dApp basada en meta-transacciones

Vote4Photo

Roberto Sánchez Custodio

Máster U. en Ciberseguridad y Privacidad
Sistemas de Blockchain

Jordi Guirao Muns
Victor Garcia Font

01/2024

Universitat Oberta de Catalunya

FICHA DEL TRABAJO FINAL

Título del trabajo:	*dApp basada en meta-transacciones - VoteForPhoto*
Nombre del autor:	*Roberto Sánchez Custodio*
Nombre del consultor/a:	*Jordi Guirao Muns*
Nombre del PRA:	Victor Garcia Font
Fecha de entrega (mm/aaaa):	*01/2024*
Titulación o programa:	*Máster U. en Ciberseguridad y Privacidad*
Área del Trabajo Final:	Sistemas de blockchain
Idioma del trabajo:	*Castellano*
Palabras clave	*blockchain dapp metatransacciones*

Resumen del Trabajo

La dApp objeto del trabajo pretende demostrar con un enfoque práctico cómo se puede implementar mediante el uso de SmartContracts y meta-transacciones un modelo de negocio orientado al público en general, sin conocimientos en la materia y sin necesidad de criptomonedas, pero que permita con una política transparente que los usuarios puedan interactuar en un Marketplace con NFTs utilizando exclusivamente moneda fiduciaria.

Uno de los problemas que a menudo se encuentran las aplicaciones basadas en NFTs es el peaje que una persona ajena a la tecnología tiene que pagar para acceder a los mismos, la gente de a pie no entiende generalmente cómo funciona un sistema blockchain, las tarifas de gas y la desconfianza que da el uso de criptomonedas para poder acceder a la mayoría de los marketplaces de NFTs. Los usuarios podrán realizar transacciones en la blockchain indirectamente gracias a las meta-transacciones, como resultado, un usuario de la plataforma podrá adquirir un NFT, una foto original, sin la necesidad de poseer Ether u otra criptomoneda, por otro lado, el autor de la foto podrá acceder al Marketplace y ofrecer un NFT de su foto sin necesidad de Ether, lo cual facilita el acceso a cualquier fotógrafo sin conocimientos en la tecnología.

El uso de las meta-transacciones abre la puerta de la blockchain a personas no familiarizadas con la tecnología y que puedan tener reticencias, en especial por la necesidad de poseer algún tipo de criptomoneda, para poder operar en la misma y beneficiarse de sus ventajas.

Abstract

The dApp subject of this work aims to demonstrate with a practical approach how a business model oriented towards the general public, without prior knowledge in the blockchain and without the need for cryptocurrencies, can be implemented using SmartContracts and meta-transactions while allowing users to interact in a Marketplace with NFTs exclusively using fiat currency.

One of the problems often encountered by NFT-based applications is the toll that a person unfamiliar with technology has to pay to access them. The average person generally doesn't understand how a blockchain system works, gas fees, and the distrust that using cryptocurrencies can create when trying to access most NFT marketplaces. Users will be able to carry out transactions on the blockchain indirectly thanks to meta-transactions. As a result, a platform user will be able to acquire an NFT, an original photo, without the need to possess Ether or another cryptocurrency. On the other hand, the photo's author will be able to access the marketplace and offer an NFT of their photo without needing Ether, making it easier for any photographer without knowledge on the technology.

The use of meta-transactions opens the door to the blockchain for individuals unfamiliar with the technology and who may have reservations, especially due to the need to possess some form of cryptocurrency in order to operate within it, and benefit from its advantages

Índice

Lista de figuras

1 Introducción

El mundo de la tecnología blockchain ha supuesto toda una revolución en los últimos años, planteando un nuevo paradigma en la manera de concebir el concepto de transacción entre partes ya sea por un bien o por un servicio. Si bien, en los últimos tiempos, la adopción de la tecnología por empresas y ciudadanos puede haberse frenado en cierta medida por diferentes motivos, sigue siendo una tecnología con gran potencial y que evoluciona continuamente para adecuarse a nuevas necesidades.

Dentro del amplio mundo del blockchain, este proyecto se centra en las transacciones de NFTs y pretende dar una solución a uno de los problemas que presenta este tipo de transacciones, nos referimos al "coste del gas" o "tarifa del gas", coste que debe ser asumido tradicionalmente por el usuario final y que representa como mínimo un obstáculo para atraer a nuevos usuarios a la tecnología.

La solución que se plantea a la "tarifa del gas" está basada en las Meta-transacciones y a lo largo del presente documento la describiremos en detalle.

1.1 Contexto y justificación del Trabajo

El trabajo propuesto pretende dar una solución al mencionado problema con el coste del gas, creando un sistema que abstrae al usuario final de la existencia de dicho coste que además debe pagarse con la criptomoneda de la red correspondiente, tradicionalmente Ether (ETH), las transacciones que serán enviadas a la blockchain están relacionadas con NFTs (minado y transferencia), para ello vamos a desarrollar una aplicación web a modo de prueba de concepto que incluye una dApp (Distributed Application) que hará uso de meta-transacciones para el minado y transferencia de NFTs.

Para demostrar el funcionamiento de las meta-transacciones implementaremos una aplicación basada en un hipotético negocio relacionado con los concursos fotográficos, donde los concursantes crearán NFTs a partir de sus fotos. Hemos llamado al sistema Vote4Photo.

Figura 1: Logotipo aplicación Vote4Photo

El funcionamiento del sistema, Vote4Photo, actúa de manera similar a un Marketplace, pero con algunas características algo especiales, por un lado, para poder vender una foto debe ser parte de un concurso donde el resto de usuarios de la plataforma pueden votar por una o varias fotos, incluso pueden dar más de un voto para una misma foto.

El registro en la plataforma es gratuito, los usuarios solo deberán pagar por conseguir votos a usar en los concursos o para ofrecer una foto para que participe en un concurso, además el pago sería con moneda fiduciaria, en particular en euros.

La operativa de un concurso sería la siguiente:

- El administrador de la plataforma programa un concurso de fotografía para una fecha/hora determinada.
- Los usuarios de la plataforma que deseen exponer sus fotos se podrán adherir al concurso, indicando un potencial valor de venta y pagando la tasa correspondiente en euros. Al unirse al concurso se genera automáticamente un NFT de la foto que se almacena en el Wallet del fotógrafo mientras dura el concurso.
- Cuando llega la fecha programada se abre el concurso y cualquier usuario de la plataforma puede votar por las fotos expuestas. Al votar se debe indicar si se está interesado en adquirir la foto.
- Los votos son privados hasta la finalización del concurso, a la hora de votar no se saben los votos que tiene cada foto, para evitar influenciar en los votantes.
- Cuando el concurso finaliza, entre todos los votantes de la foto ganadora, se sortea el NFT de la propia foto.
- El autor de la foto ganadora recibirá un premio en metálico, proporcional a los votos recibidos, en el momento que realice la transferencia del NFT de la foto, actualmente en su Wallet. Si el fotógrafo decide no transferir la foto, el ganador del sorteo entre los votantes recibirá en su lugar el premio en metálico.
- Entre los votantes interesados en comprar las fotos, aunque no sean la ganadora, se sorteará la opción de compra. Todas las fotos del concurso pueden ser compradas.
- Todos los usuarios que hayan recibido la opción de compra y la ejecuten recibirán el NFT en su Wallet. El usuario no tendrá la necesidad de haber adquirido con anterioridad Ether u otra criptomoneda, ni tokens propios de la plataforma.
- Las fotos que se hayan quedado sin vender se mantendrán en el Wallet del autor. El autor tampoco necesitará de Ether o tokens previos para ello.

Todo el proceso está pensado para que el uso de las redes blockchain sea lo más transparente posible para los usuarios, que, si bien necesitarán de un Wallet para interactuar con la aplicación, no requieren haber adquirido o minado criptomoneda ni otro tipo de tokens, el Wallet solo se necesita para almacenar el NFT de la foto, simplificando enormemente el proceso para personas no familiarizadas con el mundo blockchain.

1.2 Qué es una meta-transacción

Todas las transacciones de la blockchain requieren del pago de lo que se conoce como tarifa del Gas [1] , que además puede tener un coste variable y no despreciable, por lo que puede llegar a ser un problema en determinados

servicios descentralizados (web3), dado que el usuario que inicia la transacción debe pagar dicha tarifa, este problema ha llevado a la necesidad de crear un mecanismo que permita evitar el pago del Gas al usuario que inicia la petición, recayendo dicho pago en un tercero, que podría ser, por ejemplo, la empresa que gestiona un determinado servicio, la cual obtiene el beneficio por otro canal. En resumen, un usuario podrá realizar transacciones en la blockchain, aunque tenga 0 ETH en su cuenta, siempre que haya un tercero dispuesto a hacerse cargo de la tarifa del Gas.

Dicho mecanismo es lo que se conoce como Meta-transacción y se define en el standard ERC-2771 [2] , en el siguiente diagrama se muestra cómo es el flujo de la información al realizar una meta-transacción:

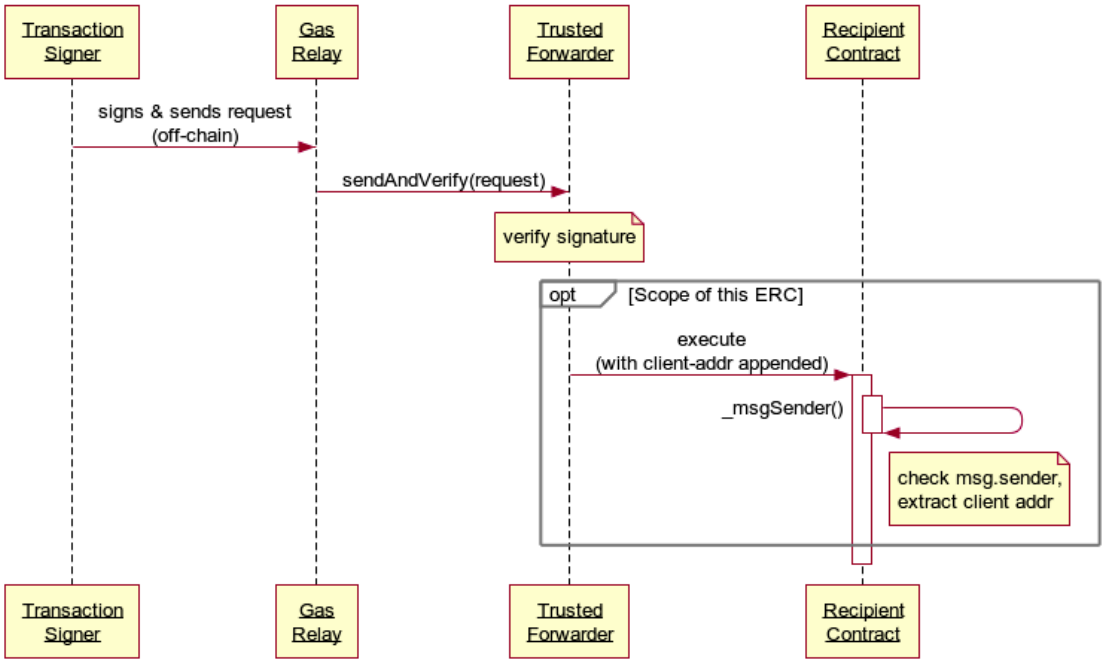

Figura 2: Diagrama de flujo de una meta-transacción.
Fuente: https://eips.ethereum.org/EIPS/eip-2771#example-flow

Los actores que intervienen en la misma son los siguientes y cada uno tiene una función determinada en la ejecución de la transacción:

- **Transaction signer**: La cuenta que inicia la meta-transacción, en nuestro caso los usuarios de la plataforma, si bien no necesitan gas para realizarla, si deben firmar la transacción, firma que será validada posteriormente.
- **Gas relay**: Es la cuenta que se hará cargo del gas y que recibe la petición del firmante off-chain, es decir, antes de entrar en la cadena de bloques, por ese motivo se habla de “meta”-transacción ya que se inicia el proceso antes de crear una transacción final.
- **Trusted Forwarder**: Es un contrato inteligente que debe verificar la firma del “Transaction signer” y enviar la transacción al contrato final que añadirá la transacción a la blockchain, utilizando la cuenta del firmante como la del cliente que originó la transacción pero pagando el gas con la cuenta del “Gas relay”.

