Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Ingeniería de Sistemas y Computación.
Proyecto de Grado.

Desarrollo de un contrato inteligente basado en protocolos DeFi que permita ofrecer un mecanismo de inversión auditable

Jean Carlos Santacruz
Juan Ignacio Gándara

Director: Dr . Juan Pablo Garcia Cifuentes

20 de julio de 2025

Santiago de Cali, 20 de julio de 2025.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Por medio de la presente me permito informarle que los estudiantes de Ingeniería de Sistemas y Computación Jean Carlos Santacruz (cod: 8962275) y Juan Ignacio Gándara (cod: 8958524) trabajan bajo mi dirección en el proyecto de grado titulado "Desarrollo de un contrato inteligente basado en protocolos DeFi que permita ofrecer un mecanismo de inversión auditable".

Atentamente,

Juan Pablo García Cifuentes

Firmado digitalmente por Juan Pablo García Cifuentes
Fecha: 2025.07.16 14:21:58 -05'00'

Dr . Juan Pablo Garcia Cifuentes

Santiago de Cali, 20 de julio de 2025.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Nos permitimos presentar a su consideración el anteproyecto de grado titulado "Desarrollo de un contrato inteligente basado en protocolos DeFi que permita ofrecer un mecanismo de inversión auditable" con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el proyecto de grado y posteriormente optar al título de Ingeniero de Sistemas y Computación.

Al firmar aquí, damos fe que entendemos y conocemos las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,

Jean Carlos Santacruz
Código: 8962275

Juan Ignacio Gándara
Código: 8958524

Resumen

Este trabajo de grado aborda la brecha entre el potencial de las Finanzas Descentralizadas (DeFi) y los desafíos prácticos de su implementación, mediante el diseño, desarrollo y validación de un prototipo de sistema de inversión basado en contratos inteligentes. El sistema permite a un colectivo de usuarios depositar fondos en un contrato común, el cual los invierte en un protocolo simulado para generar rendimientos. Al finalizar un período predefinido, el capital inicial es devuelto a todos los participantes, mientras que los intereses acumulados se asignan a un único ganador a través de un mecanismo de sorteo.

La metodología incluyó un análisis comparativo para seleccionar la plataforma blockchain más idónea, resultando en la elección de Ethereum. La arquitectura del sistema se implementó en Solidity bajo un patrón Factory para garantizar la modularidad y escalabilidad. La validación funcional se realizó de manera exhaustiva en la red de pruebas (testnet) de Sepolia, confirmando la correcta ejecución de todo el ciclo de vida de la inversión. Este trabajo no solo entrega un prototipo funcional, sino que también sirve como un caso de estudio técnico sobre la construcción de aplicaciones DeFi, aportando una base de código y una metodología documentada para futuras exploraciones en el campo de las finanzas programables.

Palabras Clave: Blockchain, Finanzas Descentralizadas, Contrato Inteligente, Patrón Factory, Solidity, Testnet, Inversión Colectiva.

Abstract

This thesis addresses the gap between the potential of Decentralized Finance (DeFi) and the practical challenges of its implementation through the design, development, and validation of a prototype investment system based on smart contracts. The system allows a collective of users to deposit funds into a common contract, which invests them in a simulated protocol to generate yield. At the end of a predefined period, the initial capital is returned to all participants, while the accumulated interest is awarded to a single winner through a lottery mechanism.

The methodology included a comparative analysis to select the most suitable blockchain platform, leading to the choice of Ethereum. The system's architecture was implemented in Solidity using a Factory pattern to ensure modularity and scalability. Functional validation was exhaustively performed on the Sepolia testnet, confirming the correct execution of the entire investment lifecycle. This work not only delivers a functional prototype but also serves as a technical case study on building DeFi applications, providing a codebase and a documented methodology for future explorations in the field of programmable finance.

Keywords: Blockchain, Decentralized Finance (DeFi), Smart Contracts, Factory Pattern, Solidity, Testnet, Collective Investment.

Índice general

1. Descripción del Problema **19**
- 1.1. Planteamiento del Problema . . . 19
  - 1.1.1. Formulación . . . 19
  - 1.1.2. Sistematización . . . 20
- 1.2. Objetivos . . . 20
  - 1.2.1. Objetivo General . . . 20
  - 1.2.2. Objetivos Específicos . . . 20
- 1.3. Justificación . . . 20
- 1.4. Delimitaciones y Alcances . . . 21
  - 1.4.1. Alcance . . . 21
  - 1.4.2. Delimitaciones . . . 22

2. Marco Teórico **23**
- 2.1. Áreas Temáticas . . . 23
- 2.2. Fundamentos conceptuales y técnicos . . . 23
  - 2.2.1. Tecnología Blockchain . . . 23
  - 2.2.2. Finanzas Descentralizadas (DeFi) . . . 24
  - 2.2.3. Contratos Inteligentes . . . 24
  - 2.2.4. Verificabilidad y Lógica Determinista . . . 25
  - 2.2.5. Mecanismos de Generación de Intereses en DeFi . . . 26
  - 2.2.6. Patrones de Diseño de Contratos Inteligentes . . . 26
  - 2.2.7. Mainnet vs. Testnet . . . 26
- 2.3. Estado del Arte y Trabajos Relacionados . . . 27
  - 2.3.1. Protocolos Fundacionales de Préstamo y Generación de Rendimiento . . . 27
  - 2.3.2. Mecanismos de Ahorro Colectivo y Juegos sin Pérdida . . . 28
  - 2.3.3. Análisis Crítico y Posicionamiento del Proyecto . . . 28
- 2.4. Resultados Esperados . . . 29

3. Análisis Comparativo y Selección de Plataforma Blockchain para el Desarrollo de Contratos Inteligentes DeFi **31**
- 3.1. Selección de Plataformas Candidatas . . . 31
- 3.2. Criterios de Inclusión y Exclusión . . . 32
- 3.3. Definición y explicación de los Criterios de Calificación . . . 33
  - 3.3.1. Criterios seleccionados . . . 33
- 3.4. Priorización y justificación metodológica . . . 34
  - 3.4.1. Justificación del enfoque multicriterio . . . 34
  - 3.4.2. Modelo de priorización adoptado . . . 35

3.4.3. Asignación de pesos a los criterios . . . 35
3.4.4. Validez metodológica . . . 36
3.5. Evaluación de las plataformas seleccionadas . . . 37
3.5.1. Metodología aplicada . . . 37
3.5.2. Evaluación por criterio . . . 37
3.5.3. Resultados de la evaluación . . . 41
3.6. Elección final de plataforma . . . 42

4. Metodología y Diseño de la Solución 43
4.1. Arquitectura de los contratos Inteligentes . . . 43
4.2. Diseño y funcionamiento de los contratos Inteligentes . . . 45
4.2.1. Contrato Factory (`DeFiLotteryETHFactory`) . . . 45
4.2.2. Contrato Base (`DeFiLotteryETH`) . . . 46
4.2.3. Contrato para el protocolo DeFi (`SimpleDeFi`) . . . 47
4.2.4. Flujo de interacción y movimiento de activos . . . 48

5. Pruebas y resultados 51
5.1. Despliegue del sistema . . . 51
5.2. Plan y Ejecución de Pruebas . . . 53
5.2.1. Tipos de pruebas . . . 54
5.2.2. Alcance y entorno de prueba . . . 54
5.2.3. Estrategia de pruebas . . . 54
5.2.4. Plan de pruebas detallado . . . 56
5.3. Resultado de las Pruebas . . . 65
5.3.1. SimpleDeFi . . . 65
5.3.2. DeFiLotteryETHFactory . . . 65
5.3.3. DeFiLotteryETH . . . 66

6. Caso de Uso funcional 67
6.1. Diseño del sistema . . . 67
6.1.1. Interfaz Web . . . 68
6.1.2. Back-end . . . 69
6.1.3. Uso de Wallet externa . . . 69
6.2. Tecnologías usada . . . 69
6.2.1. Front-end . . . 69
6.2.2. Back-end . . . 70
6.2.3. Gestor de Base de Datos . . . 71
6.2.4. Wallet . . . 71
6.3. Visualización del Sistema y casos de uso funcionales . . . 71
6.3.1. Autenticación . . . 71
6.3.2. Configuración de MetaMask . . . 72
6.3.3. Información de las Loterias . . . 74

6.3.4. Creación de una lotería 75
6.3.5. Interacción con la lotería 78

7. Conclusiones **81**

8. Trabajo Futuro **83**

Bibliografía **85**

Índice de figuras

4.1. Visión general de la arquitectura de contratos inteligentes 43

5.1. Visión general del sistema desplegado en Remix para las pruebas 53

6.1. Visión general de la arquitectura del sistema 68
6.2. Página de Login y Registro del Sistema 72
6.3. Selección de la opción de MetaMask en el perfil 73
6.4. MetaMask conectada correctamente 74
6.5. Diagrama de alto nivel de la arquitectura del sistema 74
6.6. Tarjetas de loterías activas en el panel de usuario 75
6.7. Acceso al formulario de creación de lotería 76
6.8. Confirmación de la transacción en MetaMask 77
6.9. Lotería ubicada en estado “Creada” tras su despliegue 78
6.10. Formulario de inversión y confirmación en MetaMask 79
6.11. Notificación de inversión exitosa 79
6.12. Inicio de la lotería: cambio de estado a “Iniciada” y visualización de fechas 80
6.13. Finalización de la lotería y recepción de fondos (inversión + ganancia) 80

Índice de cuadros

3.1. Resultado del primer filtro de plataformas 33
3.2. Resumen de criterios de evaluación de plataformas blockchain 37
3.3. Evaluación comparativa de plataformas blockchain 42

5.1. Ingreso funcional de balance inicial 56
5.2. Depósito funcional de inversión 57
5.3. Cálculo de ganancia aplicada 57
5.4. Balance cero sin depósito 58
5.5. Retiro exitoso con ganancia 58
5.6. Retiro inválido sin depósito 59
5.7. Ingreso de inversión válida 59
5.8. Ingreso inválido por participante no autorizado 60
5.9. Inicio de inversión en protocolo DeFi 60
5.10. Finalización temprana no permitida 61
5.11. Finalización y reparto de fondos 62
5.12. Validación de creación de instancia de lotería 63
5.13. Depósito de valor límite 0 en `SimpleDeFi` 64
5.14. Depósito de valor negativo en `SimpleDeFi` 64
5.15. Creación de lotería con duración nula 64
5.16. Inicio de inversión con zero deposits 65
5.17. Resultados de las pruebas funcionales de `SimpleDeFi.sol` 65
5.18. Resultados de las pruebas funcionales de `DeFiLotteryETHFactory.sol` 65
5.19. Resultados de las pruebas funcionales de `DeFiLotteryETH.sol` 66

Introducción

En el dominio de la inversión financiera, la confianza y la participación de los usuarios dependen en gran medida de la transparencia y la seguridad de los mecanismos utilizados. Los sistemas tradicionales operan sobre la base de intermediarios, como bancos y corredores, cuya presencia, si bien necesaria en muchos casos, introduce costos adicionales y riesgos vinculados al control centralizado de la información. Esta estructura puede dar lugar a errores, manipulación de datos y accesos restringidos, afectando especialmente a los pequeños inversores.

Frente a estas limitaciones, las Finanzas Descentralizadas (DeFi) surgen como una alternativa tecnológica que permite realizar transacciones sin intermediarios, mediante el uso de contratos inteligentes. Estos programas autónomos, ejecutados sobre *blockchain*, permiten establecer condiciones de ejecución programadas en el propio código, lo cual reduce la necesidad de confianza en terceros. Esta arquitectura promueve una mayor trazabilidad de las operaciones, menores barreras de acceso y una disminución de los costos operativos.

Aunque las propiedades técnicas de DeFi son prometedoras, su implementación práctica presenta desafíos importantes. La construcción de sistemas verificables, reproducibles y funcionales requiere de diseños robustos que puedan ser validados en entornos controlados antes de proyectarse hacia aplicaciones reales. Se hace evidente una brecha entre los principios teóricos del ecosistema DeFi y la necesidad de contar con prototipos funcionales que los reflejen adecuadamente sin involucrar riesgos financieros.

Este trabajo propone el diseño, desarrollo y validación de un prototipo de inversión colectiva sobre la red de pruebas Ethereum (Sepolia), utilizando Ether de *testnet*. El sistema permite la agrupación de aportes en un contrato inteligente que simula la generación de rendimiento. Al finalizar un periodo definido, los fondos son devueltos a todos los participantes, mientras que los intereses simulados se asignan a uno de ellos mediante un mecanismo de selección aleatoria automatizado.

Para ello, se implementa una arquitectura modular en Solidity basada en el patrón *Factory*, que permite desplegar múltiples instancias independientes del contrato. El proceso de desarrollo incorpora herramientas estándar del ecosistema Ethereum —como Remix, Hardhat y MetaMask— y contempla pruebas funcionales e integradas para verificar el flujo completo de interacción, desde la creación de la inversión hasta la distribución de los resultados. Este prototipo no pretende competir con protocolos DeFi existentes, sino ofrecer una implementación replicable y documentada que pueda servir como base académica y técnica para futuras investigaciones y proyectos educativos en el ámbito de las finanzas programables.

Capítulo 1

Descripción del Problema

1.1. Planteamiento del Problema

El sistema financiero tradicional opera en base a un modelo de intermediación, que aunque permite realizar las transacciones, conlleva también la opacidad necesaria y los riesgos de la contraparte. Estos agentes centralizados, como bancos y corredores, son los que controlan la infraestructura de liquidación, custodia y validación y, por ello, los costos, fricciones operativas y barreras de entrada de osallistujat al sistema, tal como afirman Harvey, Ramachandran y Santoro (2021), "los usuarios tienen que confiar en instituciones donde la transparencia es limitada y sus incentivos no siempre van con los de sus clientes"[1].

Frente a esta problemática, las Finanzas Descentralizadas (DeFi) emergen como un paradigma alternativo que aprovecha la tecnología blockchain para construir un ecosistema financiero más abierto, eficiente y accesible [2]. La premisa fundamental de DeFi es la eliminación de intermediarios mediante el uso de contratos inteligentes: programas autónomos que ejecutan acuerdos de forma automática y determinista en la blockchain [3]. El ecosistema DeFi se construye sobre estos contratos como si fueran "legos de dinero", permitiendo la creación de aplicaciones financieras complejas y componibles [4]. Esto traslada la necesidad de confianza desde las instituciones hacia protocolos de código públicamente verificables [5].

Sin embargo, la transición hacia este nuevo paradigma no está exenta de complejidades. A pesar del potencial teórico de los contratos inteligentes, su implementación práctica en el ecosistema DeFi presenta notables desafíos técnicos y de seguridad. La literatura académica ha documentado exhaustivamente que las vulnerabilidades en el código, los ataques de gobernanza y los riesgos sistémicos son obstáculos importantes para la adopción masiva y la madurez del sector [6]. Se configura así el problema central: existe una brecha significativa entre la promesa de la tecnología DeFi y la realidad de su implementación segura y confiable. Por ello, la exploración metodológica para el diseño y desarrollo de aplicaciones DeFi funcionales y auditables no es solo un ejercicio técnico, sino una necesidad para la consolidación del campo.

1.1.1. Formulación

La pregunta ahora es: **¿Cómo desarrollar un contrato inteligente basado en protocolos DeFi que permita ofrecer un mecanismo de inversión auditable?**

1.1.2. Sistematización

Para resolver esta pregunta se deben tener en cuenta algunas subpreguntas:

- ¿Cuáles son las plataformas existentes más adecuadas para implementar un contrato inteligente?
- ¿Qué criterios deben considerarse para seleccionar la plataforma más eficiente y segura para desplegar el contrato inteligente?
- ¿Cómo se desarrolla un contrato inteligente en la plataforma seleccionada?
- ¿Cómo se puede probar el funcionamiento del contrato inteligente en un entorno de prueba?
- ¿Qué caso de uso funcional se puede desarrollar para demostrar la utilidad del contrato inteligente diseñado?

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un sistema basado en contratos inteligentes y protocolos DeFi para gestionar inversiones en criptomonedas de manera automatizada, permitiendo la generación y retiro de intereses en un entorno auditable, utilizando una testnet para validar su funcionamiento.

1.2.2. Objetivos Específicos

1. Identificar las plataformas existentes más adecuadas para implementar el contrato inteligente.
2. Definir los criterios para la selección de la plataforma más eficiente y segura para desplegar el contrato.
3. Desarrollar el contrato inteligente en la plataforma seleccionada.
4. Implementar y probar el contrato en una testnet para validar su correcto funcionamiento.
5. Desarrollar un prototipo funcional de un contrato inteligente basado en protocolos DeFi para gestionar inversiones de forma auditable en un entorno testnet.

1.3. Justificación

El sistema financiero tradicional opera sobre una base de intermediarios que, si bien facilitan las transacciones, también introducen puntos de opacidad y riesgo de contraparte [7]. En respuesta a esto, las Finanzas Descentralizadas (DeFi) proponen un modelo alternativo que utiliza la tecnología blockchain para crear sistemas financieros más abiertos. Este enfoque aprovecha las propiedades inherentes de la blockchain, como la inmutabilidad de su registro y la capacidad de verificación

pública de las transacciones [1, 2].

El componente central de esta arquitectura son los contratos inteligentes, que son programas autónomos cuyo código y ejecuciones se registran de forma permanente en la blockchain [3]. Esta característica fundamental es la que dota al sistema de una cualidad de **auditabilidad intrínseca**: cualquier participante puede verificar de forma independiente que las operaciones se han ejecutado exactamente como el código lo estipula, sin depender de la confianza en un intermediario [5]. La confianza, por tanto, se desplaza de las instituciones a la verificabilidad del protocolo.

Sin embargo, el hecho de que la tecnología ofrezca estas propiedades no garantiza que su aplicación sea trivial o esté exenta de desafíos de diseño y lógica [6]. Por lo tanto, este proyecto se justifica en la necesidad de realizar una exploración práctica sobre cómo estructurar y aplicar estos principios para un caso de uso concreto. El objetivo no es inventar nuevas técnicas de seguridad, sino desarrollar un prototipo funcional que demuestre la viabilidad de utilizar las herramientas existentes de DeFi para construir un mecanismo de inversión simple y, sobre todo, auditable. La creación de este prototipo en un entorno académico controlado aporta valor al servir como un caso de estudio documentado sobre la implementación de estos sistemas, sentando bases prácticas para futuras aplicaciones que busquen aprovechar la naturaleza verificable de la blockchain [4].

1.4. Delimitaciones y Alcances

1.4.1. Alcance

El presente proyecto se centra en el diseño, implementación y validación de un sistema de inversión colectiva basado en contratos inteligentes desplegados en una red de prueba blockchain. El prototipo será desarrollado en el lenguaje Solidity y ejecutado en la red *Sepolia*, perteneciente al ecosistema Ethereum.

El sistema permitirá a múltiples participantes aportar fondos en Ether de prueba, simular un proceso de inversión y distribuir los intereses acumulados a uno de los participantes seleccionados aleatoriamente. Para ello, se implementará una arquitectura modular basada en el patrón *Factory*, que facilitará la creación de múltiples instancias independientes del contrato.

El alcance incluye la validación funcional del sistema a través de pruebas controladas, utilizando herramientas estándar como Remix, Hardhat y MetaMask. Este enfoque busca demostrar la viabilidad técnica del sistema en un entorno seguro, sin involucrar activos reales ni condiciones de producción. El prototipo se presenta como una prueba de concepto con valor académico, susceptible de ser adaptado o extendido en desarrollos futuros.

1.4.2. Delimitaciones

Este proyecto se desarrolla exclusivamente en un entorno de pruebas (*testnet*), por lo que no contempla aspectos propios de un despliegue en producción. En consecuencia, no se abordarán desafíos relacionados con tarifas reales de transacción (*gas*), manejo de criptomonedas con valor económico, ni riesgos de seguridad en contextos abiertos o comerciales.

La solución trabajará únicamente con la red Ethereum en su versión de pruebas (*Sepolia*), sin contemplar interoperabilidad entre cadenas (*cross-chain*) ni integración con activos externos como tokens ERC-20, monedas fiduciarias o protocolos oraculares.

Tampoco se considerará el diseño de mecanismos de consenso, desarrollo de infraestructura blockchain propia ni optimizaciones para alta escalabilidad. El objetivo es demostrar el funcionamiento lógico del sistema de inversión en un entorno simulado, bajo restricciones técnicas y de tiempo acordes con el contexto académico en el que se enmarca este trabajo.

