TFG — Finalización de la integración de Microsoft Entra ID en DecentraLabs: del login a las reservas completas

Duración estimada: 4–5 meses
Dificultad: Media-alta. Requiere comprender el flujo de identidad completo del sistema (OIDC, JWT, WebAuthn, intents EIP-712, Spring Boot). El login ya está implementado; el trabajo consiste en completar la integración end-to-end.
Idioma de referencia del código fuente: JavaScript (Next.js 15), Java (Spring Boot 3.5 / Java 21)
CONTACTO: Luis de la Torre (ldelatorre@dia.uned.es)

---

1. Resumen

DecentraLabs es un marketplace de laboratorios remotos y modelos de simulación. Tiene implementado el **flujo de login con Microsoft Entra ID** en la rama `feature/entra-id`: el usuario puede autenticarse via OIDC Authorization Code + PKCE, obtener un `CanonicalPrincipal` mapeado desde los claims de Entra, y recibir una cookie de sesión con la misma forma que la sesión SAML. Sin embargo, esa sesión es actualmente una sesión de segunda clase: **no puede usarse para realizar reservas**.

El motivo es que los routes de finalización de intents (`intents/actions/finalize` y `intents/reservations/finalize`) tienen una dependencia hardcodeada a `session.samlAssertion`, que no existe en una sesión Entra. Además, `blockchain-services` espera recibir una SAML assertion en los payloads de intent y la valida con `SamlValidationService`. Sin resolver estas dos capas, el usuario de Entra ID puede entrar al marketplace pero no puede reservar ni acceder a ningún laboratorio.

Este TFG completa esa integración: adapta los intent flows del Marketplace y el backend de `blockchain-services` para que una sesión Entra sea funcionalmente equivalente a una sesión SAML en todos los flujos de negocio relevantes.

---

2. Estado actual detallado de la rama `feature/entra-id`

2.1 Qué está implementado y funciona

Módulo	Archivo	Estado
Inicio del flujo OIDC (PKCE)	`src/app/api/auth/entra/login/route.js`	Completo
Callback OIDC: intercambio de código, validación de `id_token`	`src/app/api/auth/entra/callback/route.js`	Completo
Adapter de claims Entra → `CanonicalPrincipal`	`src/utils/auth/entraIdAdapter.js`	Completo

Módulo	Archivo	Estado
Modelo canónico de principal (`PrincipalType`, `AuthMethod`)	`src/utils/auth/principal.js`	Completo
Registro de IdPs y reglas de mapeo de roles por tenant	`src/utils/auth/idpRegistry.js`	Completo
Mapper de roles desde claims de IdP	`src/utils/auth/roleMapper.js`	Completo
Cliente OIDC con helpers PKCE	`src/utils/auth/oidcClient.js`	Completo
Generación del JWT operativo local para sesión Entra	`src/utils/auth/marketplaceJwt.js` (`generateEntraAuthToken`)	Completo
Botón de login Microsoft en la UI	`src/components/auth/EntraLoginButton.js`	Completo
Integración del botón en el modal de login	`src/components/auth/Login.js`	Completo
Guards de autorización actualizados para `institutionId` de Entra	`src/utils/auth/guards.js`	Parcial

2.2 Qué falta

Problema	Archivo(s) afectado(s)	Impacto
`session.samlAssertion` requerido para finalizar una acción	`intents/actions/finalize/route.js`	Bloquea acceso a laboratorios para usuarios Entra
`session.samlAssertion` requerido para finalizar una reserva	`intents/reservations/finalize/route.js`	Bloquea reservas para usuarios Entra
`blockchain-services` valida SAML assertion en todos los intents	`SamlValidationService.java`, `IntentService.java`	Bloqueo a nivel de backend
Adapter Okta	`src/utils/auth/idpAdapters/oktaAdapter.js`	Stub — fase 2 no implementada
Tests E2E de reserva completa con sesión Entra	`cypress/e2e/`	No existen
Documentación de configuración de tenant para administradores	—	No existe

3. Motivación y contexto

3.1 Valor del trabajo

El login con Entra ya funciona, lo que significa que el alumno no parte de cero ni tiene que diseñar la arquitectura de identidad. La dificultad del trabajo está en el "último tramo": entender cómo funciona el flujo de intents EIP-712 de DecentraLabs (que hoy está profundamente ligado a SAML), diseñar la abstracción correcta que permita expresar "autorización de intent" tanto para una sesión SAML como para una sesión Entra, e implementarla en las dos capas (Marketplace y `blockchain-services`).

3.2 El patrón de diseño a implementar

La decisión arquitectónica de partida es:

- `blockchain-services` **no valida directamente los tokens de Entra**. Sigue siendo la única fuente de confianza.
- El Marketplace actúa como broker: valida el token Entra, construye el `CanonicalPrincipal`, y emite un **JWT operativo local** firmado con la clave del Marketplace.
- `blockchain-services` valida ese JWT local (ya soportado: `/auth/jwks`) y extrae la identidad del principal.

