OAuth 2.0 Single Sign-On

Integration Specification Document

Cross-Team Implementation Guide

App A (Java Authorization Server) $\longleftrightarrow$ App B (Next.js Client)

Document Owner: Team B (Next.js Application)
Version: 1.0.0

Date: January 3, 2026

Document Status	Draft / Under Review / Approved
Classification	Internal / Confidential
Last Updated	January 3, 2026
Review Date	February 3, 2026

Contents

1 Executive Summary

1.1 Purpose

This document serves as the definitive technical specification for implementing OAuth 2.0 Single Sign-On (SSO) between two applications maintained by separate teams:

- **App A (Authorization Server):** Java Web Application – Maintained by Team A
- **App B (Client Application):** Next.js Application – Maintained by Team B

1.2 Scope

This specification covers:

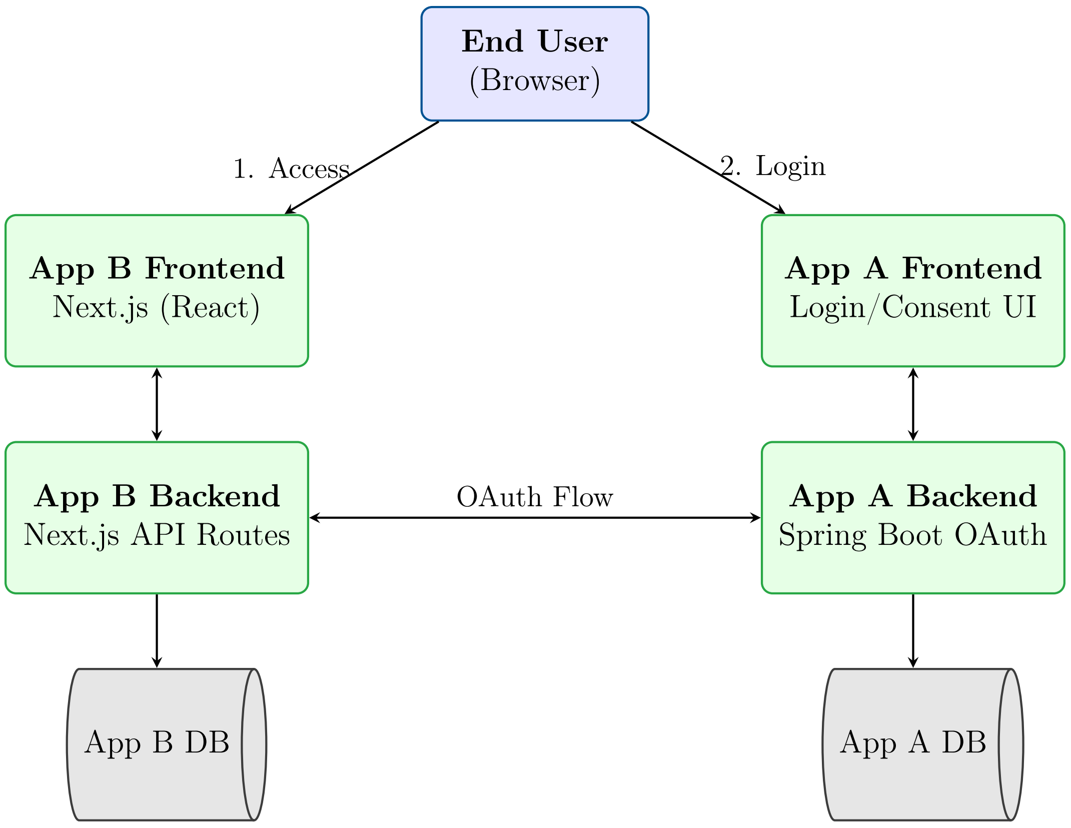
- Complete OAuth 2.0 Authorization Code Flow with PKCE
- All API endpoints required from both teams
- Frontend URLs and routes
- Data exchange formats and contracts
- Security requirements
- Integration testing procedures

