DesignLibre

OAuth Authentication Setup Guide

Integrating Claude API with OAuth 2.0 + PKCE

This guide covers:

- Registering your application with Anthropic
- Configuring OAuth client credentials
- Creating the OAuth callback page
- Testing the authentication flow

Version 1.0
January 2026

Contents

1 Introduction

DesignLibre integrates with Claude AI through Anthropic's API. While API keys provide a simple authentication method, OAuth 2.0 with PKCE (Proof Key for Code Exchange) offers a more secure, user-friendly authentication flow—similar to how Claude Code CLI authenticates users.

1.1 Why OAuth?

- **No API key management**: Users authenticate with their Anthropic account
- **Secure**: PKCE prevents authorization code interception attacks
- **User-friendly**: Familiar "Sign in with..." flow
- **Token refresh**: Automatic token renewal without re-authentication
- **Scoped access**: Request only the permissions needed

1.2 Prerequisites

Before proceeding, ensure you have:

1. An Anthropic account with API access
2. A deployed web application with HTTPS (required for OAuth)
3. Access to your web server to add the callback page

Warning

OAuth requires HTTPS in production. For local development, `localhost` is typically allowed, but production deployments must use a valid SSL certificate.

2 Registering Your Application with Anthropic

2.1 Step 1: Access the Anthropic Console

Navigate to the Anthropic Console at:

`https://console.anthropic.com`

Sign in with your Anthropic account credentials.

2.2 Step 2: Navigate to OAuth Applications

1. Click on **Settings** in the left sidebar
2. Select **OAuth Applications** (or **API & OAuth**)
3. Click **Create New Application**

Info

If you don't see the OAuth Applications option, your account may need to be upgraded or you may need to request access from Anthropic. Contact support@anthropic.com for

assistance.

2.3 Step 3: Configure Application Details

Fill in the application registration form:

Field	Value
Application Name	DesignLibre
Description	AI-powered design tool with Claude integration
Application Type	Single Page Application (SPA)
Homepage URL	`https://your-domain.com`
Redirect URIs	`https://your-domain.com/auth/callback`

Table 1: OAuth Application Configuration

2.4 Step 4: Configure Redirect URIs

Add all redirect URIs where your application will receive OAuth callbacks:

```
# Production
https://designlibre.app/auth/callback
https://www.designlibre.app/auth/callback

# Staging
https://staging.designlibre.app/auth/callback

# Local Development
http://localhost:3000/auth/callback
http://localhost:5173/auth/callback
```

Example Redirect URIs

Warning

Each environment needs its own redirect URI. The redirect URI in the OAuth request must **exactly match** one of the registered URIs—including trailing slashes and protocol.

2.5 Step 5: Request Scopes

Select the OAuth scopes your application needs:

Scope	Description	Required
`api:read`	Read access to Claude API	Yes
`api:write`	Write access (send messages)	Yes
`user:read`	Read user profile information	Optional

Table 2: OAuth Scopes

2.6 Step 6: Obtain Client Credentials

After creating the application, you'll receive:

- **Client ID**: A public identifier for your application

- **Client Secret**: *Not used for public clients with PKCE*

Success

Save your **Client ID**! You'll need it to configure the OAuth client in DesignLibre.
Example: `dl_oauth_abc123xyz789`

2.7 Step 7: Update DesignLibre Configuration

Update the OAuth client configuration in your codebase:

```
const ANTHROPIC_OAUTH_CONFIG: OAuthConfig = {
  clientId: 'your-client-id-here',  // Replace with your Client ID
  authorizationEndpoint:
     'https://console.anthropic.com/oauth/authorize',
  tokenEndpoint: 'https://console.anthropic.com/oauth/token',
  redirectUri: `${window.location.origin}/auth/callback`,
  scopes: ['api:read', 'api:write'],
};
```

src/ai/auth/oauth-client.ts

3 Creating the OAuth Callback Page

The callback page receives the authorization code from Anthropic and posts it back to the parent window (your main application).

