DESARROLLO DE SMART CONTRACTS SEGUROS Y EFICIENTES PARA E-COMMERCE

SECURE AND EFFICIENT DEVELOPMENT OF SMART CONTRACTS FOR E-COMMERCE

TRABAJO FIN DE GRADO
CURSO 2024-2025

AUTORES
HUGO SILVA CACABELOS
ALMUDENA GÓMEZ-SANCHA LÓPEZ-BARAJAS

DIRECTORES
ELVIRA ALBERT ALBIOL
PABLO GORDILLO ALGUACIL

CALIFICACIÓN: 7.8

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DESARROLLO DE SMART CONTRACTS SEGUROS Y EFICIENTES PARA E-COMMERCE

SECURE AND EFFICIENT DEVELOPMENT OF SMART CONTRACTS FOR E-COMMERCE

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTORES
HUGO SILVA CACABELOS
ALMUDENA GÓMEZ-SANCHA LÓPEZ-BARAJAS

DIRECTORES
ELVIRA ALBERT ALBIOL
PABLO GORDILLO ALGUACIL

CONVOCATORIA: SEPTIEMBRE 2025

CALIFICACIÓN: 7.8

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

9 DE SEPTIEMBRE DE 2025

DEDICATORIA

A nuestras familias que gracias a ellas
hemos llegado hasta aquí

AGRADECIMIENTOS

Gracias a todas las personas que nos han acompañado durante la carrera y este trabajo, especialmente a nuestras familias y amigos. Gracias a Pablo y Elvira que nos han guiado y aconsejado durante todo este trabajo.

RESUMEN

Desarrollo de smart contracts seguros y eficientes para E-commerce

El E-commerce, presente en plataformas como Amazon o eBay, forma parte esencial de la vida cotidiana, pero sigue presentando problemas relacionados con la seguridad y la eficiencia de las transacciones. Este Trabajo de Fin de Grado aborda dichas limitaciones mediante el uso de la tecnología blockchain, en particular de los smart contracts sobre la red Ethereum.

Como contribución principal, se ha desarrollado un software funcional que aplica contratos inteligentes al ámbito del comercio electrónico, permitiendo almacenar de forma inmutable la información de las transacciones de compraventa en la blockchain. Con ello se refuerza la confianza del usuario y se mejora la seguridad respecto a sistemas tradicionales. Además, se ha realizado un análisis del consumo de gas y de las propiedades de seguridad de la solución propuesta.

Palabras clave

Ethereum, Contratos inteligentes, Cadena de bloques, E-commerce, Gas

ABSTRACT

Secure and Efficient Development of Smart Contracts for E-commerce

E-commerce, present in platforms such as Amazon or eBay, has become an essential part of everyday life, but it still faces challenges related to transaction security and efficiency. This Final Degree Project addresses these limitations through the use of blockchain technology, specifically smart contracts on the Ethereum network.

As its main contribution, a functional software has been developed that applies smart contracts to electronic commerce, enabling the immutable storage of purchase–sale transaction data on the blockchain. This approach strengthens user trust and improves security compared to traditional systems. In addition, an analysis of gas consumption and the security properties of the proposed solution has been carried out.

Keywords

Ethereum, Smart contracts, Blockchain, E-commerce, Gas

ÍNDICE DE CONTENIDOS

Contenidos

Dedicatoria III

Agradecimientos V

Resumen VII

Abstract IX

Índice de contenidos XI

Índice de figuras XIII

Índice de tablas XV

Capítulo 1 - Introducción 1

1.1 Motivación 1

1.2 Objetivos 3

1.3 Plan de trabajo 5

1.4 Estructura del documento 6

Capítulo 2 - Preliminares 7

Capítulo 3 - Arquitectura y diseño de la DApp 15

3.1 Arquitectura y componentes 15

3.1.2 Componentes y responsabilidades 18

3.2 Diseño DApp 20

3.2.1 Diseño de los contratos inteligentes 20

3.2.2 Diseño del frontend 21

Capítulo 4 - Implementación 25

4.1 Herramientas 25

4.2 Estructura del repositorio 28

4.3 Blockchain 29

4.3.1 EscrowFactory 29

4.3.2 Escrow 31

4.3.3 MockPriceFeed 33

4.4 Frontend 33

4.4.1 Casos de uso 33

4.5 Despliegue del proyecto 40

Capítulo 5 - Funcionalidades 43

5.1 Seguridad 43

5.2 Gas y eficiencia 44

Capítulo 6 - Trabajo relacionado 47

6.1 Comercio electrónico 47

6.2 Comercio tradicional 50

Capítulo 7 - Conclusiones y trabajo futuro 53

Contribuciones Personales 65

Bibliografía 65

Apéndices 70

ÍNDICE DE FIGURAS

Figura 2-1. Esquema de funcionamiento de la EVM. 11

Figura 3-1. Arquitectura DApp...16

Figura 4-1. Fragmento de código de contrato Escrow. 32

Figura 4-2. Barra de navegación. 34

Figura 4-3. Página Enviar. 34

Figura 4-4. Cuentas importadas a MetaMask. 35

Figura 4-5. Envío de dinero. 35

Figura 4-6. Confirmación de envío con MetaMask. 36

Figura 4-7. Mensaje de confirmación de envío. 36

Figura 4-8. Página transacciones pendientes. 37

Figura 4-9. Página liberar. 37

Figura 4-10. Página transacciones completadas. 38

Figura 4-11. Ejemplo transacción reembolsada y otra completada respectivamente...38

Figura 4-12. Ejemplo comercio electrónico. 39

Figura 4-13. Pago integrado en tienda. 39

Figura 4-14. Instalación nodo. 40

Figura 4-15. Despliegue de red de blockchain. 40

Figura 4-16. Cuentas de prueba resultantes. 41

Figura 4-17. Red local creada. 41

Figura 4-18. Comando para despliegue factoría de contratos e inclusión en el front-end. 41

Figura 4-19. Dirección del contrato incluido en frontend/src/contract/factoryAddress. 41

Figura 4-20. Inicio sesión en MetaMask. 42

ÍNDICE DE TABLAS

Tabla 3-1. Componentes y responsabilidades..19

Capítulo 1 - Introducción

El presente capítulo tiene como objetivo contextualizar el trabajo desarrollado, exponiendo las motivaciones que han impulsado su realización, los objetivos perseguidos y la planificación seguida para alcanzarlos. La sección 1.1 presenta la motivación del proyecto y la necesidad de explorar soluciones basadas en *blockchain* para el ámbito del comercio electrónico. La sección 1.2 recoge los objetivos generales y específicos que guían este Trabajo de Fin de Grado. Finalmente, la sección 1.3 describe el plan de trabajo seguido, desde el estudio preliminar hasta el diseño, implementación y evaluación de la plataforma propuesta.

1.1 Motivación

En la actualidad se observan dos tendencias que nos han llevado a la propuesta de este trabajo: el avance en las tecnologías *blockchain,* cada vez más utilizadas en distintos ámbitos como la sanidad, donde se emplean para garantizar la trazabilidad de historiales médicos o el control de la cadena de suministro de medicamentos, o las finanzas, con aplicaciones en pagos internacionales, gestión de activos y sistemas de liquidación; y, por otro lado, la necesidad de medios de pago descentralizados, transparentes, más seguros y con lógicas de liquidación distintas a las actuales.

A la vez que aumenta la confianza en las tecnologías descentralizadas, aumenta la fricción con las instituciones financieras tradicionales. Las altas comisiones, la falta de transparencia y la inestabilidad económica están impulsando la búsqueda de alternativas a estas instituciones. Esto, junto con el desarrollo de las tecnologías *blockchain*, crea un nicho de necesidad de plataformas de pago descentralizadas como las que puede ofrecer Ethereum.

Estas plataformas tienen grandes ventajas frente a las actuales: son más transparentes ya que las operaciones realizadas se apuntan en un registro público e inmutable; están descentralizadas por lo que la red es más resistente a fallos o ataques ya que no depende de los servidores de una sola empresa o institución; se reduce la complejidad y las tarifas en el proceso de pago ya que elimina la necesidad de

intermediarios; y no hay necesidad de confianza entre usuarios gracias al mecanismo de consenso de la tecnología.

Además, Ethereum tiene una gran ventaja frente a tecnologías similares: no solo actúa como medio de transmisión de valor, sino que permite la ejecución de código en la cadena de bloques gracias a su lenguaje *cuasi-Turing-completo*.

Existiendo esta demanda hemos querido desarrollar una plataforma de pago electrónico basada en contratos inteligentes de Ethereum, orientada a integrarse en comercios electrónicos. La plataforma permite que un cliente deposite el importe de una compra en un contrato *escrow*, el cual retiene los fondos hasta que un tercero neutral (agente) verifica que se ha cumplido la condición acordada, por ejemplo, la entrega del producto y autoriza la liberación del pago al comercio. Con este diseño se refuerza la confianza en la transacción y se eliminan intermediarios financieros. Paralelamente, hemos investigado los dos grandes retos que aún presentan este tipo de contratos: la eficiencia en el uso de recursos (gas) y la seguridad frente a las vulnerabilidades, con el objetivo de seguir desarrollando y mejorando la plataforma en estos aspectos de cara a futuras versiones.

1.2 Objetivos

El objetivo general es diseñar, implementar y evaluar un prototipo de plataforma de pago con contratos inteligentes, basado en Ethereum, que sea integrable en comercios electrónicos. El objetivo de la plataforma es que sea segura, eficiente y fácil de integrar.

Los objetivos específicos son los siguientes:

- Investigación y estudio sobre el estado de la cuestión y sobre las opciones tecnológicas para su implementación.

  Realizar una revisión de la literatura y de plataformas existentes con el fin de identificar los retos y soluciones previas en el uso de *smart contracts* para pagos electrónicos. Paralelamente, analizar las distintas opciones tecnológicas disponibles (lenguajes, entornos de desarrollo, librerías, *wallets* y herramientas de despliegue) para fundamentar las decisiones de diseño del proyecto.

- Diseño de la arquitectura del sistema.

  Definir una arquitectura en capas que separa la interfaz de usuario, la gestión de cuentas y firmas y la lógica de negocio implementada en la blockchain mediante contratos inteligentes. Además, diseñar los actores involucrados y sus interacciones a través del contrato *escrow*.

- Implementación de los contratos.

  Desarrollar contratos inteligentes en Solidity que gestionan la creación de depósitos de custodia, la liberación de fondos y la devolución de estos en caso de incumplimiento. Aplicar patrones de seguridad para prevenir ataques de reentrada, así como buenas prácticas para reducir el consumo de gas.

- Implementación de interfaz gráfica.

  Construir una aplicación descentralizada (*DApp*) con un *frontend* que permita a los usuarios (clientes, comercios y agentes) interactuar con los contratos de manera sencilla, conectando sus carteras y gestionando pagos desde una interfaz web intuitiva.

- Integración de contratos e interfaz.

  Construir una capa de integración entre los contratos desplegados en la red de pruebas y la interfaz gráfica, permitiendo que la DApp capture direcciones de contratos, gestione estados de transacciones, ofrezca retroalimentación en tiempo real y muestre el histórico de operaciones realizadas.

- Seguridad y uso de gas.

  Evaluar la robustez de los contratos frente a vulnerabilidades comunes en Ethereum (como reentrada o uso ineficiente del almacenamiento) y medir el consumo de gas de las operaciones, proponiendo optimizaciones que contribuyan a un sistema más seguro y eficiente.

- Investigación sobre herramientas parecidas.

  Estudiar plataformas que ofrecen servicios de pago con contratos inteligentes para comparar enfoques, identificar buenas prácticas y contrastarlos con la propuesta de este trabajo.

- Investigación sobre mejoras futuras.

  Explorar líneas de evolución del prototipo en base al potencial y las limitaciones de este.

1.3 Plan de trabajo

Al inicio del proyecto se realizó una exploración amplia de alternativas: desde construir un marketplace descentralizado completo hasta desarrollar componentes aislados de pagos. Al final se decidió enfocarse en una plataforma de pago integrable en comercios electrónicos. El objetivo general guía todo el plan: diseñar, implementar y evaluar un prototipo en Ethereum que sea seguro, eficiente y fácil de integrar.

Con esa meta, se arrancó primero con el estudio de Ethereum y Solidity para crear una base de conocimiento sólida sobre la que después trabajar. Tras esto se investigó las opciones tecnológicas (carteras, librerías, patrones de escrow y costes en gas) para decidir qué aplicar en el presente proyecto. A partir de ahí se procedió al diseño de la arquitectura: separación estricta entre DApp (presentación), cartera (firma) y contratos (EscrowFactory/Escrow), definición de roles (payer, payee, agente) y de la máquina de estados con sus eventos. Sobre ese diseño, se abordó la implementación de los contratos en Solidity aplicando patrones para evitar reentradas, cuidando las escrituras en storage para contener el gas. En paralelo, se desarrolló la interfaz gráfica y una capa API de modo que orquesta la firma con MetaMask y reconstruye listados desde eventos sin depender de backend.

A continuación, se continuó con la integración de contratos e interfaz: validaciones en el formulario, feedback de transacción (hash, estado), captura de la dirección del nuevo escrow desde el receipt y vistas de seguimiento/resolución. Este ciclo se completó con una revisión de seguridad y uso de gas.

Finalmente, se investigó sobre trabajos relacionados y otras plataformas existentes, así como líneas de mejora futura alineadas con el objetivo de integración.

1.4 Estructura del documento

El presente documento se organiza en siete capítulos, además de las secciones preliminares de resumen, abstract, dedicatoria y agradecimientos:

- Capítulo 1 – Introducción. Presenta la motivación del trabajo, los objetivos planteados, el plan de trabajo seguido y la estructura general del documento.
- Capítulo 2 – Preliminares. Revisa los conceptos fundamentales de la tecnología blockchain y los smart contracts, así como las principales plataformas y herramientas relacionadas, destacando sus ventajas y limitaciones en el ámbito del e-commerce.
- Capítulo 3 – Diseño. Describe la arquitectura de la plataforma, sus componentes principales y el diseño de la aplicación descentralizada (DApp), incluyendo la interacción con la red blockchain y la interfaz web.
- Capítulo 4 – Implementación. Expone las herramientas empleadas, la organización del repositorio y la implementación detallada de los contratos inteligentes y la DApp.
- Capítulo 5 – Funcionalidades. Analiza las principales funcionalidades logradas en los aspectos de seguridad y eficiencia en el uso de gas.
- Capítulo 6 – Trabajo relacionado. Examina soluciones existentes en el sector que abordan problemas similares mediante contratos inteligentes, con el fin de contextualizar y comparar la propuesta de este trabajo.
- Capítulo 7 – Conclusiones y trabajo futuro. Resume los logros alcanzados, valora las limitaciones detectadas y plantea posibles mejoras y líneas de trabajo que podrían desarrollarse en el futuro.