- **Recipient contract**: Es el contrato final destinatario de la transacción.

En el Anexo III. Meta-transacción por usuario, podemos ver cómo percibiría el usuario final (el “Transaction signer”) un uso práctico de una meta-transacción en la creación de un NFT, las interacciones que debe realizar y el resultado final de la misma.

Alternativas para implementación de meta-transacciones

Para implementar las meta-transacciones hemos optado por los contratos base ofrecidos por OpenZeppelin, pero existen otras alternativas que también se han valorado y que comentamos a continuación, como es el caso de Open GSN [3] (Gas Station Network), que ofrecen su propia implementación de los contratos “Forwarder” (Forwarder.sol) y “Recipient” (ERC2771Recipient.sol) para el standard ERC2771, a diferencia de Open Zeppelin, el enfoque para el uso de meta-transacciones resulta algo más ambicioso, planteando un escenario con un HUB de posibles Relayers de confianza y donde podemos añadir nuestro propio Relayer, con este enfoque el contrato que se hace cargo del gas es independiente del Relayer, Paymaster contract, toda la arquitectura del sistema propuesto se puede ver en el diagrama de la documentación oficial:

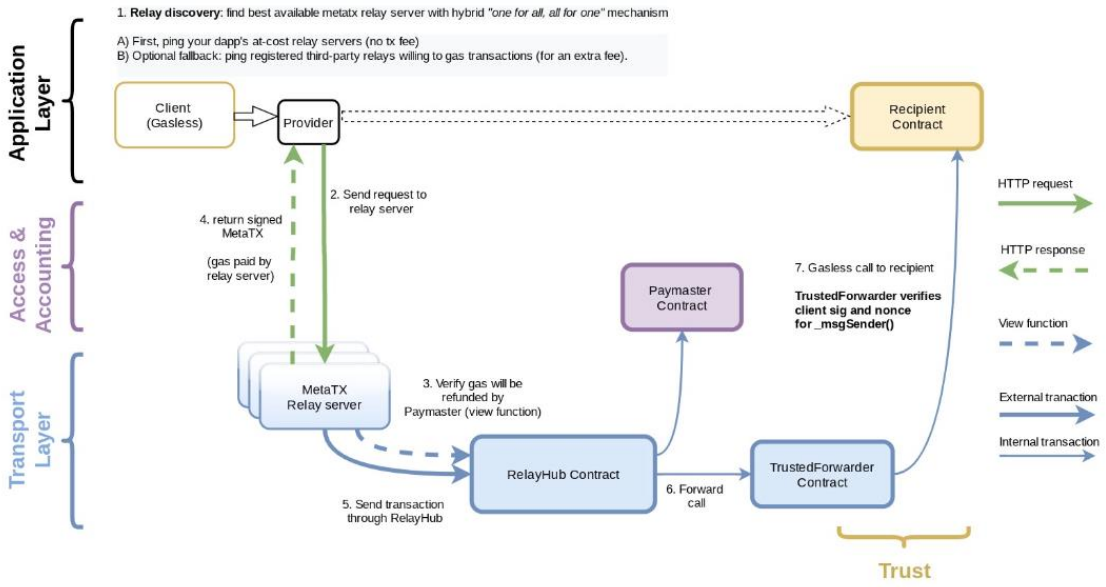

Figura 3: Arquitectura meta-transacciones en Ethereum Gas Station Network
Fuente: https://docs.opengsn.org/#the-problem

Si bien el planteamiento es interesante para poder reutilizar la arquitectura ya disponible de OpenGSN con todos los controles de seguridad ya implementados para un uso a gran escala, para nuestro caso resultaba más adecuado un escenario donde el Relayer y el Paymaster fueran el mismo contrato, simplificando de esta manera el despliegue sin depender de infraestructura de terceros.

Alternativas a las meta-transacciones

Si bien las meta-transacciones dan una solución al problema del pago del gas por parte de los usuarios de una plataforma web3, pueden tener cierto problema a la hora de ser aplicadas en contratos ya existentes, dichos problemas están relacionados con la obtención de la dirección del firmante por parte del "Trusted Forwarder" y los datos originales de la transacción, el problema [4] del standard ERC-2771 es que no es retroactivo y aplicarlo en contratos ya desplegados y en uso puede ser costoso y potencialmente peligroso.

Para solucionar este escenario se ha creado otro standard, el ERC-4337 [5] que introduce el concepto de "Account Abstraction", el cual consiste en que los usuarios en lugar de realizar directamente transacciones en la blockchain hacen uso de lo que se conoce como UserOperation, que en cierto modo representa la "intención" de realizar una transacción, la cual acaba en un *mempol* específico para UserOperations para enviar operaciones (mensajes) a un contrato inteligente que implementa el standard, dichas operaciones se agrupan en un *bundle* y un tercero se encarga de crear la transacción y grabar las operaciones en la blockchain, asumiendo éste el coste del Gas.

Un diagrama del proceso sería el siguiente:

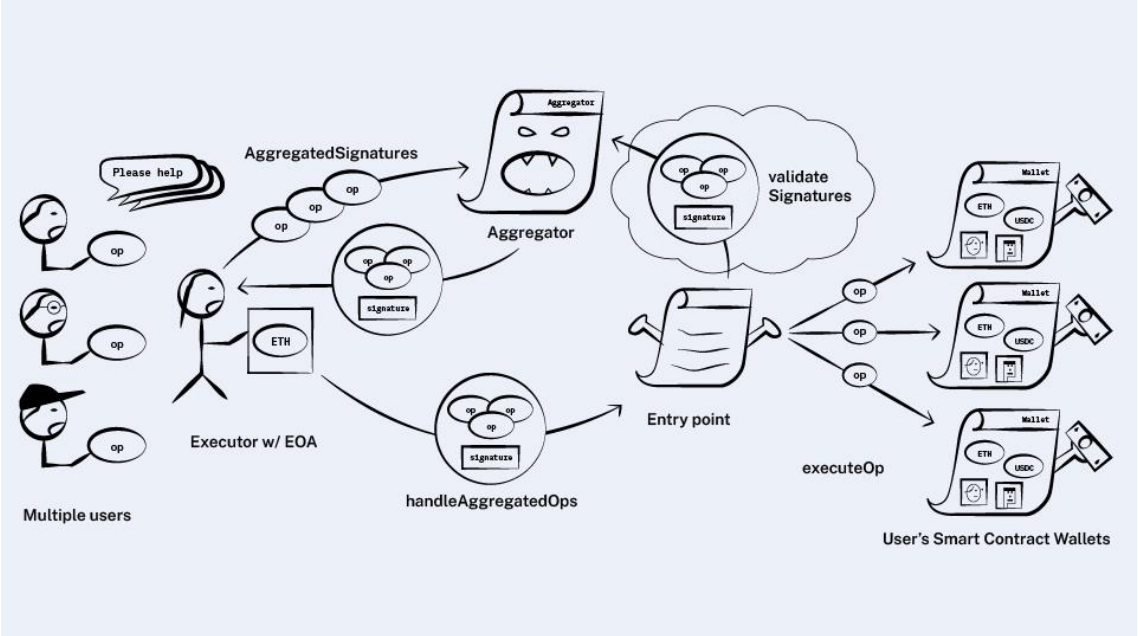

Figura 4: Diagrama proceso de Account Abstraction
Fuente: https://www.alchemy.com/overviews/what-is-account-abstraction

Podemos encontrar un análisis más profundo del concepto de "Account Abstraction" en el paper publicado recientemente (septiembre-2023) "Account Abstraction, Analysed" [6], en el estudio concluye la mejora en seguridad y eficiencia, además de un impacto reducido en la infraestructura preexistente para poder aplicar mecanismos de ejecución de operaciones sin necesidad de pagar la tarifa del Gas de la transacción por parte de los usuarios finales, es decir, podemos decir que el nuevo standard añade nuevas soluciones al problema del gas, sin embargo, a diferencia de las meta-transacciones, el standard es aún un borrador en el momento de iniciar el presente proyecto, por lo que podemos

considerar la solución de las meta-transacciones con un punto más de madurez, si bien ya existen algunas implementaciones como son:

- Eth-infinitism: https://github.com/eth-infinitism/account-abstraction
- Stackup: https://www.stackup.sh/

En cualquier caso, pese al potencial problema de las meta-transacciones, en un escenario como el planteado en el TFM, donde se va a implementar el servicio desde cero, demostraremos que son una solución viable, sencilla y asequible para solucionar el problema del pago del gas.

1.3 Objetivos del Trabajo

Los objetivos que se pretenden conseguir con el trabajo son los siguientes:

- Demostrar con un caso práctico cómo se pueden realizar transacciones delegando el coste del gas en un tercero.
- Diseñar un servicio que resulte atractivo y sencillo de utilizar a un tipo de usuario que potencialmente puede beneficiarse de las ventajas de las NFTs, como es el caso de los amantes de la fotografía o artistas digitales, ya sean profesionales o aficionados. Mostrando de esta manera que la blockchain es más que criptomonedas y se pueden plantear servicios útiles sin los riesgos derivado de las mismas.
- Mostrar un modelo de negocio sostenible, donde se pueden intercambiar NFTs sin el uso de criptomonedas ni siquiera para el pago del gas de la transacción por parte de los usuarios finales, de manera que usuarios que buscan ampliar sus bibliotecas de NFTs, puedan hacerlo con un coste reducido y sin preocuparse de sobrecostes indirectos por las transacciones.

1.4 Impacto en sostenibilidad, ético-social y de diversidad

Sostenibilidad

Las redes de blockchain y el minado de criptomonedas en particular han sido objeto de críticas en lo referente a la gran cantidad de energía necesaria para minar los bloques de la cadena, sin embargo, en particular la red Ethereum donde correría una supuesta versión final del sistema descrito, ha cambiado el mecanismo por el que se minan nuevos bloques, que entre otras consecuencias resulta en un ahorro de energía de órdenes de magnitud. El nuevo Ethereum 2.0 basa la aceptación de los bloques por un mecanismo Proof-of-Stake [7] en lugar de utilizar el tradicional Proof-of-Work [8], mucho más exigente en cuanto a necesidad de cómputo y por tanto con un consumo mucho mayor de energía.

Adicionalmente, el despliegue de la aplicación web en una plataforma Serverless como Cloudflare Workers/Pages basada en un mecanismo de aislamiento de los procesos de usuario (Isolate model [9]) que optimiza en gran medida el *overhead* del sistema para correr cada proceso, tiene como consecuencia el ahorro en el uso de recursos, en especial el uso de la memoria se reduce drásticamente,

mejorando así el coste efectivo del sistema y su eficiencia energética en relación a otras soluciones basadas en máquinas virtuales o contenedores.