Capítulo 2

Marco Teórico

2.1. Áreas Temáticas

Para enmarcar formalmente este trabajo dentro de la disciplina, se identifican las siguientes áreas de conocimiento según la clasificación de la ACM (Association for Computing Machinery):

- **Software and its engineering → Software design engineering:** El proyecto aplica principios de diseño de software a través de la implementación de patrones reconocidos como el Patrón Factory y el Patrón Guard, fundamentales para crear una arquitectura modular y robusta.
- **Security and privacy → Cryptography:** Aunque el proyecto no implementa nuevos algoritmos, se fundamenta en los principios de la criptografía de clave pública para la gestión de la propiedad de activos y la autorización de transacciones, elementos esenciales para el funcionamiento de cualquier aplicación en blockchain.
- **Applied computing → Financial services:** El proyecto se sitúa directamente en el dominio de la computación aplicada a los servicios financieros, explorando nuevos mecanismos de inversión basados en la tecnología de Finanzas Descentralizadas (DeFi).

2.2. Fundamentos conceptuales y técnicos

En este proyecto se aborda el diseño y desarrollo de un sistema de inversión descentralizado basado en Finanzas Descentralizadas (DeFi) y contratos inteligentes, utilizando la tecnología blockchain. La investigación explorará conceptos de DeFi, contratos inteligentes, plataformas de blockchain y los beneficios y desafíos de implementar soluciones DeFi en el contexto de inversión. Este marco teórico proporciona un contexto detallado y estructurado sobre estos temas, destacando los aspectos computacionales necesarios para la implementación del proyecto.

2.2.1. Tecnología Blockchain

La tecnología Blockchain es un libro de registro abierto y distribuido que puede asentar transacciones entre dos partes de manera eficiente, verificable y permanente [8]. Funciona como una base de datos descentralizada cuya información se agrupa en bloques enlazados de forma cronológica y segura mediante criptografía [9]. Este diseño fue introducido por primera vez para la criptomoneda Bitcoin, y su principal innovación es permitir que los miembros de una red de pares (*peer-to-peer*)

lleguen a un consenso sobre el estado del registro sin necesidad de una autoridad central [10].

Una de las características fundamentales de esta tecnología es la inmutabilidad, que asegura que una transacción, una vez registrada en la cadena de bloques, no puede ser alterada [9]. Cada bloque contiene un *hash* criptográfico del bloque que le precede, creando una cadena robusta. Si se intentara modificar un bloque, su *hash* cambiaría, lo que rompería la integridad de todos los bloques posteriores y alertaría a la red sobre la manipulación [10].

La segunda propiedad clave es la descentralización, lograda al distribuir y sincronizar copias del registro entre todos los participantes de la red. Este enfoque elimina la dependencia de intermediarios tradicionales, como los bancos, y establece un nuevo paradigma donde la confianza se deposita directamente en el protocolo criptográfico subyacente [11]. Por su capacidad para ofrecer transparencia y seguridad, la blockchain se considera una tecnología fundacional con el potencial de transformar industrias enteras [8].

2.2.2. Finanzas Descentralizadas (DeFi)

Las Finanzas Descentralizadas (DeFi) representan un cambio de paradigma con respecto a las finanzas tradicionales, al proponer un ecosistema de aplicaciones y mercados financieros construidos sobre redes blockchain públicas y de código abierto [1]. El objetivo principal de DeFi es la desintermediación: la recreación de servicios financieros complejos —como préstamos, intercambios de activos y derivados— sin depender de intermediarios centralizados tales como bancos o corredores [1].

La funcionalidad de este ecosistema se fundamenta en el uso de contratos inteligentes, programas autónomos y auto-ejecutables que codifican la lógica de las operaciones financieras directamente en la blockchain [2]. Esto da lugar a varias de sus propiedades definitorias. Primero, son sistemas abiertos y sin permisos (permissionless), lo que significa que cualquier usuario con una conexión a Internet y una billetera compatible puede participar sin necesidad de aprobación previa [2]. Segundo, ofrecen una transparencia radical, ya que todas las transacciones y el código de los contratos son públicamente auditables en la cadena de bloques [2]. Finalmente, una de las propiedades más innovadoras de DeFi es la componibilidad (composability): los protocolos se asemejan a "legos de dinero", que pueden combinarse e interconectarse para construir productos financieros cada vez más sofisticados de forma flexible y modular [1, 4].

2.2.3. Contratos Inteligentes

Un contrato inteligente es un programa auto-ejecutable que se almacena en una infraestructura blockchain, diseñado para reforzar y automatizar los términos de un acuerdo cuando se satisfacen una serie de condiciones predefinidas [12, 13]. La aplicación fundamental de estos protocolos computacionales es la automatización de flujos de trabajo y procesos de negocio, lo que permite optimizar la eficiencia, reducir costos operativos y mitigar la dependencia de intermediarios fiduciarios en la

validación de transacciones [13].

Desde una perspectiva técnica, estos programas residen en una dirección específica de la blockchain y se componen de código, que define su lógica operativa, y de datos, que representan su estado en un momento dado [12]. Esta arquitectura, en la que el contrato es replicado y procesado por una red descentralizada, le confiere un conjunto de características determinantes para su fiabilidad:

- **Autonomía:** Una vez desplegado, el contrato opera sin la necesidad de control o intervención humana. La Fundación Ethereum ilustra este concepto mediante la analogía de una máquina expendedora digital, la cual garantiza la ejecución de un resultado si se proveen las entradas correctas, eliminando al intermediario del proceso [12].

- **Inmutabilidad:** El registro criptográfico en la blockchain asegura que los términos del contrato no puedan ser alterados con posterioridad a su despliegue. Esta propiedad es fundamental para prevenir el fraude y la manipulación de los acuerdos establecidos entre las partes [13].

- **Transparencia:** Aunque no toda la información es necesariamente pública, las transacciones que se ejecutan sobre el contrato son registradas en un libro mayor compartido y distribuido, lo que permite a las partes autorizadas tener visibilidad sobre el progreso de los acuerdos [13].

En consecuencia, la implementación de contratos inteligentes representa un cambio paradigmático en la ejecución de acuerdos, desplazando la confianza desde las entidades y los procesos humanos hacia la certeza criptográfica del código.

2.2.4. Verificabilidad y Lógica Determinista

La propiedad de **verificabilidad on-chain** se sustenta en la transparencia inherente de las cadenas de bloques públicas. En una red como la testnet Sepolia de Ethereum, cada operación —desde el depósito inicial hasta el reparto periódico de intereses— queda registrada de forma inmutable en un libro mayor distribuido que cualquier persona puede inspeccionar a través de un explorador de bloques [14]. Esta visibilidad facilita auditorías independientes y continuas, puesto que todo el historial contable y el código del contrato inteligente se encuentran abiertos al escrutinio público [15].

Por otro lado, los contratos inteligentes siguen una **lógica determinista**: dados el mismo estado global y las mismas entradas, todos los nodos llegarán a un resultado idéntico. Este comportamiento está garantizado por la Ethereum Virtual Machine (EVM), cuya función de transición de estado está especificada matemáticamente en el protocolo [3]. La ejecución determinista elimina la ambigüedad y reduce la superficie de manipulación, porque las reglas de reparto de fondos quedan fijadas en el código y no pueden modificarse una vez que el contrato ha sido desplegado.

2.2.5. Mecanismos de Generación de Intereses en DeFi

Uno de los mecanismos más habituales para la obtención de rendimientos en las Finanzas Descentralizadas (*DeFi*) son los **protocolos de préstamo**. Estos sistemas reúnen los activos depositados por los usuarios en *pools* de liquidez; posteriormente, otros participantes pueden tomar préstamos de dichos *pools* pagando una tasa de interés que se redistribuye entre los proveedores de liquidez, lo que genera un rendimiento pasivo para ellos [16], Un caso representativo es el protocolo **Aave**[17].

El contrato `SimpleDeFi.sol` desarrollado en este proyecto abstrae esta lógica para simular la acumulación de intereses de manera programática.

2.2.6. Patrones de Diseño de Contratos Inteligentes

El desarrollo de contratos inteligentes seguros y mantenibles se apoya en **patrones de diseño** establecidos, los cuales facilitan la modularidad, reducen costes de gas y mitigan vulnerabilidades. Entre los catálogos más citados se encuentra la colección de *Contract Structural Patterns* de CSIRO, donde se clasifica el patrón *Factory* como mecanismo on-chain para generar instancias de otros contratos [18]. Complementariamente, la biblioteca OpenZeppelin documenta módulos de seguridad como `ReentrancyGuard`, que implementan defensas prácticas contra ataques de reentrada [19].

En este proyecto se incorporan dos patrones clave:

- **Patrón *Factory*.** Un contrato "fábrica" despliega dinámicamente nuevas instancias de otro contrato y conserva un registro de sus direcciones. Nuestro contrato `DeFiLotteryETHFactory` crea instancias independientes de `DeFiLotteryETH`, aislando el estado de cada inversión y centralizando la trazabilidad [18].
- **Patrón *Guard* (Reentrancy Guard).** Los ataques de reentrada permiten que un contrato malicioso vuelva a invocar una función antes de que finalice la primera ejecución, pudiendo drenar fondos. Siguiendo las recomendaciones de OpenZeppelin, heredamos de `ReentrancyGuard` y protegemos las funciones sensibles con el modificador `nonReentrant`, bloqueando llamadas recursivas dentro de la misma transacción [19].

2.2.7. Mainnet vs. Testnet

En el ecosistema Ethereum existen distintos **tipos de redes** que ejecutan el mismo protocolo, pero con propósitos diferentes.

- **Mainnet** es la red pública de producción donde se registran transacciones con Ether (ETH) de valor económico real. Cualquier operación efectuada allí —desde transferencias simples hasta la ejecución de contratos— queda inscrita de forma inmutable y con implicaciones financieras directas.

- Las **testnets públicas** (p. ej. Sepolia o Hoodi) replican el comportamiento de Mainnet, pero usan ETH sin valor de mercado. Sirven como entornos de *staging* para desarrollar, depurar y auditar contratos inteligentes antes de llevarlos a producción, evitando riesgos y costes de gas reales.

La documentación oficial de Ethereum recomienda *siempre* desplegar y probar el código en una testnet antes de migrarlo a Mainnet, pues así se valida la lógica del contrato en condiciones casi idénticas a las de producción sin exponer fondos reales ni la integridad de la red principal [20].

2.3. Estado del Arte y Trabajos Relacionados

El ecosistema de las Finanzas Descentralizadas (DeFi) ha madurado hasta convertirse en un campo robusto con una variedad de primitivas financieras componibles. Para contextualizar el presente proyecto, es esencial analizar los protocolos fundacionales que han validado los mecanismos de generación de rendimiento y de ahorro colectivo. Este análisis se estructura en dos áreas clave: los protocolos de préstamo que sirven como motor de rendimiento y los modelos de "juegos sin pérdida" que se alinean directamente con la lógica de nuestro contrato.

2.3.1. Protocolos Fundacionales de Préstamo y Generación de Rendimiento

La capacidad de generar intereses sobre activos digitales de forma descentralizada es una de las piedras angulares de DeFi. Dos de los protocolos más influyentes en esta área son Aave y Compound.

Aave: Es un protocolo de mercado monetario descentralizado y de código abierto donde los usuarios pueden participar como depositantes o prestatarios [17]. Los depositantes proveen liquidez al mercado para obtener un ingreso pasivo, mientras que los prestatarios pueden tomar préstamos utilizando una garantía sobrecolateralizada. El protocolo utiliza un modelo algorítmico para determinar las tasas de interés en función de la oferta y la demanda dentro de cada pool de liquidez [17].

Compound Finance: De manera similar, Compound es un protocolo algorítmico y autónomo que permite a los usuarios y aplicaciones obtener intereses sobre sus criptomonedas o tomar préstamos contra ellas [21]. Los saldos de cada activo en el protocolo se agregan y los intereses se devengan en cada bloque de la blockchain, convirtiéndose en un componente fundamental de la infraestructura DeFi [21].

Estos protocolos son relevantes para nuestro proyecto porque validan el principio de que es posible depositar un activo digital y obtener un rendimiento a lo largo del tiempo de forma automatizada y regida por código. Nuestro contrato `SimpleDeFi.sol` simula este mismo mecanismo, actuando como una representación funcional de un motor de rendimiento.

2.3.2. Mecanismos de Ahorro Colectivo y Juegos sin Pérdida

Más allá de la generación de rendimiento, ha surgido un modelo innovador que combina el ahorro con elementos de gamificación. El ejemplo más destacado de esta categoría es PoolTogether.

PoolTogether: Es un protocolo de ahorro basado en premios que permite a los usuarios depositar fondos con la posibilidad de ganar premios, similares a una lotería, pero sin perder nunca su depósito inicial [22]. El mecanismo funciona de la siguiente manera: los depósitos de todos los usuarios se agrupan y se invierten en protocolos de generación de rendimiento (como los mencionados Aave o Compound). Los intereses generados por el capital colectivo constituyen el premio. Al final de un período, se selecciona aleatoriamente a un ganador que recibe todos los intereses acumulados, mientras que todos los participantes, incluido el ganador, pueden retirar su depósito original completo. Este modelo se conoce popularmente como "lotería sin pérdida"(no-loss lottery) [22].

2.3.3. Análisis Crítico y Posicionamiento del Proyecto

El estado del arte revela que los componentes de nuestro sistema se basan en mecanismos ya validados por el mercado. Protocolos como **Aave** y **Compound** han establecido la infraestructura para la generación de rendimiento de manera fiable y descentralizada. Por otro lado, **PoolTogether** ha validado con éxito el modelo de negocio de los "juegos sin pérdida", demostrando que es posible combinar el ahorro con la posibilidad de obtener una ganancia significativa a través de un sorteo.

El objetivo de esta tesis, sin embargo, no es la creación de un producto comercial para competir con estas plataformas. La contribución de este trabajo es de naturaleza académica e ingenieril: se centra en el **diseño, desarrollo y documentación de un prototipo funcional desde cero** que implementa un modelo de inversión colectiva con sorteo. El valor de este proyecto radica en:

1. **El Diseño Arquitectónico:** Se propone y valida una arquitectura modular basada en el patrón *Factory*, que aísla cada ciclo de inversión en un contrato independiente, ofreciendo un modelo de gestión escalable y ordenado.

2. **La Implementación de Referencia:** Se provee una implementación completa y documentada en Solidity, sirviendo como un caso de estudio práctico sobre cómo estructurar la lógica de negocio, gestionar los estados del ciclo de vida de la inversión y manejar los flujos de fondos entre contratos.

3. **La Validación en un Entorno Controlado:** Se demuestra la viabilidad técnica y el correcto funcionamiento de la lógica del sistema en una red de pruebas (testnet), validando cada paso del proceso, desde el depósito inicial hasta la distribución final de los fondos y los intereses.

En resumen, mientras que PoolTogether ofrece una solución robusta a nivel de producto, este proyecto ofrece una **radiografía detallada del proceso de ingeniería de software** necesario

para construir un sistema de este tipo, aportando valor como un recurso educativo y un caso de estudio técnico.

2.4. Resultados Esperados

Al culminar este proyecto de grado, se habrán materializado una serie de resultados tangibles que demuestran el cumplimiento de los objetivos planteados. Estos entregables se centran en el desarrollo del prototipo y en la documentación del proceso de ingeniería, tal como se describe a continuación:

1. **Un prototipo funcional del sistema de contratos inteligentes.** Este es el resultado técnico principal del proyecto. Consiste en un sistema de tres contratos inteligentes interoperables, escritos en Solidity y desplegados exitosamente en la red de pruebas (testnet) de Sepolia. El sistema está compuesto por:
   - Un contrato fábrica (`DeFiLotteryETHFactory`) para la creación y registro de nuevas instancias de inversión.
   - Un contrato base (`DeFiLotteryETH`) que gestiona el ciclo de vida de una inversión grupal.
   - Un contrato de simulación (`SimpleDeFi`) que emula un protocolo de generación de rendimiento.
2. **La validación del prototipo, documentada en el capítulo de Pruebas.** En lugar de un informe externo, este resultado se materializa en el capítulo de pruebas de esta tesis. Dicho capítulo contiene la evidencia de la correcta operación del sistema, incluyendo:
   - Los casos de prueba diseñados y ejecutados para verificar la **funcionalidad** de cada contrato de manera aislada.
   - La descripción de las pruebas de **integración** que validan el flujo completo del sistema, asegurando que los contratos interactúan correctamente entre sí.
   - Un análisis básico del **consumo de gas** de las funciones principales, demostrando la viabilidad de las operaciones en un entorno blockchain.
3. **La documentación de la arquitectura y la implementación, integrada en este documento.** El diseño técnico del sistema se presenta como un resultado en sí mismo, contenido en los capítulos de Metodología y Diseño de esta tesis. Esto incluye:
   - La descripción y justificación de la **arquitectura modular** seleccionada, incluyendo el uso del Patrón Factory.
   - El detalle de las **funciones, eventos y variables de estado** más relevantes de cada contrato inteligente, explicando su propósito y funcionamiento.

Capítulo 3

Análisis Comparativo y Selección de Plataforma Blockchain para el Desarrollo de Contratos Inteligentes DeFi

3.1. Selección de Plataformas Candidatas

Para identificar una plataforma adecuada para el desarrollo del contrato inteligente con fines DeFi, se estableció un conjunto de plataformas blockchain candidatas basado en su madurez tecnológica, soporte para contratos inteligentes, aplicabilidad en sistemas financieros descentralizados y respaldo académico. Esta selección contempla tanto plataformas ampliamente reconocidas por la literatura como aquellas con potencial creciente en el ecosistema DeFi.

Las plataformas seleccionadas son: **Ethereum**[23], **Hyperledger Fabric**[24], **Corda**[25], **Cardano**[26], **Solana**[27] y **Polkadot**[28].

Ethereum es considerada la plataforma pionera en la implementación de contratos inteligentes gracias a su lenguaje Solidity y su máquina virtual (EVM), la cual proporciona un entorno Turing-completo para la ejecución de lógica compleja. Es la plataforma más analizada académicamente, con amplio respaldo en evaluaciones comparativas y estudios sobre seguridad, escalabilidad y adopción DeFi [29, 30, 31, 32, 33, 34, 35, 36, 37].

Hyperledger Fabric es una plataforma permissioned desarrollada por la Linux Foundation, orientada a entornos empresariales. Su arquitectura modular y soporte para canales privados la posicionan como una alternativa robusta en contextos donde la privacidad y la gobernanza son esenciales. Aunque no está orientada nativamente a DeFi, su flexibilidad ha permitido casos de uso financieros en consorcios cerrados [29, 30, 31, 32, 33].

Corda, desarrollada por R3, se diferencia por su enfoque en transacciones financieras entre partes conocidas y su modelo de notary-based consensus. Si bien no utiliza una blockchain tradicional, permite contratos inteligentes que operan con alta eficiencia y privacidad, características relevantes para simular sistemas financieros auditables [30, 31, 32, 33].

Cardano implementa el protocolo de consenso *Ouroboros*, basado en pruebas formales y revisa-

do por pares. Utiliza un modelo eUTxO extendido y Plutus como lenguaje de contratos inteligentes. Diversos estudios destacan sus propiedades de seguridad, verificabilidad formal y su idoneidad para aplicaciones financieras auditables [32, 34, 38, 39].

Solana combina el algoritmo de consenso *Proof of History* con *Proof of Stake*, permitiendo alta escalabilidad. Su modelo basado en Rust sobre BPF y su uso en diversas dApps DeFi han sido estudiados en revisiones técnicas recientes, que analizan tanto su rendimiento como los riesgos asociados al desarrollo de contratos [32, 40, 41, 35].

Polkadot está diseñada como una red multichain interoperable, con parachains que se comunican mediante una cadena de relevo y contratos desarrollados principalmente con Ink!. Estudios recientes analizan su arquitectura para integrar liquidez entre cadenas y evaluar la viabilidad de contratos inteligentes interoperables [30, 32, 36, 37].

3.2. Criterios de Inclusión y Exclusión

Para delimitar el conjunto de plataformas que serán comparadas en profundidad, se establecieron criterios objetivos de inclusión y exclusión alineados con los requerimientos.

Criterios de inclusión (I)

- **I1.** Lenguaje de contratos inteligentes que permita implementar lógica DeFi (condicionales, cálculos, gestión de estados).
- **I2.** Disponibilidad de *testnet* pública y gratuita para despliegue y pruebas.
- **I3.** Red *permissionless* o, al menos, capacidad nativa para aplicaciones DeFi abiertas.
- **I4.** Ecosistema activo de desarrollo: documentación, herramientas y comunidad.
- **I5.** Cobertura académica suficiente (artículos revisados por pares o libros blancos reconocidos).

Criterios de exclusión (E)

- **E1.** Arquitectura *permissioned* estricta que limite la composabilidad DeFi.
- **E2.** Ausencia de *testnet* pública o requerimiento de infraestructura privada.
- **E3.** Ecosistema DeFi incipiente o inexistente, dificultando la obtención de métricas relevantes.
- **E4.** Escasa atención académica reciente.