Capítulo 2 - Preliminares

Ethereum[1] es una máquina de estados distribuida cuyo objetivo declarado es desarrollar una tecnología generalizada en la que se pueden construir todas las máquinas de estado basadas en transacciones. Además, facilita las transacciones entre individuos que, en otra situación, no tendrían medios para confiar uno en otro. Esto se consigue gracias a la transparencia y universalidad que proporciona un intérprete algorítmico.

Ethereum tiene un estado global que es un mapeo entre las direcciones y los estados de las cuentas. Las direcciones son identificadores de 160 bits que representan cuentas. Estas pueden ser de dos tipos: de propiedad externa (EOA) que controla un usuario con su clave privada y de contrato que está controlada por su código. Los usuarios usan sus cuentas para intercambiar Ether o para desplegar contratos. Por otro lado, la cuenta de un contrato se crea cuando este es desplegado y es activado por llamadas de otras cuentas.

Los estados de las cuentas son estructuras de datos que se componen de cuatro campos: *nonce* que representa, en las cuentas EOA, el número de transacciones enviadas y, en las cuentas de contratos, los contratos creados; *balance* que es el Ether total poseído por la cuenta; *storageRoot* que es un hash que apunta al almacenamiento de los datos asociados a una cuenta de contrato; y *codeHash* que, para contratos, es un hash inmutable que es utilizado por la máquina virtual para acceder al código y, para usuarios, es una cadena vacía. La máquina de estados empieza en un estado inicial y, a través de transacciones, va cambiando de estado al actual.

Un contrato inteligente es, básicamente, un programa y su estado guardados en la cadena de bloques[2]. Este se ejecuta en la máquina virtual de Ethereum y, tras ser desplegado, usuarios y otros contratos pueden invocar sus funciones. Una vez desplegados no se pueden cambiar. Quien inventó el termino *smart contract* fue Nick Szabo y definió un mercado digital con procesos automáticos y seguros sin necesidad de intermediarios[3].

> *A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs.*
>
> *Nick Szabo, 1994[4]*

Desplegar el contrato consiste en, primero, enviar una transacción con el código y, opcionalmente, los parámetros del constructor. A continuación, se crea en la red una dirección del contrato, con su estado y almacenamiento inicial. Una vez desplegado, cualquier cuenta, ya sea de usuario o de contrato, puede enviarle una transacción a este ya que su dirección es pública. Además, se puede consultar gratis las funciones de un contrato ya que esto no altera el estado del contrato. El contrato cambia su estado según las transacciones recibidas y puede emitir eventos para notificar a clientes o aplicaciones externas en las que está integrado.

Los eventos emitidos por un contrato no se guardan en almacenamiento, sino en un registro especial llamado *log*. Este registro es mucho menos costoso que almacenar datos en el contrato y permite que aplicaciones externas puedan escucharlos y recoger esta información. Además, permite indexar datos para una posterior lectura y filtrado más rápido[5].

Los contratos inteligentes tienen grandes ventajas como la ejecución automática, la ausencia de ambigüedad, el registro público y la posibilidad de construcción modular gracias a que los contratos pueden llamarse entre sí. Además, al utilizar un lenguaje *cuasi-Turing-completo* los contratos se pueden aplicar a casi cualquier caso de uso. El lenguaje en sí es Turing completo pero, efectivamente, no lo es por el límite de gas que se impone para no entrar, por ejemplo, en bucles infinitos.

Algunos ejemplos de caso de uso común de la aplicación de los contratos inteligentes son:

- Los tokens fungibles pueden representar virtualmente cualquier cosa desde activos financieros hasta las habilidades de un personaje en un juego. Se crean usando el estándar ERC-20 que hace que cada token sea igual en tipo y valor que otro para que sean intercambiables entre sí y compatibles con todas las aplicaciones de Ethereum[6]. Un ejemplo concreto y destacado son las *stablecoins* que son tokens diseñados para mantener un valor fijo, aunque el valor del Ether cambie[7].
- Las finanzas descentralizadas (DeFi) recrean servicios financieros tradicionales utilizando contratos inteligentes y sin intermediarios, es decir, proveen una alternativa financiera descentralizada con total visibilidad y acceso a cualquier persona con conexión a Internet[8]. Un ejemplo destacado es Uniswap[9] que funciona como mercado de intercambio de tokens ERC-20 y criptomonedas.
- Los tokens no fungibles (NFT) representan la propiedad de ítems digitales únicos. Estos se crean según el estándar ERC-721 que hace que cada NFT sea único y pueda tener un valor diferente a otro el mismo contrato ya sea, por ejemplo, por su antigüedad o características[10]. Esta naturaleza ha hecho que el uso predominante sea el arte digital y los coleccionables.
- Las aplicaciones descentralizadas (DApp) son las construidas en una red descentralizada que combina contratos inteligentes con una interfaz de usuario. En Ethereum tiene su backend corriendo en la cadena de bloques y el frontend hace llamadas a este para interactuar[11]. Estas se usan, por ejemplo, para implementar finanzas descentralizadas como Uniswap.
- Los contratos escrow son un mecanismo financiero que actúan como depósito temporal que puede controlar un tercero neutral en una transacción entre comprador y vendedor. El contrato retiene los fondos y el tercero elegido por las dos partes regula la transacción hasta que se cumplan las condiciones acordadas por las partes interesadas. También se puede automatizar el proceso completamente con llamadas desde otros contratos o aplicaciones para la liberación de los fondos tras verificarse automáticamente el cumplimiento de las condiciones acordadas. Este mecanismo reduce el riesgo de incumplimiento del acuerdo y es

completamente transparente ya que se registra en la cadena de bloques. También se puede modelar como una retención de depósitos de ambas partes: los fondos por parte del comprador y el bien por parte del vendedor. Esto se puede aplicar cuando se trata de bienes digitales y se liberan los fondos cuando el comprador confirma la entrega correcta sin necesidad de un tercero[12].

Para programar los contratos inteligentes se utilizan principalmente dos lenguajes: Solidity[13] que es el estándar en la práctica, este tiene amplias librerías y documentación, es de alto nivel, orientado a objetos; y Vyper[14] que tiene una sintaxis similar a la de Python[15] con menos funciones con el objetivo de que los contratos sean más seguros y fáciles de auditar. Por otro lado, para optimizaciones de bajo nivel se usan Yul/Yul+ pero solo se recomiendan para desarrolladores más familiarizados con las prácticas de seguridad recomendadas y el funcionamiento de la máquina virtual que ejecuta el código de los contratos[16].

La Ethereum Virtual Machine, también conocida como EVM, es la máquina virtual que permite la ejecución de código en la cadena y su comportamiento está formalizado en el *Yellow Paper*[1]. Se ejecuta como una máquina de pila operando con palabras de 256 bits, lo cual simplifica la verificación criptográfica ya que los 256 bits se alinean con el tamaño de enteros que usa Ethereum para las firmas digitales, las funciones de hash y la indexación del estado global.

La EVM tiene tres regiones de memoria (véase Figura 2-1): pila, memoria volátil que se mantiene durante la ejecución por la llamada, pero no entre transacciones, y almacenamiento que está asociado a la cuenta del contrato y es persistente en la cadena, es decir, forma parte del estado global.

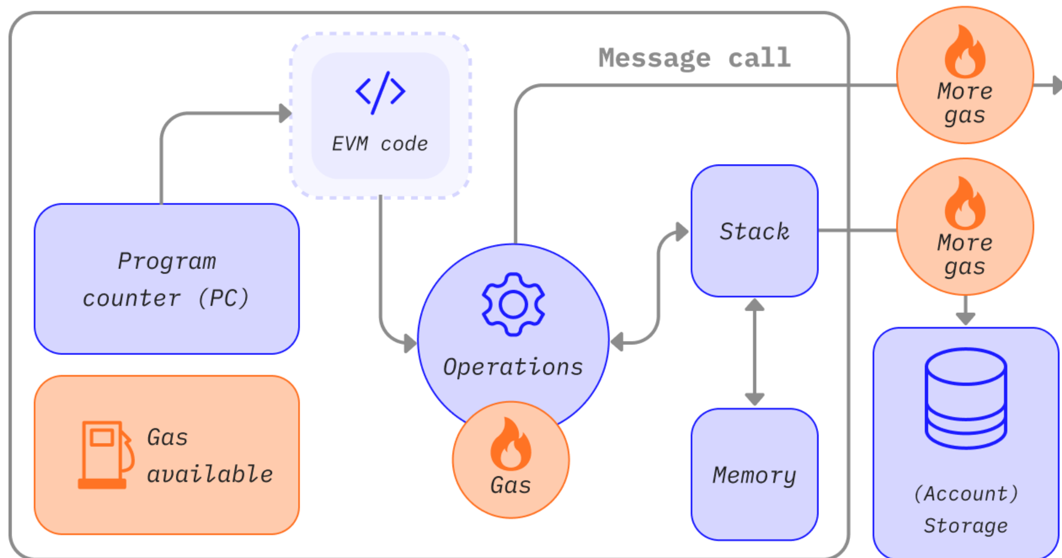

Figura 2-1. Esquema de funcionamiento de la EVM[1].

El lenguaje máquina de la EVM es el *bytecode* que es una secuencia de datos y *opcodes* (códigos de operación) de bajo nivel que indican qué operación concreta debe ejecutar la máquina virtual. Esto lo genera el compilador y es lo que se despliega y queda inmutable en la cadena de bloques. Cada opcode tiene asociado un coste de gas que se explicará más adelante.

La ejecución empieza con una llamada a un contrato o una transacción de creación. En una llamada el contrato recibe los datos de esta, el emisor, el valor en ETH y un presupuesto de gas para su ejecución. Entonces, el código del contrato, utilizando la pila y memoria, puede leer o escribir de almacenamiento, emitir eventos y realizar llamadas a otras cuentas. Cada llamada que realice tiene su propio presupuesto de gas y devuelve datos o errores.

Una transacción es una transición válida entre estados de la máquina mencionada anteriormente. Las transacciones realizadas por un usuario tienen una firma digital que verifica que ha sido autorizada por el propietario de la cuenta que la realiza. Esta firma resulta del siguiente proceso: primero se pasan los datos de la

---

[1] https://ethereum.org/content/developers/docs/gas/gas.png

transacción por un algoritmo de hash, después el usuario utiliza su clave privada para firmar esta transacción y, entonces, se genera la firma digital que es única. Este sistema permite que, con la clave pública, se pueda verificar si la firma es válida y, a la vez, es imposible obtener la clave privada a través de la firma.

Al definir lo que es una transacción se enfatiza su validez ya que hay cambios de estados que son inválidos como que aumente una cuenta sin disminuir otra, la firma sea incorrecta o no haya saldo suficiente para cubrir el coste de la operación. De manera más técnica, una transacción es una instrucción firmada criptográficamente por un actor externo que despliega código o invoca funciones. Hay dos tipos de transacciones: las provocadas por llamadas de mensajes y por creación de contratos [1].

Un bloque es una colección de transacciones que se une a otros con hashes criptográficos formando así la cadena de bloques. Además, también contienen otra información como el hash del bloque padre o el gas total utilizado por transacciones en el bloque. Esta estructura encadenada es lo que hace que sea inmutable, es decir, que no se permitan las reescrituras ya que estas romperían la cadena de hashes.

Ethereum, así como otras tecnologías blockchain, funciona gracias a una red de personas (nodos) que lo guardan y se encargan de ejecutar las operaciones computacionales que los usuarios quieren realizar. Para que este sistema funcione tiene que existir un incentivo y, en consecuencia, una manera de transmitir valor. Aquí entran en juego los Ether (ETH) y el gas.

El Ether es la criptomoneda nativa de Ethereum que se puede usar como moneda de cambio o para utilizar la red. El gas mide el coste computacional de cada operación ejecutada en la EVM y tiene distintos objetivos: pagar a los validadores y el uso de energía, desincentivar el uso de almacenamiento excesivo o repetitivo, acotar la ejecución frente a ataques de denegación de servicio y evitar la sobrecarga de Ethereum[17]. Cada instrucción del *bytecode* tiene un precio fijo de gas. Por ejemplo, una suma (ADD) cuesta tan solo 3 unidades de gas, en cambio almacenar un valor de manera persistente (SSTORE) cuesta entre 5000 y 20000 unidades de gas[18]. Esto incentiva a tan solo almacenar la información necesaria y esencial en la cadena. Además, durante la ejecución del código en la EVM se pueden producir sobre costes de gas por expansión de la memoria o llamadas a otros contratos.

El usuario que envía una transacción debe proporcionar con antelación un límite de gas que está dispuesto a pagar por la transacción. Si la ejecución consume más gas que el definido se produce una *out-of-gas exception* y se revierten todos los cambios, pero el gas no se reembolsa. Este mecanismo protege a la red frente a bucles infinitos o contratos maliciosos.

Para calcular el gas total de una transacción se multiplica la tarifa de gas, que debe de ser pagada en Ether, y las unidades de gas utilizadas. A su vez, la tarifa se divide en dos: tarifa base que se establece por el protocolo para realizar una transacción y tarifa prioritaria que es la propina que se puede añadir para incentivar a los validadores. Si no se incluye propina es improbable que la transacción se incluya en la cadena ya que no existe incentivo, sobre todo ante altas demandas. Además, la tarifa base se consume al ejecutarse, es decir, los Ethers que representa ya no estarán en circulación tras la creación del bloque[19].

Las personas que ejecutan las transacciones y, en consecuencia, reciben las recompensas y deciden qué transacciones entran en cada bloque, actualmente, son los validadores. Anteriormente esto no era así ya que depende del mecanismo de consenso vigente. En los primeros años fueron mineros con el modelo *Proof of Work* (prueba de trabajo) y, tras la bifurcación Paris en 2022, conocida como *La Fusión*, son los validadores con el nuevo mecanismo de consenso, el *Proof of Stake* (prueba de participación)[20].

