Third-Party Integration Guide for Nexira Game Integration

(Version 1.0)

Objective

This document is a standalone integration guide for third-party game teams that want to connect their game to Nexira authentication and cross-game SSO.

Use this guide as the single source for partner onboarding. It includes the required architecture, frontend SDK usage, backend SDK/API usage, security requirements, error handling, and go-live checklist.

This document is safe to share with external engineering partners. Replace example domains, client IDs, and callback URLs with the production values provided during partner registration.

Quick Start Summary

1. Register your game with Nexira and receive a client_id and client_secret.
2. Configure your exact callback URL, for example https://game-x.example.com/auth/callback.
3. From your game frontend, redirect users to Nexira authorization.
4. Nexira redirects the user back to your callback URL with an authorization code.
5. Your game backend exchanges the code for a Nexira token using client_secret.
6. Your backend creates your own game session and returns only that game session to the frontend.

Important: the frontend must never receive the client_secret or raw Nexira access token.

Integration Roles

Owner	Responsibility
Game frontend	Starts login, stores/validates state, handles callback URL, sends code to game backend
Game backend	Stores client_secret, exchanges authorization code, creates game session, protects game APIs
Nexira Auth	Authenticates user, issues authorization code, exchanges code/token, supports optional SSO exchange

---

1) What You Need Before Integration

- A registered game client (client_id) from Nexira admin registration.
- A one-time client_secret stored only in your backend.
- An exact registered callback URL (redirect_uri).
- Auth public base URL for browser redirect (AUTH_PUBLIC_BASE_URL).
- Auth backend base URL for server-to-server exchange (AUTH_BE_BASE_URL).

Example values:

- AUTH_PUBLIC_BASE_URL=https://auth.example.com
- AUTH_BE_BASE_URL=https://auth.example.com/api

Recommended partner environment variables:

```
NEXIRA_CLIENT_ID=game_x
NEXIRA_CLIENT_SECRET=replace_with_secret_from_nexira
NEXIRA_REDIRECT_URI=https://game-x.example.com/auth/callback
NEXIRA_AUTH_PUBLIC_BASE_URL=https://auth.example.com
NEXIRA_AUTH_BE_BASE_URL=https://auth.example.com/api
```

Store NEXIRA_CLIENT_SECRET only in backend/server-side secret storage.

---

2) Integration Architecture (Required)

- Frontend redirects users to Nexira authorize endpoint.
- Nexira login flow returns code to your callback URL.
- Your frontend sends code to your backend.
- Your backend exchanges code with Nexira Auth backend using client_secret.
- Your backend creates its own game session and returns only that game session to frontend.

Do not expose client_secret or raw Nexira token to frontend.

---

3) Frontend Integration (Web SDK Contract)

3.1 Initialize SDK

Configure the frontend SDK with your game registration values:

```
interface NexiraConfig {
  clientId: string;
  redirectUri: string;
  authBaseUrl?: string; // default: https://auth.shibawars.com
  gameBeUrl: string;
  gameId: number;
}
```

Example:

```
Nexira.init({
  clientId: 'game_x',
  redirectUri: 'https://game-x.example.com/auth/callback',
  authBaseUrl: 'https://auth.example.com',
  gameBeUrl: 'https://api.game-x.example.com',
  gameId: 1001,
});
```

3.2 Start Login Redirect

Browser redirect target:

```
GET {AUTH_PUBLIC_BASE_URL}/oauth/authorize
```

Query params:

- client_id
- redirect_uri (URL encoded)
- response_type=code
- state (opaque random CSRF value)

Example:

```
https://auth.example.com/oauth/authorize?client_id=game_x&redirect_uri=https%3A%2F%2Fgame-x.example.com%2Fcallback&response_type=code&state=random_state_123
```

SDK-style usage:

```
await Nexira.login();
```

The SDK should generate and store a random state value before redirecting. If you implement the redirect manually, generate a cryptographically random state, store it temporarily in browser storage, and validate it on callback.