3.1 Step 1: Create the Callback HTML File

Create a new file at `/auth/callback/index.html` (or configure your router to serve this page at `/auth/callback`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
     initial-scale=1.0">
  <title>Authenticating... - DesignLibre</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
                   Roboto, Oxygen, Ubuntu, sans-serif;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
    }

    .container {
```

```
      background: white;
      padding: 3rem;
      border-radius: 16px;
      box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
      text-align: center;
      max-width: 400px;
      width: 90%;
    }

    .spinner {
      width: 50px;
      height: 50px;
      border: 4px solid #e0e0e0;
      border-top-color: #667eea;
      border-radius: 50%;
      animation: spin 1s linear infinite;
      margin: 0 auto 1.5rem;
    }

    @keyframes spin {
      to { transform: rotate(360deg); }
    }

    h1 {
      color: #333;
      font-size: 1.5rem;
      margin-bottom: 0.5rem;
    }

    p {
      color: #666;
      font-size: 0.95rem;
    }

    .error {
      background: #fee;
      border: 1px solid #fcc;
      color: #c00;
      padding: 1rem;
      border-radius: 8px;
      margin-top: 1rem;
      display: none;
    }

    .error.visible {
      display: block;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="spinner" id="spinner"></div>
    <h1 id="title">Authenticating...</h1>
    <p id="message">Please wait while we complete the sign-in
       process.</p>
    <div class="error" id="error"></div>
  </div>

```

```
<script>
  (function() {
    // Parse URL parameters
    const params = new URLSearchParams(window.location.search);
    const code = params.get('code');
    const state = params.get('state');
    const error = params.get('error');
    const errorDescription = params.get('error_description');

    // UI elements
    const spinner = document.getElementById('spinner');
    const title = document.getElementById('title');
    const message = document.getElementById('message');
    const errorDiv = document.getElementById('error');

    function showError(msg) {
      spinner.style.display = 'none';
      title.textContent = 'Authentication Failed';
      message.textContent = 'Unable to complete sign-in.';
      errorDiv.textContent = msg;
      errorDiv.classList.add('visible');
    }

    function showSuccess() {
      spinner.style.display = 'none';
      title.textContent = 'Success!';
      message.textContent = 'You can close this window.';
    }

    // Handle OAuth error response
    if (error) {
      showError(errorDescription || error);
      return;
    }

    // Validate required parameters
    if (!code || !state) {
      showError('Missing authorization code or state parameter.');
      return;
    }

    // Post message to parent window
    if (window.opener) {
      try {
        window.opener.postMessage({
          type: 'oauth_callback',
          code: code,
          state: state
        }, window.location.origin);

        showSuccess();

        // Auto-close after short delay
        setTimeout(() => {
          window.close();
        }, 1500);
      } catch (err) {
        showError('Failed to communicate with the application.');
```