1.3 Audience

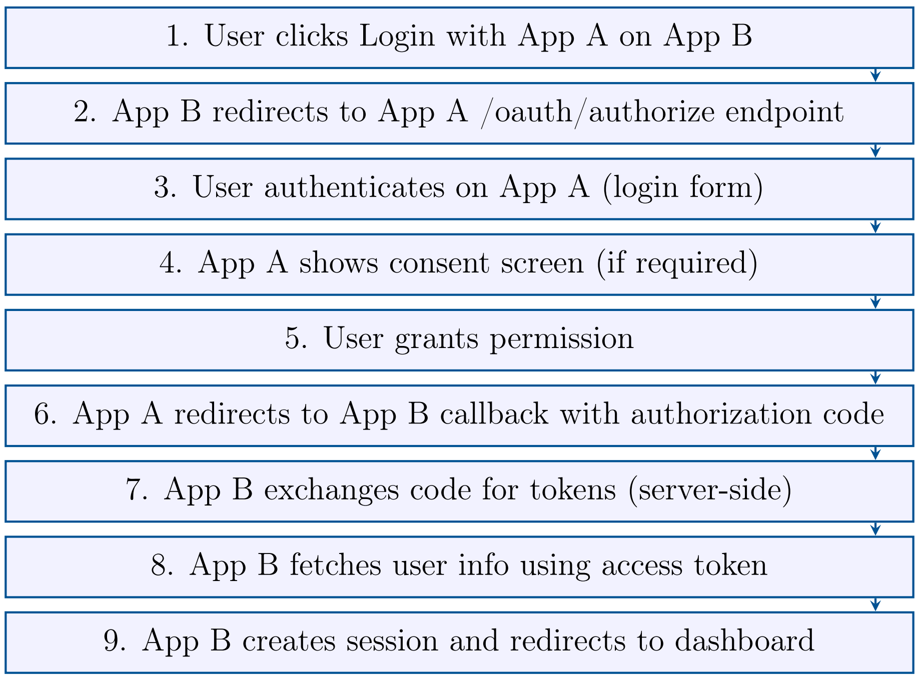
- Backend Developers (Team A and Team B)
- Frontend Developers (Team A and Team B)
- DevOps Engineers
- Security Architects
- Project Managers

2 System Overview

2.1 Architecture Diagram



2.2 OAuth 2.0 Flow Sequence


1. User clicks Login with App A on App B
2. App B redirects to App A /oauth/authorize endpoint
3. User authenticates on App A (login form)
4. App A shows consent screen (if required)
5. User grants permission
6. App A redirects to App B callback with authorization code
7. App B exchanges code for tokens (server-side)
8. App B fetches user info using access token
9. App B creates session and redirects to dashboard

3 What Team B (Next.js) Provides to Team A

Team B Deliverables

This section details all information, URLs, and configurations that Team B must provide to Team A for OAuth client registration.

3.1 Application Registration Information

Field	Description	Example Value
Application Name	Display name shown on consent screen	My Next.js Application
Application Description	Brief description of what App B does	A modern web application
Application Logo URL	Logo to display on consent screen	https://app-b.com/logo.png
Application Website	Main website URL	https://app-b.com
Privacy Policy URL	Link to privacy policy	https://app-b.com/privacy
Terms of Service URL	Link to terms of service	https://app-b.com/terms
Technical Contact	For technical issues	dev@app-b.com
Support Contact	For user support	support@app-b.com

3.2 Redirect URIs (Callback URLs)

Critical: Exact Match Required

Redirect URIs must match EXACTLY. Pay attention to:

- Protocol (http vs https)
- Trailing slashes
- Port numbers
- Case sensitivity

3.2.1 Production Environment

```
https://app-b.example.com/auth/callback
https://app-b.example.com/api/auth/callback
```