3.3 Handle Callback in FE

At callback page:

1. Parse code, state, and optional error from URL.
2. Validate state against stored value.
3. Send code (and state if your backend validates it) to your backend callback endpoint.
4. Remove auth params from browser URL.

Expected callback result shape:

```
interface CallbackResult {
  success: boolean;
  sessionToken?: string;
  error?: string;
```

```
}
```

Example callback handling:

```
const result = await Nexira.handleCallback();

if (!result.success) {
  console.error('Nexira login failed:', result.error);
  // Show retry/login message to user.
  return;
}

// Store only your game session token in frontend storage.
// Do not store or expose raw Nexira tokens in the browser.
saveGameSession(result.sessionToken);
```

3.4 FE Error Mapping

Condition	Error
URL contains error	Use URL error value
Missing code/state	invalid_callback
State mismatch	state_mismatch
Backend exchange/network failure	upstream_error or network_error

3.5 SPA Callback Routing Note

If your FE is SPA-based:

- Use root callback URL, or
- Ensure callback path rewrites to app entry (index.html).

Without rewrite, callback path can return 404 before your FE handles query params.

---

4) Backend Integration (Java SDK Contract)

Your backend is responsible for exchanging the short-lived authorization code with Nexira and creating your own game session.

Recommended backend callback endpoint:

```
POST /login/callback
Content-Type: application/json

{
  "code": "AUTH_CODE_FROM_CALLBACK",
  "state": "STATE_FROM_CALLBACK"
}
```

Recommended backend response to frontend:

```
{
  "success": true,
  "sessionToken": "YOUR_GAME_SESSION_TOKEN"
}
```

On failure:

```
{
  "success": false,
  "error": "invalid_callback"
}
```

4.1 Initialize SDK Client

```
NexiraAuthClient authClient = new
NexiraAuthClient("https://auth.example.com/api");
```

4.2 Exchange Authorization Code (Flow 1)

Use:

```
NexiraToken token = authClient.exchangeCodeForToken(clientId, clientSecret,
code);
```

Equivalent HTTP API:

```
POST {AUTH_BE_BASE_URL}/oauth/token
Content-Type: application/json

{
 "grantType": "authorization_code",
 "code": "AUTH_CODE",
 "clientId": "YOUR_CLIENT_ID",
 "clientSecret": "YOUR_CLIENT_SECRET"
}
```

Expected successful response shape:

```
{
 "accessToken": "NEXIRA_ACCESS_TOKEN",
 "tokenType": "Bearer",
 "expiresIn": 3600,
 "subject": "nexira_user_id"
}
```

Token handling rules:

- Store Nexira token server-side only.
- Use the Nexira subject/user ID to map or create a user in your game backend.
- Issue your own game session token to the frontend.
- Do not use the Nexira token as the browser/game client session token.

4.3 Optional Cross-Game SSO Exchange (Flow 2)

Use:

```
NexiraToken token = authClient.exchangeSsoToken(targetClientId,
targetSecret, sourceClientId, sourceToken);
```

Equivalent HTTP API:

```
POST {AUTH_BE_BASE_URL}/sso/exchange
Content-Type: application/json
```

```
{
  "sourceToken": "SOURCE_GAME_TOKEN",
  "sourceClientId": "source_game",
  "targetClientId": "target_game",
  "clientSecret": "TARGET_GAME_SECRET"
}
```

If SSO exchange fails with 401 invalid_token, redirect user through Flow 1 again.

Expected SSO exchange success response uses the same token shape as the authorization-code exchange.

Recommended SSO fallback logic:

1. Try SSO exchange.
2. If exchange succeeds, create game session.
3. If exchange returns 401 invalid_token, start normal login redirect.
4. For other errors, show retry or contact support message.

