En este examen final, los estudiantes ampliarán el contrato original `KipuBank`, aplicando técnicas avanzadas de Solidity e integrando buenas prácticas en arquitectura de smart contracts, seguridad y documentación.

¿Por qué esto importa?

Este proyecto es tu oportunidad de tomar todo lo que has aprendido y demostrar tu capacidad de evolucionar un smart contract existente hacia una aplicación más completa, segura y usable. Simula el proceso real de desarrollar, mantener y escalar smart contracts en entornos de producción.

En lugar de seguir un conjunto rígido de instrucciones, comenzarás desde la primera versión de `KipuBank` y tomarás decisiones de diseño fundamentadas para mejorarlo. También presentarás tu solución actualizada de manera profesional en GitHub, tal como lo harías en un equipo real o en una colaboración de código abierto.

Objetivos del proyecto

- Identificar limitaciones en un smart contract existente.
- Aplicar características avanzadas de Solidity y patrones de diseño seguros.
- Refactorizar y extender el contrato para introducir funcionalidades significativas.
- Seguir las mejores prácticas en estructura de código, documentación y despliegue en testnet.
- Comunicar tu implementación a través de un repositorio claro en GitHub.

Descripción de la tarea y requisitos

Tu tarea es actualizar el smart contract original `KipuBank` hacia una versión más lista para producción. Se espera que tomes decisiones fundamentadas sobre qué y cómo mejorar, utilizando todo lo que has aprendido a lo largo del curso.

Algunas áreas que puedes considerar mejorar o extender incluyen (pero no se limitan a):

- **Control de Acceso:** Introducir funciones administrativas con roles o accesos restringidos, usando contratos de OpenZeppelin.
- **Soporte Multi-token:** Ampliar el soporte más allá de tokens nativos para incluir activos ERC-20, permitiendo depósitos y retiros diferenciados.
- **Contabilidad Interna:** Mejorar cómo se gestionan los balances de los usuarios permitiendo contabilidad multi-token. Usar `address(0)` como dirección de token para depósitos en `ether`.
- **Eventos y Manejo de Errores:** Usar eventos y errores personalizados para mejorar la observabilidad y depuración.
- **Data Feeds:** Usar Chainlink Data Feeds para convertir montos en `ETH` a `USD` y así controlar el límite global del banco.
- **Conversión de Decimales:** Manejar diferentes decimales de los activos y convertirlos a decimales de **USDC** para la contabilidad interna.

- **Seguridad y Eficiencia:** Aplicar patrones conocidos como *checks-effects-interactions*, uso de variables `immutable` y `constant`, optimización de gas y manejo seguro de transferencias nativas.

Tú decidirás cómo implementarlo. La prioridad es que el código sea **limpio, legible, seguro y bien documentado**.

Entregables

Debes enviar lo siguiente a través de la plataforma:

1. **URL del repositorio en GitHub**

   Un repo público llamado `KipuBankV2` que contenga:

   - Tu código del smart contract actualizado dentro de la carpeta `/src`. Debe incluir estos nuevos componentes:
     - Control de Acceso.
     - Declaraciones de tipos.
     - Instancia de Oracle de Chainlink.
     - Variables constantes.
     - Mappings anidados.
     - Función de conversión de decimales y valores.
   - Un `README.md` que incluya:
     - Una explicación de alto nivel de las mejoras realizadas y el porqué.
     - Instrucciones de despliegue e interacción.
     - Notas sobre decisiones importantes de diseño o *trade-offs*.
2. **Dirección del contrato desplegado**

   En una testnet pública con el código verificado en un block explorer.