3.2.2 Staging Environment

```
https://staging.app-b.example.com/auth/callback
https://staging.app-b.example.com/api/auth/callback
```

3.2.3 Development Environment

```
http://localhost:3000/auth/callback
http://localhost:3000/api/auth/callback
http://127.0.0.1:3000/auth/callback
```

3.3 Post-Logout Redirect URIs

```
Production:
https://app-b.example.com
https://app-b.example.com/auth/login

Staging:
https://staging.app-b.example.com
https://staging.app-b.example.com/auth/login

Development:
http://localhost:3000
http://localhost:3000/auth/login
```

3.4 Required OAuth Scopes

Scope	**Required**	**Purpose**
openid	Yes	Enable OpenID Connect authentication
profile	Yes	Access user name, picture
email	Yes	Access user email address
offline_access	Optional	Enable refresh tokens

3.5 Technical Requirements

Requirement	**Value/Details**
PKCE Support	Required (code_challenge_method: S256)
Token Storage	Server-side only (HTTP-only cookies)
State Parameter	Will be used (cryptographically random)
Grant Types Needed	authorization_code, refresh_token
Response Type	code

4 What Team A (Java) Provides to Team B

Team A Deliverables

This section details all credentials, endpoints, and configurations that Team A must provide to Team B after client registration.

4.1 OAuth Client Credentials

Credential	Description	Example Format
Client ID	Public identifier for App B	app-b-prod-abc123xyz
Client Secret	Secret key for token exchange	sk_live_AbCdEf123456...

Security Warning

Client Secret Handling:

- NEVER commit to version control
- NEVER expose in frontend/client-side code
- Store in environment variables or secret manager
- Transmit via secure channel (encrypted email, secret manager)

4.2 Authorization Server Base URLs

Environment	Base URL
Production	https://auth.example.com
Staging	https://auth-staging.example.com
Development	http://localhost:4000

4.3 OAuth Endpoints (Team A Must Implement)

4.3.1 Authorization Endpoint

GET /oauth/authorize

Full URL: `https://auth.example.com/oauth/authorize`
Purpose: Initiates OAuth flow, displays login/consent UI
Query Parameters:

Parameter	Required	Type	Description
client_id	Yes	string	App B client ID
redirect_uri	Yes	string	Callback URL (registered)
response_type	Yes	string	Must be code
scope	Yes	string	Space-separated scopes
state	Yes	string	Random CSRF token
code_challenge	Yes	string	PKCE challenge (base64url)
code_challenge_method	Yes	string	Must be S256

Success Response: Redirects to redirect_uri with:

```
https://app-b.com/auth/callback?code=AUTH_CODE&state=
    STATE_VALUE
```

Error Response: Redirects to redirect_uri with:

```
https://app-b.com/auth/callback?error=ERROR_CODE&
    error_description=DESC&state=STATE
```

4.3.2 Token Endpoint

POST /oauth/token

Full URL: `https://auth.example.com/oauth/token`
Purpose: Exchange authorization code for tokens / Refresh tokens
Headers:

```
Content-Type: application/x-www-form-urlencoded
```

Request Body (Authorization Code Grant):

```
grant_type=authorization_code
&code=AUTHORIZATION_CODE
&redirect_uri=https://app-b.com/auth/callback
&client_id=CLIENT_ID
&client_secret=CLIENT_SECRET
&code_verifier=CODE_VERIFIER
```

Request Body (Refresh Token Grant):

```
grant_type=refresh_token
&refresh_token=REFRESH_TOKEN
&client_id=CLIENT_ID
&client_secret=CLIENT_SECRET
```

Success Response (200 OK):

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "dGhpcyBpcyBhIHJlZnJlc2ggdG9rZW4...",
    "scope": "openid profile email",
    "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Error Response (400 Bad Request):

```
{
    "error": "invalid_grant",
    "error_description": "The authorization code has expired"
}
```