Etico-social

El mundo del blockchain ha venido en los últimos tiempos siendo objeto de una gran polémica, criticado a menudo por la volatilidad de criptomonedas como Bitcoin o Ethereum, llevándose por el camino a muchos pequeños y no tan pequeños inversores.

Figura 5: Evolución del valor de Ethereum en los últimos 5 años.
Fuente: Google Finance: https://g.co/finance/ETH-USD?window=5Y

Figura 6: Evolución del valor del Bitcoin en los últimos 5 años
Fuente: Google Finance: https://g.co/finance/BTC-USD?window=5Y

También hemos vivido recientemente la quiebra de grandes empresas del sector como FTX [10] en 2022 que han contribuido a dar una imagen de inestabilidad y desconfianza a gran parte de la sociedad que no ha favorecido la adopción de nuevos usuarios para el mundo de blockchain y para el de las NFTs en particular.

En el escenario actual pensamos que la proliferación de servicios similares al expuesto en el presente TFM que eviten el uso de ningún tipo de criptomoneda por parte de los usuarios del mismo favorecería la imagen de los NFTs, desvinculándolos de la volatilidad asociada a las mencionadas criptomonedas, además de ofrecer un canal de venta para profesionales y aficionados del mundo de la fotografía, donde pueden exponer su trabajo en un canal que para muchos puede resultar complicado o que puede transmitir desconfianza.

Diversidad
Una de las ventajas evidentes de un servicio basado en la blockchain es la privacidad inherente de la tecnología, esto facilita el acceso a personas celosas de sus datos personales, pudiendo interactuar con otros usuarios de manera segura sin necesidad de prestar sus datos personales, ya que el wallet necesario para la operación en la aplicación es una dirección en una cadena de bloques, no contiene ningún tipo de dato personal.

Este entorno completamente privado y abierto favorece la inclusión de todo tipo de usuarios, sin importar el origen, la raza, creencias políticas o religiosas o cualquier otra característica que pueda ser objeto de exclusión en otro tipo de comunidad.

1.5 Enfoque y método seguido

Se ha elegido una estrategia con un enfoque puramente práctico para poder evaluar de la manera más realista posible lo adecuado de la solución técnica elegida (meta-transacciones) para resolver el problema planteado, además de mostrar un modelo de negocio plausible para poner los NFTs al alcance de todo tipo de usuarios sin necesidad de tratar con el uso de criptomonedas.

Para el desarrollo del proyecto se va seguir una metodología agile, similar a Scrum, una metodología basada en el aporte continuo de valor, aunque adaptada a los compromisos de entrega de las diferentes PEC del TFM, es decir, el trabajo se dividirá en Sprints que se sincronizarán con cada una de las PECs, como todo proyecto agile, se partirá de un backlog con las historias de usuario o tareas que se implementarán en el trabajo, las cuales pueden acabar evolucionando o cambiando conforme avance el proyecto. En el apartado Metodología describimos en detalle el enfoque aplicado.

En el siguiente apartado se enumeran las tareas a incluir en cada Sprint (o PEC), junto con la fecha de entrega asociada.

Para la PEC1 hemos considerado una especie de “Sprint 0” o *SetUp*, que se utilizará para preparar los entornos, el backlog y planificar el resto de Sprints.

1.6 Planificación del Trabajo

Las tareas a realizar durante la realización del TFM son las siguientes, organizadas por PECs

- PEC1 (10-oct-2023)
    - Planteamiento dApp (incluye modelo de negocio)
    - Elaboración plan de trabajo
    - Preparación entorno de desarrollo
    - Elaboración memoria inicial TFM con requisitos PEC1
- PEC2 (7-nov-2023)
    - Ampliación memoria TFM con requisitos PEC2
    - Creación Historias de usuario
    - Diseño SmartContracts para meta-transacciones (Recipient & Forwarder)
    - PoC meta-transacciones en local (Entorno HRE de Hardhat)
    - Creación estructura base front + back
    - Diseño API-REST
    - Diseño modelo de datos
- PEC3 (5-dic-2023)
    - Ampliación memoria TFM con requisitos PEC3
    - Implementación pantallas principales front
    - Integración dApp con smart-contracts
    - Integración con Metamask
    - Implementación API REST en back
- PEC4 (9-ene-2023)
    - Ampliación memoria TFM con requisitos PEC4
    - Implementación pantallas secundarias front (registro, profile, etc)
    - Implementación de proceso batch
    - Pruebas generales del sistema
    - Despliegue en la nube
- Presentación (16-ene-2024)
    - Finalización memoria TFM
    - Creación de video demo
- Defensa (26-ene-2024)

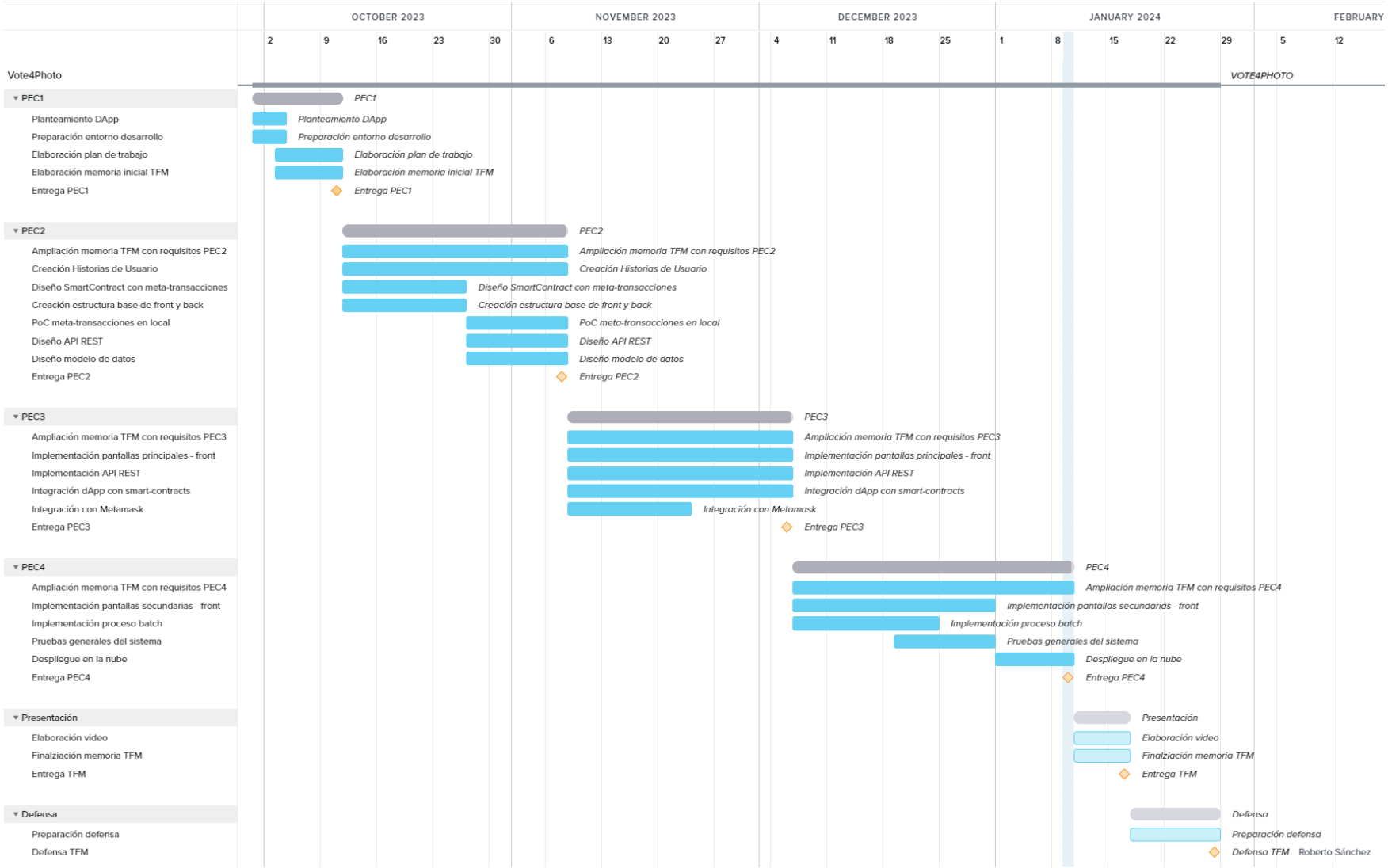

Figura 7: Diagrama de Gantt con la planificación del TFM

1.7 Breve sumario de productos obtenidos

A la finalización del trabajo, los productos o artefactos generados deben ser los siguientes:

- Aplicación web headless donde se va a separar el front del back. El front se implementará con el framework svelte+flowbite y el back con nodejs+hono, el lenguaje de desarrollo en ambos casos será Typescript.
- dApp que se incluirá dentro de la aplicación web y se encargará de interactuar con los SmartContracts.
- SmartContract que genera NFTs, desarrollado con Solidity y que implementa el standard ERC2771 (“Recipient”)
- SmartContract que implementa el reenvío de la meta-transacción al “Recipient”, “Forwarder”, desarrollado con Solidity.

1.8 Breve descripción de los otros capítulos de la memoria

En la presente memoria se ha respetado la estructura base original, pero se han añadido nuevos apartados dentro de las secciones principales predefinidas que describen en detalle el proyecto desarrollado, los capítulos añadidos son los siguientes:

- Introducción:
  - **1.2 ¿Qué es una meta-transacción?** Explicamos en qué consiste una meta-transacción y cómo resuelve el problema de la tarifa del gas.
- Materiales y métodos:
  - **2.1 Arquitectura general**. Definición de alto nivel de la arquitectura elegida para el desarrollo
  - **2.2 Metodología**. Explicación de la metodología utilizada para el desarrollo
  - **2.3 Lenguajes / frameworks / libs.** Enumeramos los lenguajes de programación que vamos a utilizar, así como los frameworks y libs principales
  - **2.4 Plataforma de despliegue y backends**. Elección de la plataforma de despliegue argumentando los motivos para dicha elección
  - **2.5 Herramientas y otros servicios para el desarrollo**. Breve descripción de las herramientas o servicios que vamos a usar en el proyecto
- Resultados:
  - **3.1 Código fuente**. Repositorios con el código fuente de la aplicación desarrollada.
  - **3.2 Historias de usuario**. Enumeración de las historias de usuarios desarrolladas, solo los títulos, el detalle de cada historia se movió al Anexo I.

    - **3.3 Servicios REST**. Enumeración de los servicios REST desarrollados con el path y los métodos HTTP de cada uno, la definición y detalle de cada uno se movió al Anexo II.
    - **3.4 Modelo de datos.** Modelo de datos relacional de la aplicación
    - **3.5 Estructura front y back**. Estructura de directorios y ficheros principales de los proyectos front y back.
    - **3.6 Diseño SmartContracts.** Explicación del diseño elegido para los SmartContracts, resaltando las características principales.
- Conclusiones y trabajos futuros
    - **4.1 Problemas encontrados**. Descripción de los problemas más destacados que hemos encontrado en el desarrollo
    - **4.2 Conclusiones**. Conclusiones respecto a los objetivos esperados y los obtenidos
    - **4.3 Trabajos futuros**. Propuesta de posibles evoluciones del trabajo realizado.
- Anexos:
    - **Anexo I. Historias de usuario**. Descripción funcional de la aplicación desarrollada con cierto nivel de detalle a nivel de pantallas, procesos, etc.
    - **Anexo II. API Servicios REST**. Descripción de los servicios REST desarrollados con los parámetros de entrada y salida.
    - **Anexo III. Meta-transacción por usuario**. Descripción de los pasos que haría un usuario final para realizar una meta-transacción, concretamente, la generación de un NFT a partir de una foto.

