Departamento de Elearning
Facultad Politécnica
Universidad Nacional de Asunción

Actividad 3. Final

Identificación

Asignatura: Blockchain
Eje: DeFi y contratos inteligentes

Objetivos

1. Aplicar Solidity para crear y administrar un protocolo de préstamos con garantía colateral.
2. Desarrollar un cliente web que interactúe con los contratos usando **ethers.js** y **MetaMask**.
3. Desplegar los contratos en la red **Ephemery Testnet** y simular casos de uso básicos (depósito, préstamo y repago).

Actividad

1. Introducción

En este mini-proyecto vas a construir una DApp de préstamos descentralizados que permita a los usuarios:

- Depositar tokens como colateral (p. ej. cUSD).
- Solicitar préstamos en otro token (p. ej. dDAI) con un ratio de colateralización del **150%**.
- Pagar su préstamo con interés fijo y retirar su colateral.
- Todo sin oráculos externos ni liquidadores automáticos.

Nota: no se utilizarán oráculos de precios. El ratio de precios es fijo (1 cUSD = 1 dDAI).

2. Primeros pasos

2.1 Instalación

```
npm init -y
```

```
npm install --save-dev
hardhat@nomicfoundation/hardhat-toolbox

npm install ethers@5.7 react vite
```

2.2 Proyecto Hardhat

```
npx hardhat init
```

Selecciona “Create a basic sample project”.

2.3 Configura Ephemery

En `hardhat.config.js`, agrega:

```
networks: {
  ephemery: {
    url: "https://rpc.ephemery.dev",
    accounts: [process.env.PRIVATE_KEY]
  }
}
```

2.4 Clona el boilerplate con Vite + React (directorio `web_app/`).

2.5 Variables de entorno

Crea `.env` con:

```
PRIVATE_KEY=

VITE_CONTRACT_ADDRESS=

VITE_RPC_URL=
```

3. Componentes

3.1 Contrato (contracts/LendingProtocol.sol)

Función	Descripción
`depositCollateral(uint256 amount)`	El usuario deposita tokens colaterales (cUSD).
`borrow(uint256 amount)`	Permite pedir prestado hasta el 66% del valor del colateral.
`repay()`	El usuario paga el préstamo con interés (fijo).
`withdrawCollateral()`	Retira colateral si no hay deuda activa.
`getUserData(address user)` (view)	Devuelve saldo colateral, deuda actual e interés acumulado.

Nota: Implementa lógica de interés fijo (p. ej. 5% semanal, sin composición). No se requiere fecha/tiempo exacto, puede simularse con múltiples llamadas de test.

3.2. Implementación de dos tokens ERC20

Descripción y cambios necesarios:

- Crear contratos de tokens ERC20:
  1. `CollateralToken.sol`: Token ERC20 para el colateral (por ejemplo, cUSD).
  2. `LoanToken.sol`: Token ERC20 para el préstamo (por ejemplo, dDAI).

- Integración con el contrato principal:
    1. El contrato `LendingProtocol.sol` deberá referenciar las direcciones de ambos tokens.
    2. Los usuarios depositarán `CollateralToken` y recibirán préstamos en `LoanToken`.
- Pasos de implementación:
    1. Desarrollar los contratos ERC20:
        - Usar el estándar OpenZeppelin para garantizar seguridad y compatibilidad.
        - Cada contrato debe tener una función `mint` (controlada solo por el owner) para facilitar la distribución inicial.
    2. Desplegar los tokens:
        - Antes de desplegar el `LendingProtocol`, desplegar ambos tokens y registrar sus direcciones en el contrato principal.
    3. Actualizar el contrato principal:
        - Añadir variables para almacenar las direcciones de los tokens.
        - Modificar las funciones de depósito, préstamo y repago para interactuar con los tokens correctos.

2. Tests unitarios con 100% de cobertura de líneas

- Desarrollo de tests unitarios:
    1. Usar Hardhat y plugins como `@nomicfoundation/hardhat-toolbox` para testing.
    2. Escribir tests para todas las funciones del contrato principal (`LendingProtocol.sol`) y de los tokens ERC20.
- Requisito de cobertura:
    1. Utilizar una herramienta como `solidity-coverage` para medir la cobertura de líneas.
    2. Asegurar que todos los caminos de ejecución estén cubiertos, incluyendo casos de error y validaciones.
- Pasos de implementación:
    1. Instalar dependencias:
    2. bash

```
npm install --save-dev solidity-coverage
```

3. Configurar scripts de test y cobertura:
    - Agregar un script en el `package.json` para ejecutar la cobertura:

```
"scripts": {
  "coverage": "hardhat coverage"
}
```

4. Escribir tests completos:
    - Cubrir todas las funciones públicas y modificadores.
    - Incluir casos de prueba para:
        - Depósito de colateral.
        - Solicitud de préstamo.
        - Repago y retiro de colateral.
        - Manejo de errores (por ejemplo, depósitos insuficientes, préstamos no autorizados).
5. Verificar la cobertura:
    - Ejecutar `npm run coverage` y revisar que el reporte muestre 100% de cobertura de líneas para los contratos principales.

3.2 Cliente (web_app/src/App.jsx)

Función	Descripción
`connectWallet()`	Conecta MetaMask y guarda la cuenta activa.
`loadUserData()`	Carga la posición de deuda/colateral del usuario.
`deposit()`	Ejecuta `depositCollateral`.
`borrow()`	Ejecuta `borrow` si cumple el ratio de colateralización.
`repay()`	Ejecuta `repay` con interés fijo.
`withdraw()`	Retira colateral si no hay deuda pendiente.

4. Requisitos generales

1. Front-end debe mostrar: monto de colateral, deuda, interés acumulado y botones para cada acción.
2. Conexión a MetaMask mediante `window.ethereum`.
3. Contrato desplegado en **Ephemery Testnet**.
4. Ratios fijos: 1:1 cUSD/dDAI, 150% colateralización, 5% de interés semanal.
5. Usa `uint256` y evita strings innecesarios. Mantén el uso de gas bajo.

5. Carga del código fuente

Repositorio: https://github.com/mdvillagra/blockchain-2025

- Sube el código a un **fork** del repo `blockchain-2025`, en la carpeta `Proyectos/Examen Final/<nombre-del-grupo>` y crea un Pull Request.
- Sube tu código fuente (contrato, tests, cliente web, README).

Plazo

19 de Junio de 2025, hasta 1 semana después como fecha límite.

Se puede realizar en grupo de **hasta 2 personas**.

Evaluación

La siguiente rúbrica se utilizará para la corrección.

Criterio	Puntos
Contratos ERC20	2
Funcionalidad de depósito y préstamo correcta	2
Interés fijo correctamente aplicado	1
Lógica de colateralización y límites implementada	1
Front-end funcional y conectado a MetaMask	1
Interfaz clara: muestra deuda, colateral, interés	1
Repago y retiro de colateral funcionales	1
Pruebas unitarias completas	1
Despliegue exitoso en Ephemery (dirección en README)	1
Código limpio y documentación clara	1
Test unitarios	1
Cobertura de los tests del 100%	1
Total	**13 puntos**