Cuadro 3.1: Resultado del primer filtro de plataformas

Plataforma	I1	I2	I3	I4	I5	**Resultado**
Ethereum [23]	✓	✓	✓	✓	✓	**Incluida**
Cardano [26]	✓	✓	✓	✓	✓	**Incluida**
Polkadot [28]	✓	✓	✓	✓	✓	**Incluida**
Solana [27]	✓	✓	✓	✓	✓	**Incluida**
Hyperledger Fabric [24]	✓	✗	✗	✓	✓	**Excluida (E1, E2, E3)**
Corda [25]	✓	✗	✗	✓	✓	**Excluida (E1, E2, E3)**

Aplicación del filtro

Como se observa en la Tabla 3.1, **Ethereum**, **Cardano**, **Polkadot** y **Solana** cumplen todos los criterios de inclusión y pasan al análisis comparativo detallado. **Hyperledger Fabric** y **Corda** se excluyen por requerir redes *permissioned* sin *testnet* pública, lo que limita su idoneidad para un protocolo DeFi abierto y auditable.

3.3. Definición y explicación de los Criterios de Calificación

La selección de una plataforma blockchain adecuada para desplegar contratos inteligentes vinculados a protocolos de finanzas descentralizadas (DeFi) no puede basarse únicamente en popularidad o disponibilidad tecnológica. Dado que este proyecto tiene como objetivo el desarrollo de un contrato inteligente auditable, que funcione como mecanismo de inversión seguro y verificable, se requiere una evaluación sistemática y fundamentada de las plataformas candidatas, con base en criterios técnicos, prácticos y estratégicos.

Diversos trabajos académicos recientes han propuesto enfoques rigurosos para comparar plataformas de contratos inteligentes [42, 43, 44, 45, 46, 47, 48]. En conjunto, estas fuentes sirven como fundamento para definir los siguientes criterios.

3.3.1. Criterios seleccionados

- **Lenguaje y herramientas de desarrollo:** Considera la disponibilidad, madurez y extensibilidad del ecosistema de desarrollo, incluyendo frameworks, entornos de prueba, soporte de lenguajes y recursos de depuración. Este aspecto es determinante en la productividad del desarrollo de contratos inteligentes y su capacidad de mantenimiento a largo plazo [43, 45, 46, 47, 48].

- **Facilidad de despliegue y configuración:** Evalúa el grado de complejidad requerido para poner en funcionamiento un contrato inteligente, incluyendo creación de cuentas, uso de testnets, documentación y disponibilidad de entornos preconfigurados. Este criterio es especial-

mente relevante para reducir la barrera de entrada en entornos académicos o de prototipado rápido [43, 46, 47].

- **Costos de transacción y escalabilidad:** Mide la eficiencia económica del uso de la plataforma y su capacidad para sostener crecimiento en la demanda. Esto incluye las tarifas de red, el impacto económico de la ejecución frecuente de contratos, y los mecanismos arquitectónicos que permiten escalar sin comprometer el rendimiento [43, 46, 47, 48].
- **Velocidad y rendimiento de red:** Analiza el desempeño transaccional medido en tiempo de confirmación, latencia y transacciones por segundo (TPS). Estos parámetros afectan directamente la experiencia de usuario en aplicaciones financieras que requieren inmediatez y confiabilidad [43, 46, 48].
- **Seguridad y estabilidad:** Evalúa la robustez técnica de la red y su historial frente a ataques o fallos. Incluye la presencia de auditorías formales, mecanismos de verificación y protocolos de recuperación ante errores. Este criterio es esencial para garantizar la integridad de contratos vinculados a activos financieros [44, 46, 47, 48].
- **Mecanismo de consenso y gobernanza:** Se refiere al proceso mediante el cual se valida la información y se gestionan las actualizaciones del protocolo. Este criterio cobra relevancia en contextos de sostenibilidad, adaptación a cambios regulatorios y participación comunitaria estructurada [44, 46, 47, 48].
- **Comunidad y soporte para desarrolladores:** Mide la existencia de una base activa de usuarios técnicos, disponibilidad de documentación, foros, recursos abiertos y contribución colaborativa. Una comunidad fuerte facilita la solución de problemas, acelera el desarrollo y promueve buenas prácticas [43, 45, 46, 47].

Cada uno de estos criterios ha sido cuidadosamente seleccionado no solo por su relevancia académica, sino también por su impacto directo en la viabilidad técnica, económica y operativa del contrato inteligente propuesto en este proyecto. En la siguiente sección se describe cómo estos criterios serán ponderados y aplicados para priorizar la plataforma más adecuada.

3.4. Priorización y justificación metodológica

Una vez definidos los criterios de calificación, el siguiente paso consiste en establecer una metodología estructurada para priorizar las plataformas blockchain candidatas. Esta priorización busca minimizar la subjetividad, permitir una comparación transparente y asegurar que la decisión final esté fundamentada en argumentos técnicos, metodológicos y estratégicos.

3.4.1. Justificación del enfoque multicriterio

La selección de tecnologías en entornos complejos, como el ecosistema blockchain, implica analizar variables heterogéneas que no pueden ser evaluadas por un único indicador. Por esta razón,

múltiples autores recomiendan enfoques multicriterio para la toma de decisiones tecnológicas. Voloder valida la pertinencia de esta estrategia al aplicar puntajes ponderados sobre diferentes plataformas en función de escenarios reales de uso de contratos inteligentes [42]. Asimismo, T. Hiemstra destaca que las decisiones en torno a plataformas blockchain deben considerar simultáneamente factores como rendimiento, escalabilidad, usabilidad y comunidad técnica [45].

De manera complementaria, el enfoque de evaluación multicriterio ha sido respaldado por investigaciones en ingeniería de decisiones. Berumen y Llamazares [49] resaltan que métodos como AHP permiten estructurar el proceso de toma de decisiones mediante comparaciones pareadas jerárquicas, facilitando el análisis de múltiples criterios tanto cualitativos como cuantitativos sin suplantar el juicio experto, sino haciéndolo explícito y coherente. Esta capacidad para organizar variables diversas en un modelo claro y reproducible fortalece la confiabilidad de la metodología empleada en este trabajo.

3.4.2. Modelo de priorización adoptado

La metodología adoptada consiste en una *matriz de puntuación ponderada*, donde a cada plataforma se le asigna una calificación entre 1 (muy bajo) y 5 (muy alto) por cada criterio, y dicha calificación se multiplica por el peso asignado. La suma de los productos genera una *puntuación final*, que permite comparar objetivamente las plataformas en función de su adecuación al contexto del proyecto.

Este enfoque ha sido utilizado en investigaciones previas con éxito [43], y resulta especialmente pertinente para proyectos académicos, ya que no requiere transformaciones complejas ni herramientas matemáticas especializadas, manteniendo la trazabilidad del proceso de decisión.

3.4.3. Asignación de pesos a los criterios

La distribución de los pesos se justifica con base en el alcance real del proyecto: se desarrollará un contrato inteligente auditable en una red de pruebas (testnet), sin realizar un despliegue en producción ni manejar fondos reales. En consecuencia, se da prioridad a los aspectos que inciden directamente en la implementación, documentación y análisis del sistema, más que a su viabilidad operativa en entornos comerciales. Bajo estas condiciones, los pesos asignados son los siguientes:

- **Lenguaje y herramientas de desarrollo – 20 %:** Este criterio es crítico, ya que determina la facilidad de implementación, la curva de aprendizaje y la capacidad del equipo para crear, probar y mantener el contrato.
- **Seguridad y estabilidad – 20 %:** Aunque no se trabajará con activos reales, esta valoración es relevante para establecer la confiabilidad potencial del contrato en contextos reales y validar la calidad del entorno técnico.

- **Facilidad de despliegue – 15 %:** Dado que el proyecto se ejecutará en testnet, es fundamental que el entorno de pruebas sea accesible, bien documentado y compatible con flujos de trabajo ágiles. Plataformas que simplifican el setup inicial y la interacción con contratos obtienen mejor puntuación.

- **Comunidad y soporte – 15 %:** Una comunidad activa facilita el aprendizaje, la solución de errores y el uso de bibliotecas de código abierto. Este criterio es esencial en entornos académicos o de prototipado rápido, donde el acceso a recursos compartidos marca una diferencia significativa.

- **Velocidad y rendimiento – 10 %:** Aunque en entornos de prueba el rendimiento real no se experimenta, este criterio sigue siendo útil para evaluar la adecuación de la plataforma a posibles escenarios futuros con mayor carga operativa.

- **Gobernanza y mecanismo de consenso – 10 %:** Se valora con el fin de entender cómo evoluciona la plataforma y cómo se integran los cambios. Es clave para estimar la sostenibilidad del proyecto en caso de futura migración a red principal.

- **Costos de transacción y escalabilidad – 10 %:** Aunque no se incurre en gastos en testnet, este criterio conserva relevancia para el análisis comparativo y el estudio de la viabilidad futura del sistema en producción.

3.4.4. Validez metodológica

El modelo de puntuación ponderada permite adaptar la evaluación a las necesidades reales del proyecto, sin perder rigor académico. Además, su flexibilidad facilita la incorporación de criterios tanto cuantitativos como cualitativos, lo cual es particularmente útil en tecnologías emergentes como blockchain. En plataformas como Solana, Cardano, Polkadot o Ethereum, donde existen modelos de gobernanza on-chain complejos y estructuras de desarrollo dinámicas, esta metodología permite comparar sin imponer restricciones arbitrarias de normalización o escalado [43, 44].

En síntesis, la metodología seleccionada asegura una decisión coherente con los objetivos del proyecto: elegir la plataforma que, sin ser evaluada exclusivamente por su popularidad o velocidad, ofrezca el mejor entorno de desarrollo, aprendizaje y análisis para contratos inteligentes auditables y orientados a aplicaciones DeFi.

Para facilitar la comprensión y consulta rápida del modelo de evaluación, en la Tabla 3.2 se presenta un resumen de los criterios seleccionados, su descripción abreviada y el peso asignado en la priorización.

Cuadro 3.2: Resumen de criterios de evaluación de plataformas blockchain

Criterio	**Descripción resumida**	**Peso (%)**
Lenguaje y herramientas	Facilidad de programación, depuración y pruebas con IDEs, frameworks y librerías activas.	20
Seguridad y estabilidad	Robustez técnica, historial de fallos, uso de auditorías y verificación formal.	20
Facilidad de despliegue	Documentación clara, soporte para testnet, rapidez en la configuración inicial.	15
Comunidad y soporte	Tamaño, actividad y diversidad de la comunidad técnica; acceso a foros, ejemplos y bibliotecas.	15
Velocidad y rendimiento	Latencia, TPS, consistencia y adaptabilidad bajo carga.	10
Gobernanza y consenso	Mecanismos para validar transacciones y actualizar protocolos; apertura y participación comunitaria.	10
Costos y escalabilidad	Costo por transacción y capacidad para adaptarse a demandas crecientes.	10

3.5. Evaluación de las plataformas seleccionadas

3.5.1. Metodología aplicada

A partir de los criterios definidos y ponderados en la sección anterior, se procede a evaluar comparativamente cuatro plataformas blockchain seleccionadas: **Ethereum**, **Cardano**, **Polkadot** y **Solana**. La elección de estas plataformas responde tanto a su relevancia en el ecosistema DeFi como a su respaldo académico y técnico.

3.5.2. Evaluación por criterio

- **Lenguaje y herramientas de desarrollo (20 %):**

  Ethereum presenta el ecosistema de desarrollo más consolidado entre las plataformas analizadas. El uso del lenguaje *Solidity*[50], junto con la *Ethereum Virtual Machine* (EVM)[51], ha permitido la creación de entornos de desarrollo ampliamente adoptados como Remix[52], Hardhat[53] y Truffle[54]. Esta combinación ha convertido a Ethereum en la principal referencia para la programación de contratos inteligentes y aplicaciones descentralizadas [46, 47, 48, 43, 45].

  Cardano emplea un enfoque más académico, con el uso de *Haskell*[55] y la máquina virtual *Plutus*[56]. Si bien este diseño proporciona garantías formales de seguridad, su complejidad sintáctica y semántica representa una barrera significativa para desarrolladores. La disponibilidad de entornos de desarrollo y herramientas es limitada en comparación con Ethereum, lo cual restringe su adopción en aplicaciones comerciales [47, 48, 45].

Solana basa su desarrollo en lenguajes como *Rust*[57] y *C*[58], que ofrecen alto rendimiento pero requieren habilidades técnicas avanzadas. La escasez relativa de herramientas accesibles y la menor disponibilidad de librerías y frameworks específicos dificultan su adopción por parte de desarrolladores menos especializados [48, 43, 45].

Polkadot utiliza *Rust*[57] y el framework *Substrate*[59], lo que proporciona una gran flexibilidad y capacidad de personalización en el diseño de blockchains. Sin embargo, su orientación técnica avanzada y la complejidad inherente del entorno reducen la accesibilidad para desarrolladores sin experiencia previa en sistemas distribuidos o lenguajes de bajo nivel [44, 45].

- **Seguridad y estabilidad (20 %):** Ethereum ha mejorado significativamente su robustez técnica tras la transición a *Proof of Stake*, incorporando mecanismos como *slashing* para penalizar validadores maliciosos y mejorar la seguridad de la red. A pesar de su historial de incidentes tempranos (como el caso de The DAO)[60], su ecosistema ha madurado con auditorías frecuentes y un modelo de ejecución ampliamente probado. Estos avances han fortalecido su estabilidad operativa general [44, 45, 47].

  Cardano se distingue por su énfasis en la verificación formal y el uso de pruebas matemáticas a través del protocolo Ouroboros[38], validado por revisiones académicas. Su enfoque de diseño basado en Haskell y fundamentos teóricos le proporciona una de las estructuras de seguridad más rigurosas del ecosistema blockchain. Si bien su despliegue es más conservador, prioriza la estabilidad a largo plazo [44, 45, 47].

  Solana ha enfrentado desafíos importantes en términos de estabilidad. A pesar de mantener una infraestructura criptográfica sólida, su arquitectura de alto rendimiento ha resultado en múltiples interrupciones de red debido a sobrecargas o ataques de spam. Estas fallas han puesto en cuestión su fiabilidad operativa, aunque la red ha tomado medidas técnicas para mitigarlas [43, 45, 48].

  Polkadot implementa un sistema de seguridad compartida a través de su Relay Chain[61], que coordina y protege todas las parachains[62] conectadas. Este modelo, junto con el consenso *Nominated Proof of Stake* (NPoS), reduce la probabilidad de ataques y errores sistémicos. Además, su diseño modular permite aislar incidentes dentro de una parachain sin comprometer la estabilidad global del ecosistema [44, 45].

- **Facilidad de despliegue y configuración (15 %):**

  Ethereum ofrece una experiencia de despliegue madura y accesible, con una amplia documentación oficial, entornos como testnets activas (Goerli, Sepolia [63, 64]) y herramientas como Hardhat y Truffle que automatizan flujos de trabajo desde la compilación hasta el despliegue. Su compatibilidad con entornos EVM facilita la integración en distintos entornos sin necesidad de ajustes técnicos significativos [46, 47, 48, 43, 45].

  Cardano ha mejorado progresivamente su accesibilidad para desarrolladores, pero su proceso de despliegue sigue siendo más técnico. A pesar de contar con testnets como Preprod[65] y herramientas en desarrollo como Plutus Playground[66], el entorno de desarrollo es menos

estandarizado y más dependiente del conocimiento técnico del lenguaje Haskell. Esto dificulta una configuración inicial ágil en comparación con otras plataformas [47, 48, 45].

Solana permite una configuración rápida para desarrolladores familiarizados con Rust o C, y ofrece documentación técnica extensa, así como acceso a testnets funcionales [67]. Sin embargo, la falta de herramientas de alto nivel y la necesidad de configurar entornos de desarrollo específicos limitan la accesibilidad para perfiles menos técnicos. Su ecosistema sigue en expansión, pero aún carece de la estandarización vista en Ethereum [48, 43, 45].

Polkadot, mediante su framework Substrate, permite la creación de blockchains personalizadas, pero su despliegue inicial requiere conocimientos técnicos avanzados. Aunque ofrece herramientas como Polkadot-JS[68] y documentación modular, la configuración inicial implica gestionar entornos complejos y componentes interrelacionados. La barrera de entrada sigue siendo alta para desarrolladores sin experiencia previa en arquitecturas multicadena [44, 45].

- **Comunidad y soporte para desarrolladores (15 %):**

  Ethereum cuenta con la comunidad de desarrolladores más amplia y activa del ecosistema blockchain. Su longevidad y liderazgo en innovación han dado lugar a una enorme disponibilidad de recursos, foros especializados, bibliotecas abiertas, hackathons frecuentes y soporte constante en plataformas como StackOverflow[69], GitHub[70] y Discord[71]. Esta red de soporte facilita el aprendizaje, la colaboración y la resolución de problemas técnicos [46, 47, 48, 43, 45].

  Cardano posee una comunidad técnicamente sólida, con fuerte presencia académica y un enfoque en el desarrollo formal. Aunque su base de desarrolladores es más reducida en comparación con Ethereum, ha crecido de forma consistente y cuenta con documentación detallada y foros como Cardano Forum[72] y Discord. Su naturaleza más especializada limita en parte la diversidad técnica dentro del ecosistema [47, 48, 45].

  Solana ha experimentado un crecimiento rápido en su comunidad de desarrolladores, impulsado por su atractivo técnico y sus programas de incentivos como hackathons y fondos de ecosistema. A pesar de esto, la relativa juventud del proyecto implica una menor madurez en términos de documentación comunitaria y foros especializados como Solana Forum [73]. La comunidad sigue expandiéndose, aunque aún depende de canales técnicos específicos [48, 43, 45].

  Polkadot posee una comunidad activa centrada en desarrolladores con conocimientos técnicos intermedios o avanzados, en parte debido al uso de Substrate. Si bien el ecosistema fomenta la colaboración a través de herramientas como Polkadot-JS, GitHub y foros como Polkadot Forum [74], la barrera técnica ha limitado su crecimiento masivo. No obstante, la comunidad demuestra una alta calidad técnica y organización estructurada [44, 45].

- **Velocidad y rendimiento (10 %):**

  Ethereum, en su capa base, procesa entre 15 y 30 transacciones por segundo (TPS), lo que limita su capacidad de respuesta en momentos de alta demanda. Aunque las soluciones de

segunda capa como Rollups mejoran considerablemente el rendimiento, la red principal aún sufre congestión y aumento en los tiempos de confirmación bajo carga elevada. La transición a *Proof of Stake* ha mejorado la eficiencia energética, pero no ha resuelto completamente las limitaciones de escalabilidad inmediata [46, 47, 48, 43].

Cardano ofrece un rendimiento teórico de hasta 270 TPS gracias a su protocolo Ouroboros. Sin embargo, su arquitectura basada en UTXO y su modelo de ejecución secuencial restringen la paralelización de transacciones. En la práctica, esto se traduce en un rendimiento estable pero no competitivo frente a arquitecturas diseñadas específicamente para alta velocidad [46, 48, 45].

Solana se posiciona como una de las plataformas más rápidas, con un rendimiento teórico que supera las 50,000 TPS y tiempos de bloque de aproximadamente 400 milisegundos. Estas cifras son posibles gracias a la combinación de *Proof of History* con *Proof of Stake* y una arquitectura optimizada para procesamiento paralelo. No obstante, su alta velocidad ha generado problemas de congestión y caídas de red bajo condiciones extremas de tráfico [48, 43, 45].

Polkadot logra un rendimiento competitivo mediante su diseño multicadena, donde las *parachains* procesan transacciones en paralelo. Si bien el TPS agregado varía según el número y capacidad de las parachains activas, esta estructura permite escalar horizontalmente y mantener una latencia controlada. Su rendimiento es alto en teoría, pero depende de la implementación específica de cada parachain [46, 44, 45].

- **Gobernanza y mecanismo de consenso (10 %):**

  Ethereum emplea un mecanismo de consenso basado en *Proof of Stake* desde su migración a Ethereum 2.0, lo que ha mejorado la eficiencia energética y la seguridad de la red. En términos de gobernanza, utiliza un modelo abierto sin autoridad central, donde las decisiones se coordinan a través del proceso de Ethereum Improvement Proposals (EIPs), permitiendo la participación de la comunidad técnica. No obstante, esta informalidad también puede ralentizar procesos críticos de actualización [46, 47, 48, 45].

  Cardano utiliza el protocolo Ouroboros como base para su consenso *Proof of Stake*, el cual ha sido verificado formalmente y revisado académicamente. Su modelo de gobernanza está diseñado en fases (Voltaire, por ejemplo) y contempla votaciones en cadena financiadas por una tesorería descentralizada. Esta estructura permite participación comunitaria estructurada, aunque su implementación aún está en evolución [46, 47, 48, 45].

  Solana combina *Proof of History* con *Proof of Stake*, priorizando velocidad en la validación. Su gobernanza, sin embargo, es más centralizada en comparación con otras redes, y las decisiones clave recaen en gran medida sobre el equipo de desarrollo y validadores técnicos. Esta estructura reduce la transparencia en la actualización del protocolo y limita la participación de la comunidad más amplia [48, 43, 45].

