TFG — Sistema de valoraciones y reseñas de laboratorio verificadas en DecentraLabs

Duración estimada: 4–5 meses
Dificultad: Media. Full-stack que toca las tres capas del sistema (Smart Contract, Spring Boot, Next.js) pero en tramos bien delimitados. No requiere experiencia previa en blockchain.
Idioma de referencia del código fuente: Solidity 0.8.33 (Foundry), Java (Spring Boot 3.5), JavaScript (Next.js 15)
CONTACTO: Luis de la Torre (ldelatorre@dia.uned.es)

---

1. Resumen

DecentraLabs es un marketplace de laboratorios remotos y modelos de simulación. Mantiene una **reputación automatizada de laboratorios** basada en eventos on-chain (completions y cancellaciones de reservas). El problema es que esa puntuación es completamente opaca para el usuario: no hay ningún mecanismo que permita a un consumidor dejar una valoración cualitativa tras una sesión, ni consultar qué opinan otros usuarios sobre un laboratorio. Esta ausencia es especialmente llamativa porque la plataforma opera con contratos inteligentes que podrían garantizar la **autenticidad de cada valoración**: el contrato puede verificar que quien valora ha completado efectivamente una reserva, y registrar que esa reserva ya fue usada para valorar, haciendo imposible el voto múltiple o las reseñas falsas.

Este TFG propone diseñar e implementar el sistema completo de valoraciones verificadas:

1. **Smart Contract**: nuevo facet `LabReviewFacet` que acepta valoraciones ligadas a `reservationId`, las rechaza si la reserva no fue completada o ya fue usada para valorar, y emite el evento on-chain correspondiente.
2. **Backend**: servicio en `blockchain-services` que indexa las valoraciones y expone una API de lectura con paginación y estadísticas.
3. **Marketplace**: prompt post-sesión para dejar la valoración + sección de reseñas en la ficha de laboratorio.

---

2. Motivación y contexto

2.1 El problema de confianza en los marketplaces

En cualquier marketplace peer-to-peer (Airbnb, Fiverr, Amazon Marketplace), el sistema de reseñas es el principal mecanismo de confianza entre partes que no se conocen. En DecentraLabs, el laboratorio de un proveedor puede ser excelente o deficiente, pero el consumidor no tiene más indicación que la descripción del propio proveedor.

La diferencia respecto a marketplaces web2 es que aquí **la autenticidad de cada valoración es verificable on-chain sin depender de la plataforma**: cualquier tercero puede comprobar que el `reservationId` asociado a una reseña existe, fue completado, y no ha sido usado para valorar antes. Esto elimina el problema de reseñas falsas desde la raíz.

2.2 Estado actual del código

Componente	Estado actual
Smart Contracts — reputación automatizada (completions/cancellaciones)	Implementado: `LibReputation.sol`, `LabReputationFacet.sol`
Smart Contracts — valoración cualitativa de usuario post-sesión	No existe
`blockchain-services` — indexación de valoraciones	No existe
Marketplace — visualización de puntuación de reputación	Parcial: `getMarketLabsSnapshot.js` llama a `getLabReputation`
Marketplace — reseñas de usuarios visibles y prompt post-sesión	No existe

La base on-chain está puesta (`LibReputation`, `LabReputationFacet`). El TFG extiende esa base con la dimensión cualitativa que falta.

2.3 Decisiones de diseño pendientes

El alumno tiene libertad de diseño en las siguientes decisiones (justificadas en la memoria):

- ¿Se almacena el texto de la reseña on-chain o solo su hash (con el texto en base de datos)?
- ¿Contribuye la valoración del usuario a la puntuación numérica de `LibReputation`, o es un canal separado?
- ¿Cuál es el modelo de moderación (reseñas visibles inmediatamente vs. revisión previa)?