Lo que hay que diseñar es qué reemplaza a `samlAssertion` en el payload de intent cuando el usuario viene de Entra. La respuesta lógica es: el JWT operativo local ya contiene toda la información de identidad necesaria; el `assertionHash` puede calcularse desde ese JWT en lugar de desde la SAML assertion.

---

4. Objetivos

Objetivo principal

Completar la integración de Microsoft Entra ID en DecentraLabs para que un usuario autenticado con Entra pueda realizar el flujo completo: login → onboarding → reserva → acceso al laboratorio, con la misma experiencia funcional que un usuario SAML.

Objetivos específicos

1. Analizar en detalle el flujo de intents EIP-712 en el Marketplace y `blockchain-services` para identificar todas las dependencias de `samlAssertion`.
2. Diseñar el mecanismo de autorización de intents para sesiones Entra: qué sustituye a `samlAssertion` y cómo se calcula el `assertionHash` equivalente de forma compatible con el contrato.
3. Adaptar los routes de finalización de intents en el Marketplace para soportar ambos tipos de sesión (SAML y Entra) sin duplicar lógica.

4. Adaptar `blockchain-services` para aceptar intents autorizados con el JWT operativo local del Marketplace (ya firmado con la clave del Marketplace) como alternativa a la SAML assertion.
5. Implementar el adapter Okta (fase 2): completar `oktaAdapter.js` y los routes `/api/auth/okta/login` y `/api/auth/okta/callback` siguiendo el patrón Entra ya establecido.
6. Escribir tests end-to-end (Cypress) que cubran el flujo completo de reserva con sesión Entra.
7. Documentar el proceso de configuración de un tenant Entra ID para administradores de DecentraLabs.

---

5. Alcance y tareas

Fase 1: Análisis del flujo de intents (3 semanas)

- Leer y mapear el flujo completo de un intent de reserva institucional:
  - Marketplace: `intents/reservations/prepare/route.js` → `intents/reservations/finalize/route.js`
  - `blockchain-services`: `IntentService.java` → `SamlValidationService.java` → `IntentAuthorizationService.java`
  - Smart Contracts: `IntentTypes.sol`, `LibIntent.sol`, `ReservationIntentFacet.sol`
- Identificar **exactamente** qué campos del payload de intent dependen de la SAML assertion y con qué propósito (validación de identidad, cálculo de hash, firma EIP-712).
- Identificar si el contrato tiene dependencias directas con el formato SAML o si el `assertionHash` es genérico.
- Entrega: diagrama de flujo anotado del intent de reserva con las dependencias SAML marcadas.

Fase 2: Diseño de la abstracción de autorización de intents (2 semanas)

- Definir la abstracción `IntentAuthorizationCredential` que puede ser:
  - `{ type: 'saml', assertion: '...' }` — sesión SAML (compatibilidad backward)
  - `{ type: 'oidc', operationalJwt: '...', principalSub: '...' }` — sesión Entra/Okta
- Diseñar la función de cálculo del `assertionHash` para el tipo OIDC (debe producir un hash determinista y verificable en `blockchain-services`).
- Diseñar los cambios mínimos necesarios en `blockchain-services` para aceptar el nuevo tipo de credencial sin romper las sesiones SAML existentes.

- Revisión del diseño con el tutor antes de implementar.

Fase 3: Adaptación de los routes de intents en el Marketplace (3 semanas)

- Refactorizar `intents/actions/finalize/route.js` y `intents/reservations/finalize/route.js` para:
  - detectar el tipo de sesión (`authMethod` en la cookie),
  - construir el `IntentAuthorizationCredential` correcto según el tipo,
  - enviar el credential al backend en el formato diseñado en Fase 2.
- Actualizar `intents/reservations/prepare/route.js` si el prepare también depende de SAML.
- Tests Jest para los routes adaptados con mock de sesión Entra y sesión SAML.

Fase 4: Adaptación de blockchain-services (4 semanas)

- Implementar `OidcIntentAuthorizationStrategy` en `blockchain-services`:
  - valida el JWT operativo local del Marketplace (via JWKS del Marketplace, ya disponible),
  - extrae el `CanonicalPrincipal` del JWT,
  - calcula el `assertionHash` de forma compatible.
- Refactorizar `IntentService.java` para usar un patrón Strategy que seleccione `SamlAuthorizationStrategy` u `OidcIntentAuthorizationStrategy` según el campo `authType` del payload.
- Migración: ninguna (el contrato no necesita cambios; el `assertionHash` es un `bytes32` genérico).
- Tests unitarios e integración en `blockchain-services`.