  Polkadot implementa un modelo de gobernanza en cadena altamente estructurado, con roles como el Consejo, el Comité Técnico y los referendos votados por la comunidad. Las actualizaciones de protocolo pueden realizarse sin hard forks, y su sistema de consenso basado en

Nominated Proof of Stake permite una validación segura y participativa. Este diseño posiciona a Polkadot como una de las plataformas más avanzadas en gobernanza descentralizada [44, 45, 46].

- **Costos de transacción y escalabilidad (10 %):**

Ethereum presenta uno de los costos de transacción más altos del ecosistema, especialmente en momentos de congestión. Aunque su transición a *Proof of Stake* ha reducido significativamente el consumo energético, el alto costo del gas sigue siendo una barrera, incluso con la implementación parcial de mejoras como EIP-1559 y soluciones de segunda capa. Estas limitaciones impactan directamente en su escalabilidad práctica sin depender de tecnologías complementarias [46, 47, 48, 43].

Cardano mantiene costos por transacción significativamente más bajos que Ethereum, y su diseño orientado a eficiencia le permite sostener esta ventaja bajo condiciones normales de uso. Sin embargo, su arquitectura limita la ejecución paralela de transacciones, lo que puede restringir su escalabilidad horizontal ante un aumento masivo de la demanda. Su modelo es eficiente en costo, pero conservador en su capacidad de escalar dinámicamente [46, 48, 45].

Solana se destaca por sus bajos costos por transacción (generalmente fracciones de centavo) y un alto rendimiento de red, lo que le permite absorber picos de tráfico sin incrementos drásticos en tarifas. Esta combinación ha sido clave en su adopción en sectores como los NFT y DeFi. Sin embargo, esta escalabilidad se logra a costa de requisitos técnicos elevados y una menor descentralización de infraestructura [48, 43, 45].

Polkadot logra una buena relación entre costos y escalabilidad gracias a su enfoque multicadena. El procesamiento paralelo de parachains permite distribuir la carga y reducir cuellos de botella, manteniendo tarifas relativamente estables. Aunque el costo por transacción puede variar según la parachain específica, el diseño general del sistema está optimizado para crecimiento horizontal eficiente [44, 45, 46].

3.5.3. Resultados de la evaluación

La Tabla 3.3 presenta la evaluación consolidada de cada plataforma en función de los criterios descritos anteriormente, aplicando los pesos correspondientes.

Cuadro 3.3: Evaluación comparativa de plataformas blockchain

Criterio	**Ethereum**	**Cardano**	**Polkadot**	**Solana**
Lenguaje y herramientas (20 %)	5 (1.00)	3 (0.60)	3 (0.60)	3 (0.60)
Seguridad y estabilidad (20 %)	4 (0.80)	5 (1.00)	4 (0.80)	2 (0.40)
Facilidad de despliegue (15 %)	5 (0.75)	3 (0.45)	2 (0.30)	3 (0.45)
Comunidad y soporte (15 %)	5 (0.75)	3 (0.45)	3 (0.45)	4 (0.60)
Velocidad y rendimiento (10 %)	2 (0.20)	3 (0.30)	4 (0.40)	5 (0.50)
Gobernanza y consenso (10 %)	3 (0.30)	4 (0.40)	5 (0.50)	2 (0.20)
Costos y escalabilidad (10 %)	2 (0.20)	4 (0.40)	4 (0.40)	5 (0.50)
Puntaje total	**4.00**	**3.60**	**3.45**	**3.25**

3.6. Elección final de plataforma

La aplicación del marco de evaluación ponderada arroja los siguientes puntajes finales: Ethereum (4.00), Cardano (3.60), Polkadot (3.45) y Solana (3.25). Con base en esta evidencia cuantitativa, se selecciona **Ethereum** como la plataforma tecnológica para la implementación del proyecto.

La elección se fundamenta en el dominio de Ethereum en los criterios que reflejan las prioridades estratégicas del proyecto: la agilidad del desarrollo y la mitigación de riesgos. Obtuvo la máxima calificación (5/5) en **Lenguaje y herramientas (20 %)**, **Facilidad de despliegue (15 %)** y **Comunidad y soporte (15 %)**, lo que demuestra una madurez y una robustez de ecosistema inigualables, a esta fortaleza se suma una alta calificación (4/5) en **Seguridad y estabilidad (20 %)**, confirmando su liderazgo en el entorno del proyecto.

Si bien las plataformas competidoras presentan ventajas notables en métricas aisladas —*como la máxima seguridad en Cardano, la gobernanza en Polkadot o la velocidad en Solana*—, ninguna ofrece el balance integral de Ethereum. La decisión, por tanto, se basa en la combinación de un ecosistema de desarrollo superior con un alto nivel de seguridad comprobada, posicionándolo como la opción más completa y estratégica para el éxito del proyecto.

Capítulo 4

Metodología y Diseño de la Solución

4.1. Arquitectura de los contratos Inteligentes

En esta sección se describe la arquitectura modular empleada en el sistema de contratos desarrollados. El diseño busca maximizar la reutilización de código, facilitar la escalabilidad y aislar cada instancia de lotería en un contrato independiente.

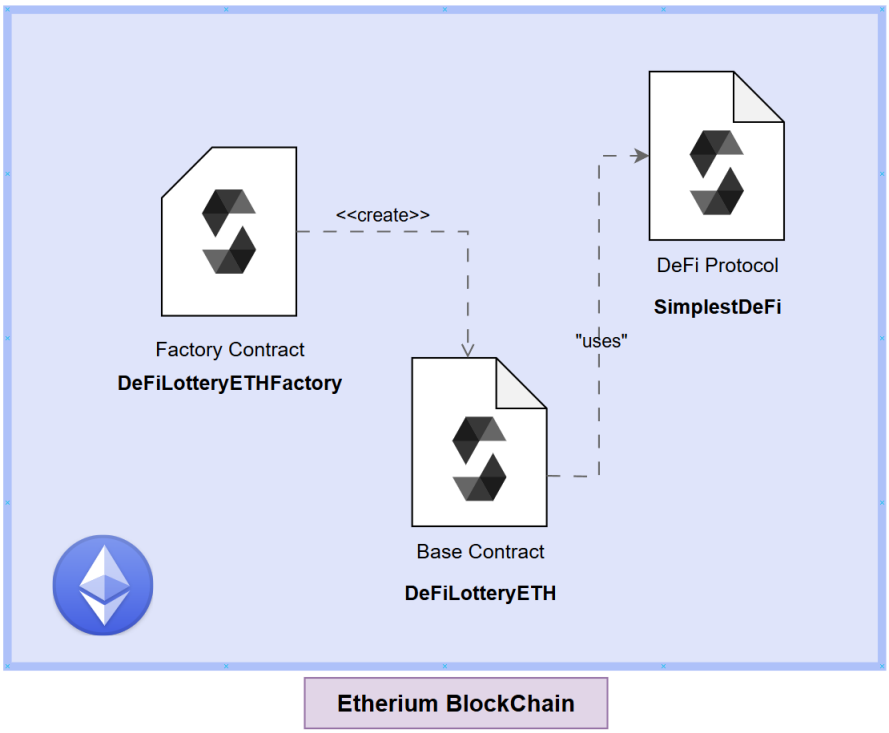

Figura 4.1: Visión general de la arquitectura de contratos inteligentes

En la arquitectura propuesta, se ha optado por utilizar un contrato Factory y un contrato Base para la gestión de transacciones en la blockchain de Ethereum. Esta decisión se fundamenta en principios de modularidad, escalabilidad y eficiencia en la administración de contratos inteligentes y nos permitió manejar cada inversión como un contrato único en lugar de manejar un único contrato

que manejara todas las inversiones. Un contrato Factory actúa como un generador de instancias del contrato Base, permitiendo crear múltiples contratos de transacción sin necesidad de duplicar código. Evitar un contrato monolítico reduce el tamaño del bytecode desplegado y, en consecuencia, los costos de gas en la red Ethereum [75]. Cada contrato de transacción es independiente, lo que minimiza riesgos de corrupción de datos o manipulación indebida. En un futuro, si se requieren modificaciones en la lógica de los contratos Base, el contrato Factory puede ser actualizado para generar versiones mejoradas sin afectar las transacciones previas lo que facilita la evolución del sistema sin comprometer la estabilidad de las transacciones históricas.

En la fase inicial se contempló integrar múltiples protocolos DeFi externos para ofrecer distintas opciones de inversión y conversión de activos; sin embargo, al intentar adaptar nuestra lógica a interfaces heterogéneas, surgieron varios retos comunes en la interoperabilidad DeFi que si bien son solucionables y hacen parte de los obstáculos inherentes a una solución de este tipo, para el propósito y el contexto de esta entrega se salía de nuestros alcances iniciales:

- **Falta de estandarización más allá de ERC-20:** aunque ERC-20 define funciones básicas (transfer, approve, transferFrom), muchos protocolos añaden extensiones (permit, flashLoan, flashMint) o emplean decimales distintos, lo que obliga a implementar adaptadores específicos para cada infraestructura externa.
- **Variabilidad en gestión de activos y conversión:** ciertos protocolos manejan pools multi-activo y swaps internos, mientras otros requieren rutas de intercambio complejas (por ejemplo Uniswap v3 vs. Curve), aumentando la complejidad de nuestros contratos y encareciendo el gas.
- **Limitaciones del entorno testnet Sepolia:** la disponibilidad de tokens de prueba estaba restringida a unos pocos ERC-20 estándar, impidiendo simular flujos cross-asset y afectando la calidad de las pruebas de conversión y liquidez.

Ante estos obstáculos, se optó por diseñar e implementar un protocolo DeFi simplificado —`SimpleDeFi.sol`— que sirviera como "mundo de pruebas" interno. Su lógica básica, inspirada en esquemas de interés fijo, consiste en:

1. El usuario invierte un monto M.
2. Por cada hora transcurrida desde la inversión, el contrato aplica un incremento fijo del 30 % sobre el saldo acumulado:
$$M_{\text{nuevo}} \;=\; M_{\text{actual}} \;\times\; 1.30$$
3. Al retirar la inversión el usuario recibe el monto actualizado.

Este protocolo propio permitió validar el flujo de valoración de inversiones, la emisión de eventos de interés y la interacción con `DeFiLotteryETH` sin depender de librerías externas. Además, al mantener la lógica contenida en un solo contrato, se redujeron los riesgos de incompatibilidad y se optimizó el consumo de gas para las operaciones de cálculo de rendimiento.

4.2. Diseño y funcionamiento de los contratos Inteligentes

El sistema de contratos inteligentes se compone de tres componentes principales que colaboran para gestionar todo el ciclo de inversión y selección de ganadores en un entorno DeFi de prueba. En primer lugar, el *Contrato Factory* funciona como punto de entrada: su única responsabilidad es crear nuevas instancias del contrato base y mantener un registro de las direcciones desplegadas. Cada vez que un usuario inicia una nueva lotería, la fábrica genera un contrato independiente, lo que garantiza que cada evento de inversión quede aislado y no comparta estado con otros.

A continuación, cada *Contrato Base* maneja la lógica específica de una lotería individual: recibe las inversiones de los participantes, controla el cambio de estado (por ejemplo, de `Created` a `Pending`, luego a `Started` y finalmente a `Completed`), y ejecuta el proceso de selección de un ganador y reembolso de los fondos. Gracias a esta separación, es posible actualizar o reemplazar el diseño de la lotería sin afectar otras instancias ya activas.

Por último, el *Contrato de Protocolo DeFi* proporciona la mecanización de los rendimientos de las inversiones mediante un modelo simplificado de interés fijo, simulando un entorno DeFi interno. Este contrato se invoca desde el contrato base para calcular y aplicar el rendimiento acumulado según el tiempo transcurrido.

En conjunto, estos tres contratos siguen un patrón modular que maximiza la reutilización de código, minimiza el tamaño de cada despliegue y facilita el mantenimiento evolutivo: la fábrica gestiona instancias, el contrato base orquesta el flujo de la lotería y el protocolo DeFi abstrae la lógica de cálculo de intereses. A continuación se describe en detalle el diseño y las características de cada uno de estos contratos.

4.2.1. Contrato Factory (`DeFiLotteryETHFactory`)

- **Estado:**
  - `address[] public lotteries;` Array dinámico que almacena las direcciones de todas las loterías creadas hasta el momento.
- **Evento:**
  - `event LotteryCreated(address indexed lotteryAddress);` Se emite cada vez que se despliega una nueva instancia de `DeFiLotteryETH`.
- **Función principal:**

```
function createLottery(
    address[] memory _participants,
    uint256 _duration,
    address _defiProtocol
```

```
    ) external {
        DeFiLotteryETH newLottery = new DeFiLotteryETH(
            _participants,
            _duration,
            _defiProtocol
        );
        lotteries.push(address(newLottery));
        emit LotteryCreated(address(newLottery));
    }
```

- Recibe la lista inicial de participantes, la duración en segundos y la dirección del contrato de protocolo DeFi.
- Despliega una nueva instancia de `DeFiLotteryETH`, pasa los parámetros al constructor y almacena su dirección.
- Emite un evento para notificar a interfaces off-chain o indexadores.

4.2.2. Contrato Base (`DeFiLotteryETH`)

Este contrato encapsula la lógica de una única lotería sobre ETH y un protocolo DeFi simulado. Gestiona el flujo completo: inversión, espera, generación de rendimiento, selección de ganador y reembolso.

- **Interfaces y herencia:**
  - Implementa `ReentrancyGuard` para protegerse contra ataques de reentrancia.
  - Utiliza la interfaz `ISimpleDeFi` para interactuar con el contrato de protocolo DeFi.
- **Variables de estado:**
  - `address[] public participants;`
  - `mapping(address =>uint256) public investments;`
  - `uint256 public deadline;`
  - `enum State { Created, Pending, Started, Failed, Completed };`
  - `State public currentState;`
  - `address public winner;`
  - `ISimpleDeFi private immutable defiProtocol;`
- **Eventos:**
  - `event Joined(address indexed player, uint256 amount);`
  - `event InvestmentStarted(uint256 deadline);`

- `event LotteryWinnerSelected(address indexed winner, uint256 prize, uint256 totalBalance);`

- **Flujo de funciones:**

1. `constructor(address[] _participants, uint256 _duration, address _defiProtocol)` Inicializa estado, lista de participantes, fecha límite y dirección de protocolo DeFi.
2. `join() external payable` Permite a cada participante invertir ETH antes de que comience la lotería. Cambia el estado a `Pending` tras la primera inversión y emite `Joined`.
3. `startInvestment() external` Invierte el saldo total en el protocolo DeFi, fija el `deadline = block.timestamp + duration` y cambia el estado a `Started`, emitiendo `InvestmentStarted`.
4. `endInvestment() external nonReentrant` Solo tras superar el `deadline`.
   - Llama a `defiProtocol.withdraw()` para recuperar principal + ganancias.
   - Selecciona aleatoriamente un ganador entre `participants`.
   - Transfiere la ganancia al ganador y devuelve el principal a los demás.
   - Cambia el estado a `Completed` y emite `LotteryWinnerSelected`.
5. `fallback / receive` Rechaza envíos directos de ETH fuera de `join()` para evitar fondos perdidos.

4.2.3. Contrato para el protocolo DeFi (`SimpleDeFi`)

Implementa un esquema de interés fijo para simular el rendimiento de una inversión en un protocolo DeFi. Se utiliza únicamente durante la fase de prueba.

- **Constantes:**
  - `ONE = 1e18` — factor para operaciones con punto fijo.
  - `INTEREST_RATE = 1.3 * ONE` — tasa de interés fija del 30 %.
  - `ONE_HOUR = 600` — periodo simulado de 10 minutos por hora de prueba.
- **Estructuras y mapeos:**
  - `struct DepositInfo { uint256 amount; uint256 timestamp; }`
  - `mapping(address =>DepositInfo) public deposits;`
  - `mapping(address =>bool) public hasDeposited;`
- **Eventos:**
  - `event Deposited(address indexed user, uint256 amount);`
  - `event Withdrawn(address indexed user, uint256 amount);`
- **Funciones:**

- `deposit() external payable` Registra el monto y la marca de tiempo, emite `Deposited` y evita duplicados en `investorAddresses`.
- `getCurrentBalance(address user) public view returns (uint256)` Calcula el saldo acumulado:
- `withdraw() external` Recupera el saldo calculado, elimina el registro para prevenir reentrancia y emite `Withdrawn`.

4.2.4. Flujo de interacción y movimiento de activos

A continuación se describe el recorrido que sigue un ciclo completo de lotería, desde la creación de la instancia hasta la distribución de fondos. Se indican los puntos en los que entran los ETH y cómo se almacenan y transfieren entre los tres contratos.

1. **Creación de la lotería (Factory)**
   - Un usuario invoca `DeFiLotteryETHFactory.createLottery( ... )`.
   - El *Factory* despliega un nuevo contrato `DeFiLotteryETH` (contrato Base) y guarda su dirección en el array `lotteries`.
   - En este punto aún no hay ETH depositados: la instancia existe pero está vacía.
2. **Ingreso de fondos (Contrato Base)**
   - Cada participante llama `DeFiLotteryETH.join()` enviando ETH en `msg.value`.
   - El contrato Base recibe y registra cada aportación en `investments[msg.sender]`, y emite el evento `Joined`.
   - Los ETH quedan almacenados en el balance del propio contrato Base.
3. **Despliegue en protocolo DeFi (Contrato Base - SimpleDeFi)**
   - Cuando todos los participantes han entrado (o tras un trigger manual), se invoca `startInvestment()`.
   - El contrato Base transfiere el total de ETH acumulados a `SimpleDeFi.depositvalue: total`.
   - `SimpleDeFi` guarda internamente cada depósito en `deposits[user].amount` y marca la `timestamp`.
   - A partir de ahora, los fondos y su cálculo de intereses quedan bajo la responsabilidad de `SimpleDeFi`.
4. **Finalización y reparto (Contrato Base - SimpleDeFi)**
   - Tras transcurrir el periodo definido (`deadline`), se llama a `endInvestment()`.
   - El contrato Base ejecuta `SimpleDeFi.withdraw()` para recuperar el principal más el rendimiento.

- `SimpleDeFi` devuelve al Base el monto acumulado calculado según la fórmula de interés fijo.
- El Base selecciona aleatoriamente un ganador de entre `participants`.
- Se distribuyen los ETH:
  - *Ganador*: recibe su inversión inicial más la parte proporcional de la ganancia.
  - *Restantes*: cada uno recupera únicamente su inversión inicial.
- El contrato Base emite `LotteryWinnerSelected` y actualiza su estado a `Completed`.

5. **Estado final y limpieza**

- Tras el reparto, el contrato Base queda con saldo cero (todos los ETH han sido transferidos).
- La instancia Base sigue accesible para consulta de eventos y estados, pero ya no maneja más fondos.
- La Factory conserva el historial de direcciones en `lotteries`, sin almacenar ETH.

Este flujo modular garantiza que cada lote de inversión se maneje en su propio contrato, que los fondos se aíslen correctamente y que la lógica de cálculo de rendimiento resida únicamente en el contrato `SimpleDeFi`. De este modo se facilita la auditoría, la actualización de componentes y la trazabilidad de cada evento de inversión.

Capítulo 5

Pruebas y resultados

5.1. Despliegue del sistema

Para la fase de despliegue y primeras pruebas se utilizó Remix IDE, un entorno de desarrollo basado en web que ofrece un compilador integrado, un panel de despliegue e interacción con contratos y un explorador visual de transacciones. Remix facilitó tanto la escritura y depuración de los contratos en Solidity como la gestión manual de fondos en la red de prueba.

1. **Compilación y configuración de ambiente**
   - Se seleccionó la versión de Solidity 0.8.30 en el compilador de Remix y se habilitó el optimizador de bytecode para reducir el consumo de gas.
   - Se configuró el proveedor de Ethereum en la pestaña *Deploy & Run Transactions* para la red Sepolia (testnet) y se conectó la cuenta mediante Metamask, disponiendo de tokens de prueba mediante faucet.
2. **Despliegue inicial de contratos auxiliares**
   - Se compiló y desplegó primero el contrato `SimpleDeFi` para que actúe como *almacén de activos* y motor de cálculo de rendimiento.
   - A continuación se desplegó el contrato `DeFiLotteryETHFactory`, encargado de crear instancias de loterías.
   - Ambas transacciones aparecen listadas en el panel de Remix, mostrando las direcciones asignadas y el balance inicial de cada contrato.
3. **Financiación del protocolo DeFi de prueba**
   - Antes de crear loterías, fue necesario enviar una cantidad de ETH de prueba al contrato `SimpleDeFi` mediante su función `deposit()`, de modo que dispusiera de un saldo suficiente para simular reinversiones y retiros durante las pruebas.
   - La cantidad exacta no es crítica, siempre que supere el total de inversiones previstas en la lotería (por ejemplo, 5 ETH en Sepolia).

