Browser Network tab'ını kullanarak request/response'ları inceleyin
- OAuth sağlayıcılarının developer dashboard'larındaki logs'ları kontrol edin

5.3 Faydalı Linkler

Google OAuth Documentation: developers.google.com/identity/protocols/oauth2

X.com OAuth Documentation: developer.twitter.com/en/docs/authentication/oauth-2-0

@react-oauth/google: npmjs.com/package/@react-oauth/google

FastAPI Security: fastapi.tiangolo.com/tutorial/security

Sonuç

Bu rehberi takip ederek gistify.pro uygulamanıza Google ve X.com sosyal giriş özelliklerini başarıyla entegre edebilirsiniz. Herhangi bir sorunla karşılaşırsanız, sorun giderme bölümünü ve resmi dokümantasyonları inceleyin.

□ *Production'a geçmeden önce tüm güvenlik önlemlerini uyguladığınızdan emin olun!*

— Rehber Sonu —