# Okta Integration Plan

**For: MIS + Entitlement + Internal Applications (RBC Insight, SPARC, ATIS, ADS)** **Purpose**: Implement secure, unified identity and access management across internal tools using Okta.

This integration aims to centralize authentication and authorization for all internal applications. Both MIS and Entitlement systems will serve as gatekeepers for access provisioning and will rely on Okta as the single source of identity. All downstream applications (RBC Insight, SPARC, ATIS, ADS) will consume user identity and access rights from this shared identity provider setup.

## 1. Okta as the Central Identity Provider (IdP)

**Key Features:**

| **Feature** | **Description** |
| --- | --- |
| Application Type | OIDC Web Application (React frontend + Spring Boot backend) |
| Grant Type | Authorization Code Flow with PKCE |
| Okta Tenant | Example: https://your-company.okta.com |
| User Directory | All users managed centrally in Okta |
| Groups | Used to represent roles and app-specific access |
| Authentication Policies | Configurable MFA, session timeouts, and IP restrictions as required |

## 

## 

## 2. Application Mapping under Okta

| **Application** | **Type** | **Okta Integration Method** | **Provisioning Managed By** |
| --- | --- | --- | --- |
| MIS | Internal Web App | OIDC | MIS Backend (Spring Boot) |
| Entitlement | Internal Web App | OIDC | Entitlement Backend |
| RBC Insight | Internal Web App | OIDC or SAML | MIS / Entitlement |
| SPARC, ATIS, ADS | Internal Tools | OIDC (or Okta SSO) | Entitlement System |

### Notes:

* Applications can either authenticate directly via Okta or rely on a secure token exchange from the MIS or Entitlement system acting as an authentication broker.
* Apps not refactored for OIDC may use SAML or Secure Web Authentication (SWA).

## 3. Okta Application Configuration

Each internal app will be registered as a separate Okta application. MIS and Entitlement are mandatory. Others (e.g., RBC Insight) can be added progressively.

**Configuration Details:**

| **Setting** | **Value** |
| --- | --- |
| Redirect URIs | https://yourdomain.com/callback |
| Logout Redirect URIs | https://yourdomain.com/logout |
| Client Type | Public (for React); Confidential (for Spring Boot) |
| Scopes | openid, profile, email, groups, offline\_access |

**Enabling Group Claims in ID Token:** To propagate group membership in the login token, configure custom claims like:

json

CopyEdit

"groups": ["MIS\_Admins", "Entitlement\_Admins", "SPARC\_Viewers"]

This allows both frontend and backend to derive roles and permissions from the token.

## 4. User Provisioning

Provisioning is automated through Okta SCIM integration or the Admin API. The MIS and Entitlement systems handle this logic through their respective interfaces and workflows.

### Internal Users

* Users created via the MIS system (e.g., IT team onboarding)
* Automatically added to Okta with relevant groups such as MIS\_Research, SPARC\_Approver, etc.

### External Users

* Created through Entitlement application UI
* Okta Admin API is used to:  
  + Create the user
  + Assign relevant default group(s)
  + Trigger any onboarding workflows

**Sample Java Code for User Creation:**

java

CopyEdit

@PostMapping("/users")

public ResponseEntity<?> createUser(@RequestBody User user) {

oktaClient.createUser(user); // Uses Okta Java SDK

assignGroup(user, "SPARC\_Viewers");

return ResponseEntity.ok().build();

}

## 5. Authentication Flow (OIDC with PKCE)

### Frontend (React: MIS / Entitlement)

1. User initiates login via Okta SDK
2. Redirect to Okta with PKCE challenge
3. Okta presents login screen (with MFA if required)
4. Upon successful login, Okta redirects back with an authorization code
5. Code is exchanged for access and ID tokens
6. Tokens are stored in secure memory or session

### Backend (Spring Boot: MIS / Entitlement)

* Verifies the access token on every API request
* Extracts user information from token claims
* Maps group claims to roles
* Enforces route-level and business logic authorization accordingly

## 6. Authorization (Role-Based Access Control)

All role-based access is derived from the group claim embedded in the ID token. Each application enforces its own permissions.

**Examples:**

| **Group Name** | **Permissions Description** |
| --- | --- |
| MIS\_Admins | Manage all MIS users and access settings |
| Entitlement\_Admins | Review and approve entitlement requests |
| SPARC\_Approver | Approve user access to SPARC |
| RBCInsight\_User | Allow access to RBC Insight application |

**Enforcement Locations:**

* Frontend: via route guards and UI logic
* Backend: using annotations like @PreAuthorize("hasRole('MIS\_Admin')")

## 7. Optional Directory Sync (HR / LDAP Integration)

For enterprise environments where HR systems (e.g., Workday) or internal directories (LDAP / Active Directory) exist, Okta supports syncing via prebuilt connectors.

Example Use Case:

* Automatically assign users in the Research department to MIS\_Research group
* Users updated in AD automatically reflect in Okta and downstream access control

## 8. Token and Session Lifecycle

| **Token Type** | **Purpose** | **Expiry** |
| --- | --- | --- |
| ID Token | Contains user identity claims | 1 hour |
| Access Token | Used to authorize API requests | 1 hour |
| Refresh Token | Optional for long-term sessions | 30 days |

**Session Management:**

* Logout endpoint must call Okta's .well-known/logout URL
* Properly revokes tokens and clears Okta session

## 9. SDKs and Tools Used

### Frontend (React)

* @okta/okta-auth-js
* @okta/okta-react

### Backend (Spring Boot)

* spring-security-oauth2-client
* okta-spring-boot-starter

### Provisioning & Admin

* okta-sdk-java (to create users, assign groups, etc.)

Summary of Responsibilities

| **Responsibility Area** | **Owned By** |
| --- | --- |
| User Identity Source | Okta |
| User Creation | MIS / Entitlement |
| Access Provisioning | Entitlement workflows |
| Authentication Mechanism | Okta (OIDC with PKCE) |
| Group / Role Management | Okta Admin + APIs |
| Authorization Enforcement | Individual applications (React + Spring Boot) |

### How Okta (OIDC with PKCE) works:

1. **Client (e.g., React app)** generates:  
   * code\_verifier: a random string
   * code\_challenge: a hash of that string (SHA256)
   * The client sends the user to the Okta **authorization endpoint**:  
     CopyEdit  
     https://your-okta-domain.com/oauth2/default/v1/authorize

?client\_id=...

&code\_challenge=... (from step 1)

&code\_challenge\_method=S256

&response\_type=code

1. The user logs in on Okta and is redirected back to your app **with an authorization code**.

Your app then exchanges the authorization code **(plus the original code\_verifier)** at the token endpoint:  
CopyEdit  
POST /token

{

code: auth\_code\_from\_redirect,

code\_verifier: original\_random\_string,

...

}

1. Your app then exchanges the authorization code **(plus the original code\_verifier)** at the token endpoint:

POST /token

{

code: auth\_code\_from\_redirect,

code\_verifier: original\_random\_string,

...

}

1. Okta verifies the code challenge and returns:  
   * **ID Token** (who the user is)
   * **Access Token** (what the user can access)
   * **Refresh Token** (if enabled)