2 Materiales y métodos

2.1 Arquitectura general

La aplicación web que sustenta el sistema se desarrollará con una arquitectura *headless* [11], es decir, separando el desarrollo del *back* y del *front*, este tipo de enfoque facilita el mantenimiento y la escalabilidad de la aplicación, además de favorecer la creación eventual de otros clientes como aplicaciones móviles o la comunicación con otros sistemas.

- **Back**: Es el núcleo del sistema, encargado de la seguridad y del acceso a los datos, tanto a la base de datos como a los SmartContracts utilizados. Se creará un API REST para ser consumido por el front.
- **Front**: Es la capa visual o de presentación, encargada de mostrar la información al usuario, obtendrá los datos a mostrar del back.

El sistema dispondrá de 2 tipos de backend, por un lado, una base de datos relacional para almacenar la información necesaria para la operativa del sistema y por otro lado un par de SmartContracts basados en la red Ethereum que correrán en una testnet que describiremos más adelante.

2.2 Metodología

La metodología de desarrollo elegida está basada en la metodología Agile Scrum, aunque con algunas variaciones motivadas por las características particulares que tiene un proyecto para un TFM. Creemos que una metodología Agile tiene ciertas características que se ajustan bien al modelo de trabajo de un TFM, como son las entregas periódicas, la adaptabilidad de la funcionalidad en función de las circunstancias que afecten al trabajo y el concepto de entrega continua de valor ligado al desarrollo Agile.

Por otro lado, en cualquier proyecto software donde el equipo de desarrollo es unipersonal, el encaje con las metodologías habituales, ya sean *agile* o *waterfall* es limitado, no son metodologías pensadas para este tipo de escenarios, en todas existen diferentes roles y ceremonias que tienen poco sentido en el escenario que nos ocupa, sin embargo, por los motivos antes expresados, consideramos que el espíritu detrás de las metodologías *Agile* pueden resultar más adecuadas para el éxito del TFM.

A modo de resumen, las directrices que vamos a seguir de una metodología como Scrum son las siguientes:

- Creación del backlog inicial en Sprint0
- Equivalencia entre las PECs y los Sprints
- Ceremonias de refinamiento eventuales con el tutor para aclarar puntos específicos del alcance.
- Ceremonia de planning después de cada PEC para reevaluar las tareas a incluir en el siguiente Sprint (PEC)

- Ceremonias de Reviews “opcionales” excepto en la entrega final, donde habrá una review off-line en video y en la defensa donde será on-line.

2.3 Lenguajes / frameworks / libs

Los lenguajes principales que se van a utilizar en el trabajo son:

- **Typescript** (https://www.typescriptlang.org/): Lenguaje creado por Microsoft que añade tipado a Javascript, mejorando la DX (Developer eXperience), ampliamente utilizado a nivel mundial y compatible con todos los frameworks de front y back que vamos a utilizar a lo largo del proyecto.
- **Solidity** (https://soliditylang.org/): Lenguaje líder para el desarrollo de SmartContracts y compatible con la red Ethereum y derivadas.

Respecto a los frameworks y las principales libs que vamos a utilizar podemos enumerar desde el front al back las siguientes:

- **Svelte + SvelteKit** (https://svelte.dev/): Framework front moderno que ha irrumpido con fuerza en un mercado dominado por los conocidos React, Angular y Vue.js, Svelte propone un enfoque distinto a los antes mencionados que le han convertido en uno de los frameworks web más apreciados por los desarrolladores que lo utilizan. Es un framework sencillo de aprender y excepcionalmente rápido.
- **Vitejs** (https://vitejs.dev/): Framework para desarrollo web que aporta herramientas al programador para facilitar las tareas cotidianas.
- **Flowbite** (https://flowbite-svelte.com/): Completa biblioteca de componentes de interfaz de usuario para una mejor experiencia y completamente integrado con Svelte (entre otros frameworks)
- **Hono** (https://hono.dev/): Sencillo y rápido framework para el back, que permite desarrollar de manera ágil y sencilla una capa de servicios REST y además es compatible con diferentes *engines* javascript, no solo nodejs, sino también Deno y sobretodo, Cloudflare Workers/Pages, que será la plataforma de despliegue que explicaremos en el siguiente apartado.
- **Drizzle ORM** (https://orm.drizzle.team/): ORM para Typescript compatible con las principales BBDD Open Source, como SQLite, en nuestro caso utilizaremos Cloudflare D1 (sintaxis compatible con SQLite) que explicaremos más adelante.
- **Viem** (https://viem.sh/): Lib para interactuar con redes basadas en Ethereum mediante Typescript.
- **ERC2771Context / ERC2771Forwarder** (OpenZeppelin Metax): Para el desarrollo del SmartContract que realizarán las meta-transacciones necesitamos implementar el standard ERC-2771 [2] y por ese motivo utilizaremos los contratos base creados por OpenZeppelin para facilitar la implementación del “Forwarder” y para la obtención del firmante en el “Recipient”, que es el que inicia la transacción pero no el que realiza el pago del Gas (en nuestro caso sería la cuenta del usuario), el mencionado pago del gas cual corre a cargo del “Gas Relay” (en nuestro caso será la cuenta propietaria del sistema).

- **ERC721** (https://docs.openzeppelin.com/contracts/5.x/api/token/erc721): Utilizaremos el standard ERC-721 [12] para la creación de los NFT basados en las fotos de los usuarios, para ello haremos uso de los contratos que implementan dicho standard por OpenZeppelin.

2.4 Plataforma de despliegue y backends

La plataforma elegida para el despliegue es **Cloudflare Worker&Pages** (https://www.cloudflare.com/developer-platform/products/), una plataforma Serverless increíblemente eficiente y rápida en comparación a la ofrecida por los grandes actores de la nube (AWS Lambda, Google cloud functions o Azure functions) que además dispone de una cuota de recursos gratuitos bastante amplia y que proporciona un conjunto de herramientas y servicios adicionales que facilitarán el desarrollo y despliegue de la aplicación. Se desplegará la aplicación web en la plataforma Cloudflare Pages y el proceso batch que monitoriza los concursos en Cloudflare Workers, (el motivo para utilizar un despliegue independiente es que Cloudflare Pages no soporta el uso de "cron triggers", es necesario utilizar un Worker, aunque comparte código y recursos con la aplicación principal).

Uno de los servicios que aprovecharemos de Cloudflare es la base de datos relacional **D1** (https://developers.cloudflare.com/d1/) una base de datos compatible con la sintaxis de SQLite pero que corre en la nube y que Cloudflare ofrece con una generosa capa de uso gratuita que excede cualquier necesidad que podamos tener para la realización y puesta en funcionamiento en la nube del TFM.

Adicionalmente a la base de datos relacional utilizaremos un Object Storage para el almacenamiento de las fotos y de los descriptores de los tokens (ficheros JSON). Cloudflare incluye un servicio compatible con el pseudo-standard S3 de Amazon llamado **R2** (https://developers.cloudflare.com/r2/), Amazon fue pionera en este tipo de almacenamiento, este tipo de servicios resulta bastante idóneo para almacenar datos binarios como los ficheros de fotos y nos puede servir como alternativa más simple (y económica) frente al uso de un servicio IPFS, el cual sería probablemente más adecuado en un entorno real productivo de un sistema como el descrito, pero hemos optado por la alternativa descrita por para simplificar el sistema resultante y focalizar el esfuerzo el "core" del sistema, que es la implementación de las meta-transacciones.

La elección de Cloudflare como proveedor principal para el despliegue en la nube viene motivada por la gran calidad de los servicios que ofrece, junto con la generosa capa gratuita de uso para todos los servicios implicados y además por la existencia de un nuevo servicio relacionado con el mundo blockchain llamado Web3 (https://developers.cloudflare.com/web3/) el cual ofrece sendos API Gateways para acceder a la red IPFS y Ethereum (incluidas testnets) respectivamente, los cuales pueden resultar de utilidad eventualmente para el desarrollo del proyecto y en general para cualquier proyecto relacionado con la Web3 que precise interactuar con la plataforma con un API confiable.

La red de cadena de bloques que utilizaremos para el despliegue en la nube es **Sepolia** (https://sepolia.dev/) es una testnet relativamente reciente, basada en

un mecanismo de consenso proof-of-authority y es la testnet recomendada [13] oficialmente para desarrollo por la web de Ethereum. Adicionalmente para el entorno de desarrollo local utilizaremos la testnet **Hardhat** (https://hardhat.org/) que ofrece un entorno de desarrollo ágil y sencillo para probar el despliegue y uso los SmartContracts antes de desplegarlos en la testnet "final".

2.5 Herramientas y otros servicios para el desarrollo

Haremos uso de diferentes herramientas y servicios a lo largo del proyecto, algunas de las más significativas serán:

- **VS Code**: IDE (Integrated Development Environment) para desarrollo del proyecto, tanto front como back como de SmartContracts.
- **GitHub repositories**: Servicio en la nube para proporcionar repositorios de código fuente y herramientas para la gestión del proyecto.
- **Docker engine**: Utilizaremos Docker para la encapsulación de los backends necesarios en contenedores en el entorno de desarrollo, como bases de datos o testnets.

3 Resultados

La aplicación desarrollada está disponible en el siguiente enlace:

https://vote4photo.dvlpr.tech/

Para registrarse se puede utilizar cualquier correo ficticio como, por ejemplo: alice@uoc.edu, bob@uoc.edu, … Al no encontrarnos en un entorno real, la aplicación no valida los correos ni envía ningún tipo de comunicación vía email.

En los siguientes capítulos enumeramos los resultados obtenidos en el trabajo a partir de la metodología y la planificación realizada.

3.1 Código fuente

Se han creado 2 repositorios en GitHub para almacenar el código fuente, ambos repositorios son Open Source con licencia MIT, con el ánimo de facilitar el acceso y distribución a todo aquel desarrollador/a que pueda interesarse por la tecnología de las meta-transacciones.

- **v4p-contracts** (https://github.com/dvlprtech/v4p-contracts): Repositorio para contener los contratos inteligentes necesarios para el proyecto:
  - **V4PForwarder.sol**
  - **PhotoNFT.sol**
- **vote4photo** (https://github.com/dvlprtech/vote4photo): Aplicación web de tipo headless donde se ha separado el front del back, es decir, a efectos técnicos son proyectos independientes (el código del front se encuentra en el directorio "FRONT" del repositorio)

En las secciones Diseño SmartContracts y Estructura front y back se detalla la estructura y contenido de cada uno de los repositorios.

3.2 Historias de usuario

Siguiendo la metodología agile, enumeramos las historias de usuario que definirán el comportamiento del sistema en los diferentes escenarios comentados inicialmente, añadiendo el detalle necesario para afrontar su desarrollo.

- US-1: Alta de usuario
- US-2: Login de usuario
- US-4: Transferencia de fondos
- US-5: Compra de derechos de voto
- US-6: Creación de concursos
- US-7: Añadir nueva foto a concurso
- US-8: Añadir foto existente a concurso
- US-9: Monitorización de concursos
- US-10: Home de la aplicación
- US-11: Perfil del usuario
- US-12: Detalle de un concurso
- US-13: Votar por una foto
- US-14: Operación de traspaso de foto ganadora
- US-15: Operación de compra
- US-16: Operación de venta

El detalle funcional de cada una de estas historias se ha incluido en el ANEXO I

La definición del sistema funcionalmente podría evolucionar o variar conforme se avance en el desarrollo del proyecto, tal y como suele ser habitual en proyectos con metodología agile, el contexto o las circunstancias pueden hacer cambiar el enfoque de alguna de las historias definidas además de incluir ciertos matices o ampliaciones en algunas de las historias existentes una vez se acometa su desarrollo.