Fase 5: Adapter Okta (2 semanas)

- Implementar `buildCanonicalPrincipal(claims)` en `oktaAdapter.js` siguiendo las instrucciones del stub existente.
- Crear `src/app/api/auth/okta/login/route.js` y `src/app/api/auth/okta/callback/route.js` (el stub los describe explícitamente).
- Añadir `OktaLoginButton` siguiendo el patrón de `EntraLoginButton`.
- Tests unitarios del adapter Okta.

Fase 6: Tests E2E y validación (2 semanas)

- Tests Cypress del flujo completo con sesión Entra (mock del IdP externo con un servidor OIDC local, por ejemplo `oidc-provider` npm):

    - login → sesión creada,
    - reserva → intent finalizado correctamente,
    - acceso al lab → session ticket generado.
- Verificar que los tests E2E SAML existentes siguen pasando.

Fase 7: Documentación y memoria (3 semanas)

- Guía de configuración de tenant para administradores: variables de entorno, configuración del App Registration en Azure, allow-list de tenants.
- Memoria con análisis del estado del arte de integración OIDC en plataformas de e-learning/laboratorio y justificación de las decisiones de diseño.

---

6. Requisitos técnicos y conocimientos previos

Área	**Nivel requerido**
JavaScript / Next.js (App Router, API Routes)	Intermedio-avanzado
OIDC / OAuth 2.0 / JWT	Básico (se amplía en la Fase 1)
Java / Spring Boot	Básico-intermedio (se aprende con el código existente)
EIP-712 / firma de intents	Ninguno (solo se necesita entender el campo `assertionHash`)
Git (branches, merge)	Intermedio
Cypress (E2E)	Básico

---

7. Resultados esperados

- Flujo completo de reserva funcional para usuarios autenticados con Entra ID.
- Adapter Okta completo y funcional (login + callback + sesión).
- Tests Jest de los routes adaptados + tests E2E Cypress del flujo de reserva.
- Adaptación de `blockchain-services` para soportar el nuevo tipo de credencial.
- Guía de configuración de tenant para administradores.
- Memoria con análisis de diseño y estado del arte.

---

8. Archivos y módulos del repositorio relevantes

`Marketplace/` — rama `feature/entra-id`

- `src/app/api/auth/entra/login/route.js` y `callback/route.js` — implementación completa del login (punto de partida)
- `src/utils/auth/entraIdAdapter.js` — adapter de claims (implementado, referencia)
- `src/utils/auth/principal.js` — modelo canónico de principal
- `src/utils/auth/idpRegistry.js` — registro de IdPs
- `src/utils/auth/idpAdapters/oktaAdapter.js` — **stub a completar** (instrucciones dentro del archivo)
- `src/app/api/backend/intents/actions/finalize/route.js` — **bloqueo principal #1**
- `src/app/api/backend/intents/reservations/finalize/route.js` — **bloqueo principal #2**
- `src/app/api/backend/intents/reservations/prepare/route.js` — revisar dependencias SAML

`Lab Gateway/blockchain-services/`

- `src/main/java/decentralabs/blockchain/service/intent/IntentService.java` — **bloqueo backend**
- `src/main/java/decentralabs/blockchain/service/auth/SamlValidationService.java` — estrategia a abstraer
- `src/main/java/decentralabs/blockchain/service/auth/JwtService.java` — emisión de JWT (referencia)
- `src/main/java/decentralabs/blockchain/controller/intent/` — endpoints de intents

`Smart-Contracts/`

- `contracts/libraries/IntentTypes.sol` — estructura del intent (verificar si `assertionHash` es genérico)
- `contracts/libraries/LibIntent.sol` — validación de intents en el contrato

Documentos de referencia

- `PLAN_ENTRA_OKTA_SUPPORT.md` — decisiones arquitectónicas de partida
- `INFORME_ENTRA-ID.md` — análisis previo
- `PLAN_M2M_SUPPORT.md` — modelo de principal canónico (contexto)

- `PLAN_VCs_EUDI.md` — trabajo paralelo en `feature/identityEvidence` (no solapar)

---

9. Bibliografía y referencias de partida

- OpenID Connect Core 1.0 — https://openid.net/specs/openid-connect-core-1_0.html
- RFC 7519 — JSON Web Token (JWT)
- RFC 7636 — PKCE for OAuth 2.0
- Microsoft Identity Platform — Authorization Code + PKCE flow — https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow
- `openid-client` v6 documentation — https://github.com/panva/openid-client
- EIP-712: Typed structured data hashing and signing — https://eips.ethereum.org/EIPS/eip-712
- Documentación interna: `PLAN_ENTRA_OKTA_SUPPORT.md`, `INFORME_ENTRA-ID.md`, código en `feature/entra-id`