Con la prueba de trabajo (PoW), los mineros eran los nodos que competían entre sí para crear el siguiente bloque resolviendo un acertijo criptográfico. El primero en resolverlo conseguía el derecho a agregar el bloque a la cadena y recibía la recompensa en Ether. Este sistema era seguro ya que alterar la cadena requería rehacer todo el trabajo computacional más rápido que el resto de la red. Esto es prácticamente imposible sin controlar más del 51% de la potencia de minado. Pero el problema con este sistema era el alto coste energético ya que era aproximadamente lo mismo que tenía toda la República Checa[21].

La prueba de participación (PoS) funciona de manera muy distinta. Ya no hay mineros sino validadores y estos ya no compiten con poder computacional sino con valor económico en juego. Los validadores primero deben retener 32 ETH en un contrato

inteligente de depósito. Entonces, el protocolo selecciona aleatoriamente a un validador para introducir un bloque mientras que, paralelamente, otros validadores estudian su validez. Los validadores que trabajan correcta y honestamente reciben recompensas proporcionales a su participación, pero los que intentan introducir bloques inválidos pierden los Ether que tienen bloqueados. En consecuencia, la seguridad se mantiene ya que atacar al sistema sería arriesgar un capital enorme[22].

Gracias a la prueba de participación se redujo el consumo de energía en un 99,95%[23]. Además, consigue que los incentivos de los validadores se alineen con los de Ethereum ya que tienen interés en que la red prospere porque su propio capital está en juego.

Por último, otro cambio causado por *La Fusión* fue la separación de la arquitectura en dos capas: la capa de ejecución y la capa de consenso. La de ejecución define las reglas para actualizar e interactuar con la EVM así como las reglas de gas y la de consenso define las reglas para determinar la historia canónica de los bloques de Ethereum[1]. Esta división hace que la estructura del protocolo sea más clara y permita la escalabilidad.

Capítulo 3 - Arquitectura y diseño de la DApp

Este capítulo explica la arquitectura, los componentes, sus interacciones y el diseño de la web de la plataforma de pago elaborada. En la sección 3.1 se definen los componentes principales y su organización e interacción, así como las capas de la aplicación y los actores involucrados. Y, en la sección 3.2, se detallará los modelos y patrones que se han aplicado en su diseño. La implementación detallada de todo esto se presentará en el siguiente capítulo.

En los pagos en comercios electrónicos a través de contratos inteligentes, participan tres actores: el pagador o cliente, el cobrador o comercio y el escrow que es un contrato que funciona como depósito temporal ya que retiene los fondos y los libera al cumplirse una condición. Esto es un gran atractivo para los clientes ya que se aseguran de que se cumpla lo acordado en el contrato como, por ejemplo, la entrega del producto comprado. Las reglas y condiciones para la liberación o reembolso de fondos quedan fijadas con anterioridad.

El objetivo principal de la DApp es proporcionar una plataforma de pago que incremente la confianza entre cliente y vendedor. En lugar de transferir directamente los fondos al comercio en el momento de la compra, el importe queda depositado en un contrato inteligente en la cadena de bloques que es desplegado automáticamente a través de una factoría. Este dinero solo será liberado cuando una compañía de entrega, elegida por el cliente, confirme la entrega de lo comprado. Este contrato protege a ambas partes: el comercio tiene la garantía de que los fondos están retenidos y disponibles en la cadena y el cliente se asegura que no se libera su dinero hasta que recibe su paquete.

3.1 Arquitectura y componentes

La plataforma de pago implementa un pago condicionado con escrow para integrarse en el checkout de un comercio electrónico. De manera simple, la plataforma funciona de manera que al principio se despliega en la red local una factoría de contratos y se guarda esta dirección en el frontend para que pueda enseñar la información. Desde este se puede hacer una transacción o pago que crea un contrato

que contiene la información de esta y que un tercero independiente debe autorizar cuando se cumpla la condición acordada entre el cliente y el comercio. Por ejemplo, que se produzca la entrega del paquete, actuando como agente la empresa de entregas.

En la figura 3-1 se representa de manera visual el funcionamiento de la DApp, incluyendo las capas y actores principales que se detallarán a continuación. La figura representa los tres flujos principales: en negro, el pago a través de la plataforma realizado por el cliente; en verde, la liberación o reembolso realizado por el agente; y, en azul, la consulta y visualización del historial de transacciones.

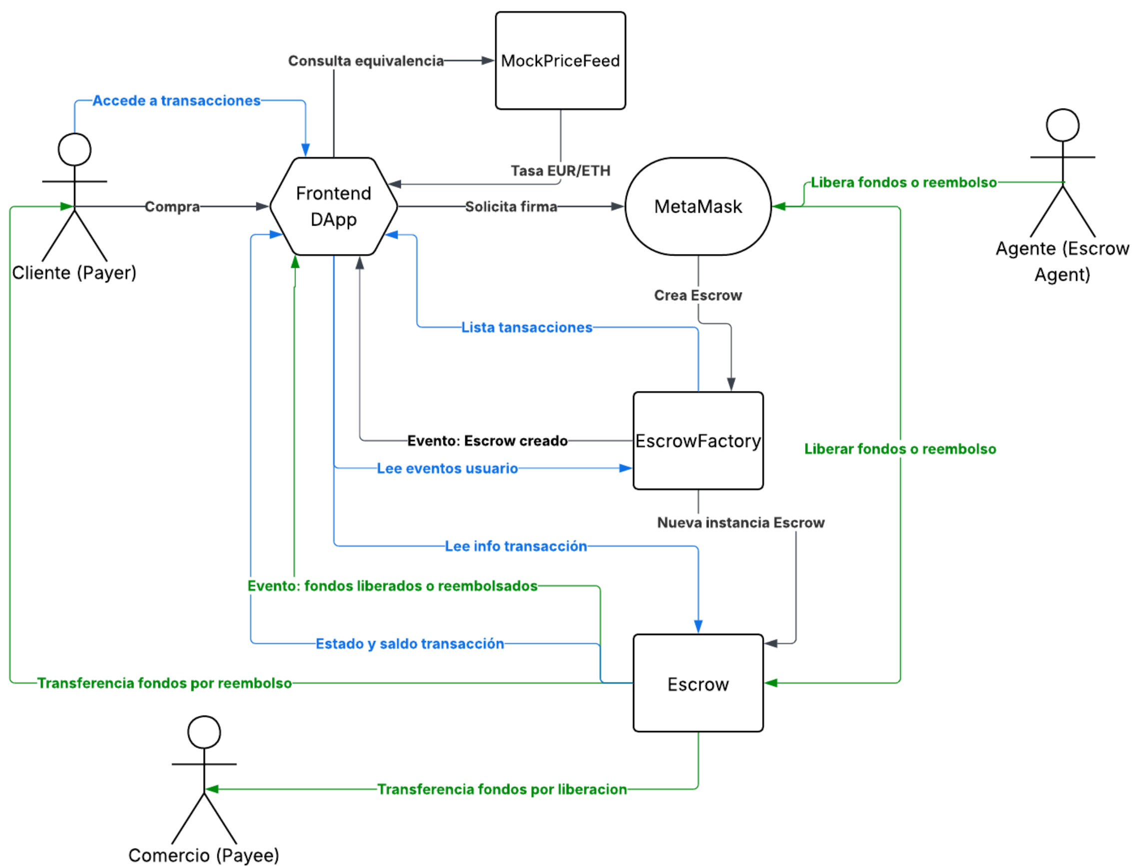

Figura 3-1. Arquitectura DApp.

La DApp se comunica, desde el frontend, con la cartera conectada para solicitar firmas y con los contratos desplegados para realizar lecturas y transacciones. EscrowFactory es el punto de entrada y recibe la llamada *createEscrow(payee, agent)*. Esta llamada crea una instancia de *Escrow* que retiene los fondos a transmitir y emite el

evento *EscrowCreated* con la dirección de la nueva instancia. Escrow tiene como datos *payer, payee, agent, amount y estado* y tiene dos funciones: *releaseFunds()* para pagar al comercio y *refund()* para devolver el dinero al cliente y emite eventos al realizaras para mantener la transparencia y permitir la auditoría.

3.1.1.1 Capas de la arquitectura

La arquitectura está formada por tres capas:

- Frontend (DApp) que se comunica con MetaMask[24] para solicitar firmas y con la cadena de bloques para realizar lecturas y mostrar las transacciones pendientes y completadas.
- Autorización (MetaMask) que gestiona las cuentas y firma de transacciones y actúa como puente hacia un nodo de Ethereum. Es la única capa con claves ya que, por seguridad, la DApp no debe custodiar secretos.
- Custodia y ejecución (contratos en Ethereum):
  - EscrowFactory: es el punto de creación de los depósitos y se despliega al arrancar la plataforma.
  - Escrow: se crea una instancia por pedido y se encarga de custodiar el importe y ejecuta el envío o reembolso de fondos según el control del agente.
  - MockPriceFee: es un componente auxiliar que simula la conversión de euros a Ether.

Esta separación reduce el riesgo, facilita la predicción del coste por operación y otorga máxima transparencia al poder ver todos los eventos en la cadena de bloques, pero manteniendo, a su vez, la privacidad.

3.1.1.2 Actores

A alto nivel tenemos tres actores que interactúan entre sí:

- Payer: es el pagador, es decir, el cliente del comercio. Este inicia la compra y deposita sus fondos en el escrow.

- Payee: es el comercio que recibe el pago en caso de liberación.
- Escrow Agent: es un tercero neutral que autoriza la liberación o reembolso de los fondos según las condiciones establecidas.

3.1.2 Componentes y responsabilidades

La DApp guía el proceso, la cartera muestra y firma las transacciones y la capa de blockchain tenemos una factoría de contratos desplegada que es la que despliega una instancia de Escrow por pedido.

De esta manera conseguimos reducir las operaciones costosas de escritura a solamente la creación de la transacción y la autorización o reembolso de esta, es decir, la actualización del estado del contrato de transacción. Tan solo se realizan lecturas de la cadena de bloques desde el frontend ya que este tiene registrada la dirección de la factoría de bloques.

En general la plataforma se compone de cuatro componentes principales: la DApp, MetaMask, EscrowFactory y Escrow. A continuación, en la tabla 3-1, se recoge las responsabilidades, interacciones y datos que guardan en o fuera de la cadena de bloques de cada componente.

Componente	Responsabilidades	Interacciones	Datos on/off-chain
DApp	Interfaz Validaciones Visualización de estado e historial	Lecturas Solicitud cuentas y firmas a MetaMask Invocación de la función para la creación de la instancia de Escrow	On: lecturas Off: metadatos de pedido y direcciones de contratos
MetaMask	Gestión de cuentas Gestión de firmas	Comunicación entre DApp y nodos	On: no aplica Off: almacenamiento

	Puente al nodo en Ethereum		de las cuentas y claves
EscrowFactory	Punto de entrada de pagos Despliegue de escrows Emisión de eventos	Recepción de la llamada para crear una instancia de Escrow Emisión evento para registrar el Escrow creado	On: evento *EscrowCreated* Off: no aplica
Escrow	Custodia de fondos y transacciones Ejecuta la liberación o reembolso	Recibe llamadas del agente Transfiere fondos al cliente o comercio Emisión evento de fondos depositados, liberados o reembolsados	On: atributos y eventos Off: no aplica

Tabla 3-1. Componentes y responsabilidades.

Gracias a esta arquitectura se mantiene la confianza. El usuario solo interactúa con la DApp la cual no guarda claves ni firmas ya que las autorizaciones pasan por MetaMask. La DApp solo guarda la dirección de la instancia del *Escrow* creado por el pago. Además, solo se guarda en la cadena lo esencial para el pago y los datos sensibles del comprador y su compra o del vendedor se mantienen fuera preservando la privacidad y minimizando también así el gasto de gas. Y, en relación al coste, el contrato no incluye bucles de longitud variable lo cual hace que el coste por operación sea más predecible.

3.2 Diseño DApp

El diseño de la DApp se ha orientado a desarrollar una solución segura, eficiente y fácil de usar. La aplicación combina contratos inteligentes eficientes y auditables con una interfaz intuitiva.

3.2.1 Diseño de los contratos inteligentes

Los contratos son el núcleo de la aplicación ya que se encargan de custodiar los fondos y ejecutar las transacciones. El diseño de estos se centra en mantener una lógica simple a través de una factoría que despliega escrows independientes y un modelo de roles y estados que garantiza la seguridad y transparencia en las transacciones.

3.2.1.1 Patrón factoría

Se adopta un patrón de factoría en lugar de un contrato único ya que tiene varias ventajas:

- Permite gestionar múltiples pagos concurrentes.
- Simplifica la auditoría ya que cada contrato se puede analizar de manera independiente.
- Al aislar los fondos, un posible fallo no afecta a otros.

3.2.1.2 Modelo de roles y estados

Cada contrato Escrow define roles inmutables (*Payer*, *Payee*, *EscrowAgent*) que son fijados en el constructor al crearse la transacción. Esto asegura que no se puedan modificar tras ser desplegado y reduce el coste de lectura de estos datos.

La lógica del contrato Escrow tiene una máquina de estados simple que garantiza que el contrato no pueda reutilizarse una vez completada la transición. El estado tiene tres posibles valores:

- *AWAITING_RELEASE*: es el estado inicial tras su despliegue y representa que está esperando a que el agente apruebe la liberación o reembolso de fondos.

- *COMPLETE*: es un estado final al que pasa el contrato cuando el agente ejecuta la liberación de los fondos, transfiriéndolos al comercio.
- *REFUNDED*: es un estado final al que pasa el contrato cuando el agente ejecuta el reembolso de los fondos, devolviéndolos al cliente.

El paso del estado inicial a cualquiera de los dos finales es irreversible. Esto garantiza que cada instancia del contrato Escrow solo puede usarse para un pago único, evitando la reutilización indebida. Con este modelo se consigue un buen equilibrio entre seguridad y simplicidad.

3.2.1.3 Gestión de eventos

En Solidity, los contratos inteligentes pueden emitir eventos que las DApps pueden escuchar y actuar en consecuencia. Además, estos se pueden indexar para que puedan buscarse más adelante[5].

Los contratos de la DApp emiten los siguientes eventos como mecanismo de trazabilidad:

- *EscrowCreated:* lo emite EscrowFactory al desplegar una instancia de Escrow.
- *FundsReleased:* lo emite Escrow al liberar los fondos al comercio.
- *RefundIssued:* lo emite Escrow al reembolsar los fondos al cliente.

El frontend reconstruye el historial de transacciones y sus estados a partir de estos eventos en vez de hacer lecturas del almacenamiento. Esto minimiza los costes de gas y facilita el filtrado y lectura del historial de transacciones.