3.3 Servicios REST

En base a la funcionalidad descrita en las historias de usuario, planteamos una capa de servicios REST que proveerán de información a las diferentes pantallas e interactuarán con el backend para actualizar los datos correspondientes en cada operación.

Hemos organizado los servicios REST en 4 grupos distintos:

- Servicios relacionados con los usuarios
  - `POST /api/account/signup`
  - `POST /api/account/signin`
  - `GET /api/account/{user_id}`
  - `PUT /api/account/{user_id}`
  - `POST /api/account/funds`
  - `POST /api/account/buyvotes`
- Servicios relacionados con los concursos
  - `POST /api/contest`
  - `GET /api/contest/{contest_id}`
  - `PUT /api/contest/{contest_id}`
  - `POST /api/contest/{contest_id}/init`
  - `POST /api/contest/{contest_id}/addphoto`
  - `POST /api/contest/{contest_id}/vote`
- Servicios relacionados con las operaciones
  - `POST /api/operation`
  - `GET /api/operation/{operation_id}/data_to_sign`
  - `POST /api/operation/{operation_id}/signature`
  - `POST /api/operation/{operation_id}/execute`
  - `POST /api/operation/{operation_id}/reject`
- Servicios relacionados con las fotos
  - `GET /api/photo/{photoKey}`
  - `POST /api/photo/prepare`
  - `DELETE /api/photo/{photoKey}`
- Otros servicios
  - `GET /api/config/votes_pricing`
  - `GET /api/config/fees`
  - `GET /api`

La descripción de cada servicio, así como el detalle de los parámetros de entrada y estructura de salida se ha incluido en el Anexo II.

3.4 Modelo de datos

El modelo de datos consta de las siguientes entidades y tablas de relación:

- **user**: Tabla con los usuarios que acceden al sistema
- **user_photos**: Contiene las fotos de los usuarios, y la información del token asociado
- **contest**: Contiene información de cada concurso de fotografía
- **contest_photo**: Almacenará las fotos que participan en cada concurso
- **user_votes**: Contiene el total de votos de cada usuario a cada foto.
- **operations**: Operaciones resultantes de cada concurso, como por ejemplo opción de compra de fotos o transferencia de la foto ganadora al ganador del sorteo.

- **votes_pricing**: Preciario para el coste de compra de derechos de voto.
- **log_votes**: Registro de los votos emitidos por los usuarios

Estas tablas se relacionan entre sí como se muestra en el Diagrama Entidad-Relación de la siguiente figura :

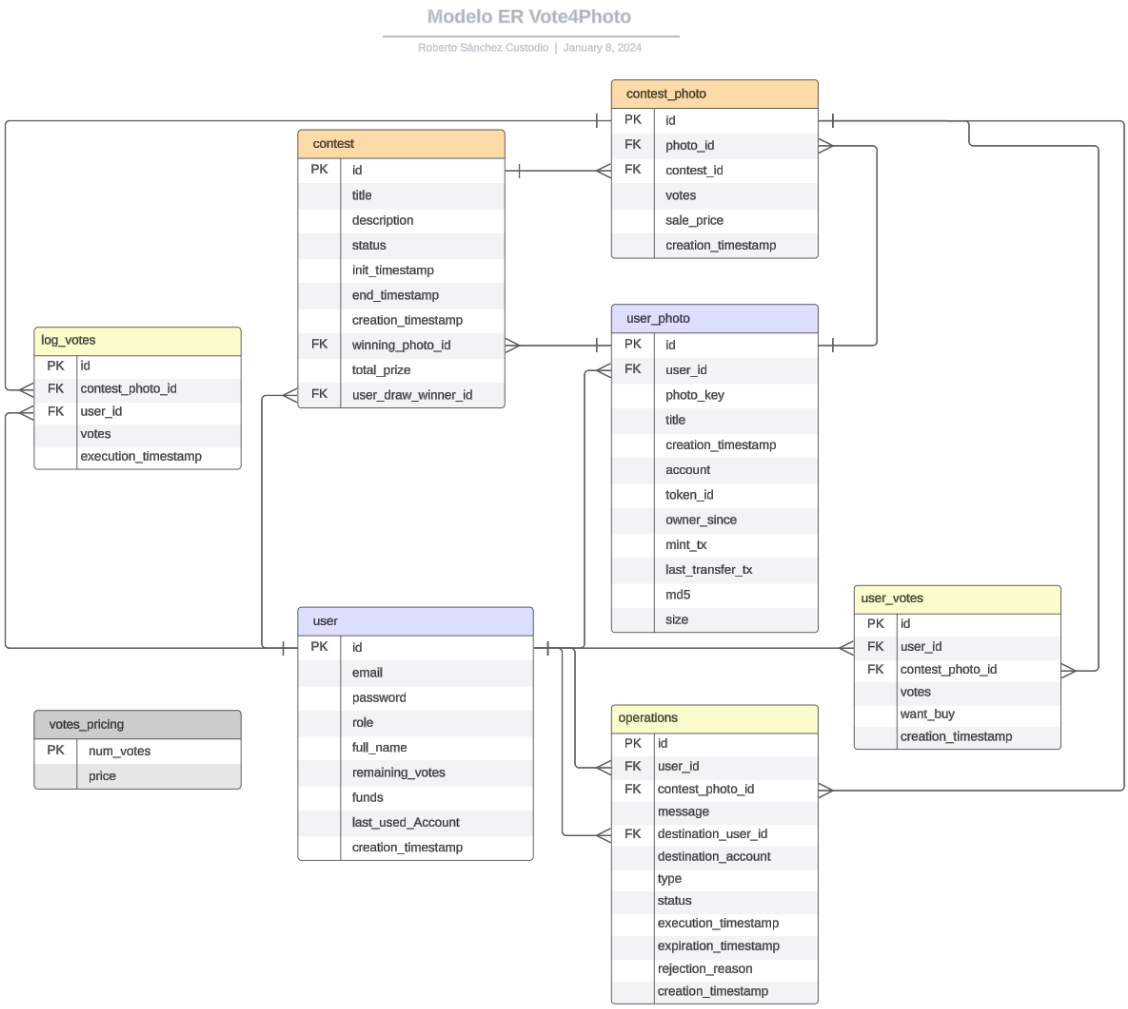

Figura 8: Diagrama ER Vote4Photo

3.5 Estructura front y back

Para el desarrollo del proyecto se utilizarán 2 repositorios GIT, uno específico para los contratos inteligentes y otro para la aplicación web.

v4p-contracts

Repositorio para la implementación de los SmartContracts que usaremos en nuestro sistema, para los que utilizaremos las herramientas provistas por HardHat para montar los entornos de testnet en local y poder probarlos antes de subirlos a la testnet definitiva, Sepolia.
La estructura del proyecto consta de los siguientes directorios y ficheros:

- **artifacts**: Artefactos generados para desplegar los contratos en una cadena de bloques.
- **cache**: Directorio temporal
- **contracts**: Directorio con la fuente de los contratos a desarrollar en Solidity
    - PhotoNFT.sol: SmartContract que implementa el token NFT y actúa como “Recipient”
    - V4PForwarder.sol: SmartContract que actúa como “Forwarder” en la ejecución de la meta-transacción
- **scripts**: Scripts para operaciones como el despliegue en cadena de bloques
- **test**: Ficheros para tests unitarios en typescript.
    - PhotoNFT.ts: Test unitarios sobre el smartcontract PhotoNFT
    - V4PForwarder.ts: Test unitarios sobre el SmartContract V4PForwarder, entre otras validaciones simulan las llamadas a meta-transacciones.
- **node_modules**: Directorio con las dependencias de terceros utilizadas
- **hardhat.config.ts**: fichero de configuración de HardHat
- **package.json**: Fichero con información sobre las dependencias y comandos para el desarrollo
- **pnpm-lock.yaml**: Fichero generado por el gestor de dependencias con la versión efectiva utilizada en cada caso
- **README.md**: fichero con información básica sobre el proyecto.
- **tsconfig.json**: Fichero de configuración para desarrollar con typescript

vote4photo

Repositorio que va a contener tanto la parte front como back, a efectos prácticos son 2 proyectos independientes, con toolkits distintos, pero que uniremos a la hora de realizar el despliegue en CloudFlare Pages, por lo que resulta conveniente que ambos convivan en el mismo repositorio.

La estructura de directorios y ficheros es la siguiente:

- **FRONT**: Directorio con el proyecto front basado en SvelteKit y Flowbite. Se describirá a continuación.
- **functions**: Directorio con la estructura definida para el API REST, basado en directorios para acceder a las diferentes rutas del API
- **pages**: Directorio con el contenido estático a desplegar, se actualizará con los ficheros generados por el FRONT, es decir, ficheros HTML, assets, bundles javascript, etc.
- **src**: Directorio con el código fuente de todos los componentes del back que no son una “Function”, en este directorio se encuentra toda la lógica de negocio de la aplicación.
- **src/scheduled**: Contiene el “entrypoint” que se desplegará como un Worker independiente para ejecutar los procesos batch de monitorización.
- **node_modules**: Directorio con las dependencias de terceros utilizadas

- **package.json**: Fichero con información sobre las dependencias y comandos para el desarrollo
- **pnpm-lock.yaml**: Fichero generado por el gestor de dependencias con la versión efectiva utilizada en cada caso
- **README.md**: fichero con información básica sobre el proyecto.
- **tsconfig.json**: Fichero de configuración para desarrollar con typescript

El directorio FRONT tiene a su vez una estructura de proyecto independiente, aunque lo adaptaremos para que los ficheros generados en el proceso de construcción se almacenen sobre la carpeta “pages” del proyecto principal, así se mantendrán ambos proyectos sincronizados. La aplicación FRONT será de tipo SPA (Single Page Application) [14]

La estructura de directorios y ficheros del front la podemos ver a continuación:

- **src**: Ficheros con el código fuente de la aplicación front, con la siguiente organización:
  - **app.d.ts**: Información de configuración de la aplicación utilizado por SvelteKit
  - **app.html**: Página base que actuará como index
  - **app.postcss**: Estilos css globales a incluir en la app
  - **lib**: Componentes comunes utilizados por las diferentes pantallas
  - **routes**: Componentes visuales organizados en directorios para determinar las rutas de cada elemento
- **static**: Recursos estáticos, principalmente imágenes
- **node_modules**: Directorio con las dependencias de terceros utilizadas
- **svelte.config.js**: Fichero de configuración de SvelteKit
- **tailwind.config.cjs**: Ficheros de configuración de Tailwind
- **vite.config.ts**: Ficheros de configuración de Vite
- **postcss.config.cjs**: Fichero de configuración de postcss, plugin utilizado por Svelte para configurar ciertos aspectos de los estilos de la aplicación
- **package.json**: Fichero con información sobre las dependencias y comandos para el desarrollo
- **pnpm-lock.yaml**: Fichero generado por el gestor de dependencias con la versión efectiva utilizada en cada caso
- **README.md**: fichero con información básica sobre el proyecto.
- **tsconfig.json**: Fichero de configuración para desarrollar con typescript

3.6 Diseño SmartContracts

El sistema hace uso de 2 SmartContracts, uno que actúa como “Forwarder” en la meta-transacción que será el encargado de validar la firma del mensaje y enviar la transacción al “Recipient”, que es el contrato que gestiona los NFT de las fotos mediante el standard ERC721.