4. **Creación de una instancia de lotería**

   Con el contrato `DeFiLotteryETHFactory` ya desplegado, se llamó a su método `createLottery`, el cual tiene la siguiente firma:

   ```
   createLottery(address[] participants, uint256 duration, address defiProtocol)
   ```

   Los parámetros requeridos son los siguientes:

   - `participants`: Arreglo de direcciones de los usuarios que participarán.
   - `duration`: Duración de la fase de inversión (en segundos).
   - `defiProtocol`: Dirección del contrato `SimpleDeFi` previamente desplegado.

   Una vez ejecutada la función, Remix mostró el evento `LotteryCreated(address)` en la consola, indicando la dirección del nuevo contrato `DeFiLotteryETH`.

5. **Interacción con la lotería**

   - Cada participante invoca `join()` enviando su aporte en `msg.value`. Remix registra cada transacción y muestra el balance acumulado en el contrato Base.
   - Una vez registrados todos los participantes, se llama a `startInvestment()`, que transfiere el balance al contrato `SimpleDeFi` y establece el `deadline`.
   - Tras el tiempo especificado, se ejecuta `endInvestment()`, recuperando el monto con rendimiento, seleccionando al ganador y distribuyendo los fondos según la lógica definida.

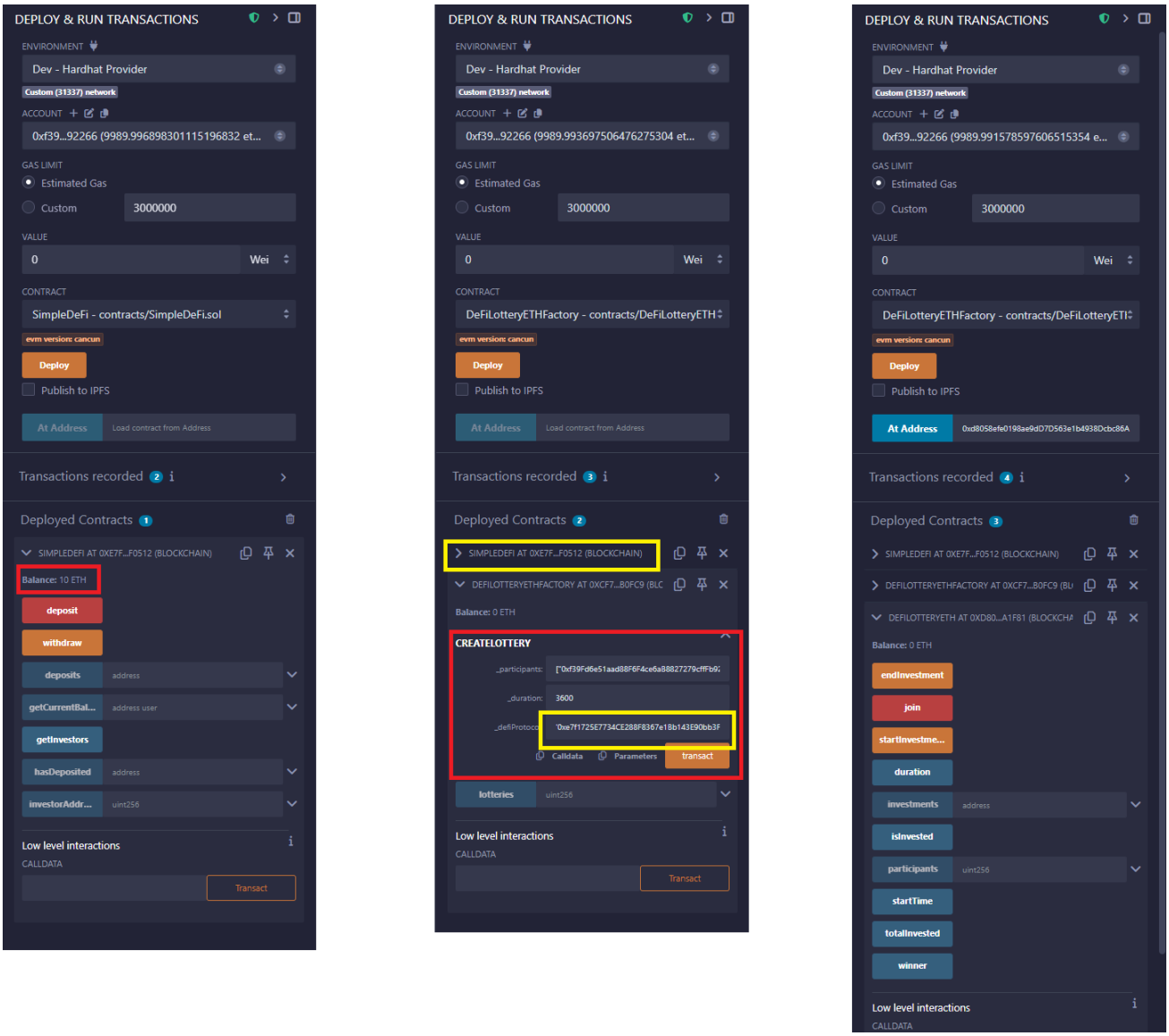

Figura 5.1: Visión general del sistema desplegado en Remix para las pruebas

5.2. Plan y Ejecución de Pruebas

El objetivo de este plan es validar la corrección funcional, la robustez y el comportamiento en términos de los tres contratos inteligentes (`DeFiLotteryETHFactory`, `DeFiLotteryETH`, `SimpleDeFi`) en un entorno de testnet Sepolia. Se busca garantizar que:

- Cada función exponga el comportamiento esperado y maneje adecuadamente estados límite

y condiciones de error.

- El flujo completo de creación, inversión, cálculo de rendimiento, selección de ganador y reembolso se ejecute sin fallos.
- El consumo de gas permanezca dentro de límites razonables para cada operación clave.

5.2.1. Tipos de pruebas

Para este proyecto se han definido dos categorías principales de pruebas:

Pruebas funcionales: Verifican de forma aislada cada unidad de funcionalidad del contrato (función o método) para asegurar que, dado un conjunto de entradas válidas o inválidas, el comportamiento (salidas, eventos, cambios de estado) sea el esperado.

Pruebas de integración: Validan la interacción conjunta entre varios contratos (Factory, Base y Protocolo DeFi), comprobando el flujo end-to-end desde el despliegue hasta la distribución de fondos, y detectando posibles problemas en la comunicación o sincronización de estados entre ellos.

5.2.2. Alcance y entorno de prueba

- **Alcance:**
  - Pruebas funcionales (unitarias) de cada contrato por separado.
  - Pruebas de integración de todo el flujo entre Factory, Base y Protocolo DeFi.
- **Entorno:**
  - *Remix IDE* y *Hardhat* sobre Sepolia.

5.2.3. Estrategia de pruebas

1. **Pruebas funcionales**
   - *`SimpleDeFi`* : Para el contrato `SimpleDeFi` se validarán las operaciones básicas de depósito, consulta de saldo y retiro. Se comprobará que el contrato recibe correctamente ETH mediante `deposit()`, que `getCurrentBalance()` devuelve el valor esperado antes y después del periodo de interés, y que `withdraw()` transfiere el monto adecuado. Estas pruebas confirman que el modelo simplificado de interés fijo funciona de extremo a extremo y se comporta según lo diseñado en escenarios normales y límite.
   - *`DeFiLotteryETH`*: En el contrato de lotería `DeFiLotteryETH` se evaluará el flujo completo de la inversión: incorporación de participantes autorizados con `join()`, despliegue de fondos al protocolo DeFi con `startInvestment()` y reparto de ganancias con

`endInvestment()`. Además, se verificarán los mecanismos de control de acceso (rechazo de direcciones no registradas), las protecciones contra llamadas fuera de estado (por ejemplo, finalizar antes del deadline) y el manejo de transferencias directas de ETH. Estas pruebas aseguran la robustez de la lógica de lotería y la integridad de los fondos.

- *`DeFiLotteryETHFactory`*: Para el contrato fábrica `DeFiLotteryETHFactory` se centrará en la creación de instancias y el registro de direcciones. Se comprobará que cada invocación a `createLottery()` genera exactamente un evento `LotteryCreated` con la dirección de la nueva lotería. Estas pruebas validan la capacidad de la fábrica para escalar y mantener un historial fiable de todas las loterías desplegadas.

2. **Pruebas de integración**

- Flujo completo:

  a) Despliegue de `SimpleDeFi` y financiación.

  b) Despliegue de `DeFiLotteryETHFactory`.

  c) Llamada a `createLottery(...)` y verificación del evento `LotteryCreated`.

  d) Inversiones con `join()` de varios participantes.

  e) Invocación de `startInvestment()` y comprobación de depósito en `SimpleDeFi`.

  f) Simulación de avance de tiempo; llamada a `endInvestment()`.

  g) Verificación de balances finales, evento `LotteryWinnerSelected` y estado `Completed`.

Además de los casos funcionales e integración ya definidos, aplicaremos dos técnicas adicionales para asegurar la robustez ante valores extremos y entradas inválidas:

- **Análisis de valores límite (Boundary Value Analysis):** Probaremos los campos numéricos clave con valores en el límite (0, 1, duración mínima) y justo fuera del límite (–1, duración negativa) para verificar que el contrato rechaza o maneja correctamente estas condiciones.

- **Partición de equivalencia (Equivalence Partitioning):** Dividiremos el rango de entradas válidas e inválidas en particiones (por ejemplo, depósitos pequeños vs. grandes, duraciones cortas vs. largas) y seleccionaremos casos representativos de cada partición.

5.2.4. Plan de pruebas detallado

5.2.4.1. Casos de prueba funcionales para `SimpleDeFi.sol`

TC-ID	TC-SD-F01
Nombre	Ingreso de balance inicial: 10 ETH al contrato
Funcionalidad	`deposit()`
Precondición	Contrato desplegado en Hardhat, saldo inicial del contrato = 0 ETH.
Entradas	Llamada a `deposit()` con `msg.value = 10 ETH` desde cuenta A.
Pasos	1. Leer saldo del contrato antes de la llamada. 2. Invocar `deposit()` (10 ETH) desde cuenta A. 3. Leer saldo del contrato después de la llamada.
Resultado esperado	▪ Emisión de `Deposited(A, 10 ETH)`. ▪ Saldo del contrato incrementado en 10 ETH.
Notas	Establece el pool inicial de fondos.

Cuadro 5.1: Ingreso funcional de balance inicial

TC-ID	TC-SD-F02
Nombre	Depósito/inversión de 1 ETH por participante
Funcionalidad	`deposit()`
Precondición	TC-SD-F01 completado; saldo del contrato = 10 ETH.
Entradas	Llamada a `deposit()` con `msg.value = 1 ETH` desde cuenta B.
Pasos	1. Leer saldo del contrato antes de la llamada. 2. Invocar `deposit()` (1 ETH) desde cuenta B. 3. Leer saldo del contrato después de la llamada.
Resultado esperado	▪ Emisión de `Deposited(B, 1 ETH)`. ▪ Saldo del contrato incrementado en 1 ETH (de 10 a 11 ETH).
Notas	Simula una inversión adicional tras el pool inicial.

Cuadro 5.2: Depósito funcional de inversión

TC-ID	TC-SD-F03
Nombre	Balance con ganancia tras periodo (30 %)
Funcionalidad	`getCurrentBalance(address)`
Precondición	TC-SD-F02 completado; simular avance de tiempo `ONE_HOUR`.
Entradas	Dirección de cuenta B.
Pasos	1. Esperar una hora. 2. Llamar a `getCurrentBalance(B)`.
Resultado esperado	Retorno de 1.3 ETH (1 ETH + 30 %).
Notas	Validar únicamente el valor retornado.

Cuadro 5.3: Cálculo de ganancia aplicada

TC-ID	TC-SD-F04
Nombre	Consulta de balance sin depósito
Funcionalidad	`getCurrentBalance(address)`
Precondición	Contrato recién desplegado, sin depósitos.
Entradas	Dirección sin historial de depósito.
Pasos	1. Llamar a `getCurrentBalance(user)`.
Resultado esperado	Retorno de 0 ETH.
Notas	Asegura manejo correcto de cuentas vacías.

Cuadro 5.4: Balance cero sin depósito

TC-ID	TC-SD-F05
Nombre	`withdraw()` transfiere capital+ganancia
Funcionalidad	`withdraw()`
Precondición	TC-SD-F02 y TC-SD-F03 completados.
Entradas	Llamada a `withdraw()` desde cuenta B.
Pasos	1. Capturar saldo de la cuenta B antes de la llamada. 2. Invocar `withdraw()`. 3. Capturar saldo de la cuenta B después de la llamada.
Resultado esperado	▪ Cuenta B recibe exactamente 1.3 ETH. ▪ Saldo del contrato disminuye en 1.3 ETH. ▪ Emisión de `Withdrawn(B, 1.3 ETH)`.
Notas	Verificar diferencia de saldo en cuenta y contrato.

Cuadro 5.5: Retiro exitoso con ganancia

TC-ID	TC-SD-F06
Nombre	`withdraw()` sin depósito revierte
Funcionalidad	`withdraw()`
Precondición	Contrato desplegado, sin depósitos.
Entradas	—
Pasos	1. Invocar `withdraw()` desde cuenta sin historial.
Resultado esperado	Transacción revierte con mensaje de error apropiado (p. ej. "No deposit").
Notas	Confirma manejo de llamadas inválidas.

Cuadro 5.6: Retiro inválido sin depósito

5.2.4.2. Casos de prueba funcionales para `DeFiLotteryETH.sol`

TC-ID	TC-DLE-F01
Nombre	`join()` válido por participante autorizado
Funcionalidad	`join()`
Precondición	▪ Factory ha creado una instancia de `DeFiLotteryETH` con participantes [A,B]. ▪ Estado inicial: `Created`.
Entradas	Llamada a `join()` con `msg.value = 1 ETH` desde A.
Pasos	1. Leer estado y balance antes. 2. A invoca `join()` enviando 1 ETH. 3. Leer estado, emitir evento y balance tras la llamada.
Resultado esperado	▪ Evento `Joined(A,1 ETH)`. ▪ Balance del contrato aumenta en 1 ETH.
Notas	Valida invitación solo a direcciones predefinidas.

Cuadro 5.7: Ingreso de inversión válida

TC-ID	TC-DLE-F02
Nombre	`join()` desde dirección no participante revierte
Funcionalidad	`join()`
Precondición	Misma creación que TC-DLE-F01; estado = `Created`.
Entradas	Llamada a `join()` con `msg.value = 1 ETH` desde C (no listado).
Pasos	1. C invoca `join()` con 1 ETH.
Resultado esperado	La transacción revierte (p. ej. "Not a participant") y balance permanece.
Notas	Asegura control de acceso.

Cuadro 5.8: Ingreso inválido por participante no autorizado

TC-ID	TC-DLE-F03
Nombre	`startInvestment()` desplaza fondos al protocolo DeFi
Funcionalidad	`startInvestment()`
Precondición	▪ Dos participantes A y B han hecho `join(1 ETH)` cada uno. ▪ Estado = `Pending`. ▪ `SimpleDeFi` desplegado en P.
Entradas	Llamada a `startInvestment()` por el creador.
Pasos	1. Leer balance de Base y SimpleDeFi antes. 2. Invocar `startInvestment()`. 3. Leer balances después y capturar evento.
Resultado esperado	▪ Evento `InvestmentStarted(deadline)` emitido. ▪ Balance de Base pasa a 0 ETH. ▪ Balance de SimpleDeFi incrementa en 2 ETH. ▪ Estado cambia a `Started`.
Notas	Verifica transferencia correcta a `SimpleDeFi`.

Cuadro 5.9: Inicio de inversión en protocolo DeFi

TC-ID	TC-DLE-F04
Nombre	`endInvestment()` antes de deadline revierte
Funcionalidad	`endInvestment()`
Precondición	▪ Estado = `Started`; antes de alcanzar `deadline`.
Entradas	Invocación de `endInvestment()`.
Pasos	1. Llamar a `endInvestment()` sin adelantar tiempo.
Resultado esperado	Reversión (p. ej. “Too early”) y sin cambios de balances.
Notas	Asegura respeto al período de inversión.

Cuadro 5.10: Finalización temprana no permitida

TC-ID	TC-DLE-F05
Nombre	`endInvestment()` tras deadline reparte fondos y elige ganador
Funcionalidad	`endInvestment()`
Precondición	▪ TC-DLE-F03 completado; Estado = `Started`. ▪ Simular avance de tiempo duración.
Entradas	Llamada a `endInvestment()`.
Pasos	1. Adelantar tiempo con `evm_increaseTime`. 2. Invocar `endInvestment()`. 3. Capturar evento `LotteryWinnerSelected(winner, prize, total)`. 4. Leer balances en cuentas A y B y del contrato.
Resultado esperado	▪ El evento muestra un ganador válido (A o B), ‘prize ¿1 ETH‘. ▪ El ganador recibe principal + ganancia; el otro recupera sólo 1 ETH. ▪ Balance del contrato queda en 0 ETH.
Notas	Verifica reparto equitativo y limpieza de estado.

Cuadro 5.11: Finalización y reparto de fondos

5.2.4.3. Casos de prueba funcionales para `DeFiLotteryETHFactory.sol`

TC-ID	TC-FAC-F01
Nombre	Creación de nueva lotería emite evento y registra dirección
Funcionalidad	`createLottery(address[] , uint256, address)`
Precondición	Contrato `DeFiLotteryETHFactory` desplegado en red local (Hardhat/Remix).
Entradas	▪ Array de participantes: [A, B, C] (direcciones válidas). ▪ Duración: 3600 segundos. ▪ Dirección de protocolo DeFi: P (dirección válida de `SimpleDeFi` desplegado).
Pasos	1. Leer longitud inicial de `factory.lotteries()` (debe ser 0). 2. Invocar `factory.createLottery([A,B,C], 3600, P)`. 3. Capturar el evento `LotteryCreated(lotteryAddr)` en la transacción.
Resultado esperado	▪ Se emite exactamente un evento `LotteryCreated` con la dirección de la nueva lotería.
Notas	Se asume que tanto las Address de los participantes como la del protocolo DeFi son correctas.

Cuadro 5.12: Validación de creación de instancia de lotería

5.2.4.4. Pruebas de valores límite para `SimpleDeFi.sol`

TC-ID	TC-SD-F07
Nombre	Depósito de 0 ETH revierte
Funcionalidad	`deposit()`
Precondición	Contrato desplegado, saldo inicial = 0
Entradas	`msg.value = 0 ETH`
Pasos	1. Invocar `deposit()` con 0 ETH.
Resultado esperado	Reversión con mensaje de error (“monto invalido”).
Notas	Verifica validación de mínimos en depósitos.

Cuadro 5.13: Depósito de valor límite 0 en `SimpleDeFi`

TC-ID	TC-SD-F08
Nombre	Depósito con valor negativo (inválido)
Funcionalidad	`deposit()`
Precondición	Contrato desplegado
Entradas	Simulación de `msg.value = {1` (test negativo)
Pasos	1. Intentar invocar `deposit()` con valor negativo.
Resultado esperado	Reversión o rechazo de la transacción; sin cambio de saldo.
Notas	Comprueba manejo de entradas no numéricas o mal formadas.

Cuadro 5.14: Depósito de valor negativo en `SimpleDeFi`

5.2.4.5. Pruebas de valores límite para `DeFiLotteryETH.sol`

TC-ID	TC-DLE-F07
Nombre	Creación con duración = 0 revierte
Funcionalidad	Constructor / `createLottery()`
Precondición	Factory desplegada
Entradas	`_duration = 0`
Pasos	1. Llamar a `createLottery([A], 0, P)`.
Resultado esperado	Reversión con mensaje “duracion ¿0”.
Notas	Verifica validación de duración mínima.

Cuadro 5.15: Creación de lotería con duración nula

TC-ID	TC-DLE-F08
Nombre	StartInvestment sin fondos revierte
Funcionalidad	`startInvestment()`
Precondición	Lotería creada pero sin `join()`
Entradas	—
Pasos	1. Invocar `startInvestment()`.
Resultado esperado	Reversión con mensaje “sin fondos”.
Notas	Comprueba manejo de inicio sin inversiones.