---

3. Objetivos

Objetivo principal

Diseñar e implementar un sistema de valoraciones post-sesión en DecentraLabs que garantice la autenticidad de cada reseña mediante verificación on-chain del `reservationId` y sea presentable al consumidor como un indicador de calidad fiable.

Objetivos específicos

1. Diseñar el modelo de datos de la valoración: qué campos almacena el contrato, qué campos almacena la base de datos, y cuál es la fuente de verdad de cada parte.
2. Implementar `LabReviewFacet` en el Diamond:
    - `submitReview(uint256 reservationId, uint8 stars, bytes32 commentHash)` — registra la valoración, verifica que la reserva está completa y no fue usada ya para valorar.
    - `hasReviewed(uint256 reservationId)` — consulta de estado.

    - `getLabReviewStats(uint256 labId)` — media de estrellas y número total de valoraciones.
3. Implementar en `blockchain-services` el servicio de indexación y consulta de valoraciones:
    - escucha de eventos `ReviewSubmitted` para indexar en base de datos relacional,
    - API paginada `GET /labs/{labId}/reviews`,
    - endpoint de envío `POST /reviews` que construye y ejecuta la transacción on-chain.
4. Implementar en el Marketplace:
    - componente de prompt post-sesión (aparece al expirar la reserva),
    - sección de reseñas paginadas en la ficha de laboratorio,
    - indicador de puntuación media visible en la tarjeta del laboratorio en el catálogo.
5. Garantizar que una misma reserva solo puede generar una valoración (verificado en el contrato, también en el backend).

---

4. Alcance y tareas

Fase 1: Análisis y diseño (3 semanas)

- Estudiar `LibReputation.sol`, `LabReputationFacet.sol` y `LibAppStorage.sol` para entender la estructura de almacenamiento existente del Diamond.
- Revisar las reservas institucionales completadas: qué datos están disponibles en el contrato para verificar la autoría de una valoración.
- Revisar `getMarketLabsSnapshot.js` para entender cómo se consume la reputación actual en el Marketplace.
- Tomar las tres decisiones de diseño clave (almacenamiento on-chain vs. hash, relación con `LibReputation`, modelo de moderación) y documentarlas con sus trade-offs.
- Entrega: documento de diseño con diagrama de flujo de submit y modelo de datos.

Fase 2: Implementación Smart Contract (4 semanas)

- Añadir a `LibAppStorage.sol`:
    - `mapping(uint256 reservationId => bool) reviewSubmitted`
    - `mapping(uint256 labId => LabReviewStats) labReviewStats` (totalStars, reviewCount)
- Implementar `LabReviewFacet.sol`:
    - `submitReview(uint256 reservationId, uint8 stars, bytes32 commentHash)` con validaciones: reserva completada, no revisada anteriormente,

`stars` en rango [1, 5], llamante == reservante.

- `hasReviewed(uint256 reservationId)` y `getLabReviewStats(uint256 labId)`.
- Evento `ReviewSubmitted(uint256 indexed labId, uint256 indexed reservationId, uint8 stars, bytes32 commentHash, address reviewer)`.

- Registrar el facet en `InitFacet` y actualizar `deploy.ps1`.
- Tests Foundry: submit válido, doble-submit rechazado, submit con reserva incompleta rechazado, submit con reserva de otro usuario rechazado. Cobertura ≥ 90%.

Fase 3: Implementación backend en blockchain-services (4 semanas)

- Migración Flyway: tabla `lab_reviews` (reservationId, labId, reviewerAddress, stars, commentText, commentHash, txHash, blockTimestamp).
- `LabReviewEventListener`: escucha el evento `ReviewSubmitted` on-chain e indexa en base de datos.
- `LabReviewService`:
  - `submitReview(...)` — construye y envía la transacción on-chain, persiste el comentario en texto claro.
  - `getReviews(labId, page, size)` — consulta paginada con el texto de la reseña.
  - `getStats(labId)` — media, conteo y distribución de estrellas.
- `LabReviewController` con los endpoints REST correspondientes.
- Tests de integración con Testcontainers (PostgreSQL + Anvil).