V4PForwarder
Nuestro “Forwarder” es un contrato que extiende de ERC2771Forwarder, la implementación que ofrece OpenZeppelin del Forwarder de una meta-transacción.

Existen 2 métodos principales en el Forwarder que son:

- **verify(request)**: Comprueba que la petición enviada es correcta, incluyendo la firma de la misma.
- **execute(request)**: Valida que la petición y la firma son correctas y envía la transacción subyacente al Recipient, que describimos a continuación.

El código del contrato es realmente sencillo dado que la implementación de los métodos principales corre a cargo de la clase base de la que se extiende, sin embargo queremos destacar la sobrecarga del método interno "_validate()" para añadir un control de seguridad que evita que el contrato sea llamado por un relayer distinto del que creó el propio contrato, de esta forma nos aseguramos que el contrato solo pueda ser invocado por la cuenta que lo creó ("trustedRelayer").

```
contract V4PForwarder is ERC2771Forwarder('Vote4Photo') {

    address private trustedRelayer;
    constructor() {
        trustedRelayer = msg.sender;
    }

    /// @inheritdoc ERC2771Forwarder
    function _validate(ForwardRequestData calldata request) override internal view virtual
     returns (bool isTrustedForwarder, bool active, bool signerMatch, address signer) {
        require(msg.sender == trustedRelayer, "Unknown relayer");
        return super._validate(request);
    }
}
```

Figura 9: Código Forwarder con trustedRelayer

De igual manera que el "Recipient" implementa un control de seguridad para habilitar únicamente las meta-transacciones que son recibidas desde el "Forwarder" de confianza, decidimos aplicar un control análogo al propio "Forwarder" para proteger así la cadena de llamadas al completo.

PhotoNFT

Este contrato es el encargado de gestionar los NFTs de las fotos mediante el standard ERC721, que son el núcleo de nuestro sistema y además implementa el standard ERC2771 para poder aceptar meta-transacciones de un "Forwarder" de confianza. La dirección del "Forwarder" se añade en el constructor por lo que el despliegue debe hacerse a continuación del despliegue de V4PForwarder, utilizando su dirección en la testnet.

Existen 2 métodos en el contrato que serán invocados mediante meta-transacción:

- **mintPhoto(newTokenURI)**: El llamante crea un nuevo token a partir de la URI de la foto.
- **transferToken(to, tokenId)**: El llamante transfiere la propiedad de un token a la dirección indicada, se verifica que el llamante es el propietario del token.

El código de estos métodos es más o menos sencillo, pero destacamos el uso del método "_msgSender()", implementado en el contrato base ERC2771Recipient, en sustitución del habitual "msg.sender", para recuperar los datos de la cuenta que firma la meta-transacción y que a efectos de este contrato es la cuenta que actúa como llamante.

```
contract PhotoNFT is ERC721("V4P Photos", "Photo"), ERC2771Recipient {

    (...)
    /**
     * Crea un nuevo token a partir de una URI de una foto
     */
    function mintPhoto(string memory newTokenURI) external onlyAllowedMinter {
        uint256 tokenId = tokenCounter++;
        _mint(_msgSender(), tokenId);
        tokenURIs[tokenId] = newTokenURI;
        emit newPhotoToken(_msgSender(), tokenId, newTokenURI);
    }

    /**
     * Transfiere un token a un nuevo propietario
     */
    function transferToken(address to, uint256 tokenId) external {
        require(ownerOf(tokenId) == _msgSender(), "Not the owner of the token");
        _transfer(ownerOf(tokenId), to, tokenId);
    }
    (...)
}
```

Figura 10: Métodos llamados desde meta-transacciones

Por motivos de seguridad, en el caso de que el método para crear el token sea invocado desde una transacción standard, es decir, no desde una meta-transacción, hemos añadido un modificador al método, "onlyAllowedMinter", que limita la invocación únicamente a "mineros" permitidos, que en principio será únicamente el propietario del contrato, en términos prácticos solo permite llamadas del propietario o del "Forwarder" de confianza definido en el constructor.

4 Conclusiones y trabajos futuros

4.1 Problemas encontrados

En la parte central del proyecto no se encontraron dificultades reseñables, toda la implementación de las meta-transacciones y servicios relacionados fue relativamente bien, de hecho, se ha seguido (con algún cambio menor) la planificación inicial, sin embargo, encontramos algún problema con la infraestructura de despliegue que habría que tener en cuenta de cara a una posible evolución del proyecto o para el desarrollo de otras aplicaciones que quieran utilizar una infraestructura similar.

La BBDD elegida, Cloudflare D1, no soporta transacciones de una manera standard, solo mediante un API específico, por lo que el uso de transacciones mediante el ORM utilizado, Drizzle-ORM, no es posible, siendo esta una gran limitación si se deseara un despliegue en un entorno productivo, el manejo de transacciones en BBDD en el desarrollo de toda dApp tiene cierto desafío, ya que a menudo es necesario actualizar elementos en la BBDD y también en la cadena de bloques mediante transacciones contra SmartContracts, para acometer estos escenarios es recomendable realizar los cambios en BBDD antes que la transacciones en la blockchain, siempre que sea posible ya que deshacer el cambio frente a un error en la BBDD es siempre más sencillo (y eficiente) que hacerlo en la blockchain. Para mitigar este problema en nuestro proyecto realizamos todas las comprobaciones necesarias para minimizar la posibilidad de error antes de realizar cambios en la BBDD y en la blockchain en aquellos servicios que realicen modificaciones en ambos backends.

En consecuencia, si alguien iniciara el desarrollo de una nueva dApp desde cero sería recomendable cambiar la BBDD relacional por otra que permita el uso común de transacciones y que además pueda ser utilizada desde Cloudflare Workers&Pages y sea compatible con Drizzle-ORM, una opción que cumple todos estos requisitos podría ser Xata (https://xata.io/), un sistema basado en PostgreSQL que es compatible con Drizzle-ORM y que se integra con la plataforma de Cloudflare.

Otro problema encontrado con el stack utilizado está relacionado con la construcción del “front”, el framework Svelte utilizado ha cumplido correctamente con su cometido durante todo el desarrollo de la aplicación hasta que se llegó al momento del despliegue de la aplicación ya finalizada. Al contrario de otros frameworks de front más “puros” como Angular, ReactJS o Vue.js, el enfoque por defecto de Svelte es distinto, al plantear un mecanismo “híbrido” donde se reduce al mínimo posible el código y los recursos que se ejecutan y se envían al cliente, realizando la mayor parte del trabajo en el servidor, es decir, en cierto modo, la configuración por defecto le permite actuar como front y como back, pero en nuestro escenario, al desarrollar el back con otro framework (buscando una arquitectura headless) necesitamos que Svelte actúe exclusivamente en el “front”, lo cual es posible, pero aplica ciertas limitaciones cuando se desarrolla una SPA (Single Page Application), por ejemplo, no permite rutas dinámicas como `/api/contests/{contestId}`, en nuestro proyecto se disponía de una única ruta dinámica y se ha resuelto guardando el identificador del concurso en el localStorage, no es una solución ideal para una aplicación real, pero resulta transparente y efectiva para el usuario en nuestro escenario.

Si se iniciara el proyecto desde cero, existen 2 posibilidades para evitar el problema descrito:

- Usar Svelte como framework fullstack, según la documentación oficial dispone de un adaptador para Cloudflare Pages.
- Usar una alternativa a Svelte para el “front”, como pueden ser Vue.js, ReactJS o Angular, manteniendo el desarrollo del back independiente.

4.2 Conclusiones

Se plantearon 3 objetivos al inicio de este proyecto, en base a estos objetivos podemos obtener las siguientes conclusiones:

- La meta-transacciones son relativamente sencillas de implementar, el trabajo adicional para que un contrato las soporte es reducido en comparación al trabajo que conlleva implementar un sistema alrededor del contrato o contratos relacionados, es decir, es una opción perfectamente viable si queremos solucionar el "problema de la tasa de gas" en las transacciones con SmartContracts.
- El sistema resultante resulta sencillo de usar para cualquier usuario, incluso si es ajeno al mundo blockchain/NFTs, un creador de contenidos, como puede ser un fotógrafo o un artista digital, puede exponer sus obras de manera sencilla y manteniendo el control sobre la misma en todo momento y sin necesidad de adquirir tokens u otro tipo de criptomoneda.
- Todo aficionado/a al mundo de las NFTs puede utilizar la plataforma diseñada sin riesgo de costes indirectos "sorpresa", todo está predefinido de antemano.

Respecto al impacto ético-social, de sostenibilidad y de diversidad, se mantiene el enfoque expuesto en el apartado correspondiente y teniendo en cuenta la finalización del proyecto, podemos confirmar que el trabajo resultante representa un servicio global y abierto para todo tipo de individuos, donde se respecta la privacidad y anonimato de todos los usuarios, no se comparten datos personales, solo las fotos que cada usuario quiera inscribir.

Una reflexión respecto a las conclusiones anteriores, si bien la solución técnica aplicada es viable y sencilla de implementar, para una empresa que desee diseñar un modelo de negocio relacionado con la blockchain y donde los usuarios no tengan que preocuparse del coste de la tarifa del gas, existe una incertidumbre inevitable y se trata de cómo trasladar al usuario de manera indirecta los costes del gas que asume la empresa para que el modelo de negocio sea sostenible, es por ello que hay un factor que sería importante analizar antes de optar por las meta-transacciones como solución al problema del gas y no es otro que la eficiencia global respecto al consumo de gas, teniendo en cuenta que la totalidad del coste de dicho consumo es asumido por la empresa propietaria del servicio, en el siguiente punto, como parte de posibles trabajos futuros, recomendamos comparar la solución aplicada en el presente proyecto con otra basada en el futuro standard ERC-4337 [5].

4.3 Trabajos futuros

Como evolución del trabajo se plantean diversos escenarios, por un lado sería interesante poder comparar la solución a la tasa del gas que hemos propuesto en el presente proyecto con otras opciones, como la que vimos en el apartado "Alternativas a las meta-transacciones" de la sección Qué es una meta-transacción, nos referimos al futuro standard ERC-4337 [5] que añade el concepto de "Account Abstration", realizar las transacciones que hemos implementado a

través de meta-transacciones en este proyecto utilizando el mencionado standard permitiría comparar ambos enfoques desde un punto de vista de viabilidad técnica y sobre todo de eficiencia.
De cara a optar por una u otra solución, es probable que la solución más eficiente sea también la que implique un mayor esfuerzo o dificultad técnica, por lo que será necesario buscar un equilibrio en función del tipo de servicio.

Por otro lado, de cara a plantear un posible uso real de una dApp que se base en el presente trabajo, se podría reutilizar la mayor parte del stack tecnológico propuesto, aunque también serían necesarios o al menos recomendables algunos cambios, como son:

- Uso de un BBDD relacional, p. ej: Xata, que soporte transacciones y sea compatible con el resto del stack utilizado, como ya vimos en el apartado Problemas encontrados.
- Uso de un sistema de almacenamiento de las imágenes basado en IPFS (InterPlanetary FileSystem).
- Uso de un Gateway de acceso a la blockchain, como el provisto por el propio Cloudflare, Web3, que agilice y añada robustez a la operativa con los SmartContracts y al acceso a IPFS.

