Ejercicio 2: Caso de uso basado en licitaciones públicas

Tecnologías de Registro Distribuido y Blockchain (BC)

Integrantes:

Jesús Alfredo González Salcedo
Manuel Martín Louredo Pérez

Noviembre 2025

Índice

1. Caso de uso propuesto

El caso de uso que se plantea está relacionado con un sistema de gestión de licitaciones públicas (contratación de servicios o suministros) en el que participan:

- Entidad licitadora (administración pública o empresa).
- Proveedores que presentan ofertas económicas y documentación
- Evaluadores, puntúan ofertas.
- Ciudadanía

En los sistemas tradiciones de licitación, la documentación de las ofertas se almacena en sistemas centralizados. Esto genera problemas de falta de transparencia, dependencia de un servidor central, dificultad para probar que los documentos evaluados son los que se presentaron al proveedor, etc.

Para mejorar esta situación, se propone un sistema en el que:

1. Definición de licitaciones, ofertas y evaluación se gestionan mediante un smart contract en Ethereum (PublicTenderContract).
2. La documentación pesada de las ofertas (PDFs, ZIPs, etc.) se almacena de forma descentralizada en IPFS.
3. El hash (CID) de cada documento se guarda tanto en el contrato IpfsStorage como en el contrato de licitación, de modo que cualquier modificación del fichero resultaría inmediatamente detectable.

De esta forma se consigue:

- Transparencia e inmutabilidad: el CID guardado en la blockchain prueba qué documento se usó en cada oferta.
- Descentralización del almacenamiento: los documentos no dependen de un único servidor.
- Trazabilidad del proceso de licitación de extremo a extremo (creación, ofertas, evaluación y ganador).

1.1. Arquitectura de comunicaciones

La arquitectura propuesta está formada por los siguientes componentes:

1. **Frontend Web (React)**:
   - Interactúa con MetaMask para realizar operaciones en Ethereum.
   - Se comunica con el nodo IPFS mediante la API HTTP (kubo-rpc-client).
   - Contiene las vistas para crear licitaciones, enviar ofertas y evaluar.
2. **Nodo IPFS local (Docker)**:
   - Se ejecuta la imagen `ipfs/kubo`.
   - Expone:
     - API: `http://127.0.0.1:5001/api/v0`
     - Gateway: `http://127.0.0.1:8080/ipfs/<CID>`
   - Se configura CORS para permitir el acceso desde el front (`http://localhost:3000`).

3. **Blockchain Ethereum (red de pruebas)**:
   - MetaMask gestiona las cuentas y firma las transacciones.
   - Se despliegan dos contratos:
     - **IpfsStorage**: almacena el último CID asociado a cada dirección.
     - **PublicTenderContract**: gestiona licitaciones, ofertas y evaluaciones.
4. **Roles del sistema**:
   - Propietario (crea licitaciones).
   - Proveedor (envía documentación en IPFS).
   - Evaluador (puntúa ofertas).

La comunicación se realiza mediante:

- `ethers.js` para Ethereum.
- `kubo-rpc-client` para IPFS.

1.2. 2.2 Implementación de la arquitectura

La implementación se desarrolló en React utilizando:

- **ethers 5.7.0** para interactuar con la blockchain.
- **kubo-rpc-client** para utilizar la API de IPFS.
- **buffer** para gestionar ficheros binarios.

Los contratos inteligentes utilizados fueron:

- **IpfsStorage**: mapping `address =>string`.
- **PublicTenderContract**: gestión de licitaciones, ofertas y evaluación.

El proyecto está organizado de la siguiente manera:

- `App.js`: lógica principal del frontend.
- `contracts/`: contiene ABIs y direcciones desplegadas.
- `App.css`: interfaz visual y estilo oscuro.

El sistema implementa tres pilares fundamentales:

Gestión de usuarios y roles

El front detecta mediante MetaMask si el usuario es:

- Propietario
- Evaluador
- Proveedor

Según el rol, diferentes pestañas se habilitan o deshabilitan.

Gestión de licitaciones

Se utiliza:

```
tenderCount()
getTender(id)
```

para listar todas las licitaciones y mostrar:

- Estado
- Ofertas enviadas
- Evaluaciones recibidas
- Ganador

Integración con IPFS

El usuario selecciona un fichero y el sistema:

1. Convierte el archivo en un Buffer.
2. Lo sube a IPFS:

   ```
   const result = await client.add(file);
   ```

3. Guarda el CID en la blockchain mediante:

   ```
   setFileIPFS(cid);
   submitOffer(tenderId, price, cid);
   ```

Esto garantiza que la documentación evaluada no pueda ser modificada.

1.3. 2.3 Funcionamiento del sistema

El funcionamiento se divide por roles:

1.3.1. 2.3.1 Creación de licitaciones (propietario)

1. El propietario inicia sesión con MetaMask.
2. Accede a “Crear licitación”.
3. Indica:
   - Descripción
   - Precio máximo
   - Plazo
   - Peso precio/calidad
4. El contrato registra la nueva licitación en la blockchain.

1.3.2. 2.3.2 Envío de ofertas y documentación (proveedor)

1. El proveedor accede a "Enviar oferta (IPFS)".
2. Selecciona la licitación y adjunta un archivo.
3. El front:
   a) Sube la documentación a IPFS.
   b) Obtiene su CID.
   c) Envía la oferta al contrato con dicho CID.

1.3.3. 2.3.3 Evaluación y ganador (evaluadores)

1. El evaluador introduce:
   - Tender ID
   - Dirección del proveedor
   - Puntuación
2. El contrato almacena la evaluación.
3. Finalmente se ejecuta:

   ```
   calculateWinner(tenderId)
   ```

4. El contrato calcula el ganador según precio y calidad.

1.4. 2.4 Lecciones aprendidas

Durante el proyecto se identificaron varios problemas y soluciones:

- **Versión de Node.js**: fue necesario usar Node 18.13.0 para evitar incompatibilidades con ethers 5.
- **Errores CORS en IPFS**: se solucionaron ejecutando los comandos directamente en el contenedor Docker.
- **Dirección 0.0.0.0 inválida**: el navegador no permite esa dirección, por lo que se usó `127.0.0.1`.
- **Errores "Tender does not exist"**: los IDs comienzan en 1 y no en 0.
- **Gestión de estados en React**: fue necesario mejorar la UX con estados de carga y mensajes de error.
- **Integración Ethereum + IPFS**: se aprendió a combinar almacenamiento descentralizado con registro inmutable.