3.2.2 Diseño del frontend

En el diseño de la interfaz se ha priorizado la simplicidad en el flujo de pago, así como la integración con MetaMask, con el objetivo de que sea parecido a lo que está acostumbrado el usuario en otras plataformas de pago integradas en e-commerce.

En esta sección se explicará el flujo de interacción y los principios aplicados de manera simplificada con la intención de dar una visión general. La implementación concreta y el uso de la aplicación detallada se explicarán en el siguiente capítulo.

3.2.2.1 Flujo de interacción

El flujo se ha diseñado para que el usuario realice el pago de manera similar a otras plataformas y con total visibilidad de lo que ocurre en la cadena. El proceso arranca en el checkout, confirmando el carrito y el agente seleccionado. La DApp consulta el oráculo de precios para averiguar su equivalente en Ether y mostrar el desglose antes de firmar. Al aceptar, se abre MetaMask para solicitar la firma y se envía la llamada de creación del Escrow a la factoría. El frontend escucha el evento que crea la factoría y extrae la dirección del nuevo contrato asociándola al pedido para mostrarla y que el usuario pueda consultarla.

Mientras los fondos permanecen retenidos en el depósito escrow el cliente puede ver el estado y los datos de la transacción en la sección de pendientes. Cuando se cumpla la condición establecida, por ejemplo, la entrega de la compra, el agente entra en la vista de liberación y ejecuta una de las dos posibles acciones: liberación de fondos al comercio o reembolso al cliente. La DApp guía a la firma a MetaMask y el frontend se actualiza al escuchar el evento emitido por el Escrow. El contrato cambia de estado y su registro, automáticamente, pasa a estar en la vista de transacciones completadas.

Gracias a los eventos, la DApp mantiene lo que hay en la cadena y no hay duplicación del estado ya que las pantallas se refrescan con la emisión de estos.

3.2.2.2 Principios de UX aplicados

Durante el desarrollo del frontend se han seguido los siguientes principios:

- Simplicidad: el diseño tiene una simplicidad intencionada de manera que el pago para el cliente se reduce a una firma en el checkout.
- Transparencia: siempre se muestra el estado y los detalles relevantes de cada transacción.
- Seguridad: la DApp nunca maneja claves privadas ni fondos.
- Acciones limitadas por rol: solo el agente seleccionado para cada transacción tendrá acceso a la liberación o reembolso de los fondos.
- Separación de vistas por propósitos: esto facilita el aprendizaje y reduce la confusión.

En conjunto estos principios permiten una experiencia con gran confianza y fácil uso. Además, la transparencia y la trazabilidad refuerzan la credibilidad del sistema.

Capítulo 4 - Implementación

En este capítulo se describe cómo se materializó el diseño planteado en el capítulo anterior, detallando las decisiones técnicas que permitieron construir la plataforma de pago basada en contratos inteligentes. Para ello, en primer lugar, se presentan las herramientas seleccionadas y se justifican los criterios de elección a alternativas disponibles. A continuación, se expone la organización del repositorio con el foco en los principales componentes y la separación entre los contratos y la aplicación descentralizada. Posteriormente, se profundiza en la implementación de los contratos inteligentes desarrollados, explicando sus componentes y optimizaciones. Finalmente, se incluyen los principales casos de uso y ejemplos de interacción a través de la interfaz web, mostrando cómo la solución se integra en un flujo real de comercio electrónico.

4.1 Herramientas

Al comenzar la implementación del proyecto era necesario seleccionar un entorno de desarrollo que facilitara tanto la escritura como la prueba y el despliegue de contratos inteligentes. En el ecosistema Ethereum destacan dos opciones principales. Truffle[25] y Hardhat[26]. Ambas proporcionan un marco de trabajo para compilar contratos, desplegarlos en distintas redes (locales o públicas), ejecutar pruebas automatizadas y depurarlos, lo que resulta fundamental para asegurar la calidad y seguridad del código antes de ponerlo en producción.

Truffle fue uno de los primeros entornos populares, ampliamente utilizado por su sencillez y por integrar librerías de pruebas y migraciones de contratos. Sin embargo, su mantenimiento se ha ralentizado y presenta limitaciones en cuanto a la integración con herramientas modernas. Hardhat, en cambio, se ha consolidado en los últimos años como la opción más extendida. Ofrece un sistema de compilación modular, un entorno de depuración avanzado y, además, soporte nativo para TypeScript[27], lo que aporta mayor seguridad en el desarrollo.

Por estas razones, se optó por utilizar Hardhat como herramienta principal, ya que proporciona un entorno más actualizado, flexible y eficiente para el desarrollo de este trabajo.

Otra decisión clave fue la elección de la librería para interactuar con la blockchain desde la aplicación. En el ecosistema Ethereum destacan dos opciones ampliamente utilizadas: Web3.js[28] y Ethers.js[29]. Ambas permiten conectar la DApp con la red, firmar transacciones mediante carteras como MetaMask, desplegar contratos inteligentes y consultar datos almacenados en la cadena.

Web3.js fue la primera librería popular en este ámbito, con un amplio ecosistema y soporte, aunque su tamaño es considerable y su API tiende a ser más pesada y compleja de mantener. Ethers.js, por su parte, se diseñó con un enfoque más modular y minimalista: ofrece una librería más ligera, con documentación clara, mayor facilidad de integración y un énfasis en la seguridad de las operaciones.

Dado que uno de los objetivos de este trabajo es la eficiencia, se optó por Ethers.js. Su menor tamaño, mayor simplicidad y compatibilidad nativa con Hardhat lo convierten en la opción más adecuada. Además, proporciona un soporte sólido para la integración con MetaMask, lo que facilita la interacción de los usuarios finales con la plataforma.

En cuanto a la gestión de cuentas y claves, era necesario incorporar una herramienta que permitiera a los usuarios firmar transacciones e interactuar de manera segura con la DApp. La opción más extendida en el ecosistema Ethereum es MetaMask, una cartera digital que funciona como extensión del navegador y como aplicación móvil. MetaMask facilita la conexión directa con las aplicaciones descentralizadas, permite gestionar múltiples cuentas y ofrece a los usuarios control exclusivo sobre sus claves privadas. Su gran adopción y confianza por parte de la comunidad lo convierten en un estándar de facto para la interacción con contratos inteligentes desde entornos web, lo que justificó su elección para este proyecto.

En cuanto al lenguaje de programación de los contratos inteligentes, la elección natural fue Solidity, ya que se trata del lenguaje oficial de Ethereum y el más utilizado en la industria. Diseñado específicamente para escribir contratos ejecutables en la EVM,

cuenta con una amplia documentación, un ecosistema consolidado y un soporte activo por parte de la comunidad. Estas características lo convierten en la opción más adecuada para garantizar tanto la compatibilidad con la mantenibilidad del código desarrollado.

Para la gestión de dependencias y la ejecución de desarrollo con Hardhat se empleó Node Package Manager (npm)[30]. Esta herramienta, integrada en el ecosistema de Node.js[31], permite instalar y actualizar librerías de forma sencilla, así como automatizar la ejecución de scripts necesarios para compilar, desplegar y probar contratos inteligentes. Su elección se debe a la amplia compatibilidad con las demás herramientas utilizadas (Hardhat, Ethers.js, MetaMask) y a que constituye el gestor de paquetes estándar en la mayoría de los proyectos basado en JavaScript y TypeScript.

En cuanto al frontend se utilizó Astro[32] como framework de construcción de la interfaz, lo que permitió organizar la aplicación en componentes y optimizar el rendimiento gracias a su enfoque de *islands architecture*[33]. Sobre Astro se empleó Preact[34], una alternativa ligera a React[35] totalmente compatible con su ecosistema, que facilita la creación de componentes reutilizables con menos consumo de recursos. El diseño visual se implementó con TailwindCSS[36], un sistema de utilidades que permite construir rápidamente interfaces responsivas y adaptables a distintos dispositivos, un requisito esencial en el ámbito del comercio electrónico. Toda la lógica del frontend, al igual del backend, se desarrolló en TypeScript, lo que aportó seguridad y robustez al tipado de datos y a la interacción con los contratos inteligentes.

En su conjunto, esta selección de herramientas permitió cubrir todas las capas necesarias del sistema: desde la definición y despliegue de los contratos inteligentes en la blockchain hasta la interacción con el usuario final a través de la DApp, Hardhat, Solidity, Ethers.js y MetaMask proporcionaron la base para un desarrollo seguro y eficiente en la red de Ethereum, mientras que Astro, Preact, TailwindCSS y TypeScript hicieron posible construir una interfaz ligera, responsiva y fácilmente integrable en un e-commerce. La combinación de estos entornos asegura coherencia entre backend y frontend, y sienta las bases para un prototipo funcional abierto a evolucionar en futuras versiones.

4.2 Estructura del repositorio

El proyecto está en un repositorio en GitHub: https://github.com/HSilvaC99/TFG_final. La organización sigue una separación clara entre la parte de blockchain (contratos y lógica en Solidity) y el frontend (interfaz de usuario en la DApp). Esta división reduce errores de versión y facilita el mantenimiento.

Dentro del repositorio encontramos dos carpetas principales:

- blockchain/
  - contracts/
    - Escrow.sol
    - EscrowFactory.sol
    - MockPriceFeed.sol
- frontend/
  - src/
    - components/
      - Componentes visuales del frontend.
    - config/
      - Configuración de pago del checkout de la tienda.
    - contract/
      - ABIs y direcciones de los contratos.
    - data/
      - Productos de ejemplo de la tienda
    - layouts/
      - Plantilla común de todas las páginas de la web.
    - lib/
      - Scripts funcionales de la web.
    - pages/
      - Endpoints de la web.

Se mantienen los contratos y la DApp separados y se sincronizan con un script para reducir los errores de versión.

4.3 Blockchain

Esta parte del proyecto se centra en el desarrollo de los contratos inteligentes que constituyen el núcleo de la plataforma. Su objetivo principal es definir, desplegar y gestionar la lógica de negocio que permite realizar pagos condicionados de forma descentralizada y segura en el contexto del e-commerce. Para ello, se diseñaron tres contratos.

4.3.1 EscrowFactory

Actúa como punto único de creación de pagos en custodia. Centraliza la instanciación de contratos Escrow y emite eventos que facilitan el descubrimiento, trazabilidad y auditoría de cada transacción.

4.3.1.1 Actores e invariantes

Los actores principales del contrato son:

- *payer*: es el *msg.sender* de la función *createEscrow,* es decir, la dirección de quien ha invocado la función del contrato.
- *payee* y *escrowAgent*: son los parámetros de la llamada.

Además, cumple los siguientes invariantes:

- La factoría no retiene fondos al final de la ejecución ya que todo *msg.value* se pasa al constructor de Escrow.
- Cada invocación válida produce como máximo una instancia de Escrow anunciada por un evento.

4.3.1.2 Funciones principales

La función principal de este contrato es *createEscrow(address payee, address agent) external payable returns (address escrow).* Esta despliega una nueva instancia de Escrow para un pedido concreto.

Utiliza las siguientes validaciones:

- *msg.value* > *0* que evita escrows vacíos.

- *payee != address(0)* y *agent != address(0)* que comprueba que son direcciones válidas.

Realiza la construcción de la instancia de Escrow con el constructor *Escrow{value: msg.value}(msg.sender, _payee, _agent)* depositando los fondos recibidos en este.

4.3.1.3 Controles de seguridad

Se han aplicado los siguientes controles de seguridad:

- Validación de parámetros para hacer un revert temprano ante entradas inválidas. Revert lo que hace es revertir los cambios realizados por el contrato hasta ese punto.
- Separación de responsabilidades: la factoría no tiene funciones para tratar los fondos, toda la lógica de custodia se queda en Escrow.
- No permite ataques de reentrada ya que no transfiere Ether a contratos externos solo a la instancia de Escrow creada por sí mismo.

4.3.1.4 Eficiencia en gas

Aplicamos los siguientes principios para reducir el consumo de gas:

- No tiene almacenamiento agregado por lo que evita escrituras costosas en este.
- Hace una sola transacción uniendo la creación del Escrow y el traspaso de fondos.
- Se aplica la función *external* para la firma que es más eficiente que *public*.

4.3.1.5 Eventos y observabilidad

Crea el evento *EscrowCreated* indexando la información necesaria (la dirección del contrato y el pagador) para que el frontend reconstruya la lista de escrows sin depender de estructuras en la cadena.

4.3.2 Escrow

Es el contrato de custodia de fondos por transacción. Modela el ciclo completo de un pago condicionado: depósito, liberación al vendedor o reembolso al pagador, bajo la supervisión de un agente neutral.

4.3.2.1 Actores e invariantes

Los atributos principales del contrato son:

- *payer*: quien deposita los fondos (se fija como msg.sender en el constructor).
- *payee*: destinatario de los fondos (comercio).
- *escrowAgent*: tercero neutral autorizado a resolver.
- *amount*: importe depositado.
- *currentState*: *{AWAITING_RELEASE = 0, COMPLETE = 1, REFUNDED = 2}*.

Además, cumple los siguientes invariantes:

- Solo puede existir un flujo de salida de fondos (liberación o reembolso).
- *amount* es 0 tras finalizar (para impedir dobles gastos).

4.3.2.2 Ciclo de vida

El ciclo de vida del contrato se desarrolla de la siguiente manera:

1. Despliegue (depósito): el *payer* llama al constructor con *msg.value = amount*.
    a. Validaciones: *amount > 0, payee/escrowAgent ≠ address(0)*.
    b. Evento: *FundsDeposited(uint256 amount)*.
2. Resolución por el agente**:**
    a. *releaseFunds()*: transfiere *amount* a *payee*, cambia estado a *COMPLETE* y emite *FundsReleased(amount)*.
    b. *refund()*: transfiere amount a payer, cambia estado a *REFUNDED* y emite *RefundIssued(amount)*.
    c. Restricción de acceso: solo *escrowAgent* puede invocar estas funciones.

3. Cierre: el contrato queda sin fondos (*amount* = *0*) y en estado terminal.

4.3.2.3 Controles de seguridad

Se han aplicado los siguientes controles de seguridad:

- Patrón CEI (Checks-Effects-Interactions):
  - Checks: verificar estado y permisos.
  - Effects: actualizar estado y poner amount a 0.
  - Interactions: realizar la transferencia al final.