5 Glosario

Término	Descripción
NFT	Non-Fungible Token (Token no fungible). Representa una propiedad única e indivisible de un activo digital que es almacenado en la cadena de bloques (blockchain)
Blockchain	Cadena de bloques que almacena transacciones digitales securizadas por mecanismos criptográficos que puede ser utilizada para operaciones con criptomonedas o NFTs, entre otras utilidades.
Testnet	Nos referimos con testnet a las cadenas de bloques disponibles en internet pero cuyo fin es meramente de pruebas, utilizadas por los desarrolladores de SmartCotnracts para desplegar los contratos en un entorno similar al real, pero sin la necesidad de necesitar Ether oficial, cada testnet tiene su propia moneda virtual.
Wallet	El wallet o billetera asociada al mundo blockchain representa un mecanismo para el almacenamiento y gestión de activos digitales, como criptomonedas o NFTs, por parte de los usuarios de la plataforma.
dApp	Una Aplicación Descentralizada (Decentralized Application), es un tipo de aplicación de software que se ejecuta en una red descentralizada de nodos, como una red P2P o una cadena de bloques (blockchain). Una parte importante de una dApp es el/los SmartContracts con los que debe interactuar.
SmartContract	Los contratos inteligentes son programas informáticos autónomos que se almacenan y ejecutan en cadenas de bloques (blockchain), como por ejemplo en Ethereum. En cierta manera actúan como el backend de una dApp
Solidity	Lenguaje de programación diseñado para la construcción de SmartContracts

6 Bibliografía

[1] ethereum.org, «Definición tarifa de Gas,» [En línea]. URL: https://ethereum.org/es/developers/docs/gas/. [Último acceso: 9.10.2023].

[2] ethereum.org, «Standard ERC-2771,» [En línea]. URL: https://eips.ethereum.org/EIPS/eip-2771. [Último acceso: 9.10.2023].

[3] OpenGSN.org, «ETHless transactions made possible,» [En línea]. URL: https://opengsn.org/. [Último acceso: 1.11.2023].

[4] H. Jadeja, «Problema del standard ERC-2771,» [En línea]. URL: https://www.alchemy.com/overviews/meta-transactions. [Último acceso: 9.10.2023].

[5] ethereum.org, «Standard ERC-4337,» [En línea]. URL: https://eips.ethereum.org/EIPS/eip-4337. [Último acceso: 9.10.2023].

[6] S. C. Qin Wang, «Account Abstraction, Analysed,» [En línea]. URL: https://arxiv.org/pdf/2309.00448.pdf. [Último acceso: 3.11.2023].

[7] wikipedia.org, «Algoritmo de consenso Prueba de Apuesta (Proof of Stake),» [En línea]. URL: https://es.wikipedia.org/wiki/Prueba_de_apuesta. [Último acceso: 9.10.2023].

[8] wikipedia.org, «Algoritmo de consenso Prueba de Trabajo (Proof-of-Work),» [En línea]. URL: https://es.wikipedia.org/wiki/Sistema_de_prueba_de_trabajo. [Último acceso: 9.10.2023].

[9] Zack Bloom (cloudflare.com), «Isolate model without containers,» [En línea]. URL: https://blog.cloudflare.com/cloud-computing-without-containers/. [Último acceso: 9.10.2023].

[10] A. S. (elpais.es), «Noticia de la quiebra de FTX,» [En línea]. URL: https://elpais.com/economia/2022-11-11/la-plataforma-de-criptomonedas-ftx-se-declara-en-bancarrota-y-amenaza-con-provocar-un-efecto-contagio.html. [Último acceso: 3.10.2023].

[11] vuestorefront.io, «Arquitectura headless,» [En línea]. URL: https://vuestorefront.io/blog/headless-architecture. [Último acceso: 3.10.2023].

[12] ethereum.org, «Standard ERC-721,» [En línea]. URL: https://eips.ethereum.org/EIPS/eip-721. [Último acceso: 20.10.2023].

[13] ethereum.org, «Testnet recomendada para desarrollo con Ethereum,» [En línea]. URL: https://ethereum.org/en/developers/docs/networks/#sepolia. [Último acceso: 9.10.2023].

[14] wikipedia.org, «Single Page Application,» [En línea]. URL: https://en.wikipedia.org/wiki/Single-page_application. [Último acceso: 5.11.2023].

7 Anexos

7.1 Anexo I. Historias de usuario

US-1 **Alta de usuario**

La aplicación permitirá el alta de cualquier usuario de manera gratuita, con el único requisito de disponer de un Wallet en el blockchain manejado por la aplicación, en este caso la testnet Sepolia y de un gestor de wallets como Metamask instalado en el navegador como extensión.
El alta del usuario requerirá los siguientes campos:

- Nombre completo
- Email
- Password (se requerirá confirmar la password introducida)
- Chain ID: identificador de la red blockchain utilizad, para validar que coincide con la utilizada por el sistema.
- Cuenta en la blockchain que se obtendrá automáticamente enlazando con Metamask.

No es necesario que el usuario disponga de fondos en su cuenta de la testnet.
Al crear la cuenta el usuario queda automáticamente autenticado, se generará un token análogo al creado en el proceso de login

US-2 **Login de usuario**

Cualquier usuario del sistema podrá acceder mediante su email y su password.
El proceso de login debe incluir la Chain ID y la cuenta (dirección) del usuario en la blockchain seleccionada a través del wallet del navegador.
Además de validar las credenciales del usuario se valida que la red blockchain seleccionada sea la red utilizada por el sistema
Al realizar el login se generará un token JWT que se utilizará en sucesivas llamadas al back.
Se debe incluir en el token JWT información básica del usuario como el id, nombre o role y también la cuenta en la red blockchain.

US-3 **Roles de acceso**

La aplicación manejará 2 roles:

- **USER**: Usuario standard que podrá tanto votar como ofrecer fotos para concursar

- **ADMIN**: Usuario administrador que podrá dar de alta concursos

US-4	**Transferencia de fondos**

Todos los usuarios podrán transferir fondos en moneda fiduciaria a su cuenta de la aplicación, dichos fondos se podrán utilizar para comprar derechos de voto o comprar fotografías.

La transferencia simulará conexión con un TPV virtual, no será una integración real, el objetivo es mostrar cómo sería la operativa de una manera relativamente realista, es decir, sin hacer uso de dinero real, solo simular cómo sería dicha transferencia, similar a la compra de cualquier artículo en una tienda online.

Para solicitar la transferencia se mostrará un dialogo al usuario donde se solicitará una cantidad, al aceptar la operación los fondos del usuario se incrementarán con la cantidad indicada.

US-5	**Compra de derechos de voto**

Un usuario podrá comprar derechos de voto para luego utilizar en los concursos, en cada concurso se podrá votar por más de una foto y para una misma foto se podrán enviar N votos.

Desde el perfil del usuario se podrán adquirir nuevos derechos de voto a partir de los fondos de la cuenta, el precio de cada derecho a voto variará en función de la cantidad a comprar, las cantidades y el precio será parametrizable internamente, a modo de ejemplo se mostrarán las siguientes opciones:

- Menos de 10 votos: 0.50 €/voto
- 10 votos: 3.5 €
- 35 votos: 8.75 €
- 100 votos: 15 €

US-6	**Creación de concursos**

El usuario administrador podrá crear concursos indicando los siguientes campos:

- Título
- Descripción: Indicación de temática o cualquier otra característica del concurso relevante para los usuarios.
- Fecha/hora de inicio
- Fecha/hora de fin

Existirá la opción de no indicar fecha de inicio, de manera que el concurso se iniciará automáticamente.

US-7 **Añadir nueva foto a concurso**

Los usuarios podrán subir fotos a su cuenta para asociarlas a alguno de los concursos abiertos o pendientes de iniciar. Una foto, una vez inscrita en un concurso, no podrá inscribirse en otro hasta que el concurso donde está inscrita termina y no es objeto de transferencia.

Para inscribir la nueva foto, se le solicitarán los siguientes datos:

- La foto a inscribir
- Título de la foto
- Valor de venta: En el caso de que alguien quiera comprar la foto, indicar el valor de venta de la misma.

Para incluir la foto en el concurso se precisará que el usuario pague una cuota de inscripción en dinero fiduciario de los fondos disponibles en su cuenta.

El proceso de inscripción requiere necesariamente de 2 pasos:

1. El usuario sube la foto con el título, la cual se almacena en el repositorio de fotos donde se genera una clave necesaria para montar el descriptor JSON que usaremos como URI para el token NFT. Antes de aplicar los cambios se valida que el usuario disponga de fondos suficientes. Como resultado el servidor retornará el mensaje que debe firmar el usuario para enviar a la meta-transacción.
2. La foto aparecerá en pantalla y al aceptar las condiciones el wallet del navegador firmará (con la cuenta con la que se hizo login) el mensaje recuperado y con la firma generada se envían al servidor todos los datos para realizar la meta-transacción una vez retirados los fondos de la tasa de inscripción de la cuenta del usuario. Como resultado se creará un token ERC721 (NFT) y se añadirá al wallet del usuario.

Al finalizar el proceso, el wallet del usuario que ha enviado la foto será propietario del token NFT generado con la foto enviada sin haber hecho uso de ninguna cantidad de Ether de su cuenta.

US-8 **Añadir foto existente a concurso**

Las fotos que no ganan el concurso y que tampoco son compradas se mantienen asociadas al usuario original, pudiendo inscribirlas en otros concursos que estén pendientes de iniciar o ya iniciados.

La inscripción al concurso también requerirá de una cuota, pero de menor cuantía, dado que el coste de creación del NFT de la foto no aplica en este caso.

Para inscribir la foto existente, se le solicitarán los siguientes datos:

- La foto a inscribir de la lista de fotos del usuario.
- Valor de venta: En el caso de que alguien quiera comprar la foto, indicar el valor de venta de la misma. El usuario podrá modificar el valor para cada concurso.

US-9 **Monitorización batch**

Existirá un proceso automático, que se ejecuta cada minuto para cambiar el estado de los concursos a “iniciados” o “finalizados”, según se alcancen las fechas (y horas) de inicio y fin. También cambiará el estado de las operaciones que hayan caducado a “rejected”.

En el cambio a “finalizado” de los concursos el proceso realizará las siguientes tareas:

- Determinará la foto ganadora del concurso.
- Determinará el premio para la foto ganadora, en función de los votos recibidos.
- Realizará un sorteo ponderado en función de los votos emitidos entre los votantes de la foto ganadora.
- Creará una operación de transferencia de la foto ganadora al ganador del sorteo entre los votantes.
- Creará una operación de venta de las fotos por cada foto que ha participado, excepto la ganadora, para ello se realizará un sorteo ponderado por los votos emitidos, entre todos los votantes que han marcado la opción de compra de la foto, se creará una operación por cada foto, siempre que haya al menos un votante con intención de compra.

Todas las operaciones creadas deben tener una fecha de caducidad, que en caso de alcanzarse sin haber sido resuelta por el usuario asignado se pasará a un estado “rejected”, que tendrá implicaciones distintas en función del tipo de operación.

US-10 **Home de la aplicación**

Al acceder a la aplicación, se mostrará una pantalla con diferentes secciones:

- **Operaciones pendientes:** Listado de todas las operaciones que requieren intervención del usuario conectado.

- **Operaciones ejecutadas:** Listado de operaciones que han sido ejecutadas por el usuario con la fecha de ejecución
- **Operaciones rechazadas:** Listado de operaciones que han sido rechazadas, con la fecha y el motivo de rechazo.