4.3.3 UserInfo Endpoint

GET /oauth/userinfo

Full URL: `https://auth.example.com/oauth/userinfo`
Purpose: Retrieve authenticated user profile
Headers:

```
Authorization: Bearer ACCESS_TOKEN
```

Success Response (200 OK):

```
{
    "sub": "user-uuid-12345",
    "email": "user@example.com",
    "email_verified": true,
    "name": "John Doe",
    "given_name": "John",
    "family_name": "Doe",
    "picture": "https://example.com/avatar.jpg",
    "updated_at": 1609459200
}
```

Error Response (401 Unauthorized):

```
{
    "error": "invalid_token",
    "error_description": "The access token is expired"
}
```

4.3.4 Token Revocation Endpoint

POST /oauth/revoke

Full URL: `https://auth.example.com/oauth/revoke`
Purpose: Revoke access or refresh tokens
Headers:

```
Content-Type: application/x-www-form-urlencoded
```

Request Body:

```
token=TOKEN_TO_REVOKE
&token_type_hint=refresh_token
&client_id=CLIENT_ID
&client_secret=CLIENT_SECRET
```

Response: HTTP 200 OK (always, even if token invalid)

4.3.5 JWKS Endpoint

GET /.well-known/jwks.json

Full URL: `https://auth.example.com/.well-known/jwks.json`
Purpose: Public keys for JWT verification
Response:

```
{
    "keys": [
        {
            "kty": "RSA",
            "kid": "key-id-1",
            "use": "sig",
            "alg": "RS256",
            "n": "base64url-encoded-modulus...",
            "e": "AQAB"
        }
    ]
}
```

4.3.6 OpenID Configuration Endpoint

GET /.well-known/openid-configuration

Full URL: `https://auth.example.com/.well-known/openid-configuration`
Purpose: OAuth/OIDC discovery document
Response:

```
{
    "issuer": "https://auth.example.com",
    "authorization_endpoint": "https://auth.example.com/oauth/
   authorize",
    "token_endpoint": "https://auth.example.com/oauth/token",
    "userinfo_endpoint": "https://auth.example.com/oauth/
   userinfo",
    "revocation_endpoint": "https://auth.example.com/oauth/
   revoke",
    "jwks_uri": "https://auth.example.com/.well-known/jwks.
   json",
    "response_types_supported": ["code"],
    "grant_types_supported": ["authorization_code", "
   refresh_token"],
    "scopes_supported": ["openid", "profile", "email"],
    "token_endpoint_auth_methods_supported": ["
   client_secret_post"],
    "code_challenge_methods_supported": ["S256"]
}
```

4.4 Token Configuration

Configuration	Value	Notes
Access Token Lifetime	3600 seconds (1 hour)	Refresh before expiry
Refresh Token Lifetime	604800 seconds (7 days)	Rotate on use
Authorization Code Lifetime	600 seconds (10 min)	Single use only
ID Token Algorithm	RS256	Asymmetric signing
Token Type	Bearer	Standard bearer token

4.5 Frontend Pages (Team A Must Implement)

Page	URL	Description
Login Page	/login	User authentication form
Registration Page	/register	New user registration
Consent Screen	/oauth/consent	Permission approval screen
Password Reset	/forgot-password	Password recovery flow
Error Page	/oauth/error	OAuth error display

5 What Team B (Next.js) Must Implement

Team B Implementation Requirements

This section details all endpoints, pages, and components that Team B must build.