  Este patrón mitiga riesgos de reentrancy y condiciones de carrera. En la figura 4-1 vemos como se aplica en el código.

- Estados terminales: llamadas repetidas a releaseFunds/refund se rechazan con error de estado inválido.
- Autorización estricta: solo escrowAgent ejecuta resoluciones, evitando conflictos entre partes.

```
function releaseFunds()
    external
    onlyEscrowAgent
    inState(EscrowState.AWAITING_RELEASE)
{
    // Evitamos múltiples escrituras al almacenamiento
    uint256 paymentAmount = amount;
    amount = 0; // Actualizamos antes de la interacción externa para prevenir reentrancia
    currentState = EscrowState.COMPLETE;

    (bool success, ) = payee.call{value: paymentAmount}("");
    require(success, "Transfer to payee failed");

    emit FundsReleased(paymentAmount);
}
```

Figura 4-1. Fragmento de código de contrato Escrow.

4.3.2.4 Eficiencia en gas

Aplicamos los siguientes principios para reducir el consumo de gas:

- Una única escritura crítica al cerrar (poner *amount = 0* y actualizar *currentState*), minimizando SSTORE.
- Sin bucles y uso de constantes cuando aplica.

- Variables que no cambian marcadas como *immutable/constant* cuando procede.

4.3.2.5 Eventos y observabilidad

FundsDeposited, FundsReleased, RefundIssued proporcionan telemetría on-chain para que el frontend liste estados sin depender de servicios externos.

4.3.3 MockPriceFeed

Este contrato ha sido añadido con intención de integrar a futuro un oráculo que calcule en tiempo real la equivalencia de ETH en EUR. Se utiliza como herramienta para hacer un cálculo estático y simular esto.

4.4 Frontend

El frontend constituye la capa de interacción entre el usuario y la cadena de bloques. Su objetivo es ofrecer una experiencia sencilla y similar a la de otros sistemas de pago, ocultando la complejidad técnica de los contratos inteligentes. Para ello se desarrolló una aplicación web descentralizada (DApp) que conecta con MetaMask y permite gestionar todo el ciclo de vida de las transacciones: creación, custodia, resolución y visualización de histórico.

4.4.1 Casos de uso

A continuación, detallaremos los casos de uso de la aplicación, explicando los flujos de interacción y cómo debe utilizarla el usuario.

4.4.1.1 Barra de navegación

La aplicación cuenta con una barra de navegación común a todas las vistas, que permite acceder a las diferentes secciones del sistema: Enviar, Pendientes, Liberar, Transacciones y Tienda. Esta barra constituye el punto de entrada a cada una de las funcionalidades y se muestra en la Figura 4-2.

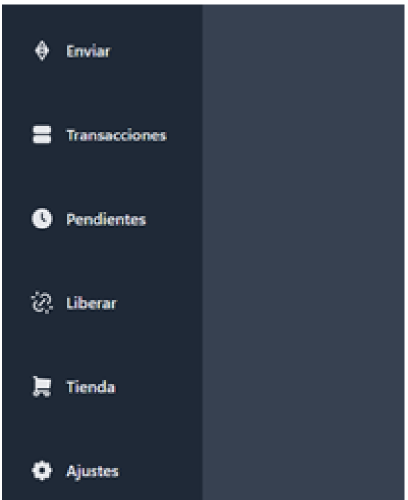

Figura 4-2. Barra de navegación.

4.4.1.2 Enviar

En esta vista el cliente puede iniciar un pago, conectando su cuenta de MetaMask y especificando la dirección del comercio, la del agente que actuará como árbitro y la cantidad a transferir en Ether. Al confirmar la operación, se invoca la función *createEscrow* del contrato factoría y se genera una nueva instancia de *escrow*. Las Figuras 4-3 y 4-4 muestran el formulario de envío y la conexión de cuentas de prueba. Una vez completado el formulario y firmada la transacción en MetaMask (Figuras 4-5 y 4-6), se despliega el contrato y los fondos quedan custodiados. El mensaje de confirmación con la dirección del contrato se observa en la Figura 4-7.

Figura 4-3. Página Enviar.

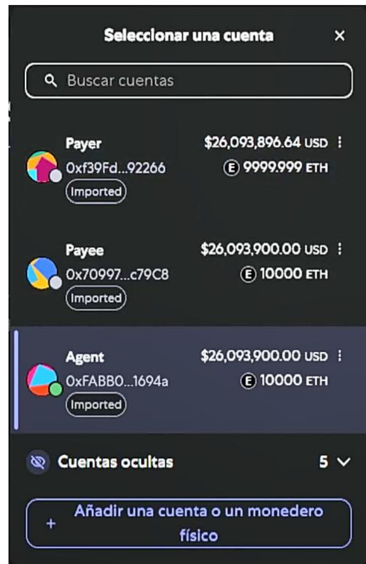

Figura 4-4. Cuentas importadas a MetaMask.

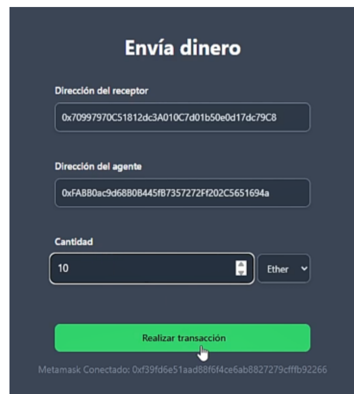

Figura 4-5. Envío de dinero.

Figura 4-6. Confirmación de envío con MetaMask.

Figura 4-7. Mensaje de confirmación de envío.

4.4.1.3 Pendientes

La sección de pendientes está destinada al agente neutral. Desde esta vista se listan las transacciones en espera de resolución, mostrando información como direcciones abreviadas de los participantes, el balance en custodia y la fecha de creación del contrato. Un ejemplo puede verse en la Figura 4-8.

Figura 4-8. Página transacciones pendientes.

4.4.1.4 Liberar

En esta vista, el agente puede decidir entre completar la transacción liberando los fondos al comercio o reembolsar al cliente si no se cumplen las condiciones pactadas. Estas acciones corresponden a las funciones *releaseFunds* y *refund* del contrato *Escrow*. La interfaz asociada se muestra en la Figura 4-9.

Figura 4-9. Página liberar.

4.4.1.5 Transacciones

El historial de operaciones resueltas se recoge en la sección de transacciones. En ella, cada usuario puede consultar los pagos en los que ha participado, con detalle de dirección del contrato, rol ejercido (pagador o receptor), cantidad transferida y fecha. La Figura 4-10 muestra un ejemplo de transacción completada, mientras que la Figura 4-11 ilustra el caso de un reembolso.

Figura 4-10. Página transacciones completadas.

Figura 4-11. Ejemplo transacción reembolsada y otra completada respectivamente.

4.4.1.6 Tienda integrada

Para mostrar la aplicabilidad práctica del sistema, se desarrolló un prototipo de tienda en línea. Desde esta sección es posible seleccionar productos, añadirlos al carrito y efectuar el pago mediante MetaMask, integrando así la lógica de *escrow* en un flujo de compra real. La Figura 4-12 enseña la vista de la tienda y la Figura 4-13 el proceso de pago integrado.

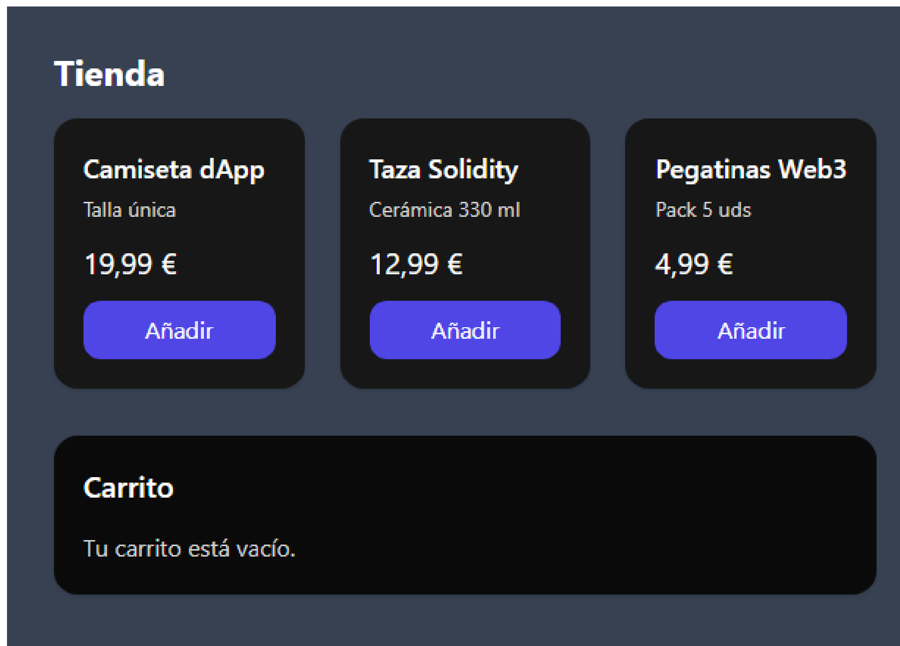

Figura 4-12. Ejemplo comercio electrónico.

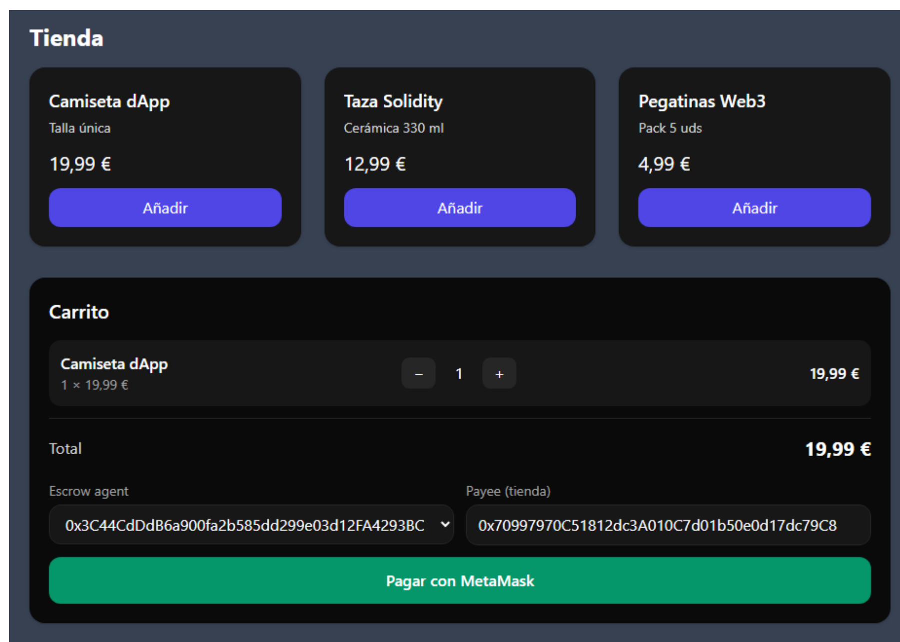

Figura 4-13. Pago integrado en tienda.

4.4.1.7 Adaptabilidad a dispositivos

Finalmente, la aplicación se diseñó como *responsive*, de forma que puede adaptarse a distintos dispositivos como tabletas o móviles, lo que resulta fundamental en un contexto de comercio electrónico.

4.5 Despliegue del proyecto

Para poner en funcionamiento la plataforma ha sido necesario realizar el despliegue tanto de la red de pruebas como de los contratos inteligentes y su integración con el frontend.

En primer lugar, se configuró un nodo local de Hardhat que simula una blockchain de desarrollo. Al iniciarlo se generan varias cuentas de prueba con sus claves privadas y saldos iniciales de Ether, lo que permite reproducir escenarios realistas de pagadores, comercios y agentes. Este proceso se observa en las Figuras 4-14, 4-15, 4-16 y 4-17.

```
# Terminal 1 (Servidor blockchain)
cd ~/Documents/TFG/blockchain && npm install && npx hardhat node
```

Figura 4-14. Instalación nodo.

```
$ cd blockchain && npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.
```

Figura 4-15. Despliegue de red de blockchain.

```
Account #12: 0xFABB0ac9d68B0B445fB7357272Ff202C5651694a (10000 ETH)
Private Key: 0xa267530f49f8280200edf313ee7af6b827f2a8bce2897751d06a843f644967b1

Account #13: 0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec (10000 ETH)
Private Key: 0x47c99abed3324a2707c28affff1267e45918ec8c3f20b8aa892e8b065d2942dd

Account #14: 0xdF3e18d64BC6A983f673Ab319CCaE4f1a57C7097 (10000 ETH)
Private Key: 0xc526ee95bf44d8fc405a158bb884d9d1238d99f0612e9f33d006bb0789009aaa

Account #15: 0xcd3B766CCDd6AE721141F452C550Ca635964ce71 (10000 ETH)
Private Key: 0x8166f546bab6da521a8369cab06c5d2b9e46670292d85c875ee9ec20e84ffb61

Account #16: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30 (10000 ETH)
Private Key: 0xea6c44ac03bff858b476bba40716402b03e41b8e97e276d1baec7c37d42484a0

Account #17: 0xbDA5747bFD65F08deb54cb465eB87D40e51B197E (10000 ETH)
Private Key: 0x689af8efa8c651a91ad287602527f3af2fe9f6501a7ac4b061667b5a93e037fd

Account #18: 0xdD2FD4581271e230360230F9337D5c0430Bf44C0 (10000 ETH)
Private Key: 0xde9be858da4a475276426320d5e9262ecfc3ba460bfac56360bfa6c4c28b4ee0

Account #19: 0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199 (10000 ETH)
Private Key: 0xdf57089febbacf7ba0bc227dafbffa9fc08a93fdc68e1e42411a14efcf23656e
```

Figura 4-16. Cuentas de prueba resultantes.

```
astro v5.13.2 ready in 833 ms

┃ Local    http://localhost:4321/
┃ Network  use --host to expose
```

Figura 4-17. Red local creada.