```
        }
      } else {
        showError('This page must be opened from DesignLibre.');
      }
    })();
  </script>
</body>
</html>
```

public/auth/callback/index.html

3.2 Step 2: Understanding the Callback Flow

The callback page performs these steps:

1. **Parse URL Parameters**: Extract `code` and `state` from the query string
2. **Validate State**: The state parameter prevents CSRF attacks
3. **Post to Parent**: Send the code to the opener window via `postMessage`
4. **Close Window**: Automatically close after successful handoff

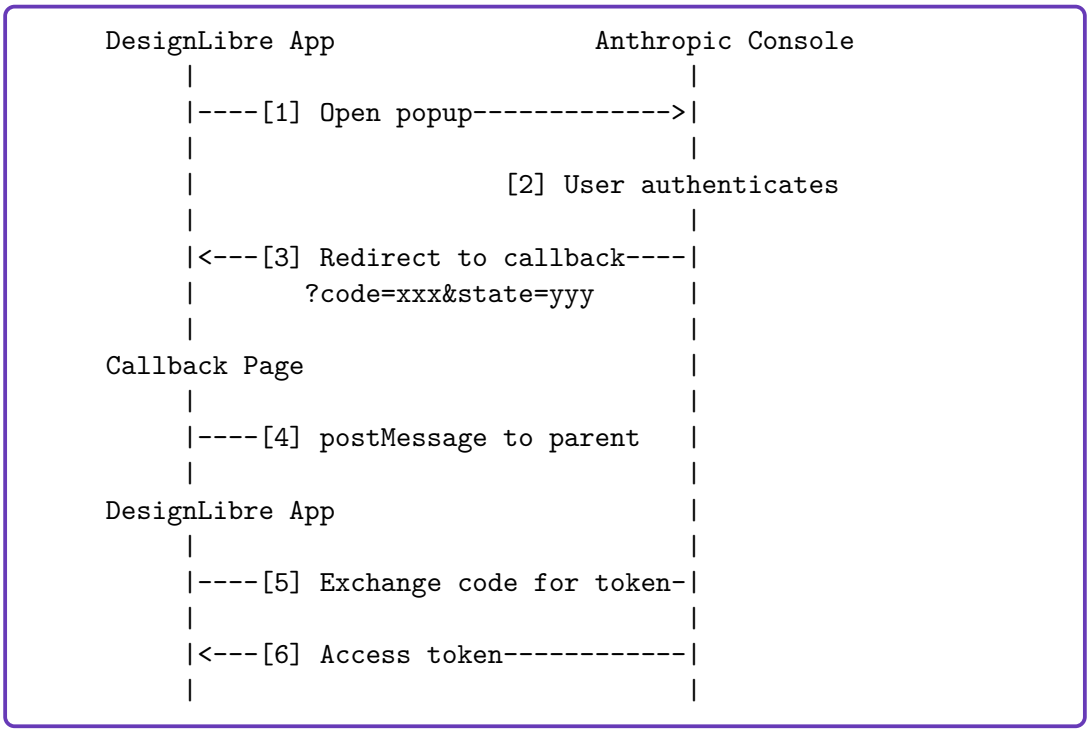

Figure 1: OAuth 2.0 + PKCE Flow Diagram

3.3 Step 3: Security Considerations

Warning

Important Security Notes:

- Always validate the `origin` in `postMessage` handlers
- Never log or expose the authorization code
- Use HTTPS in production
- The state parameter must match what was sent in the authorization request

3.4 Step 4: Server Configuration

Ensure your web server serves the callback page correctly:

3.4.1 Nginx Configuration

```
location /auth/callback {
    try_files $uri $uri/ /auth/callback/index.html;
}
```

nginx.conf

3.4.2 Apache Configuration

```
RewriteEngine On
RewriteRule ^auth/callback$ /auth/callback/index.html [L]
```

.htaccess

3.4.3 Vite/SPA Configuration

For single-page applications using Vite:

```
export default defineConfig({
  // ... other config
  build: {
    rollupOptions: {
      input: {
        main: resolve(__dirname, 'index.html'),
        callback: resolve(__dirname, 'auth/callback/index.html'),
      },
    },
  },
});
```

vite.config.ts

4 Testing the Authentication Flow

4.1 Step 1: Local Development Testing

1. Start your development server

2. Open the AI Assistant panel
3. Click "Sign in with Claude"
4. Verify the popup opens to Anthropic's authorization page
5. Complete authentication
6. Verify the callback page appears briefly
7. Verify the popup closes automatically
8. Check that the settings panel shows "Connected"

4.2 Step 2: Verify Token Storage

Open your browser's Developer Tools and check localStorage:

```
// Check if tokens are stored
const tokenData = localStorage.getItem('designlibre:auth:anthropic');
if (tokenData) {
  const tokens = JSON.parse(atob(tokenData));
  console.log('Access Token:', tokens.accessToken ? 'Present' :
     'Missing');
  console.log('Refresh Token:', tokens.refreshToken ? 'Present' :
     'Missing');
  console.log('Expires At:', new Date(tokens.expiresAt));
}
```

Browser Console

4.3 Step 3: Test API Calls

Verify that API calls use the OAuth token:

1. Open Network tab in Developer Tools
2. Send a message in the AI Assistant
3. Find the request to `api.anthropic.com`
4. Verify the `Authorization: Bearer ...` header is present

4.4 Step 4: Test Token Refresh

To test token refresh:

```
// Manually expire the token for testing
const tokenData = localStorage.getItem('designlibre:auth:anthropic');
if (tokenData) {
  const tokens = JSON.parse(atob(tokenData));
  tokens.expiresAt = Date.now() - 1000; // Set to past
  localStorage.setItem(
    'designlibre:auth:anthropic',
    btoa(JSON.stringify(tokens))
  );
  console.log('Token marked as expired. Next API call will trigger
     refresh.');
}
```

Browser Console - Force Token Expiry

5 Troubleshooting

5.1 Common Issues

Issue	Solution
"Invalid redirect URI"	Ensure the redirect URI exactly matches what's registered in Anthropic Console, including protocol and trailing slashes
"Invalid client ID"	Verify the client ID in `oauth-client.ts` matches the one from Anthropic Console
Popup blocked	Users may need to allow popups for your domain
"State mismatch"	Clear session storage and try again; may indicate CSRF attempt
Token refresh fails	The refresh token may have expired; user needs to re-authenticate
CORS errors	Ensure your domain is allowed in Anthropic's OAuth app settings

Table 3: Troubleshooting Guide

5.2 Debug Mode

Enable debug logging by adding to your OAuth client:

```
// In oauth-client.ts, add to startAuth():
console.log('[OAuth] Starting auth flow', {
  clientId: this.config.clientId,
  redirectUri: this.config.redirectUri,
  scopes: this.config.scopes,
});

// In handleCallback():
console.log('[OAuth] Received callback', { code: '***', state });
console.log('[OAuth] Token exchange successful', {
  hasAccessToken: !!tokens.accessToken,
  hasRefreshToken: !!tokens.refreshToken,
  expiresAt: tokens.expiresAt,
});
```

Debug Logging

6 Production Checklist

Before deploying to production, verify:

- OAuth application registered with Anthropic
- Production redirect URI added to allowed URIs
- HTTPS configured and working
- Callback page deployed and accessible
- Client ID updated in production config

Token storage working correctly

Error handling tested

Token refresh tested

Sign out functionality working

Analytics/monitoring in place (optional)

7 Appendix: Complete OAuth Client Code

For reference, here is the complete OAuth client implementation:

```
export class AnthropicOAuthClient {
  private config: OAuthConfig;
  private authWindow: Window | null = null;

  constructor(config: Partial<OAuthConfig> = {}) {
    this.config = { ...ANTHROPIC_OAUTH_CONFIG, ...config };
  }

  async startAuth(): Promise<void> {
    const state = generateRandomString(32);
    const verifier = generateCodeVerifier();
    const challenge = await generateCodeChallenge(verifier);

    sessionStorage.setItem(STATE_STORAGE_KEY, state);
    sessionStorage.setItem(VERIFIER_STORAGE_KEY, verifier);

    const params = new URLSearchParams({
      response_type: 'code',
      client_id: this.config.clientId,
      redirect_uri: this.config.redirectUri,
      scope: this.config.scopes.join(' '),
      state,
      code_challenge: challenge,
      code_challenge_method: 'S256',
    });

    const authUrl = `${this.config.authorizationEndpoint}?${params}`;

    // Open centered popup
    const width = 500, height = 700;
    const left = window.screenX + (window.outerWidth - width) / 2;
    const top = window.screenY + (window.outerHeight - height) / 2;

    this.authWindow = window.open(
      authUrl,
      'Claude Authentication',
      `width=${width},height=${height},left=${left},top=${top}`
    );

    return this.waitForCallback();
  }

  // ... rest of implementation
}
```

src/ai/auth/oauth-client.ts (excerpt)

Questions or Issues?

Open an issue at:
https://github.com/johnjanik/libredesign.app/issues