

Flujo de autenticación con Google OAuth 2.0

Secuencia completa

```
USUARIO → FRONTEND → GOOGLE → FRONTEND → BACKEND → GOOGLE → BACKEND → BASE  
DE DATOS → BACKEND → FRONTEND
```

Diccionario de datos

A. FRONTEND → GOOGLE

Petición inicial de autenticación:

```
GET https://accounts.google.com/o/oauth2/v2/auth
```

Parámetros:

- client_id : ID de la aplicación en Google Cloud Console
- redirect_uri : URL a donde Google redirigirá después de autenticar
- response_type : "code" (Authorization Code Flow)
- scope : "email profile" (permisos solicitados)
- state : "RANDOM_STRING" (protección CSRF - recomendado)

B. GOOGLE → FRONTEND

Respuesta después de autenticación exitosa:

```
GET {redirect_uri}?code=AUTH_CODE_ABC123&state=RANDOM_STRING
```

Parámetros:

- code : Código de autorización temporal (un solo uso)
- state : Mismo valor enviado (validar que coincida)

C. FRONTEND → BACKEND

Envío del código al servidor:

```
POST /auth/google
Content-Type: application/json

{
  "code": "AUTH_CODE_ABC123"
}
```

D. BACKEND → GOOGLE

Intercambio de código por tokens:

```
POST https://oauth2.googleapis.com/token
Content-Type: application/json

{
  "code": "AUTH_CODE_ABC123",
  "client_id": "TU_CLIENT_ID",
  "client_secret": "TU_CLIENT_SECRET",
  "redirect_uri": "TU_REDIRECT_URI",
  "grant_type": "authorization_code"
}
```

⚠ IMPORTANTE: El `client_secret` NUNCA debe estar en el frontend.

E. GOOGLE → BACKEND

Respuesta con tokens y datos del usuario:

```
{
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjE...",
  "access_token": "ya29.a0AfH6SMB...",
  "expires_in": 3600,
  "token_type": "Bearer",
  "scope": "openid https://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/userinfo.profile",
  "refresh_token": "1//0gHJ..." (opcional, solo si se solicita)
```

```
offline_access)  
}
```

Decodificación del `id_token` (JWT):

```
{  
  "sub": "google_12345678901234567890",  
  "email": "usuario@gmail.com",  
  "email_verified": true,  
  "name": "Nombre Usuario",  
  "picture": "https://lh3.googleusercontent.com/...",  
  "given_name": "Nombre",  
  "family_name": "Usuario",  
  "iat": 1638360000,  
  "exp": 1638363600,  
  "iss": "https://accounts.google.com",  
  "aud": "TU_CLIENT_ID"  
}
```

F. BACKEND → BASE DE DATOS

Búsqueda de usuario existente:

```
SELECT * FROM users  
WHERE google_sub = 'google_12345678901234567890';
```

Si no existe, registro de nuevo usuario:

```
INSERT INTO users (google_sub, email, name, avatar, email_verified,  
created_at)  
VALUES (  
  'google_12345678901234567890',  
  'usuario@gmail.com',  
  'Nombre Usuario',  
  'https://lh3.googleusercontent.com/...',  
  true,  
  NOW()  
) ;
```

Actualización de último login:

```
UPDATE users
SET last_login = NOW()
WHERE google_sub = 'google_12345678901234567890';
```

G. BACKEND → FRONTEND

Generación y envío de JWT interno:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 123,
    "email": "usuario@gmail.com",
    "name": "Nombre Usuario",
    "avatar": "https://lh3.googleusercontent.com/..."
  }
}
```

Payload del JWT interno:

```
{
  "userId": 123,
  "email": "usuario@gmail.com",
  "iat": 1638360000,
  "exp": 1638446400
}
```

H. FRONTEND → BACKEND (requests subsecuentes)

Todas las peticiones autenticadas incluyen:

```
GET /api/recurso-protégido
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Validaciones de seguridad en el backend

1. Validar el `id_token` de Google

```
// Verificar:  
✓ Firma del JWT (usando claves públicas de Google)  
✓ iss === "https://accounts.google.com"  
✓ aud === TU_CLIENT_ID  
✓ exp > tiempo actual  
✓ email_verified === true
```

2. Validar el parámetro `state`

```
// En frontend:  
const state = generateRandomString();  
sessionStorage.setItem('oauth_state', state);  
  
// Al recibir respuesta:  
if (receivedState !== sessionStorage.getItem('oauth_state')) {  
  throw new Error('CSRF attack detected');  
}
```

3. Configuración en Google Cloud Console

Authorized redirect URIs:

- `http://localhost:3000/auth/callback` (desarrollo)
- `https://tu-dominio.com/auth/callback` (producción)

Authorized JavaScript origins:

- `http://localhost:3000` (desarrollo)
- `https://tu-dominio.com` (producción)

Esquema de base de datos

```
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  google_sub VARCHAR(255) UNIQUE NOT NULL,  
  email VARCHAR(255) UNIQUE NOT NULL,  
  email_verified BOOLEAN DEFAULT false,  
  name VARCHAR(255),  
  avatar TEXT,  
  created_at TIMESTAMP DEFAULT NOW(),
```

```
last_login TIMESTAMP,  
INDEX idx_google_sub (google_sub),  
INDEX idx_email (email)  
) ;
```

Manejo de errores

Caso	Acción
Usuario cancela en Google	Redirigir con error=access_denied
Código inválido/expirado	Devolver 401, pedir re-autenticación
Token de Google inválido	Devolver 401, pedir re-autenticación
Email no verificado	Devolver 403, pedir verificación
Error de red con Google	Devolver 503, reintentar

Flujo de logout

1. Frontend elimina el JWT del localStorage/cookies
2. Backend (opcional): invalidar token en blacklist
3. Frontend (opcional): revocar token en Google:
POST <https://oauth2.googleapis.com/revoke>
token={access_token}

Mejorasopcionales

PKCE (Proof Key for Code Exchange)

Para SPAs, agregar una capa extra de seguridad:

```
// Frontend genera:  
const codeVerifier = generateRandomString(128);  
const codeChallenge = base64UrlEncode(sha256(codeVerifier));  
  
// Envía a Google:  
code_challenge: codeChallenge
```

```
code_challenge_method: "S256"

// Envía al backend junto con el code:
{ code, codeVerifier }

// Backend lo incluye al intercambiar el código:
{ code, code_verifier: codeVerifier, ... }
```

Refresh tokens

Si necesitas sesiones de larga duración:

```
// Solicitar en el scope inicial:
scope: "email profile offline_access"

// Google devolverá refresh_token
// Guardarlo en BD encriptado
// Usarlo para renovar access_token cuando expire
```