5.1 Backend API Routes

5.1.1 Login Initiation Route

GET /api/auth/login

Purpose: Generate OAuth URL and redirect to App A
Implementation Steps:

1. Generate cryptographically random `state` value
2. Generate PKCE `code_verifier` (43-128 characters)
3. Calculate `code_challenge` = base64url(SHA256(code_verifier))
4. Store state and code_verifier in HTTP-only cookies
5. Build authorization URL with all parameters
6. Redirect user to App A authorization endpoint

Response: HTTP 302 Redirect to App A

5.1.2 OAuth Callback Route

GET /api/auth/callback

Purpose: Handle callback from App A, exchange code for tokens
Query Parameters Received:

- `code` - Authorization code (on success)
- `state` - State parameter for validation
- `error` - Error code (on failure)
- `error_description` - Error details (on failure)

Implementation Steps:

1. Validate `state` matches stored value
2. Check for `error` parameter
3. Retrieve stored `code_verifier`

4. Call App A token endpoint with code
5. Call App A userinfo endpoint
6. Create session with tokens and user info
7. Clear OAuth cookies
8. Redirect to dashboard

Response: HTTP 302 Redirect to /dashboard or /auth/login?error=...

5.1.3 Token Refresh Route

POST /api/auth/refresh

Purpose: Refresh expired access token
Implementation Steps:

1. Get session from request
2. Extract refresh token
3. Call App A token endpoint with refresh_token grant
4. Update session with new tokens
5. Return success/failure

Success Response:

```
{
    "success": true
}
```

Error Response:

```
{
    "error": "refresh_failed",
    "message": "Please login again"
}
```

5.1.4 Logout Route

POST /api/auth/logout

Purpose: Revoke tokens and clear session
Implementation Steps:

1. Get session from request
2. Call App A revoke endpoint (best effort)
3. Clear session cookie
4. Return success

Response:

```
{
    "success": true
}
```

5.1.5 Current User Route

GET /api/auth/me

Purpose: Return current user info
Success Response (200 OK):

```
{
    "isAuthenticated": true,
    "user": {
        "sub": "user-uuid-12345",
        "email": "user@example.com",
        "name": "John Doe",
        "picture": "https://..."
    }
}
```

Unauthenticated Response (401):

```
{
    "isAuthenticated": false,
    "user": null
}
```

5.2 Frontend Pages

Page	Route	Description
Landing Page	/	Public homepage with login button
Login Page	/auth/login	Login button, redirects to /api/auth/login
Callback Page	/auth/callback	Handles OAuth callback (can be API-only)
Dashboard	/dashboard	Protected - requires authentication
Profile	/profile	Protected - user profile display/edit
Settings	/settings	Protected - user settings

5.3 Middleware

middleware.ts

Purpose: Protect routes and handle token refresh
Protected Routes:

```
/dashboard/*
/profile/*
/settings/*
/api/* (except /api/auth/login, /api/auth/callback)
```

Logic:

1. Check if route is protected
2. Verify session exists
3. Check if token is expired
4. If expired, attempt refresh
5. If refresh fails, redirect to login

5.4 Utility Libraries

5.4.1 OAuth Utilities (lib/oauth.ts)

```
// Functions to implement:

// Generate random state string (32+ bytes)
function generateState(): string

// Generate PKCE code verifier (43-128 chars)
```

```
function generateCodeVerifier(): string

// Generate code challenge from verifier
function generateCodeChallenge(verifier: string): Promise<string>

// Parse JWT payload (without verification)
function parseJwt(token: string): object | null
```

5.4.2 Session Management (lib/session.ts)

```
// Functions to implement:

// Create new session from tokens
function createSession(response, data): Promise<void>

// Get session from request
function getSession(request): Promise<SessionData | null>

// Update session with new tokens
function updateSession(response, updates): Promise<void>

// Clear session
function clearSession(response): void

// Check if token is expired
function isTokenExpired(session): boolean
```

5.5 React Components

Component	**Description**
AuthProvider	Context provider for authentication state
useAuth	Hook to access auth context
LoginButton	Button to initiate OAuth flow
LogoutButton	Button to logout user
ProtectedRoute	HOC/wrapper for protected pages
UserProfile	Display user info from session