Posteriormente, se desplegó el contrato *EscrowFactory* en la red local. La dirección de este contrato se guardó en el código del frontend para poder invocar desde allí la función *createEscrow* y crear nuevas instancias de custodia. Las instrucciones de despliegue y el registro de la dirección se muestran en las Figuras 4-18 y 4-19.

```
# Terminal 2 (Servidor web)
cd ~/Documents/TFG/blockchain/ && npx hardhat run ignition/modules/deployFactory.ts --network localhost | tail -1 | awk '{printf "const FACTORY_ADDRESS = \"%s\";\n\nexport default FACTORY_ADDRESS;", $NF}' > ../frontend/src/contract/factoryAddress.ts && npx hardhat run ignition/modules/deployMockPriceFeed.ts --network localhost | tail -1 | awk '{printf "const PRICE_FEED_ADDRESS = \"%s\";\n\nexport default PRICE_FEED_ADDRESS;", $NF}' > ../frontend/src/contract/priceFeedAddress.ts && cd ~/Documents/TFG/frontend/ && pnpm install && pnpm dev
```

Figura 4-18. Comando para despliegue factoría de contratos e inclusión en el frontend.

```
const FACTORY_ADDRESS = "0x5FbDB2315678afecb367f032d93F642f64180aa3";

export default FACTORY_ADDRESS;
```

Figura 4-19. Dirección del contrato incluido en frontend/src/contract/factoryAddress.

Con la red y los contratos listos, el frontend se conectó a través de MetaMask, lo que permitió a los usuarios firmar transacciones y operar con la plataforma (4-20). A partir de este momento fue posible probar el ciclo completo: creación de un pago, resolución por parte del agente y consulta de transacciones históricas.

Figura 4-20. Inicio sesión en MetaMask.

Este flujo de despliegue asegura que la plataforma pueda ejecutarse de extremo a extremo en un entorno controlado, replicando con fidelidad cómo se comportaría en un escenario real de comercio electrónico sobre Ethereum.

Capítulo 5 - Funcionalidades

En este capítulo se analizan las propiedades principales del prototipo desarrollado, centrándose en dos aspectos clave: la seguridad y la eficiencia en el uso de gas. El objetivo es mostrar cómo las decisiones de diseño e implementación descritas en el capítulo anterior se traducen en un sistema robusto frente a ataques comunes y, al mismo tiempo, optimizado para reducir los costes de ejecución en la blockchain de Ethereum. Para ello, se revisan las medidas de protección incorporadas en los contratos inteligentes y se evalúa su consumo de gas mediante herramientas especializadas, lo que permite valorar la viabilidad práctica de la solución propuesta.

5.1 Seguridad

La seguridad es un aspecto crítico en sistemas de pago, ya que una vulnerabilidad podría comprometer los fondos depositados por los usuarios. Para mitigar los ataques más comunes en Ethereum, se incorporaron varias medidas en la implementación del contrato Escrow:

- Patrón Checks-Effects-Interactions (CEI): antes de transferir fondos se realizan todas las comprobaciones necesarias (rol del agente y validez del estado) y se actualiza el estado del contrato a cerrado (amount = 0, currentState cambiado). Solo después se efectúa la interacción externa con la blockchain. Este patrón reduce el riesgo de ataques de reentrada, habituales en contratos que gestionan fondos.
- Restricción de funciones críticas: solo el agente neutral designado en el despliegue del Escrow puede invocar releaseFunds o refund. De este modo se evita que cliente o comercio manipulen unilateralmente el desenlace de la transacción.
- Estados terminales: una vez ejecutada una transición (liberación o reembolso), el contrato queda cerrado e impide cualquier operación adicional. Esto garantiza que no pueda repetirse la acción ni reabrirse el flujo de fondos.

- Eventos para auditoría: cada transición emite un evento (FundsDeposited, FundsReleased, RefundIssued) que permite reconstruir el historial en la cadena de bloques, facilitando la transparencia y el seguimiento de cada operación.
- Decisiones de diseño adicionales: se han evitado bucles de longitud variable y se han utilizado variables constantes e inmutables cuando ha sido posible, reduciendo la superficie de ataque y el coste de ejecución asociado a cada operación.

5.2 Gas y eficiencia

En el contexto del e-commerce cada instrucción se traduce en dinero y, posiblemente, en fricción con el usuario por ello hay que tener muy en cuenta el gas que consume el contrato. En consecuencia, hay que intentar seguir buenas prácticas como minimizar las escrituras en almacenamiento y no incluirlas en bucles, así como agrupar las transferencias, siempre que sea posible.

Para reducir el uso de gas se han aplicado distintas prácticas como:

- Utilización de *immutable* en los roles de *payer*, *payee* y *escrowAgent*: al declararlos así no ocupan storage, se incluyen directamente en el bycode al desplegarse. Esto hace que no se tenga que realizar una lectura de almacenamiento en las llamadas.
- Emisión de eventos vs registro on-chain: la DApp reconstruye el listado de eventos creados con su información leyendo los eventos. Esto evita realizar escrituras en la cadena que son muy costosas.
- Indexación en eventos limitada: al emitir eventos en la EVM el log puede tener parámetros indexados o no. Estos son útiles para filtrar la información ya que permiten hacerlo desde el nodo y sin necesidad de codificación. Pero cuestan gas extra ya que aumenta el tamaño del log. Por ello solo se indexan los parámetros que se utilizan para filtrar posteriormente.

Para conseguir esto también se pueden utilizar herramientas de medición, análisis y optimización. Estas herramientas son interesantes ya que van más allá que el compilador *solc* de Solidity. Este solo hace un análisis estático y estima un consumo de gas infinito en funciones más complejas o con dependencias de otros contratos, lo cual

no proporciona la información que buscamos. Algunas posibilidades son Gastap[37] que, que realiza un análisis estático del bytecode de los contratos y estima cotas superiores para el gas requerido en cada función o GASPER[18] que localiza automáticamente patrones costosos en el *bytecode*.

El consumo de gas en Ethereum determina el coste económico de ejecutar las funciones de un contrato inteligente, por lo que resulta fundamental, no solo aplicar buenas prácticas, sino analizar su comportamiento para garantizar que el sistema sea viable en un contexto de comercio electrónico. Con este objetivo, aplicamos la herramienta Gastap.

La herramienta se ejecutó sobre los contratos *Escrow*, *EscrowFactory* y *MockPriceFeed*. Los resultados generados se pueden encontrar en el apéndice A. Estos permiten diferenciar entre:

- Funciones de lectura y getters, como *currentState()*, *getBalance()*, *payer()*, *payee()*, *escrowAgent()* o *amount()*, que presentan un consumo constante bajo (entre 342 y 2.540 unidades de gas). Estos valores son esperables, dado que solo acceden a variables de estado sin modificar la cadena.
- Funciones de actualización de estado, como refund() y releaseFunds(), que registran un consumo aproximado de 74.000 unidades de gas. Este coste se debe a que implican escritura en almacenamiento (poner *amount = 0* y actualizar el estado), además de una transferencia de Ether. Aunque mayor que el de los getters, sigue siendo un valor contenido y comparable al de otros contratos escrow analizados en la literatura. En los resultados vemos que estas funciones se marcan como que sus costes dependen del número de llamadas externas que se realicen. En consecuencia, parece un coste variable, pero por la naturaleza del contrato podemos asegurar que se le llama una sola vez y es un coste constante.
- Creación de nuevas instancias, mediante la función *createEscrow(address, address)* de *EscrowFactory*, cuyo coste asciende a unas 36.652 unidades de gas, lo que confirma que la mayor parte del gasto

está en el despliegue inicial, mientras que las operaciones posteriores se mantienen en cotas bajas y predecibles. Este es un coste esperable para una factoría, pero debe advertirse al usuario.

- Funciones auxiliares del oráculo simulado (*MockPriceFeed*), como setRate(int256) (~23.687 gas), que implican escritura en almacenamiento, y latestAnswer() (~2.460 gas), que son consultas más ligeras. Las consultas son baratas y, por tanto, ideales para un uso frecuente. Pero la función setRate tiene un coste demasiado elevado y no es viable en la realidad y a gran escala. Como es una simulación no es crítico, pero se debe buscar una solución alternativa a futuro.

En conjunto, el análisis confirma que los contratos desarrollados son eficientes y predecibles en su consumo de gas lo cual es ideal para poder dar al usuario una predicción correcta del gas necesario y no incurrir en excepciones de out-of-gas. No se detectaron dependencias paramétricas ni bucles que pudieran hacer crecer el coste en función de la entrada, lo que garantiza la escalabilidad de la solución en escenarios reales de e-commerce. El único coste crítico que revela este análisis que no es justificable ni coherente con los objetivos en un escenario real es el del oráculo, pero al ser una herramienta auxiliar con vistas de mejora no tiene un impacto relevante.

Capítulo 6 - Trabajo relacionado

6.1 Comercio electrónico

En esta sección analizamos plataformas que implementan pagos tipo escrow mediante contratos inteligentes, priorizando comercio electrónico. Nos centramos en ver qué problema resuelven, cómo está diseñado el escrow on-chain y por qué han funcionado o no.

Shopify[38], en colaboración con Coinbase[39], lanzó este junio el Commerce Payments Protocol[40], una iniciativa para habilitar pagos con criptomonedas en el comercio electrónico a gran escala. Este introduce un contrato inteligente de escrow en la cadena Base[41], una cadena de bloques de L2 sobre Ethereum, desarrollada por Coinbase, que proporciona transacciones rápidas y con menores costes, manteniendo la compatibilidad total con la EVM, para manejar pagos con USDC en tiendas de Shopify. La propuesta es abierta, auditable y orientada a minoristas a gran escala. Está implementado de forma descentralizada lo cual implica que nadie (ni siquiera Shopify o Coinbase) puede desactivar o alterar el contrato una vez desplegado, lo que brinda máximas garantías de confianza. Al aprovechar las ventajas de la red Base, como la mayor rapidez de confirmación de transacciones, los bajos costes y alcance global de las criptomonedas, Shopify y Coinbase han logrado un avance significativo, aunque su adopción masiva dependerá de la comodidad de usuarios y comerciantes al usar cripto en entornos de compra cotidianos. Como es una mejora muy reciente todavía no podemos evaluar cómo de aceptado y valorado será por el público, pero es una gran propuesta. En comparación con este trabajo, el TFG plantea un prototipo más ligero y auditable, pensado para integrarse en tiendas de pequeño y mediano tamaño, mientras que la propuesta de Shopify está orientada a grandes minoristas con infraestructura global.

Origin Protocol[42]es una plataforma fundada en 2017 cuya visión es crear mercados *peer-to-peer* descentralizados para bienes y servicios. Creó un marketplace descentralizado con pagos escrow y contratos de Ethereum: el comprador deposita en el contrato el importe correspondiente a la compra, el vendedor entrega el bien o

servicio acordado y, tras confirmación del comprador de que la entrega se ha realizado correctamente, el pago se libera al vendedor. Si bien el enfoque fue pionero y técnicamente sólido (contratos, datos abiertos y flujo de compra sin backend), la adopción masiva en comercio generalista fue limitada (por fricción tecnológica y elevados costes de gas) y el proyecto pivotó hacia otros productos con mayor tracción (NFT/DeFi)[43]. En relación con este TFG, la plataforma desarrollada aborda de forma explícita la eficiencia en gas y la simplicidad de uso, aprendiendo de las limitaciones que Origin Protocol encontró en su adopción generalista.

Boson Protocol[44] implementa un escrow de dos depósitos (comprador y vendedor) y emite un NFT canjeable (que representa el derecho a recibir el bien físico); los fondos quedan bloqueados en escrow onchain hasta resolución o disputa. Se ha aplicado en experiencias como Decentraland[45] y usa Ethereum/L2. Sin embargo, su adopción ha estado limitada en parte por la fricción tecnológica, por ejemplo, operar en Ethereum implicaba tasas de gas altas para los usuarios. De hecho, la plataforma tuvo que incentivar a los usuarios compensando costes de gas – cada compra exitosa recompensaba al comprador con tokens $BOSON para mitigar las tarifas elevadas de la red Ethereum. Es un proyecto activo e innovador ya que presenta un modelo de escrow on-chain único (NFTs con fondos en custodia y depósitos) para asegurar transacciones de comercio electrónico sin terceros de confianza. A diferencia de Boson, que introduce complejidad adicional mediante NFTs y depósitos dobles, el TFG apuesta por un escrow simple y económico que resulta más directo de integrar en un comercio electrónico.

DMarket[46] es un mercado digital fundado en 2017 orientado a la compraventa de bienes virtuales, principalmente objetos de videojuegos (aspectos, objetos, cuentas). Utiliza blockchain para garantizar la trazabilidad y propiedad de los activos, y los contratos inteligentes para asegurar que el comprador deposita el importe antes de la transferencia del activo digital. Aunque logró cierta tracción en el sector *gaming*, aplicación está muy limitada a este nicho y no ha tenido impacto en el e-commerce generalista. En este sentido, la propuesta del TFG busca ser más generalista, aplicable a cualquier comercio electrónico y no únicamente a un mercado de nicho como el de los bienes digitales de videojuegos.

OpenBaazar[47] fue uno de los primeros marketplaces descentralizado *peer-to-peer*, lanzado en 2016, que permitía comprar y vender bienes digitales y físicos utilizando criptomonedas como medio de pago. Incorporaba un sistema de *escrow* mediante moderadores, que actuaban como árbitros neutrales en caso de disputa. A pesar de ser pionero, la falta de adopción masiva, la complejidad técnica para usuarios no expertos y la ausencia de un modelo económico sostenible provocaron su cierre en 2021. El TFG, en cambio, elimina la dependencia de moderadores individuales y automatiza la resolución de disputas a través del contrato inteligente, lo que reduce la complejidad y aporta mayor transparencia.

Zora[48] es una plataforma creada en 2020 con foco en los NFTs, que funcionan como un protocolo abierto para la compraventa de activos digitales. Su innovación radica en permitir mercados programables y dinámicos, en los que los contratos inteligentes gestionan tanto la propiedad como las condiciones de compraventa de los tokens. Aunque ha conseguido relevancia en el ámbito cultural y artístico digital, su orientación está restringida al mercado de NFTs, lo que limita su aplicación directa en comercio electrónico tradicional. Frente a esta orientación hacia el mercado cultural y artístico digital, el TFG se centra en bienes y servicios propios del comercio electrónico, con un diseño orientado a integrarse en flujos de compra reales.

Bitify[49] fue un *marketplace* fundado en 2013 que combinaba el uso de criptomonedas (Bitcoin, Litecoin, etc.) como medio de pago con un sistema *escrow* gestionado por la propia plataforma. Los compradores enviaban fondos a una custodia que se liberaba al vendedor solo tras confirmarse la recepción del producto o servicio. Esta combinación de cripto y escrow tradicional permitió que la plataforma operara durante años. Sin embargo, el servicio fue cerrado en enero de 2025 sin aviso previo, lo que generó descontento entre los usuarios. Mientras Bitify mantenía un modelo híbrido de custodia centralizada, el TFG propone un escrow completamente descentralizado y auditable, eliminando la necesidad de confiar en la propia plataforma.