---

5) Minimal End-to-End Example

Frontend (pseudocode)

```
// 1) Login button
Nexira.login();

// 2) On callback route/page
const result = await Nexira.handleCallback();
if (result.success) {
  // Use your game session token for game APIs/WebSocket
}
```

Backend (Java)

```
public LoginResponse handleCallback(String code) {
    NexiraToken nexiraToken = authClient.exchangeCodeForToken(clientId,
clientSecret, code);

    // Store nexiraToken.getAccessToken() only in backend storage
```

```
    // Create your own game session and return it to FE
    String gameSession = createGameSession(nexiraToken.getSubject());
    return new LoginResponse(gameSession);
}
```

Backend HTTP-only Example Flow

If you do not use the Java SDK, call Nexira Auth directly from your backend:

```
curl -X POST "https://auth.example.com/api/oauth/token" \
  -H "Content-Type: application/json" \
  -d '{
    "grantType": "authorization_code",
    "code": "AUTH_CODE",
    "clientId": "game_x",
    "clientSecret": "YOUR_CLIENT_SECRET"
  }'
```

After a successful response, create a game session in your own backend and return only that session to your frontend.

---

6) Security Rules (Must Follow)

- Keep client_secret in backend only.
- Never return raw Nexira token to frontend.
- Preserve and validate state to mitigate CSRF.
- Redact tokens/secrets in logs and error traces.
- Use HTTPS in production for all auth and callback URLs.
- Register exact callback URLs. Do not use wildcard callback URLs in production.
- Apply CORS only to trusted game frontend origins.
- Set reasonable backend timeouts and retries for server-to-server calls.
- Treat authorization codes as one-time-use and short-lived.

- Rotate client_secret immediately if exposed.

---

7) Error Handling Reference

Frontend callback errors

Error	Meaning	Recommended handling
invalid_callback	Callback URL is missing required code or state	Restart login
state_mismatch	Returned state does not match stored state	Clear local auth state and restart login
access_denied	User denied or login was cancelled	Show login retry option
network_error	Frontend could not reach game backend	Retry or show temporary outage message
upstream_error	Game backend could not complete Nexira exchange	Retry or contact support

Backend exchange errors

HTTP status	Example error	Meaning	Recommended handling

400	invalid_request	Missing or malformed request field	Validate request body and configuration
400	invalid_grant	Code is expired, reused, or invalid	Restart login
401	invalid_client	Invalid client_id or client_secret	Check backend secret configuration
401	invalid_token	SSO source token is invalid or expired	Fall back to normal login
500	server_error	Temporary upstream error	Retry with backoff or show temporary outage message

Recommended public error behavior:

- Show user-friendly messages to players.
- Do not display raw backend/Nexira error payloads to users.
- Log correlation IDs if available.
- Redact code, client_secret, access tokens, and session tokens from logs.

---

8) Go-Live Checklist

Registration and configuration

Register game and obtain client_id + client_secret.
Register exact redirect_uri.
Configure AUTH_PUBLIC_BASE_URL for browser redirects.

Configure AUTH_BE_BASE_URL for backend token exchange.
Store client_secret in backend secret storage only.

Frontend

Implement FE redirect to /oauth/authorize.
Implement FE callback parsing + state verification.
Remove code, state, and error params from browser URL after processing.
Confirm SPA callback routes are served correctly and do not return 404.

Backend

Implement BE code exchange with /oauth/token.
Store Nexira token server-side only.
Issue your own game session token to FE.
Implement optional /sso/exchange fallback logic.
Verify logs do not contain secrets or tokens.
Configure CORS for trusted game frontend origins only.
Configure retry/timeout handling for Nexira server-to-server calls.

Production validation

Test successful login from a clean browser session.
Test cancelled login or denied login.
Test expired/reused authorization code handling.
Test state mismatch handling.
Test backend behavior when Nexira Auth is temporarily unavailable.
Confirm HTTPS is used for all production URLs.