Cuadro 5.16: Inicio de inversión con zero deposits

5.3. Resultado de las Pruebas

5.3.1. SimpleDeFi

TC-ID	Fecha	Entorno	Resultado
TC-SD-F01	2025-06-18	Hardhat Network	Pasó
TC-SD-F02	2025-06-18	Hardhat Network	Pasó
TC-SD-F03	2025-06-18	Hardhat Network	Pasó
TC-SD-F04	2025-06-18	Hardhat Network	Pasó
TC-SD-F05	2025-06-18	Hardhat Network	Pasó
TC-SD-F06	2025-06-18	Hardhat Network	Pasó
TC-SD-F07	2025-06-18	Remix Sepolia	Pasó (revirtió)
TC-SD-F08	2025-06-18	Remix Sepolia	Pasó (revirtió)

Cuadro 5.17: Resultados de las pruebas funcionales de `SimpleDeFi.sol`

5.3.2. DeFiLotteryETHFactory

TC-ID	Fecha	Entorno	Resultado
TC-FAC-F01	2025-06-18	Hardhat Network	Pasó
TC-FAC-F02	2025-06-18	Hardhat Network	Pasó

Cuadro 5.18: Resultados de las pruebas funcionales de `DeFiLotteryETHFactory.sol`

5.3.3. DeFiLotteryETH

TC-ID	Fecha	Entorno	Resultado
TC-DLE-F01	2025-06-18	Hardhat Network	Pasó
TC-DLE-F02	2025-06-18	Hardhat Network	Pasó
TC-DLE-F03	2025-06-18	Hardhat Network	Pasó
TC-DLE-F04	2025-06-18	Remix Sepolia	Pasó (revirtió)
TC-DLE-F05	2025-06-18	Hardhat Network	Pasó
TC-DLE-F06	2025-06-18	Hardhat Network	Pasó
TC-DLE-F07	2025-06-18	Remix Sepolia	Pasó (revirtió)
TC-DLE-F08	2025-06-18	Remix Sepolia	Pasó (revirtió)

Cuadro 5.19: Resultados de las pruebas funcionales de `DeFiLotteryETH.sol`

Conclusión: Todas las pruebas funcionales ejecutadas sobre los contratos han sido exitosas, cumpliendo con los comportamientos y restricciones definidos. Esto confirma que el sistema es robusto, modular y funciona correctamente tanto en la gestión de depósitos e intereses como en la creación de instancias de lotería y en el flujo completo de inversión, cálculo de rendimientos y reparto de fondos. A partir de ahora, el prototipo está validado para un entorno de prueba y se considera listo para posibles ampliaciones o despliegues en testnet pública.

Capítulo 6

Caso de Uso funcional

6.1. Diseño del sistema

El diseño de un sistema es un aspecto clave en el desarrollo de cualquier aplicación, ya que define la estructura, la organización y la forma en que los diferentes componentes interactúan entre sí para cumplir con las necesidades del sistema. Una arquitectura bien diseñada no solo permite mejorar la escalabilidad y el rendimiento, sino que también facilita la mantenibilidad y la evolución del sistema a lo largo del tiempo. Según Fowler (2017), la elección de patrones arquitectónicos adecuados es esencial para garantizar que el sistema pueda adaptarse a cambios, gestionar la complejidad y soportar un alto volumen de transacciones sin comprometer su estabilidad [76]. En este sentido, la selección de una arquitectura apropiada responde a la necesidad de equilibrar eficiencia, modularidad y flexibilidad, asegurando que cada componente cumpla una función clara dentro del ecosistema del sistema.

Para la arquitectura del sistema adoptamos un diseño modular con componentes desacoplados, lo que nos garantiza escalabilidad y facilidad de mantenimiento a futuro. Aunque el alcance actual consiste en un prototipo piloto con funcionalidades básicas, esta arquitectura permite:

- **Escalabilidad horizontal:** cada componente (front-end, back-end, contratos) puede ampliarse o replicarse independientemente.
- **Desarrollo paralelo:** equipos pueden trabajar simultáneamente en la interfaz, la lógica de negocio y la capa de datos sin conflictos.
- **Evolución del sistema:** nuevos módulos (por ejemplo, aplicaciones móviles, APIs externas) pueden integrarse sin rehacer la estructura central del back-end.

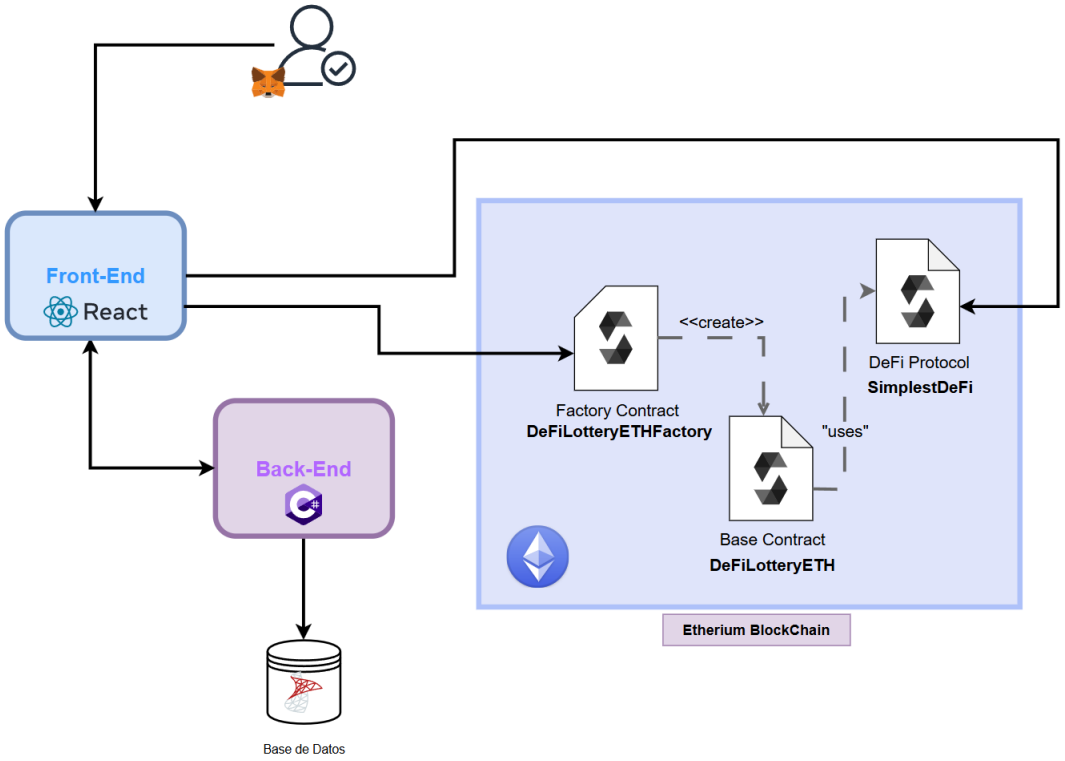

Figura 6.1: Visión general de la arquitectura del sistema

6.1.1. Interfaz Web

Centralizamos la interacción del usuario en una aplicación web por su accesibilidad universal: cualquier dispositivo con navegador puede acceder sin instalaciones adicionales, eliminando barreras de entrada [76]. Esta elección facilita también:

- **Compatibilidad multi-plataforma:** Windows, macOS, Linux, iOS y Android funcionan directamente con la misma aplicación.
- **Despliegue centralizado:** las actualizaciones se publican en un solo servidor o CDN, y todos los usuarios obtienen automáticamente la versión más reciente [76].
- **Conexión Web3:** gracias a MetaMask (u otras extensiones), el navegador actúa como puente para firmar y enviar transacciones, simplificando la experiencia descentralizada [77].

6.1.2. Back-end

El componente back-end se encarga exclusivamente de la gestión de usuarios: proporciona los servicios de registro, autenticación y almacenamiento seguro de credenciales. Mediante un API REST protegido, el sistema permite crear nuevas cuentas, validar credenciales en cada inicio de sesión y mantener un repositorio de perfiles de usuario que se vinculan posteriormente con las interacciones en la capa de contratos. Este enfoque centraliza la lógica de seguridad y acceso, asegurando que únicamente usuarios autorizados puedan invocar funcionalidades del front-end, sin almacenar ni manipular información sensible de las transacciones en blockchain.

6.1.3. Uso de Wallet externa

Con el fin de reforzar la seguridad del sistema y simplificar el desarrollo, se optó por delegar la custodia y firma de transacciones a una wallet externa gestionada por el usuario, en lugar de almacenar claves privadas en el servidor (vía HSM o KMS). Esta decisión ofrece múltiples beneficios:

- **Seguridad por descentralización:** al mantener las claves privadas en el dispositivo del usuario, se evita su exposición en servidores centrales, lo que mitiga riesgos críticos como compromisos masivos o ataques dirigidos [78].
- **Experiencia del usuario:** herramientas como MetaMask exponen el objeto `window.ethereum` en el navegador, permitiendo firmar transacciones directamente desde el front-end sin necesidad de desarrollar una infraestructura de gestión de claves.
- **Compatibilidad y flexibilidad:** el usuario puede cambiar entre redes como testnet o mainnet desde su wallet sin afectar la lógica del back-end, lo que mejora la escalabilidad y portabilidad del sistema.

Este enfoque no solo mejora la postura de seguridad general del sistema, sino que también promueve la interoperabilidad con herramientas estándar en el ecosistema Web3, reduciendo la complejidad técnica y facilitando una arquitectura modular y mantenible.

6.2. Tecnologías usada

A continuación se describen las decisiones tecnológicas adoptadas para cada componente del sistema, justificando la elección basada en rendimiento, escalabilidad y experiencia del equipo.

6.2.1. Front-end

Para la interfaz de usuario evaluamos tres opciones principales: Angular, Vue.js y React. Las tres son tecnologías maduras con amplias comunidades y facilitan la construcción de aplicaciones web dinámicas y escalables.

- **Angular** (Google): framework completo con soluciones integradas para enrutamiento, formularios y testing.
- **Vue.js** (Evan You): framework progresivo, fácil de integrar en proyectos existentes y con curva de aprendizaje suave.
- **React** (Meta): biblioteca basada en componentes y Virtual DOM, que permite crear interfaces reactivas con alto rendimiento .

Elegimos **React** porque:

- *Curva de aprendizaje y experiencia:* ya contábamos con conocimiento previo y aceleró el ritmo de desarrollo.
- *Modularidad:* su arquitectura de componentes facilita la reutilización y la integración con otras librerías.
- *Rendimiento:* el uso de Virtual DOM reduce el costo de renderizado en cambios dinámicos, esencial para futuras gráficas en tiempo real.
- *Ecosistema:* soporte en largo plazo por Meta y compatibilidad con herramientas como Next.js o Vite.

Para optimizar la construcción y el bundling, utilizamos **Vite**, una herramienta de desarrollo que ofrece tiempos de arranque veloces y un entorno de pruebas eficiente, dejando las decisiones de enrutamiento y estado en manos del desarrollador.

6.2.2. Back-end

Consideramos varias alternativas populares con soporte para WebSockets y fácil conexión a bases de datos:

- **.NET (C#)**
- **Node.js (JavaScript/TypeScript)**
- **FastAPI (Python)**
- **Golang (Go)**

Descartamos Go por falta de experiencia interna y optamos por **.NET** debido a:

- Familiaridad del equipo y velocidad de desarrollo.
- Integración nativa con **Entity Framework**, que soporta múltiples motores de base de datos sin cambios de código significativos.
- Soporte integrado para WebSockets a través de **SignalR**.
- Madurez de ASP.NET para construir APIs REST escalables.

6.2.3. Gestor de Base de Datos

Para la capa de persistencia seleccionamos un modelo relacional por su madurez y consistencia histórica. Evaluamos:

- PostgreSQL
- Microsoft SQL Server Express
- MySQL

Gracias a Entity Framework, podemos cambiar entre estos motores sin modificar la capa de acceso a datos, lo que nos ofrece flexibilidad y portabilidad. Finalmente, consideramos la capa gratuita y facilidad de despliegue al elegir la instancia que mejor se adapte al entorno de producción.

6.2.4. Wallet

Decidimos integrar una única extensión de navegador: **MetaMask**. Las razones principales son:

- Amplia adopción en Ethereum y EVM-compatible chains, facilitando la conexión y firma de transacciones [77].
- Compatibilidad nativa con **Web3.js** y **ethers.js** mediante el proveedor `window.ethereum`.
- Almacenamiento local de claves privadas por parte del usuario, evitando confiar en un custodio central y reduciendo riesgos asociados a hackeos de exchanges [78].
- Soporte para hardware wallets (Ledger, Trezor) para mayor seguridad.

Con esta pila tecnológica garantizamos un equilibrio entre rapidez de desarrollo, rendimiento y capacidades de escalado para un eventual despliegue en un entorno productivo real.

6.3. Visualización del Sistema y casos de uso funcionales

A continuación en esta sección queremos presentarles algunos casos de uso funcionales abordados desde el sistema que construimos.

6.3.1. Autenticación

Para comenzar a utilizar el sistema, el usuario debe registrarse o iniciar sesión mediante la página de autenticación.

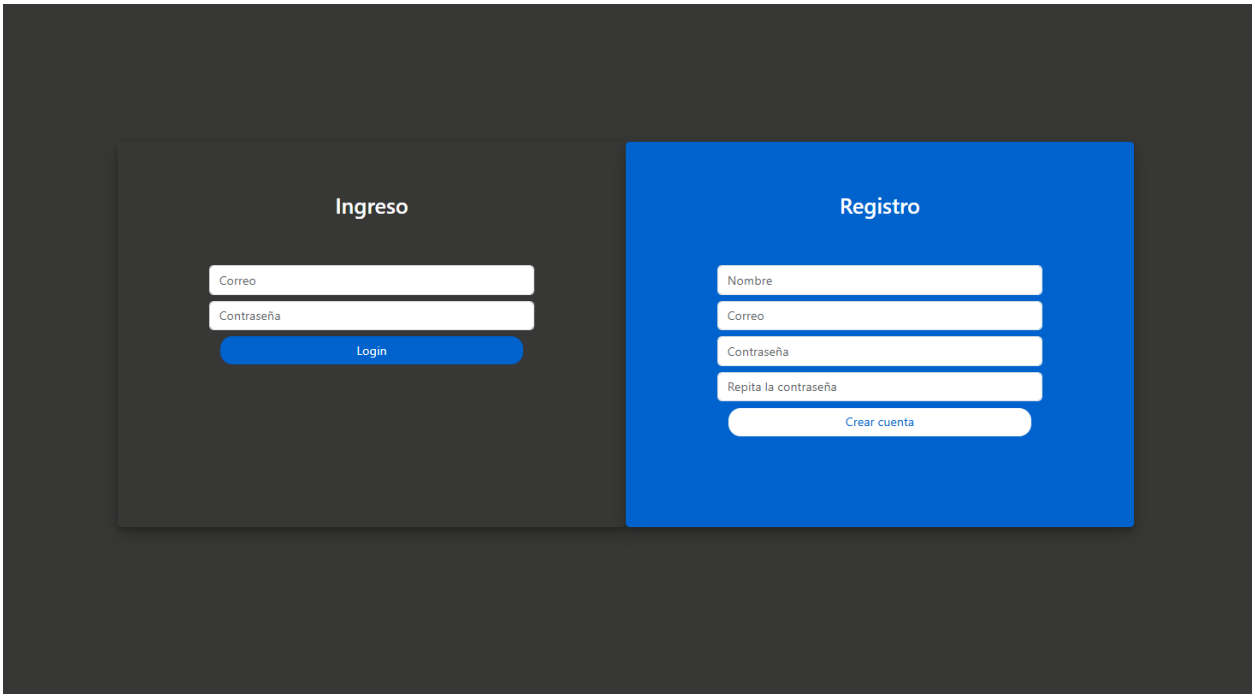

Figura 6.2: Página de Login y Registro del Sistema

En la pantalla de autenticación, el usuario tiene dos opciones:

- **Registro:** El usuario ingresa su correo y una contraseña. El front-end envía estos datos al back-end, que crea un nuevo registro en la base de datos. Para mayor seguridad, la contraseña se almacena de forma cifrada (hash) y nunca en texto plano.

- **Inicio de sesión:** El usuario introduce sus credenciales existentes. El front-end envía la solicitud al back-end, que valida el correo y la contraseña (comparando hashes). Si son correctos, genera un JSON Web Token (JWT) y devuelve al front-end la información de usuario necesaria junto con el token.

El JWT se almacena en el cliente (en `localStorage`) y se incluye en el encabezado de autorización de todas las peticiones posteriores. De este modo, el usuario permanece autenticado mientras dura la sesión y puede acceder a las funcio

6.3.2. Configuración de MetaMask

Para interactuar con los contratos inteligentes, el usuario debe conectar su wallet de MetaMask a la aplicación web. Dado que todas las transacciones se firman en el front-end, esta configuración garantiza que las solicitudes se envíen desde la cuenta correcta.

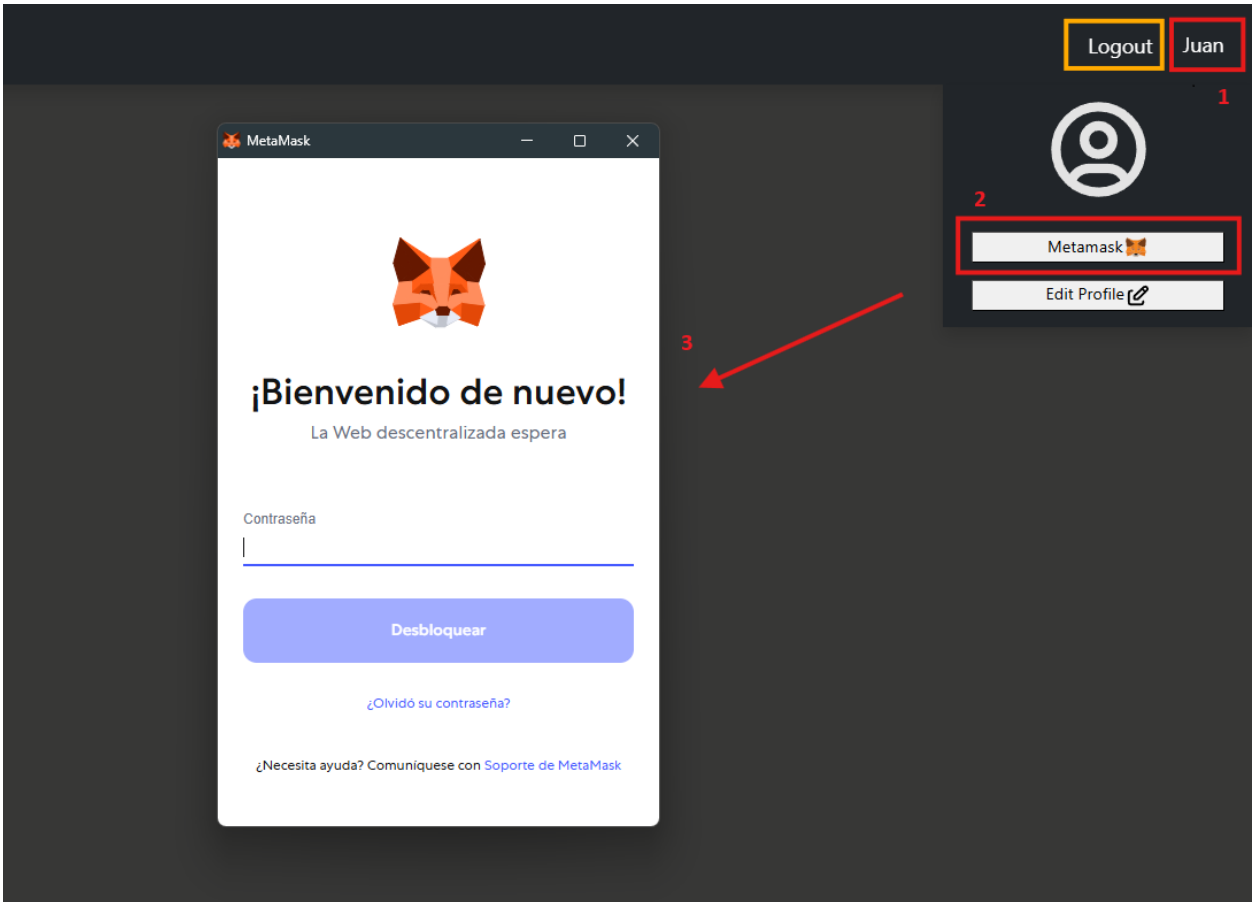

Figura 6.3: Selección de la opción de MetaMask en el perfil

En la esquina derecha del NavBar se encuentran dos botones:

- **Cerrar sesión:** finaliza la sesión actual.
- **Opciones de usuario:** despliega un menú con distintas acciones, entre las que se incluye *Configurar MetaMask*.

Al seleccionar “Configurar MetaMask”, la aplicación inicia el flujo de conexión con la extensión, solicitando al usuario que ingrese su contraseña y autorice el acceso de la dApp a su dirección.