6.2 Comercio tradicional

En el ámbito del comercio no electrónico también existen ejemplos significativos de adopción de soluciones basadas en Ethereum para mejorar la transparencia y la seguridad de los sistemas:

Carrefour[50] ha utilizado blockchain para la trazabilidad alimentaria, permitiendo a los consumidores escanear un código QR y verificar la cadena de bloques de todo el recorrido de un producto (como pollos de corral o leche fresca) desde su origen hasta el punto de venta. Esto refuerza la confianza y la transparencia en la cadena de suministro. En contraste, este TFG se centra en el ámbito de los pagos en escrow, no en la trazabilidad de productos, pero comparte el mismo objetivo de aumentar la confianza del consumidor mediante blockchain.

Maersk[51] e IBM[52] desarrollaron la plataforma TradeLens[53], basada en blockchain, para la gestión de contenedores y logística marítima. Aunque finalmente fue cerrada en 2022 por falta de adopción suficiente en la industria, demostró el potencial de la tecnología para digitalizar documentos de transporte, reducir tiempos de despacho y mejorar la trazabilidad en operaciones internacionales. El TFG, aunque en otro dominio, sigue la misma línea de aplicar blockchain para mejorar procesos críticos, en este caso en los pagos electrónicos, con mayor facilidad de adopción en escenarios de e-commerce.

Tracr[54], impulsada por De Beers[55], es una plataforma que emplea blockchain para rastrear diamantes desde la mina hasta el cliente final, asegurando la autenticidad del producto y evitando la comercialización de diamantes de origen ilícito. El TFG, en cambio, propone un sistema más generalista, aplicable a cualquier tienda online, aunque con el mismo principio de reforzar la confianza de los usuarios en la validez de la transacción.

OpenSC[56] es una iniciativa de WWF[57] y BCG[58] que utiliza blockchain para certificar la sostenibilidad y origen ético de productos como alimentos y materias primas. Permite a los consumidores verificar en tiempo real la procedencia y la huella ambiental de los bienes que adquieren. La propuesta del TFG difiere en su enfoque, ya que busca integrar

pagos escrow en el comercio electrónico, pero comparte con OpenSC el objetivo de reforzar la transparencia y la confianza en las operaciones.

En definitiva, los casos revisados ponen de manifiesto que la aplicación de blockchain y contratos inteligentes no se limita al e-commerce. Existen proyectos muy variados, desde soluciones de pago y mercados descentralizados hasta iniciativas en ámbitos como la trazabilidad alimentaria, la logística internacional o la certificación de productos. La mayoría de estas experiencias han tenido que enfrentarse a problemas comunes, como los costes de uso, las barreras tecnológicas o la dificultad para conseguir una adopción generalizada. Aun así, muestran que Ethereum puede ser una base sólida para aumentar la transparencia y la seguridad de las transacciones de distinta naturaleza. Sobre este escenario se sitúa la propuesta de este trabajo, que se centra en el comercio electrónico y plantea un prototipo práctico pensado para integrarse en tiendas online.

Capítulo 7 - Conclusiones y trabajo futuro

7.1 Conclusiones

En este Trabajo Fin de Grado se ha diseñado e implementado un prototipo funcional de plataforma de pago con contratos inteligentes en Ethereum, orientado a su integración en el comercio electrónico mediante un esquema de escrow con agente neutral. La solución combina una DApp conectada a MetaMask con una factoría de contratos y contratos Escrow independientes por pedido, priorizando la seguridad, la transparencia y la eficiencia en el consumo de gas. El enfoque de reconstrucción del histórico a partir de eventos on-chain evita almacenar información innecesaria, hace el coste por operación más predecible y preserva la privacidad del usuario. En materia de seguridad, el contrato Escrow aplica el patrón Checks-Effects-Interactions, estados terminales y autorización estricta por rol, lo que mitiga riesgos como la reentrancy y asegura que los fondos solo pueden liberarse o reembolsarse en una única dirección de flujo. En eficiencia, se han evitado bucles, se han utilizado variables constantes e inmutables cuando ha sido posible y se ha optado por un uso intensivo de eventos para reducir escrituras en almacenamiento.

Al revisar los objetivos planteados en el capítulo inicial, se puede afirmar que el objetivo general de diseñar, implementar y evaluar un prototipo seguro, eficiente y fácil de integrar se ha cumplido. El sistema resultante incluye una DApp con MetaMask y una arquitectura en capas que demuestra su integración en un flujo de compra y la trazabilidad de las operaciones a través de eventos.

En cuanto a los objetivos específicos, el primero, relativo a la investigación del estado del arte y las opciones tecnológicas, se ha alcanzado con una revisión de plataformas y soluciones similares, justificando así las decisiones adoptadas. En este punto hemos encontrado dificultades al escribir la sección de manera didáctica y comprensiva ya que no conseguíamos organizarla bien, pero con la orientación de nuestros tutores conseguimos orientarla mucho mejor.

El diseño de la arquitectura se concretó en tres capas y en un modelo de actores con patrón factoría para gestionar múltiples pagos. Los contratos EscrowFactory y

Escrow se implementaron con invariantes claros, eventos y funciones de custodia, liberación y reembolso, cumpliendo el objetivo de implementación. La interfaz gráfica cubre las vistas necesarias para enviar, revisar pendientes, liberar fondos, consultar transacciones y probar la compra en una tienda integrada, demostrando la integración entre interfaz y contratos. También se ha validado la integración mostrando feedback al usuario y reconstruyendo el histórico con eventos. En este punto encontramos dificultades en el desarrollo de un gráfico comprensivo pero simple de la arquitectura. Tras muchas pruebas y dibujos, encontramos una manera de explicar la arquitectura equilibrando la simplicidad para facilitar la comprensión con la inclusión de los componentes y flujos clave.

En cuanto a la seguridad y al uso de gas, se han introducido mecanismos de protección en el diseño y se ha realizado un análisis exhaustivo con Gastap que confirma que las operaciones críticas mantienen un consumo de gas acotado y predecible: las funciones getter muestran costes muy bajos, las funciones de liberación y reembolso rondan las setenta y cuatro mil unidades, y la creación de escrows se sitúa en torno a treinta y seis mil. Además, se ha investigado el panorama de herramientas y plataformas relacionadas, y se han planteado líneas de trabajo futuro.

Durante el desarrollo encontramos otras dificultades como en la integración de un oráculo para la conversión de divisas: no encontramos uno que pudiéramos usar en nuestro modelo. Esto lo resolvimos mediante un oráculo estático que simula uno real.

Otro gran reto fue integrar correctamente la lógica de contratos con la interfaz gráfica, en especial en lo relativo a la captura y actualización de estados en tiempo real a partir de eventos, ya que, a pesar de que la documentación de Ethers.js está bastante completa, en el momento del desarrollo decidimos transicionar de la versión 4 a la 5 la cual presentaba lagunas y ejemplos incompletos en su documentación, lo que añadió dificultad al proceso. Esto nos obligó a dedicar tiempo adicional a la investigación y la realización de pruebas, tanto para decidir cómo almacenar la información de forma eficiente sin incurrir en un alto coste, como para aprender a trabajar adecuadamente con los eventos emitidos por los contratos.

En resumen, el proyecto materializa un enfoque seguro y eficiente para pagos condicionados en comercio electrónico mediante contratos inteligentes de escrow,

con una experiencia de usuario sencilla y trazabilidad basada en eventos. La evaluación confirma la viabilidad técnica del modelo y demuestra que los objetivos planteados se han alcanzado en su totalidad, aportando una base sólida para evolucionar hacia entornos de producción más escalables, con menores costes y mayor robustez en la confianza de los actores involucrados.

7.2 Trabajo futuro

Este prototipo tiene limitaciones sobre las que se podría trabajar a futuro. En general, podríamos desplegarlo en redes L2 para abaratar las comisiones, pero manteniendo la seguridad. También podríamos incorporar incentivos para el agente para que se reduzcan posibles bloqueos. Pero a continuación exponemos las mejoras prioritarias y que creamos que aportarían mayor valor a la plataforma.

Lo primero que habría que integrar sería un oráculo real para la conversión de euros a Ether. Esto es porque el actual es estático y una simulación por lo que no representa la realidad y supone costes altos.

Por otro lado, lo que más aportaría a este prototipo sería integrar oráculos que automaticen la señal del agente. La idea es que este mande la señal al confirmar la entrega de manera automática y sin tener que entrar en la plataforma a autorizar la liberación cada vez que se realice un pago.

Otra posible expansión de la plataforma es aceptar otro tipo de condiciones para liberación de los fondos. Por ejemplo, la caducidad del pago si no llega el paquete a tiempo y que se cobre el coste del uso de la plataforma al vendedor y se devuelva al cliente. Pero manteniendo la condición de entrega del paquete ya que consideramos que es la condición más adecuada para el e-commerce.

Por último, sería interesante añadir un contrato de escrow dual para los bienes digitales. Este tipo de contrato funciona de manera que el comprador deposita los fondos en el contrato y el vendedor el bien digital. De esta manera se produciría el intercambio sin necesidad de un agente validador.

Introduction

This chapter aims to contextualize the work carried out, outlining the motivations that drove its development, the objectives pursued, and the planning followed to achieve them. Section 1.1 presents the motivation for the project and the need to explore blockchain-based solutions in the field of e-commerce. Section 1.2 sets out the general and specific objectives that guide this Final Degree Project. Finally, Section 1.3 describes the work plan followed, from the preliminary study to the design, implementation, and evaluation of the proposed platform.

Motivation

At present, two trends converge to motivate this work: on one hand, the progress of blockchain technologies, increasingly used in fields such as healthcare, where they ensure the traceability of medical records or the control of pharmaceutical supply chains or finance with applications in international payments, asset management, and settlement systems; and, on the other hand, the growing need for decentralized, transparent, and more secure means of payment, with settlement logics different from current systems.

As trust in decentralized technologies grows, so too does friction with traditional financial institutions. High fees, lack of transparency, and economic instability are driving the search for alternatives to these institutions. This, together with the development of blockchain technologies, creates a niche for decentralized payment platforms such as those offered by Ethereum.

These platforms provide major advantages over existing ones: they are more transparent, since operations are recorded in a public and immutable ledger; they are decentralized, making the network more resilient to failures or attacks as it does not rely on the servers of a single company or institution; they reduce complexity and fees in the payment process by eliminating intermediaries; and they remove the need for trust between users thanks to the consensus mechanism.

In addition, Ethereum has a significant advantage over similar technologies: it is not only a means of transferring value but also allows the execution of code on the blockchain through its quasi-Turing-complete language.

In response to this demand, we have developed an electronic payment platform based on Ethereum smart contracts, designed to be integrated into e-commerce. The platform allows a customer to deposit the amount of a purchase in an escrow contract, which holds the funds until a neutral third party (agent) verifies that the agreed condition, such as product delivery, has been met and authorizes the release of payment to the merchant. This design strengthens trust in the transaction and eliminates financial intermediaries. At the same time, we have investigated two major challenges still faced by this type of contract: resource efficiency (gas consumption) and security against vulnerabilities, with the aim of continuing to develop and improve the platform in these aspects in future versions.

Goals

The general objective is to design, implement, and evaluate a prototype payment platform using smart contracts based on Ethereum, capable of being integrated into e-commerce. The platform should be secure, efficient, and easy to integrate.

The specific objectives are as follows:

- Research and study of the state of the art and technological options for implementation.
  A literature review and analysis of existing platforms was carried out in order to identify challenges and previous solutions in the use of smart contracts for electronic payments. At the same time, different technological options (languages, development environments, libraries, wallets, and deployment tools) were analyzed to inform project design decisions.
- System architecture design.
  A layered architecture was defined to separate the user interface, account and signature management (MetaMask), and business logic implemented on the blockchain through smart contracts. The actors involved (payer,

payee, and agent) and their interactions via the escrow contract were also modeled.

- Implementation of contracts.

  Smart contracts in Solidity were developed to manage the creation of escrow deposits, the release of funds, and their return in case of non-compliance. Security patterns such as Checks-Effects-Interactions were applied to prevent reentrancy attacks, along with best practices to reduce gas consumption.

- Implementation of the graphical interface.

  A decentralized application (DApp) frontend was built to enable users (customers, merchants, and agents) to interact with the contracts in a simple way, connecting their wallets via MetaMask and managing payments from an intuitive web interface.

- Integration of contracts and interface.

  An integration layer was created between the contracts deployed on the test network and the graphical interface, enabling the DApp to capture contract addresses, manage transaction states, provide real-time feedback, and display the history of operations performed.

- Security and gas usage.

  The robustness of the contracts was evaluated against common Ethereum vulnerabilities (such as reentrancy or inefficient storage usage) and the gas consumption of operations was measured, with optimizations proposed to contribute to a more secure and efficient system.

- Research into similar tools.

  Platforms offering smart contract-based payment services (such as Origin Protocol, Boson Protocol, or the Shopify–Coinbase initiative) were studied to compare approaches, identify best practices, and differentiate this project's proposal.

- Research into future improvements.

  Potential lines of evolution for the prototype were explored, such as deployment on layer-two networks (L2) to reduce costs or the incorporation of oracles to automate the agent's signal.

Work plan

At the start of the project, a broad exploration of alternatives was conducted: from building a complete decentralized marketplace to developing isolated payment components. Ultimately, the focus was placed on creating a payment platform integrable into e-commerce. The general objective guided the entire plan: to design, implement, and evaluate a prototype on Ethereum that is secure, efficient, and easy to integrate.

With that goal in mind, the work began with a study of Ethereum and Solidity to establish a solid knowledge base. This was followed by an analysis of technological options (wallets, libraries, escrow patterns, and gas costs) to determine what to apply in the project. Based on this, the system architecture was designed: a strict separation between the DApp (presentation), wallet (signatures), and contracts (EscrowFactory/Escrow), definition of roles (payer, payee, agent), and a state machine with its events.

On that foundation, the contracts were implemented in Solidity applying patterns to prevent reentrancy and carefully managing storage writes to contain gas usage. In parallel, the graphical interface and an API layer were developed to orchestrate signing with MetaMask and rebuild listings from events without relying on a backend.

Next, integration of contracts and interface was addressed: form validations, transaction feedback (hash, state), capturing the new escrow's address from the receipt, and views for monitoring/resolution. This cycle concluded with a review of security and gas usage.