5.6 Environment Variables

```
# .env.local

# App A (Authorization Server) - Provided by Team A
NEXT_PUBLIC_AUTH_SERVER_URL=https://auth.example.com
AUTH_SERVER_URL=https://auth.example.com

# OAuth Credentials - Provided by Team A (KEEP SECRET!)
```

```
OAUTH_CLIENT_ID=app-b-prod-abc123
OAUTH_CLIENT_SECRET=sk_live_...

# App B Configuration
NEXT_PUBLIC_APP_URL=https://app-b.example.com
OAUTH_REDIRECT_URI=https://app-b.example.com/api/auth/callback

# Session Configuration
SESSION_SECRET=your-32-char-minimum-secret-key

# OAuth Scopes
OAUTH_SCOPES=openid profile email
```

6 What Team A (Java) Must Implement

Team A Implementation Requirements

This section summarizes all components Team A must build for the Authorization Server.

6.1 Database Schema

```
-- Users Table
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    picture_url VARCHAR(500),
    email_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- OAuth Clients Table
CREATE TABLE oauth_clients (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    client_id VARCHAR(100) UNIQUE NOT NULL,
    client_secret_hash VARCHAR(255) NOT NULL,
    client_name VARCHAR(255) NOT NULL,
    redirect_uris TEXT NOT NULL,
    allowed_scopes VARCHAR(500),
    access_token_validity INT DEFAULT 3600,
    refresh_token_validity INT DEFAULT 604800,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Authorization Codes Table
CREATE TABLE authorization_codes (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    code VARCHAR(255) UNIQUE NOT NULL,
    client_id VARCHAR(100) NOT NULL,
    user_id BIGINT NOT NULL,
    redirect_uri VARCHAR(500) NOT NULL,
    scope VARCHAR(500),
    code_challenge VARCHAR(255),
    code_challenge_method VARCHAR(10),
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Refresh Tokens Table
CREATE TABLE refresh_tokens (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    token_hash VARCHAR(255) UNIQUE NOT NULL,
    client_id VARCHAR(100) NOT NULL,
    user_id BIGINT NOT NULL,
    scope VARCHAR(500),
    expires_at TIMESTAMP NOT NULL,
    revoked BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6.2 API Endpoints Summary

Method	Endpoint	Purpose
GET	/oauth/authorize	Start OAuth flow, show login
POST	/oauth/token	Exchange code/refresh for tokens
GET	/oauth/userinfo	Get user profile
POST	/oauth/revoke	Revoke tokens
GET	/.well-known/jwks.json	Public keys for JWT verification
GET	/.well-known/openid-configuration	OIDC discovery document
POST	/auth/register	User registration
POST	/auth/login	User login (session-based)
POST	/auth/logout	User logout

6.3 Security Requirements

- Password hashing with bcrypt (cost factor >= 12)
- JWT signing with RS256 (RSA asymmetric)
- PKCE validation (S256 method required)
- Authorization code: single-use, short-lived (10 min)
- Strict redirect_uri validation
- Rate limiting on sensitive endpoints
- HTTPS required in production

7 Complete URL and Endpoint Reference

7.1 App A (Authorization Server) URLs

Type	**Development**	**Production**
Base URL	http://localhost:4000	https://auth.example.com
OAuth Endpoints		
Authorization	/oauth/authorize	/oauth/authorize
Token	/oauth/token	/oauth/token
UserInfo	/oauth/userinfo	/oauth/userinfo
Revoke	/oauth/revoke	/oauth/revoke
JWKS	/.well-known/jwks.json	/.well-known/jwks.json
OIDC Config	/.well-known/openid-configuration	/.well-known/openid-configuration
Frontend Pages		
Login	/login	/login
Register	/register	/register
Consent	/oauth/consent	/oauth/consent

7.2 App B (Next.js Client) URLs

Type	**Development**	**Production**
Base URL	http://localhost:3000	https://app-b.example.com
API Routes		
Login Init	/api/auth/login	/api/auth/login
Callback	/api/auth/callback	/api/auth/callback