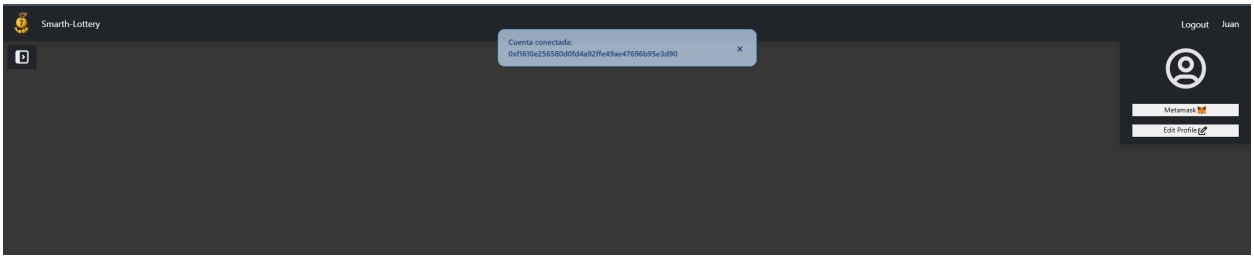

Figura 6.4: MetaMask conectada correctamente

Si la conexión es exitosa, la aplicación muestra una notificación confirmando que la wallet se ha configurado correctamente. A partir de ese momento, todas las operaciones de inversión y firma de transacciones se realizarán utilizando la cuenta seleccionada en MetaMask.

6.3.3. Información de las Loterias

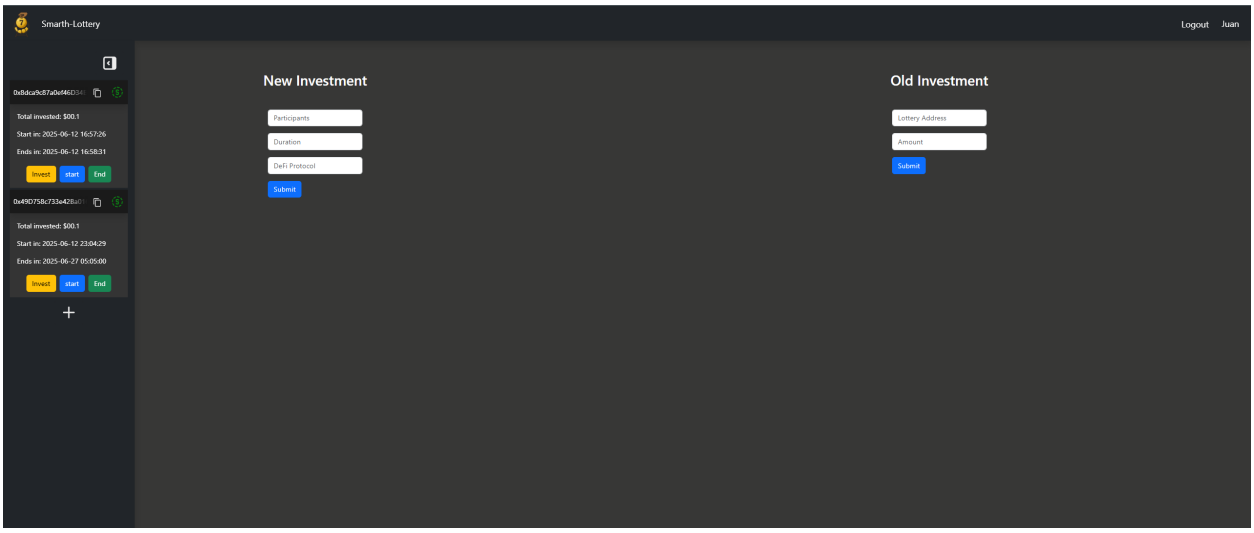

Figura 6.5: Diagrama de alto nivel de la arquitectura del sistema

A continuación se detalla la interfaz principal de la aplicación web y su interacción con los contratos inteligentes.

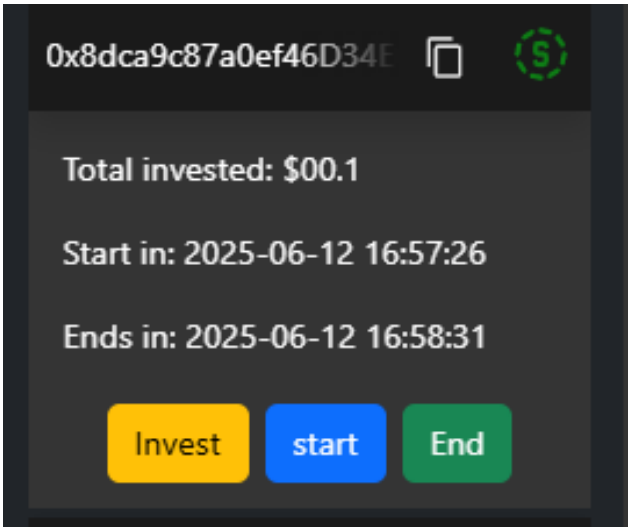

Figura 6.6: Tarjetas de loterías activas en el panel de usuario

En la vista de loterías, ubicada en el panel izquierdo, cada tarjeta muestra información clave:

- **Estado de la lotería:** Indica si la lotería está *Creada*, *Pendiente*, *Iniciada* o *Completada*.
- **Fechas de inicio y fin:** Muestra la marca de tiempo de cuando comenzó la inversión y la fecha estimada de finalización, calculada a partir de la duración especificada al crear la lotería.
- **Total invertido:** Suma de todos los aportes realizados por los participantes.
- **Acciones disponibles:**
  - *Invertir:* abre el diálogo para enviar fondos al contrato de la lotería.
  - *Iniciar:* despliega los fondos al protocolo DeFi y establece el período de inversión.
  - *Finalizar:* cierra la inversión, recupera los rendimientos y distribuye los premios.

6.3.4. Creación de una lotería

Para generar una nueva lotería, el usuario hace clic en el botón "+" situado al final del listado de loterías existentes. Esto redirige al formulario de creación o inversión.

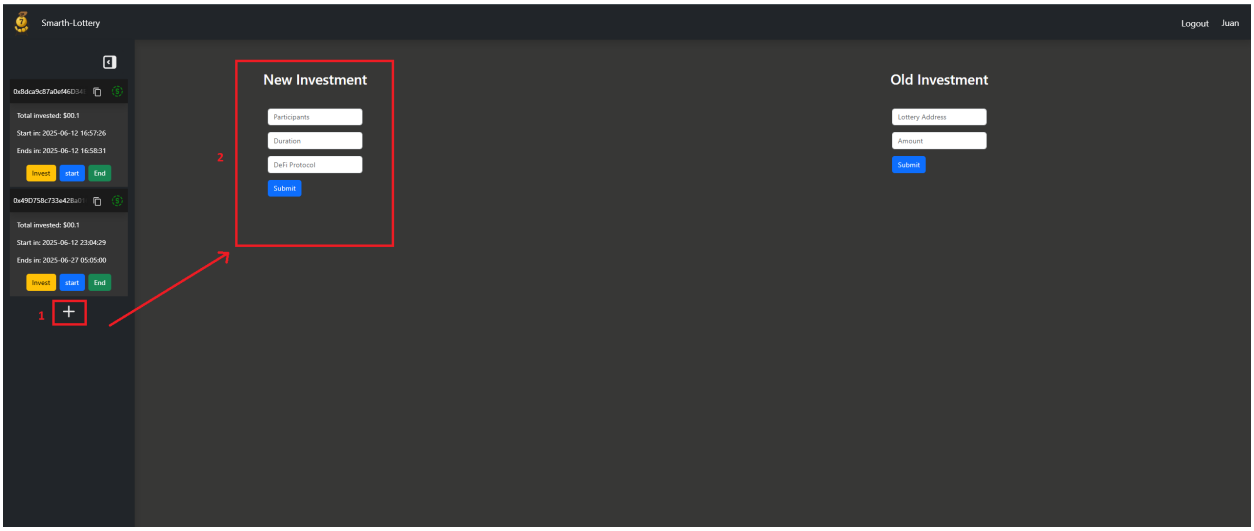

Figura 6.7: Acceso al formulario de creación de lotería

En el formulario de creación el usuario debe completar:

- **Participantes:** Direcciones de wallet separadas por comas. Permitimos desde un solo participante hasta varios, quienes serán elegibles para invertir y ganar.

- **Duración:** Tiempo en segundos durante el cual la lotería permanecerá activa en el protocolo DeFi. Se emplean segundos para facilitar pruebas rápidas en testnet.

- **Protocolo DeFi:** Dirección del contrato `SimpleDeFi` desplegado, que gestionará el rendimiento de los fondos.

Para motivos de esta prueba cambiamos las reglas de nuestro protocolo DeFi: en lugar de generar un 30 % de ganancia por hora, se genera un 10 % cada 10 minutos. Sin embargo, el funcionamiento del sistema y la interacción se mantienen iguales.

Al enviar el formulario, la aplicación invoca la función `createLottery(...)` del contrato factory, y automáticamente abre MetaMask para que el usuario confirme la transacción.

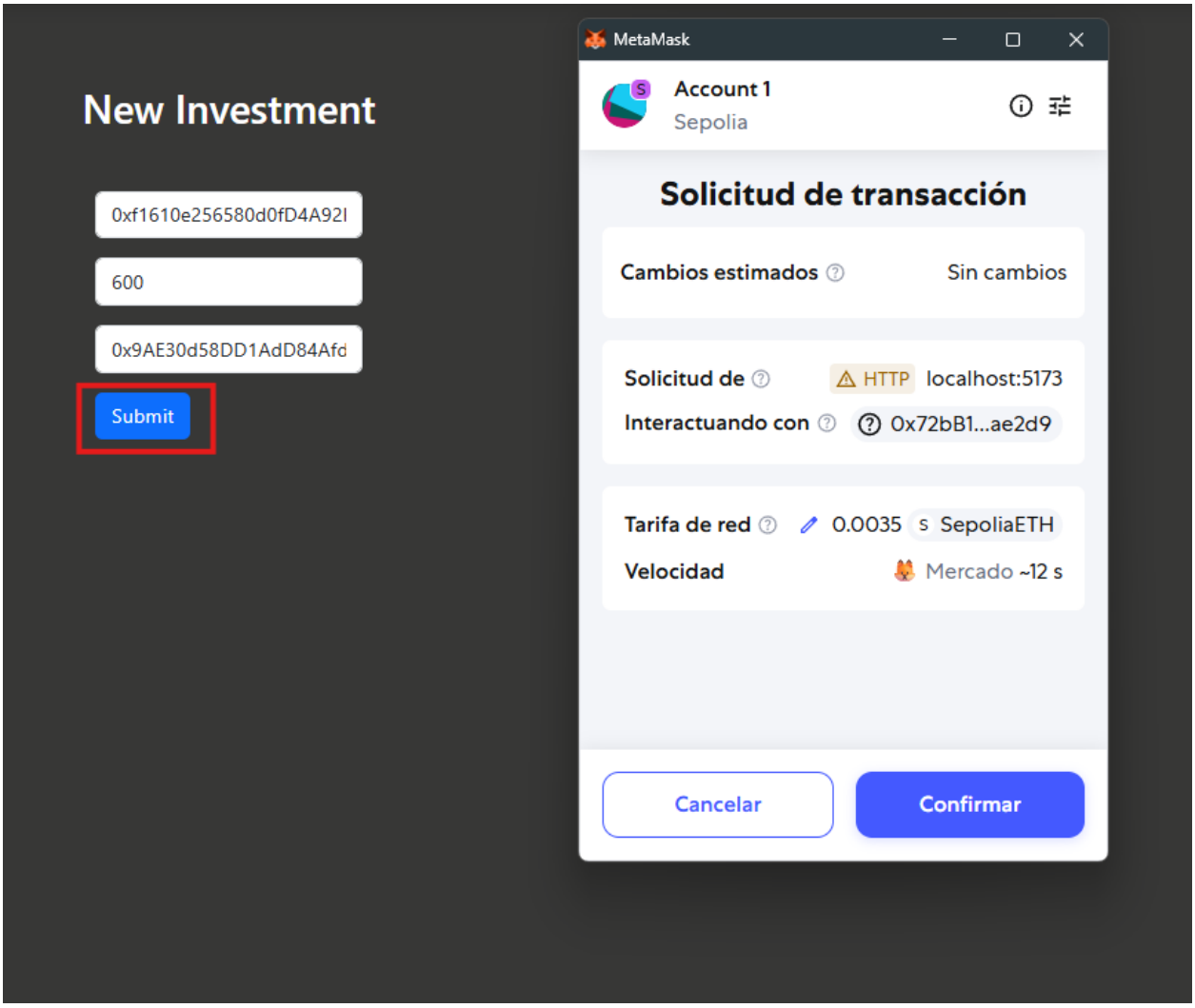

Figura 6.8: Confirmación de la transacción en MetaMask

Una vez confirmada, la lotería recién creada aparece en el panel izquierdo con:

- **Estado `Creada`:** Indica que aún no se ha iniciado la inversión.
- **Total invertido = 0:** No hay aportes hasta que comience la lotería.
- **Fechas vacías:** Las marcas de inicio y fin se agregarán cuando se invoque `startInvestment()`.

En la parte superior del Card podemos ver la dirección de ese nuevo contrato desplegado y a su derecha un botón que nos permite copiarlo en el portapapeles, esto será necesario más adelante.

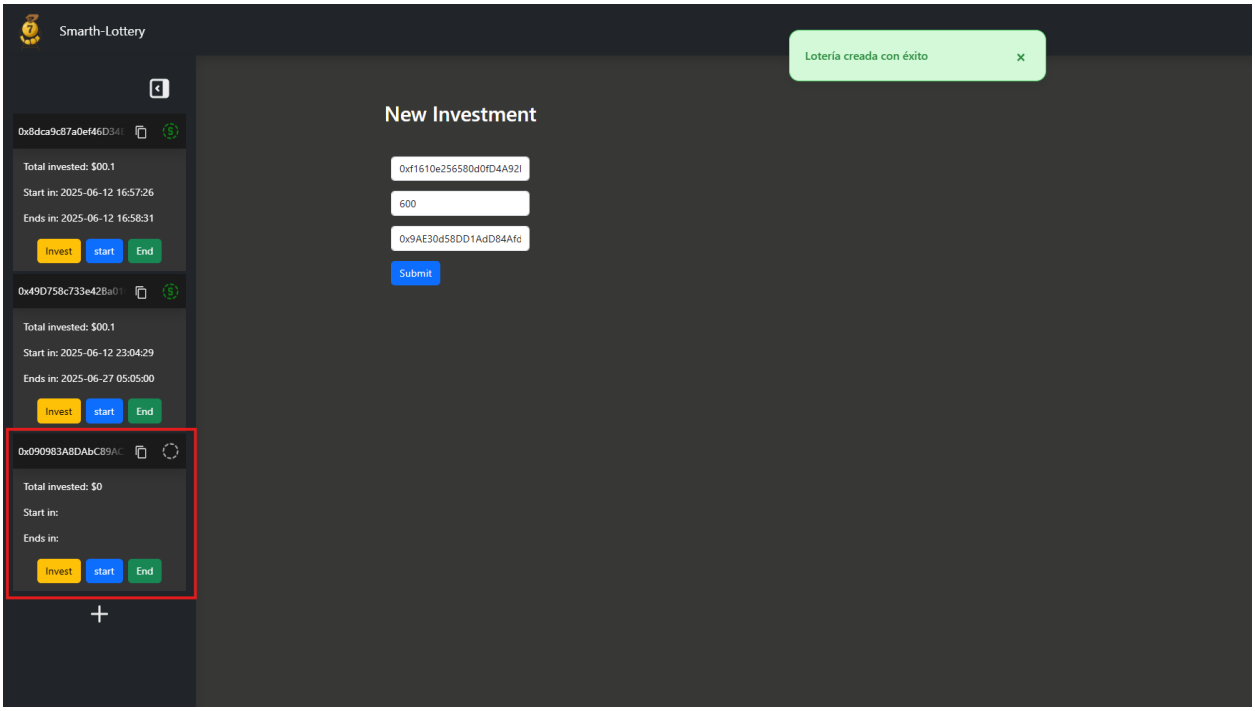

Figura 6.9: Lotería ubicada en estado “Creada” tras su despliegue

6.3.5. Interacción con la lotería

Con la lotería ya creada, los usuarios pueden participar invirtiendo ETH directamente en ella. Para ello:

1. Seleccione la lotería deseada y abra el formulario de inversión introduciendo la dirección de la lotería y el monto en ETH.

2. Al enviar, la aplicación solicita confirmación en MetaMask para autorizar la transacción.

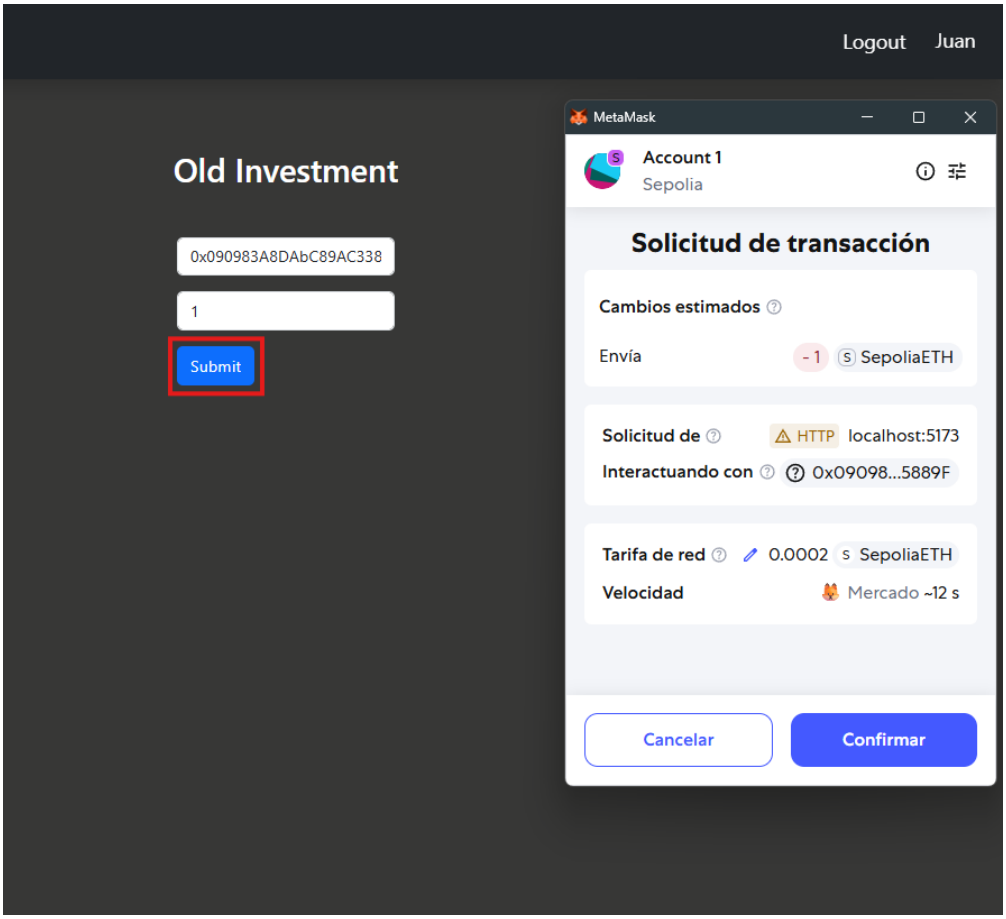

Figura 6.10: Formulario de inversión y confirmación en MetaMask

Una vez confirmada y minada la transacción, el sistema muestra una notificación informando del éxito de la operación:

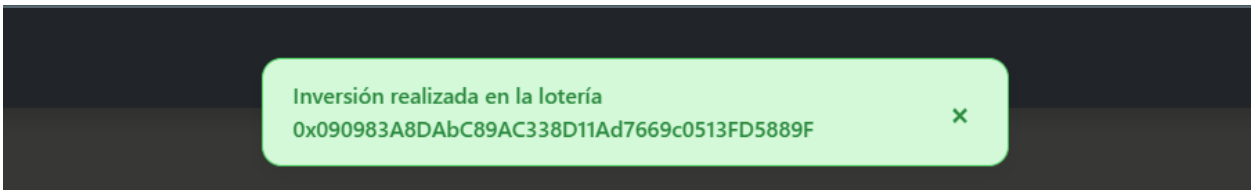

Figura 6.11: Notificación de inversión exitosa

Tras la primera inversión, el estado de la lotería cambia a **Pendiente**. En este estado, otros participantes pueden sumarse tantas veces como deseen. Cuando todas las inversiones estén realizadas, el creador de la lotería puede iniciar el periodo de inversión:

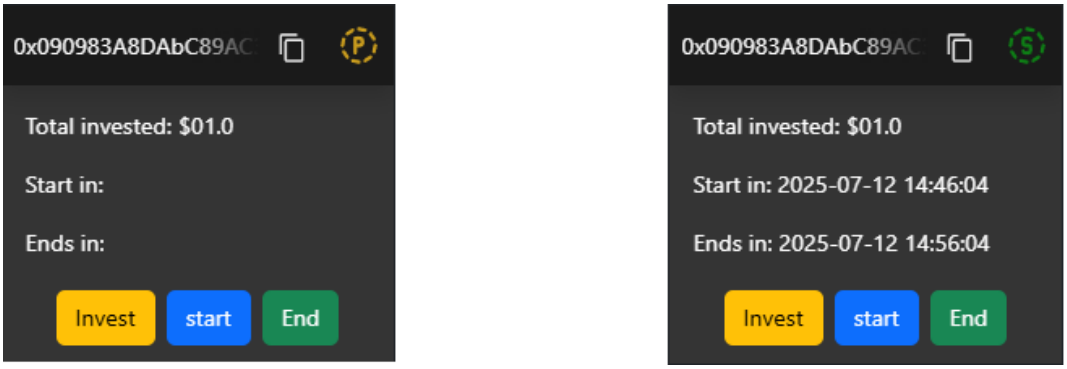

Figura 6.12: Inicio de la lotería: cambio de estado a "Iniciada" y visualización de fechas

Al cumplirse la duración configurada, el botón **Finalizar** permite cerrar la lotería y ejecutar el reparto de fondos. Nuevamente, MetaMask solicitará confirmación y, tras la transacción, el usuario recibe su inversión más la ganancia:

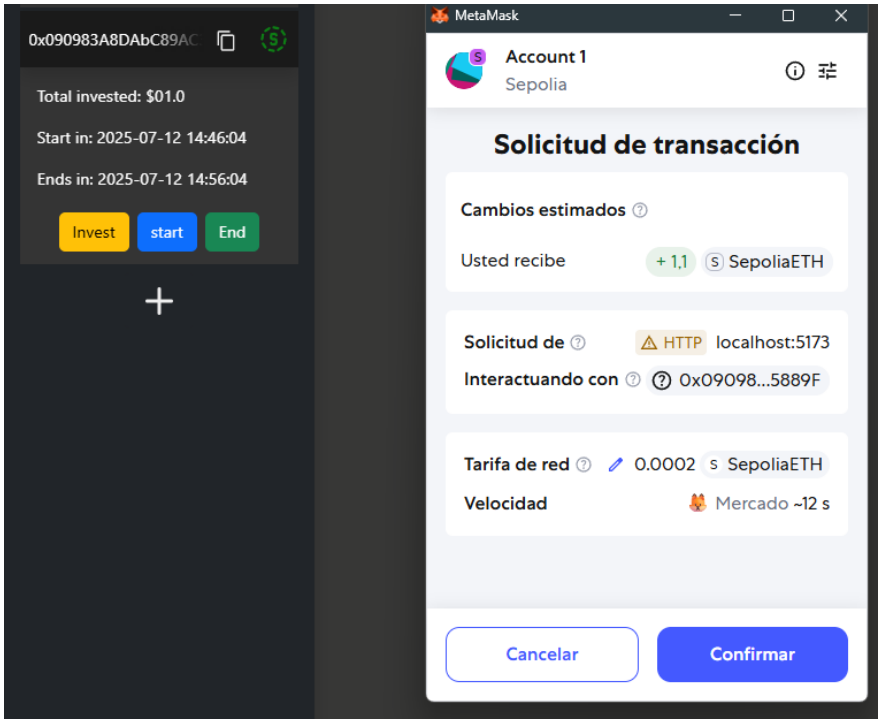

Figura 6.13: Finalización de la lotería y recepción de fondos (inversión + ganancia)

Capítulo 7

Conclusiones

El presente trabajo de grado se propuso abordar los desafíos de opacidad y riesgo de contraparte inherentes a los sistemas financieros tradicionales, mediante la aplicación de los principios de las Finanzas Descentralizadas (DeFi). Tras un proceso sistemático de diseño, desarrollo y validación, se concluye que el proyecto ha cumplido con éxito su objetivo principal: el desarrollo y validación de un prototipo funcional de un sistema basado en contratos inteligentes que ofrece un mecanismo de inversión *auditable*, operando en un entorno de prueba que demuestra la viabilidad de la propuesta.

Para alcanzar este resultado, se dio respuesta a las preguntas de investigación formuladas en la sistematización del problema, logrando así los objetivos específicos:

- Se identificaron y analizaron las plataformas blockchain más relevantes del ecosistema, estableciendo un marco de criterios ponderados que justificó la selección de Ethereum como la infraestructura más idónea para este proyecto, dando cumplimiento a los dos primeros objetivos.
- Se desarrolló una arquitectura de software modular en el lenguaje Solidity. Se implementó el Patrón Factory para la creación de instancias de inversión independientes, y se aplicaron patrones de seguridad como ReentrancyGuard para mitigar vulnerabilidades conocidas, resultando en un código robusto y bien estructurado que materializó el tercer objetivo específico.
- Finalmente, los objetivos cuatro y cinco se cumplieron mediante el despliegue del sistema en la *testnet* de Sepolia y la ejecución de un plan de pruebas exhaustivo. La validación del prototipo funcional demostró que la lógica del contrato opera según lo esperado, gestionando correctamente el ciclo de vida completo de la inversión.

A través del prototipo, se ha evidenciado cómo la inmutabilidad del registro y la ejecución determinista de los contratos inteligentes permiten eliminar la necesidad de confiar en terceros, trasladando la confianza hacia la verificabilidad pública del código. Este enfoque técnico, complementado con una arquitectura documentada y modular, entrega un caso de estudio replicable que puede servir como base para futuras investigaciones y desarrollos dentro del ecosistema DeFi.

Más allá de los logros técnicos, este trabajo también plantea posibilidades reales de aplicación en contextos donde la transparencia y la autogestión de recursos son fundamentales. El modelo desarrollado podría adaptarse a mecanismos de ahorro colectivo o fondos rotatorios comunitarios, donde la lógica automatizada y la trazabilidad de las reglas de inversión aportan confianza entre

participantes. De igual forma, su valor educativo es significativo: el sistema puede emplearse como herramienta didáctica en cursos de tecnologías financieras, contratos inteligentes o desarrollo en blockchain.

En resumen, este trabajo materializa una solución de ingeniería de software de principio a fin, y demuestra que es posible codificar reglas de inversión colectiva en una infraestructura descentralizada y verificable. Su propósito no es competir con soluciones comerciales existentes, sino ofrecer una implementación validada que sirva como punto de partida técnico y académico. A partir de esta base, el trabajo futuro podrá enfocarse en aspectos clave como la reducción de costos operativos (gas), la interoperabilidad entre redes, la integración con protocolos externos y el diseño de interfaces centradas en usuarios no técnicos, consolidando así el camino hacia una solución aplicable en escenarios reales.

Capítulo 8

Trabajo Futuro

El prototipo desplegado en testnet ha cumplido con los objetivos iniciales: demostración del flujo de inversión automatizado, cálculo de rendimientos y reparto de fondos en un entorno controlado. Para dar el salto a un entorno productivo real y garantizar que el sistema pueda crecer en funcionalidades, usuarios y redes, proponemos las siguientes líneas de mejora:

- **Conectar distintas blockchains.** Hoy el sistema funciona solo en Ethereum; en producción necesitamos que los usuarios puedan operar también en otras redes (por ejemplo, Polygon, BNB Smart Chain, Avalanche). Para ello se implementarían "puentes" y adaptadores que trasladen activos y mensajes de manera segura de una cadena a otra, manteniendo siempre la propiedad descentralizada de los fondos.

- **Soportar múltiples tipos de activos.** Además de Ether, es deseable aceptar y procesar tokens ERC-20 (como stablecoins) e incluso recibir depósitos en monedas fiduciarias. Esto implica crear módulos genéricos que gestionen distintos formatos de token (decimales, aprobaciones) y, para monedas tradicionales, integrar oráculos o servicios que conviertan valores y alimenten precios en tiempo real.

- **Adoptar estándares de token más completos.** El uso actual de tokens básicos es suficiente para un piloto, pero en un entorno real conviene soportar estándares más avanzados (por ejemplo, ERC-777 para llamadas seguras antes y después de la transferencia, o ERC-1155 para manejar distintos activos con un solo contrato). Este paso facilitará futuras extensiones, reducirá el consumo de gas en operaciones masivas y ofrecerá mayores garantías de compatibilidad.

- **Optimizar el consumo de gas.** Acciones como la creación de nuevas loterías o la ejecución de funciones complejas tienen un costo asociado en términos de gas. En redes como Ethereum Mainnet, estos costos pueden ser prohibitivos para el usuario promedio. Se sugiere estudiar la migración hacia soluciones de segunda capa (Layer 2) como Arbitrum u Optimism, o cadenas con menor costo como Polygon, a fin de mejorar la viabilidad económica del sistema.

- **Diseñar interfaces centradas en usuarios no técnicos.** Si se desea aplicar este sistema en contextos reales —por ejemplo, comunidades de ahorro colectivo— será necesario diseñar interfaces más intuitivas y accesibles. Esto incluye reducir pasos técnicos (como la gestión de wallets) y ofrecer guías visuales que ayuden a comprender el proceso sin conocimientos previos de blockchain.

Con estas mejoras, el sistema podrá escalar a múltiples redes, ofrecer una experiencia más rica en activos y prepararse para casos de uso reales en los que la interoperabilidad, la flexibilidad, la accesibilidad y la eficiencia sean requisitos indispensables.

Bibliografía

[1] C. R. Harvey, A. Ramachandran, and J. Santoro, "Defi and the future of finance." https://papers.ssrn.com/abstract_id=3711777, 2021.

[2] F. Schär, "Decentralized finance: On blockchain- and smart contract-based financial markets." https://doi.org/10.20955/r.103.153-74, 2021.

[3] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform." https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf, 2014.

[4] S. M. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. J. Knottenbelt, "Sok: Decentralized finance (defi)." https://dl.acm.org/doi/epdf/10.1145/3558535.3559780, 2022.

[5] Q. W. Z. W. Q. W. J. W. X. L. C. W. Y. D. E. Jiang, B. Qin and Y. Zhang, "Decentralized finance (defi): A survey." https://arxiv.org/abs/2308.05282, 2023.

[6] D. A. Zetzsche, D. W. Arner, and R. P. Buckley, "Decentralized finance (defi)." https://ssrn.com/abstract=3539194, 2020.

[7] ACFE, "Occupational fraud 2024: A report to the nations." https://legacy.acfe.com/report-to-the-nations/2024/, 2024.

[8] M. Iansiti and K. R. Lakhani, "The truth about blockchain." https://www.researchgate.net/publication/341913793_The_Truth_About_Blockchain, 2017.

[9] A. W. Services, "What is blockchain technology?." https://aws.amazon.com/what-is/blockchain/, 2024.

[10] Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf, 2008.

[11] PwC, "Making sense of bitcoin, cryptocurrency and blockchain." https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html, 2021.

[12] E. Foundation, "Smart contracts." https://ethereum.org/en/smart-contracts/, 2024.

[13] IBM, "What are smart contracts?." https://www.ibm.com/topics/smart-contracts, 2023.

[14] Etherscan, "Wallet on sepolia." https://sepolia.etherscan.io/address/0xf1610e256580d0fd4a92ffe49ae47696b95e3d90, 2025.

[15] C. Watsky, M. Liu, N. Ly, K. Orr, A. Seira, Z. Vida, and L. Wu, "Tokenized assets on public blockchains: How transparent is the blockchain?." https://www.federalreserve.gov/econres/notes/feds-notes/tokenized-assets-on-public-blockchains-how-transparent-is-the-blockchain-20240403.html, 2024.

[16] S. Yang and W. Cui, "An evaluation system for defi lending protocols." https://ieeexplore.ieee.org/document/10240601, 2023.

[17] Aave, "Aave protocol whitepaper." https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf, 2020. Consultado en 2024.

[18] C. Data61, "Factory contract – blockchain patterns." https://research.csiro.au/blockchainpatterns/general-patterns/contract-structural-patterns/factory-contract/, 2024.

[19] OpenZeppelin, "Security contracts api (4.x) – reentrancyguard." https://docs.openzeppelin.com/contracts/4.x/api/security, 2025.

[20] E. Foundation, "Ethereum networks." https://ethereum.org/en/developers/docs/networks/, 2025.

[21] R. Lesner and G. Hayes, "Compound: The money market protocol." https://compound.finance/documents/Compound.Whitepaper.pdf, 2019. Consultado en 2024.

[22] P. Inc., "Pooltogether documentation." https://docs.pooltogether.com/, 2024. Consultado en 2024.

[23] E. Foundation, "Ethereum official website." https://ethereum.org/. Accessed: 25-Jun-2025.

[24] H. Foundation, "Hyperledger fabric official website." https://www.hyperledger.org/use/fabric. Accessed: 25-Jun-2025.

[25] R3, "Corda official website." https://www.r3.com/corda/. Accessed: 25-Jun-2025.

[26] C. Foundation, "Cardano official website." https://cardano.org/. Accessed: 25-Jun-2025.

[27] S. Labs, "Solana official website." https://solana.com/. Accessed: 25-Jun-2025.

[28] W. Foundation, "Polkadot official website." https://polkadot.network/. Accessed: 25-Jun-2025.

[29] T. M. Fernández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the internet of things." https://ieeexplore.ieee.org/document/8370027. IEEE Access, vol. 6, pp. 32979–33001, 2018.

[30] J. L. Y. L. J. Y. R. L. B. Hu, Z. Zhang and X. Lin, "A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems." https://arxiv.org/abs/2008.13413v2, Patterns, vol. 2, 2021.

[31] M. Valenta and P. Sandner, "Comparison of ethereum, hyperledger fabric and corda." https://api.semanticscholar.org/CorpusID:46991541. FSBC Working Paper, 2017.

[32] A. G. R. Singh and P. Mittal, "A systematic literature review on blockchain-based smart contracts." https://dl.acm.org/doi/10.1145/3704741. ACM Distributed Ledger Technologies, 2024.

[33] K. Riahi, M. e. A. Brahmia, A. Abouaissa, and L. Idoumghar, "A comparative study of blockchain development platforms." https://www.researchgate.net/publication/380059420_A_Comparative_Study_of_Blockchain_Development_Platforms. Proc. ICCIP, 2024.

[34] T. Ferariu, P. Wadler, and O. Melkonian, "Validity, liquidity, and fidelity: Formal verification for smart contracts in cardano." https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.FMBC.2025.6. OASIcs FMBC, vol. 129, p 6:1–6:21, 2025.

[35] X. Li, X. Wang, T. Kong, J. Zheng, and M. Luo, "From bitcoin to solana – innovating blockchain towards enterprise applications." https://arxiv.org/abs/2207.05240. arXiv:2207.05240 [cs.CR], submitted 12 Jul 2022.

[36] H. Abbas, M. Caprolu, and R. D. Pietro, "Analysis of polkadot: Architecture, internals, and contradictions." https://arxiv.org/abs/2207.14128. CoRR abs/2207.14128, submitted July 2022.

[37] D. Morháč, K. Košťál, V. Valaštín, and I. Kotuliak, "Unispell: Universal adapter for interoperability in polkadot paraverse." https://link.springer.com/article/10.1007/s10586-025-05183-6. Cluster Computing, vol. 28, art. no. 311, Apr. 2025.

[38] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol." https://link.springer.com/chapter/10.1007/978-3-319-63688-7_12. Advances in Cryptology – CRYPTO 2017, LNCS vol. 10401, pp. 357–388, 2017.

[39] C. Foundation, "The blockchain revolution started with bitcoin. it continues now with ouroboros." https://cardano.org/ouroboros/. Cardano.org, 2025.

[40] A. Yakovenko, "Solana: A new architecture for a high performance blockchain." https://solana.com/solana-whitepaper.pdf. White Paper v0.8.13, 2017.

[41] D. P. Mishra, S. R. Behera, S. S. Behera, A. R. Patro, and S. R. Salkuti, "Solana blockchain technology: A review." https://www.researchgate.net/publication/382785733_Solana_blockchain_technology_a_review. Int. Journal of Informatics and Comm. Technology, vol. 13, no.2, pp.197–205, 2024.

[42] A. Voloder and M. D. Angelo, "Comparison of smart contract platforms from the perspective of developers." https://link.springer.com/chapter/10.1007/978-3-031-44920-8_7. Blockchain– ICBC2023, LNCSvol.14206, pp.104–118, 2023.

[43] J. Gjorgjev, N. Sejfuli-Ramadani, V. Angelkoska, P. Latkoski, and A. Risteski, "Use cases and comparative analysis of blockchain networks and layers for dapp development." https://doi.org/10.1109/MECO62516.2024.10577885. Proc. 13th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, pp. 1–5, Jun.2024.

[44] N. Oderbolz, B. Marosvölgyi, and M. Hafner, "Towards an optimal staking design: Balancing security, user growth, and token appreciation." https://arxiv.org/abs/2405.14617. arXiv:2405.14617 [econ.GN], submitted 23 May 2024.

[45] T. Hiemstra, "Blockchain platforms." https://www.theseus.fi/handle/10024/509455. Bachelor's Thesis, Metropolia University of Applied Sciences, 3 Nov.2021.

[46] T. Do, "Sok: On the evolution of public blockchains." https://eprint.iacr.org/2023/315.pdf. IACR Cryptology ePrint Archive, 2023.

[47] Z. Jurka, "Smart contracts on the blockchain: Evolution, applications, and challenges." https://is.vstecb.cz/th/iujwa/BP_-_Smart_Contracts_on_the_Blockchain_-_Evolution__Applications__and_Challenges.pdf. Bachelor Thesis, VSTE, 2024.

[48] C. Busayatananphon and B. Ekkarat, "Financial technology defi protocol: A review." https://ieeexplore.ieee.org/document/9720373. ResearchGate Preprint, 2022.

[49] S. A. B. Arellano and F. L. Redondo, "La utilidad de los métodos de decisión multicriterio (como el ahp) en un entorno de competitividad creciente." https://revistas.javeriana.edu.co/index.php/cuadernos_admon/article/view/4043, 2007.

[50] S. Team, "Solidity: Smart contract language." https://soliditylang.org/, 2024.

[51] E. Foundation, "Ethereum virtual machine (evm)." https://ethereum.org/en/developers/docs/evm/, 2024.

[52] R. Project, "Remix ide." https://remix-project.org/?lang=es, 2024.

[53] N. Foundation, "Hardhat: Ethereum development environment for professionals." https://hardhat.org/, 2024.

[54] T. Suite, "Truffle docs (archived)." https://archive.trufflesuite.com/docs/truffle/, 2023.

[55] T. H. Language, "Haskell language official website." https://www.haskell.org/, 2024.

[56] I. O. Global, "Plutus: Smart contract platform for cardano." https://plutus.iohk.io/, 2024.

[57] R. Foundation, "Rust programming language." https://www.rust-lang.org/, 2024.

[58] T. C. P. Language, "C programming language official page." https://en.cppreference.com/w/c/language, 2024.

[59] P. Technologies, "Substrate: The blockchain framework." https://substrate.io/, 2024.

[60] E. Foundation, "The dao hack explained." https://ethereum.org/en/history/#the-dao-hack, 2024.

[61] Polkadot, "Relay chain: The heart of polkadot." https://wiki.polkadot.network/docs/learn-relay-chain, 2024.

[62] Polkadot, "Parachains: Polkadot's solution for scalability." https://polkadot.network/technology/#parachains, 2024.

[63] E. Foundation, "Goerli testnet." https://goerli.net/, 2024.

[64] E. Foundation, "Sepolia testnet." https://sepolia.dev/, 2024.

[65] I. O. Global, "Cardano preprod testnet." https://book.world.dev.cardano.org/environments/preview/, 2024.

[66] I. O. Global, "Plutus playground." https://playground.plutus.iohkdev.io/, 2024.

[67] S. Foundation, "Solana testnet." https://solana.com/developers/testnet, 2024.

[68] Polkadot, "Polkadot-js apps." https://polkadot.js.org/apps/, 2024.

[69] S. Overflow, "Stack overflow: Where developers learn, share, build careers." https://stackoverflow.com/, 2024.

[70] GitHub, "Github: Where the world builds software." https://github.com/, 2024.

[71] Discord, "Discord: Your place to talk and hang out." https://discord.com/, 2024.

[72] C. Foundation, "Cardano forum." https://forum.cardano.org/, 2024.

[73] S. Foundation, "Solana community forum." https://forums.solana.com/, 2024.

[74] Polkadot, "Polkadot forum." https://forum.polkadot.network/, 2024.

[75] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger. ethereum project." https://www.the-blockchain.com/docs/Dr.%20Gavin%20Wood%20-%20Ethereum%20-%20A%20Secure%20Decentralised%20Generalised%20Transaction%20Ledger.pdf?, 2014.

[76] M. Fowler, "Patterns of enterprise application architecture." https://books.google.com.co/books?id=FyWZt5DdvFkC, 2003.

[77] ConsenSys, "Ethereum developer resources." https://consensys.io/developers/, 2023. Consultado en 2025.

[78] M. P. Ivan Homoliak, "Sok: Cryptocurrency wallets– a security review and classification based on authentication factors." https://arxiv.org/pdf/2402.17659, 2024.