Fase 4: Implementación en el Marketplace (3 semanas)

- Componente `PostSessionRatingPrompt`: aparece en el dashboard del usuario cuando una reserva pasa a estado `COMPLETED` sin valorar. Formulario de 1–5 estrellas + campo de texto opcional. Llama al endpoint de `blockchain-services` que prepara y firma la transacción vía WebAuthn/wallet.
- Componente `LabReviewSection` en la página de detalle del laboratorio: lista de reseñas paginadas con nombre de institución, fecha, estrellas y texto.
- Actualizar la tarjeta de laboratorio en el catálogo para mostrar la media de estrellas y el número de reseñas (junto con la puntuación de reputación existente).
- Tests Jest del componente `PostSessionRatingPrompt` y `LabReviewSection`.

Fase 5: Validación y pruebas end-to-end (2 semanas)

- Flujo completo en red local (Anvil): completar reserva → submit de valoración → indexación → visualización en UI.

- Verificación de los casos de error: doble submit, reserva no completada, intentar valorar la reserva de otro usuario.
- Ejecutar la suite de tests regresivos del Marketplace y de blockchain-services.

Fase 6: Memoria y presentación (3 semanas)

---

5. Requisitos técnicos y conocimientos previos

Área	**Nivel requerido**
Solidity / Foundry	Básico (se aprende en la Fase 2; la estructura del Diamond ya está resuelta)
Java / Spring Boot	Intermedio
SQL / Flyway	Básico
JavaScript / React	Intermedio
Web3j	Básico (solo para enviar una transacción y escuchar eventos)
Git	Básico

---

6. Resultados esperados

- `LabReviewFacet` desplegado en Sepolia con tests Foundry (cobertura ≥ 90%).
- API de valoraciones en `blockchain-services` con tests de integración.
- Prompt post-sesión y sección de reseñas funcionales en el Marketplace.
- Memoria del TFG con análisis comparativo de sistemas de reputación en marketplaces descentralizados.

---

7. Archivos y módulos del repositorio relevantes

`Smart-Contracts/`

- `contracts/libraries/LibReputation.sol` — lógica de puntuación actual (referencia y punto de extensión)
- `contracts/libraries/LibAppStorage.sol` — Diamond Storage (patrón a seguir para nuevos mappings)
- `contracts/facets/lab/LabReputationFacet.sol` — facet de reputación (referencia directa)
- `contracts/facets/InitFacet.sol` — registro de facets (necesita actualización)

- `scripts/deploy.ps1` — despliegue en Sepolia

`Lab Gateway/blockchain-services/`

- `src/main/java/decentralabs/blockchain/service/intent/` — flujo de reserva (referencia de integración)
- `src/main/resources/db/migration/` — migraciones Flyway existentes (referencia de estilo)

`Marketplace/`

- `src/server/market/getMarketLabsSnapshot.js` — consume `getLabReputation` (integración del nuevo `getLabReviewStats`)
- `src/app/(app)/lab/[id]/page.js` — página de detalle del laboratorio (punto de inserción de `LabReviewSection`)
- `src/app/(app)/dashboard/` — dashboard del usuario (punto de inserción del prompt post-sesión)

---

8. Bibliografía y referencias de partida

- Nakamoto, S. (2008) — *Bitcoin: A Peer-to-Peer Electronic Cash System* (contexto de confianza sin intermediario)
- Nofer et al. (2017) — *Blockchain* en Business & Information Systems Engineering
- Chevalier & Mayzlin (2006) — *The Effect of Word of Mouth on Sales: Online Book Reviews* (valor económico de las reseñas)
- Documentación OpenZeppelin — EIP-2535 Diamond Storage
- Dokumentation Foundry — https://book.getfoundry.sh/
- Documentación interna: `Smart-Contracts/contracts/libraries/LibReputation.sol`, `LabReputationFacet.sol`