Refresh	/api/auth/refresh	/api/auth/refresh
Logout	/api/auth/logout	/api/auth/logout
Current User	/api/auth/me	/api/auth/me
Frontend Pages		
Landing	/	/
Login	/auth/login	/auth/login
Callback	/auth/callback	/auth/callback
Dashboard	/dashboard	/dashboard
Profile	/profile	/profile

8 Integration Testing Checklist

8.1 Happy Path Tests

No.	Test Case	Pass/Fail
1	User can click login and is redirected to App A	☐
2	User can authenticate on App A login page	☐
3	User sees consent screen with correct permissions	☐
4	User is redirected back to App B with code	☐
5	App B successfully exchanges code for tokens	☐
6	App B retrieves user info from App A	☐
7	User session is created in App B	☐
8	User can access protected routes	☐
9	Token refresh works before expiry	☐
10	Logout clears session and revokes tokens	☐

8.2 Error Handling Tests

No.	Test Case	Pass/Fail
1	Invalid client_id shows appropriate error	☐
2	Invalid redirect_uri is rejected	☐
3	State mismatch is detected	☐
4	Expired authorization code fails gracefully	☐
5	Invalid authorization code fails gracefully	☐
6	Expired access token triggers refresh	☐
7	Expired refresh token redirects to login	☐
8	User denial on consent returns error	☐
9	Network errors are handled gracefully	☐
10	Invalid PKCE verifier is rejected	☐

9 Security Checklist

9.1 Team A Security Requirements

Requirement	**Complete**
Passwords hashed with bcrypt (cost >= 12)	☐
JWTs signed with RS256	☐
Authorization codes are single-use	☐
Authorization codes expire in <= 10 minutes	☐
PKCE validation implemented	☐
Redirect URI strictly validated	☐
Rate limiting on login/token endpoints	☐
HTTPS enforced in production	☐
Client secrets hashed in database	☐
Token revocation fully implemented	☐

9.2 Team B Security Requirements

Requirement	**Complete**
Client secret stored in environment variables only	☐
Client secret NEVER exposed in frontend code	☐
State parameter validated on callback	☐
PKCE implemented (S256)	☐
Tokens stored in HTTP-only cookies	☐
Session cookie has Secure flag (production)	☐
Session cookie has SameSite=Lax	☐
Token refresh happens server-side	☐
All OAuth calls made server-side	☐
HTTPS enforced in production	☐

10 Appendices

10.1 Error Codes Reference

Error Code	**Description**
invalid_request	Missing or invalid parameter
unauthorized_client	Client not authorized for this grant type
access_denied	User denied authorization
unsupported_response_type	Response type not supported
invalid_scope	Requested scope is invalid
server_error	Authorization server error
temporarily_unavailable	Server temporarily unavailable
invalid_client	Client authentication failed
invalid_grant	Grant (code/refresh token) is invalid
unsupported_grant_type	Grant type not supported
invalid_token	Access token is invalid
insufficient_scope	Token does not have required scope

10.2 Glossary

Term	**Definition**
OAuth 2.0	Industry-standard authorization framework
OpenID Connect	Identity layer built on OAuth 2.0
Authorization Code	Temporary code exchanged for tokens
Access Token	Credential for accessing protected resources
Refresh Token	Credential for obtaining new access tokens
ID Token	JWT containing user identity claims
PKCE	Proof Key for Code Exchange (security extension)
State	Random value preventing CSRF attacks
Scope	Permissions requested by client
JWKS	JSON Web Key Set (public keys)
Bearer Token	Token granting access to bearer

10.3 Contact Information

Team	**Role**	**Contact**
Team A	OAuth Server Development	teamA@example.com
Team B	Client Integration	teamB@example.com
Security	Security Review	security@example.com
DevOps	Deployment	devops@example.com

Document Approval

Role	Name	Signature	Date
Team A Lead			
Team B Lead			
Security Architect			
Project Manager			

End of Document