Finally, related work and other existing platforms were researched, and potential lines of future improvement aligned with the integrability objective were documented.

Conclusions

In this thesis, a functional prototype of a payment platform with smart contracts on Ethereum has been designed and implemented, aimed at its integration into e-commerce through an escrow scheme with a neutral agent. The solution combines a DApp connected to MetaMask with a contract factory and independent Escrow contracts per order, prioritizing security, transparency, and gas efficiency. The approach of reconstructing transaction history from on-chain events avoids storing unnecessary information, makes the cost per operation more predictable, and preserves user privacy. In terms of security, the Escrow contract applies the Checks-Effects-Interactions pattern, terminal states, and strict role-based authorization, which mitigates risks such as reentrancy and ensures that funds can only be released or refunded in a single flow direction. Regarding efficiency, loops have been avoided, constant and immutable variables have been used whenever possible, and events have been heavily relied upon to reduce storage writes.

When reviewing the objectives set out in the initial chapter, it can be stated that the general goal of designing, implementing, and evaluating a secure, efficient, and easily integrable prototype has been achieved. The resulting system includes a DApp with MetaMask and a layered architecture that demonstrates its integration into a purchase flow and the traceability of operations through events.

As for the specific objectives, the first, related to researching the state of the art and technological options, has been achieved through a review of platforms and similar solutions, thereby justifying the decisions made. At this stage, we encountered difficulties in writing the section in a didactic and comprehensive way, as we struggled to organize it properly, but with the guidance of our tutors we managed to give it a much better structure.

The system architecture was defined in three layers and in an actor model with a factory pattern to manage multiple payments. The EscrowFactory and Escrow contracts were implemented with clear invariants, events, and functions for custody, release, and refund, meeting the implementation goal. The graphical interface covers the necessary views to send payments, review pending ones, release funds, consult transactions, and

test purchases in an integrated store, demonstrating the integration between the interface and the contracts. Integration was also validated by showing feedback to the user and reconstructing transaction history through events. At this point, we faced difficulties in developing a comprehensive yet simple diagram of the architecture. After many tests and sketches, we found a way to explain the architecture that balanced simplicity for easier understanding with the inclusion of key components and flows.

In terms of security and gas usage, protection mechanisms were introduced in the design and a thorough analysis with Gastap was carried out, confirming that critical operations maintain a limited and predictable gas consumption: getter functions show very low costs, release and refund functions are around seventy-four thousand units, and escrow creation is around thirty-six thousand. Furthermore, the landscape of related tools and platforms was investigated, and possible lines of future work were identified.

Another major challenge was correctly integrating the contract logic with the graphical interface, especially regarding the capture and real-time updating of states from events. Although the Ethers.js documentation is fairly complete, at the time of development we decided to transition from version 4 to version 5, which presented gaps and incomplete examples in its documentation, adding difficulty to the process. This forced us to dedicate additional time to research and testing, both to decide how to store information efficiently without incurring high costs and to learn how to properly work with the events emitted by the contracts.

In summary, the project materializes a secure and efficient approach to conditional payments in e-commerce through escrow smart contracts, with a simple user experience and event-based traceability. The evaluation confirms the technical feasibility of the model and demonstrates that the proposed objectives have been fully achieved, providing a solid foundation for evolving toward more scalable production environments, with lower costs and greater robustness in the trust among the actors involved.

Future work

This prototype has limitations that could be addressed in future work. In general, it could be deployed on L2 networks to reduce transaction fees while maintaining security. Incentives could also be incorporated for the agent in order to reduce potential bottlenecks. However, the following are the priority improvements that we believe would bring the most value to the platform.

The first step would be to integrate a real oracle for the conversion from euros to Ether. The current one is static and only a simulation, so it does not represent real conditions and implies higher costs.

On the other hand, what would add the most value to this prototype would be integrating oracles that automate the agent's signal. The idea is that this signal would be sent automatically upon confirming delivery, without requiring the agent to log into the platform to authorize the release each time a payment is made.

Another possible expansion of the platform is to accept other types of conditions for the release of funds. For example, payment expiration if the package does not arrive on time, where the platform fee would be charged to the seller and refunded to the customer. However, the delivery of the package should remain the primary condition, as we consider it the most appropriate one for e-commerce.

Finally, it would be interesting to add a dual escrow contract for digital goods. In this type of contract, the buyer deposits the funds into the contract and the seller deposits the digital asset. This way, the exchange would take place without the need for a validating agent.

CONTRIBUCIONES PERSONALES

En el desarrollo de este trabajo, hemos trabajado colaborativamente de manera que se han repartido las tareas de manera equilibrada pero los dos hemos participado en todo aportando ya sea realizando la tarea de manera conjunta o aportando feedback e ideas. No obstante, cada uno hemos asumido un rol principal, según nuestros puntos fuertes, en las dos tareas generales: el desarrollo del código y el desarrollo de la memoria e investigación. Hugo ha liderado el desarrollo del código: desde la implementación de este hasta la integración de las herramientas utilizadas y Almudena ha liderado el desarrollo de la memoria e investigación: desde la investigación bibliográfica hasta la organización y redacción de la memoria.

Aún con esta división ha habido una comunicación constante y una participación completa en todas las tareas. La ventaja principal de esta división ha sido que en todo momento uno de los dos tenía un control completo de cada una de las dos partes. En consecuencia, el trabajo ha avanzado de manera fluida, evitando solapamientos y complementándonos según nuestras fortalezas. Además, así hemos aprendido más al tener ya que explicar lo aprendido al otro reforzamos el conocimiento.

BIBLIOGRAFÍA

[1] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger. Shanghai Version," 2025. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[2] S. Farokhnia, "Lazy Contracts: Alleviating High Gas Costs by Secure and Trustless Off-chain Execution of Smart Contracts," Sep. 2023, [Online]. Available: http://arxiv.org/abs/2309.11317

[3] Ethereum.org, "Introducción a los contratos inteligentes," https://ethereum.org/es/smart-contracts/.

[4] N. Szabo, "Smart Contracts," 1994, [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOT winterschool2006/szabo.best.vwh.net/smart.contracts.html

[5] Ethereum.org, "Registrar datos de contratos inteligentes con eventos," https://ethereum.org/es/developers/tutorials/logging-events-smart-contracts/.

[6] Ethereum.org, "Estándar de token ERC-20," https://ethereum.org/es/developers/docs/standards/tokens/erc-20/.

[7] Ethereum.org, "Monedas estables - Dinero digital para uso diario," https://ethereum.org/es/stablecoins/.

[8] Ethereum.org, "Finanzas descentralizadas (DeFi)," https://ethereum.org/es/defi/#what-is-defi.

[9] "Uniswap," https://app.uniswap.org/.

[10] Ethereum.org, "Estándar de token no fungible ERC-721," https://ethereum.org/es/developers/docs/standards/tokens/erc-721/.

[11] Ethereum.org, "Introducción a las DApps," https://ethereum.org/es/developers/docs/dapps/.

[12] A. Asgaonkar and B. Krishnamachari, "Solving the Buyer and Seller's Dilemma: A Dual-Deposit Escrow Smart Contract for Provably Cheat-Proof Delivery and Payment for a Digital Good without a Trusted Mediator," *2019 IEEE international conference on blockchain and cryptocurrency (ICBC)*, 2019.

[13] “Solidity,” https://soliditylang.org/.

[14] “Vyper,” https://vyperlang.org/.

[15] “Python,” https://www.python.org/.

[16] Ethereum.org, “Lenguajes de contrato inteligente,” https://ethereum.org/es/developers/docs/smart-contracts/languages/.

[17] N. B. M. C. T. Atzei, “A Survey of Attacks on Ethereum Smart Contracts (SoK),” in *Principles of Security and Trust*, M. Maffei and M. Ryan, Eds., in Lecture Notes in Computer Science. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186. doi: 10.1007/978-3-662-54455-6.

[18] T. Chen, X. Li, X. Luo, and X. Zhang, *Under-Optimized Smart Contracts Devour Your Money*. 2017.

[19] Ethereum.org, “Gas y tarifas,” https://ethereum.org/es/developers/docs/gas/#what-is-gas-limit.

[20] Ethereum.org, “La Historia de Ethereum,” https://ethereum.org/es/history/.

[21] Ethereum.org, “Prueba de trabajo (PoW),” https://ethereum.org/es/developers/docs/consensus-mechanisms/pow/.

[22] Ethereum.org, “Prueba de participación (PoS),” https://ethereum.org/es/developers/docs/consensus-mechanisms/pos/.

[23] Ethereum.org, “La Fusión,” https://ethereum.org/es/roadmap/merge/.

[24] “MetaMask,” https://metamask.io/es.

[25] “Truffle,” https://github.com/trufflesuite/truffle.

[26] “HardHat,” https://github.com/NomicFoundation/hardhat.

[27] “TypeScript,” https://www.typescriptlang.org/.

[28] “Web3.js,” https://github.com/web3/web3.js.

[29] “Ethers.js,” https://github.com/ethers-io/ethers.js.

[30] “Node Package Manager,” https://github.com/npm/cli.

[31] “Node.js,” https://github.com/nodejs/node.

[32] "Astro," https://astro.build/.

[33] Docs.astro.build, "Islands architecture," https://docs.astro.build/en/concepts/islands/.

[34] "Preact," https://preactjs.com/.

[35] "React," https://es.react.dev/.

[36] "TailwindCSS," https://tailwindcss.com/.

[37] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "Don't run on fumes—Parametric gas bounds for smart contracts," 2021.

[38] "Shopify," https://www.shopify.com/es-es.

[39] "Coinbase," https://www.coinbase.com/es-es.

[40] Shopify.engineering, "Introducing the commerce payments protocol," https://shopify.engineering/commerce-payments-protocol.

[41] "Base," https://www.base.org/.

[42] Origin Protocol Blog, "Origin DApp gets ERC 725 Identity, Transaction Steps, Escrow, and Reviews," https://blog.originprotocol.com/tech-update-dapp-gets-erc-725-identity-transaction-steps-escrow-reviews-298a7c91b7a5.

[43] Canvasbusinessmodel.com, "What is the brief history of Origin Protocol Company?," https://canvasbusinessmodel.com/blogs/brief-history/origin-protocol-brief-history?srsltid=AfmBOoqIL-hADpo40suIN-VIZSkrJqLq40wokj0rgHPLoo-GSI39_IbH#:~:text=The%20early%20growth%20of%20Origin,and%20a%20clearer%20value%20proposition.

[44] Bosonportal.io, "Boson Portal - How it works?," https://www.bosonportal.io/how-it-works.

[45] "Decentraland," https://decentraland.org/.

[46] "DMarket," https://dmarket.com/es.

[47] "OpenBazaar," https://github.com/OpenBazaar/openbazaar-desktop/releases.

[48] "Zora," https://zora.co/.

[49] "Bitify," https://shutdown.bitify.com/.

[50] Carrefour.es, "Carrefour lanza el primer blockchain de pescado fresco," https://www.carrefour.es/grupo-carrefour/sala-de-prensa/noticias.aspx?tcm=tcm:5-51092.

[51] "Maersk," https://www.maersk.com/.

[52] "IBM," https://www.ibm.com/es-es.

[53] Maersk.com, "A.P. Moller - Maersk and IBM to discontinue TradeLens, a blockchain-enabled global trade platform," https://www.maersk.com/news/articles/2022/11/29/maersk-and-ibm-to-discontinue-tradelens.

[54] "Tracr," https://www.tracr.com/.

[55] "De Beers," https://www.debeersgroup.com/.

[56] "OpenSC," https://opensc.org/.

[57] "WWF," https://wwf.org.au/.

[58] "BCG," https://www.bcg.com/.

APÉNDICES

Apéndice A - Resultados Gastap

GASTAPRES:
/home/Escrow.sol_Escrow_currentState()_block138;/home/Escrow.sol;Escrow;currentState();0x0c3f6acf;block138;ok;2540;15;0;0;0;0;1.74;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_getBalance()_block168;/home/Escrow.sol;Escrow;getBalance();0x12065fe0;block168;ok;342;15;0;0;0;0;1.607;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_payer()_block198;/home/Escrow.sol;Escrow;payer();0x123119cd;block198;ok;403;15;0;0;0;0;1.622;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_refund()_block228;/home/Escrow.sol;Escrow;refund();0x590e1ae3;block228;ok;74313+nat(call);18;0;0;0;0;2.521;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_releaseFunds()_block238;/home/Escrow.sol;Escrow;releaseFunds();0x69d89575;block238;ok;74246+nat(call);18;0;0;0;0;2.516;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_amount()_block248;/home/Escrow.sol;Escrow;amount();0xaa8c217c;block248;ok;2431;15;0;0;0;0;1.638;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_payee()_block278;/home/Escrow.sol;Escrow;payee();0xae90b213;block278;ok;402;15;0;0;0;0;1.633;0;0

GASTAPRES:
/home/Escrow.sol_Escrow_escrowAgent()_block308;/home/Escrow.sol;Escrow;escrowAgent();0xe1f6c098;block308;ok;424;15;0;0;0;0;1.629;0;0

GASTAPRES:

/home/EscrowFactory.sol_EscrowFactory_createEscrow(address,address)_block33;/home/EscrowFactory.sol;EscrowFactory;createEscrow(address,address);0x834cc6fc;block33;ok;36652;409;0;0;0;0;6.865;0;0

GASTAPRES:

/home/MockPriceFeed.sol_MockPriceFeed_setRate(int256)_block78;/home/MockPriceFeed.sol;MockPriceFeed;setRate(int256);0x2587eb4e;block78;ok;23687;24;0;0;0;0;1.309;0;0

GASTAPRES:

/home/MockPriceFeed.sol_MockPriceFeed_DECIMALS()_block106;/home/MockPriceFeed.sol;MockPriceFeed;DECIMALS();0x2e0f2625;block106;ok;316;15;0;0;0;0;0.964;0;0

GASTAPRES:

/home/MockPriceFeed.sol_MockPriceFeed_decimals()_block136;/home/MockPriceFeed.sol;MockPriceFeed;decimals();0x313ce567;block136;ok;345;15;0;0;0;0;0.957;0;0

GASTAPRES:

/home/MockPriceFeed.sol_MockPriceFeed_latestAnswer()_block166;/home/MockPriceFeed.sol;MockPriceFeed;latestAnswer();0x50d25bcd;block166;ok;2460;15;0;0;0;0;0.97;0;0