US-11 **Perfil del usuario**

El usuario podrá acceder en todo momento a su perfil donde podrá ver información básica como su nombre o email y los fondos de los que dispone actualmente, así como la lista de tokens de fotos de los que dispone, bien por ser el propietario original de la foto o bien por haberla adquirido desde la plataforma.

Desde el perfil del usuario se podrán traspasar fondos pulsando en un botón que mostrará el dialogo con la operación de traspaso de fondos.

También se podrán comprar derechos de voto desde esta pantalla.

US-12 **Detalle de un concurso**

Cada concurso tendrá una página donde se mostrarán los datos del mismo y el listado de fotos que se inscribieron.

La información a mostrar de cada foto en concursos Activos:

- Miniatura de la foto que podrá ampliarse para verla en detalle.
- Título de la foto
- Valor de venta
- Un botón para votar por la foto.
- Un indicador con los votos propios que hemos realizado en cada foto

La información a mostrar de cada foto en concursos ya finalizados:

- Miniatura de la foto que podrá ampliarse para verla en detalle.
- Título de la foto
- Premio recibido en el caso de ser la foto ganadora
- Algo que la destaque como foto ganadora si es el caso
- Premio en moneda fiduciaria que ha ganado el autor de la foto

- Fecha en la que se subió a la plataforma.

El orden de las fotos en concursos activos será aleatorio, cada vez que se visite la página, pero el orden en concursos finalizados estará determinado por el número de votos, mostrando al principio las fotos con más votos.

US-13 **Votar por una foto**

Desde el listado de fotos de un concurso se podrá votar por una de las fotos inscritas pulsando en el botón de voto de la foto en cuestión, al pulsarlo aparecerá un dialogo, con información de la foto, como:

- Título
- Precio de venta
- La propia foto
- Un input con el total de votos que se desean dar a la foto, por defecto será 1 y como máximo el número de votos disponibles en la cuenta.
- Check indicando si se tiene intención de comprar la foto.

El número de fotos afectará a la probabilidad de ser elegido como ganador del sorteo para conseguir la foto ganadora, a mayor número de votos, más probabilidad de ser elegido.

US-14 **Operación de traspaso de foto ganadora**

Al finalizar un concurso, el propietario de la foto ganadora tendrá una operación asociada para realizar el traspaso de la foto al usuario ganador del sorteo entre los votantes de la foto.

La operación pendiente tendrá una fecha de caducidad de 1 semana (parámetro interno de la app). En el caso de llegar la fecha de expiración sin haber atendido la operación, se rechazará automáticamente.

Para resolver la operación, el usuario podrá aceptarla o rechazarla, en el caso de que el premio en metálico no lo considere suficiente o por cualquier otro motivo, opcionalmente podrá indicar un motivo del rechazo.

Si la operación es aceptada, el proceso es el siguiente:

- El sistema generará una meta-transacción para el traspaso del token de la foto
- El propietario de la misma firmará la meta-transacción con Metamask. No se requieren fondos en su wallet para esta operación.

- El sistema envía la meta-transacción al contrato Forwarder que a su vez realizará la transacción de traspaso al nuevo propietario del token sobre el contrato que gestiona los tokens ERC721.
- Una vez realizada la transacción en la blockchain, el usuario que ganó el sorteo entre los votantes de la foto ganadora podrá disponer de ella dentro del sistema y aparecerá en su wallet como NFT.
- El propietario original de la foto y ganador del concurso recibirá en los fondos de su cuenta el premio en metálico (en moneda fiduciaria) por haber ganado el concurso y haber transferido el token de la foto ganadora.

Si la operación es rechazada, el proceso es el siguiente:

- El propietario de la foto la mantiene en su wallet, pero no recibe el premio en metálico por haber ganado el concurso.
- El ganador del sorteo, recibirá en su cuenta el premio en metálico en lugar de la foto.
- Se generará una operación de notificación para el usuario que recibe los fondos indicando la cantidad recibida, si el propietario rechaza el premio.

US-15 **Operación de compra**

Los votantes de las fotos que no han resultado ganadoras en el concurso podrán tener derecho a comprar la foto votada por la cantidad indicada por el propietario en la inscripción al concurso.

La operación pendiente tendrá una fecha de caducidad de 1 semana (parámetro interno de la app). En el caso de llegar la fecha de expiración sin haber atendido la operación, se rechazará automáticamente.

Si el usuario acepta la compra:

- Se generará una nueva operación de tipo venta, para que el propietario de la foto realice la transferencia.
- Se bloquearán los fondos necesarios para la compra, restándolos de los fondos de la cuenta compradora.

Si el usuario cancela la compra:

- No se realizará la transferencia del token, manteniéndose vinculado al propietario original.
- Se ingresarán los fondos necesarios para volver a inscribir la foto en otro concurso, de manera que el propietario de la foto pueda tener otra oportunidad para venderla.

US-16	**Operación de venta** Una vez aceptada la operación de compra se genera una nueva operación de venta asociada al propietario de la foto, para que proceda a realizar la transferencia de la misma al comprador. La operación pendiente tendrá una fecha de caducidad de 1 semana (parámetro interno de la app). En el caso de llegar la fecha de expiración sin haber atendido la operación, se rechazará automáticamente. Si el usuario acepta la venta: • El sistema generará una meta-transacción para el traspaso del token de la foto • El propietario de la misma firmará la meta-transacción con Metamask. No se requieren fondos en su wallet para esta operación. • El sistema envía la meta-transacción al contrato Forwarder que a su vez realizará la transacción de traspaso al nuevo propietario del token sobre el contrato que gestiona los tokens ERC721. • Una vez realizada la transacción en la blockchain, el usuario que compró la foto podrá disponer de ella dentro del sistema y aparecerá en su wallet como NFT. • El vendedor recibirá en los fondos de su cuenta el valor de venta estipulado en moneda fiduciaria. Si el usuario rechaza la venta: • No se realizará traspaso del token • El comprador recuperará los fondos retenidos para compra del token con la foto. • Se reintegrarán los votos utilizados en el concurso para no penalizar al comprador en la inversión realizada para conseguir la foto.

7.2 Anexo II. API Servicios REST

Servicios relacionados con los usuarios

Método	URI y descripción del servicio
POST	**`/api/account/signup`** Servicio de acceso público Creará una cuenta de usuario, recibiendo los siguientes campos:

	• **fullName**: Nombre completo • **email**: Email del usuario • **password**: Password de acceso • **account**: Dirección de la cuenta activa en el Wallet en la blockchain • **chainId**: identificador numérico de la red blockchain El servicio retornará el ID del usuario creado y un token JWT, la creación de la cuenta realiza un login automático: • **id**: Identificador del usuario creado • **jwt**: Token de autenticación para utilizar en las llamadas sucesivas al back
POST	**/api/account/signin** Servicio de acceso público Realizará el login en el sistema con los parámetros: • **email**: Nombre completo • **password**: Email del usuario • **account**: Dirección de la cuenta activa en el Wallet en la blockchain • **chainId**: identificador numérico de la red blockchain El servicio retornará un token JWT si la autenticación es exitosa: • **token**: Token de autenticación para utilizar en las llamadas sucesivas al back

GET	**/api/account/{user_id}**
	
	Servicio de acceso restringido, requiere un JWT válido.
	Recupera todos los datos del usuario pasado por parámetro. Para los usuarios con rol USER, se verifica que el parámetro user_id coincide con el ID del propio usuario, en caso contrario se lanza un error.
	
	Los datos retornados serán:
	• **id**: Identificador del usuario
	• **fullName**: Nombre completo
	• **email**: Email del usuario
	• **role**: Rol del usuario
	• **password**: Password de acceso
	• **funds**: Fondos disponibles
	• **remainingVotes**: Votos disponibles
	• **photos**: Lista de fotos con datos básicos:
	  ○ **id:** ID de la foto
	  ○ **title:** Título de la foto
	  ○ **photoKey:** Clave de la foto en el repositorio S3
	  ○ **tokenId:** Identificador del token NFT

	○ **size:** Tamaño en bytes ○ **account:** Cuenta de la blockchain propietaria del token ○ **mintTx:** Transacción en la que se minó el token ○ **lastTransferTx**: Transacción de la blockchain en la que se transfirió por última vez ○ **currentContestId**: Identificador del concurso actual, si se encuentra asociado a uno ○ **currentContestTitle**: Título del concurso asociado si existe ○ **ownerSince:** Fecha desde que se es propietario
PUT	**`/api/account/{user_id}`** Servicio de acceso restringido, requiere un JWT válido. Para los usuarios con rol USER, se verifica que el parámetro user_id coincide con el ID del propio usuario, en caso contrario se lanza un error. Modifica datos básicos del usuario, los parámetros permitidos son: • **fullName**: Nombre completo • **password**: Password de acceso Retorna los datos del usuario actualizados excepto el campo password que no es retornado en ningún caso, ni en este ni en otros servicios.
POST	**`/api/account/funds`** Servicio de acceso restringido, requiere un JWT válido. Añade nuevos fondos al usuario conectado • **amount**: Cantidad en moneda fiduciaria a añadir a la cuenta del usuario. Este servicio asume una teórica integración con TPV virtual de dónde obtener los fondos, en nuestro caso todo es simulado, para simplificar el sistema y no tratar con necesidad de dinero real. Retornará los fondos del usuario actualizados: • **funds**: Cantidad en moneda fiduciaria asociada a la cuenta del usuario conectado
POST	**`/api/account/buyvotes`** Servicio de acceso restringido, requiere un JWT válido. Compra derechos de voto para el usuario

	• **amount**: Cantidad de votos a comprar Retornará los votos totales actualizados, siempre que el usuario disponga de fondos suficientes para adquirir los derechos de voto. • **remainingVotes**: Votos disponibles • **funds**: Cantidad en moneda fiduciaria asociada a la cuenta del usuario conectado

Servicios relacionados con los concursos

Método	URI y descripción del servicio
POST	**/api/contest** Servicio de acceso restringido, requiere un JWT válido y rol ADMIN del usuario conectado. Creará un nuevo concurso con los siguientes datos: • **title**: Nombre completo • **description**: Email del usuario • **initTimestamp**: Fecha/hora de inicio del concurso. Si es null se utiliza la fecha del sistema y se activa el concurso. Formato ISO. Si es null se asume la fecha actual y se inicia el concurso automáticamente • **endTimestamp**: Fecha/hora de finalización del concurso. Formato ISO El servicio retornará el nuevo concurso: • **id**: Identificador del concurso • **title**: Nombre completo • **description**: Email del usuario • **initTimestamp**: Fecha/hora de inicio del concurso • **endTimestamp**: Fecha/hora de finalización del concurso • **status**: Estado del concurso (‘pending’ o ‘active’)
POST	**/api/contest/{contest_id}/init** Servicio de acceso restringido, requiere un JWT válido y rol ADMIN del usuario conectado. Adelantará la activación del concurso al momento actual, cambiando los campos initTimestamp y status. A partir de este momento los usuarios ya pueden votar las fotos inscritas. Solo será posible iniciar concursos que están en estado pendiente.

	El servicio retornará los nuevos datos del concurso: • **initTimestamp**: Fecha y hora de inicio del concurso, no puede ser anterior al momento actual. • **status**: Nuevo estado del concurso (‘active’)
PUT	**/api/contest/{contest_id}** Servicio de acceso restringido, requiere un JWT válido y rol ADMIN del usuario conectado. Modificará un concurso existente con los siguientes datos: • **title**: Título del concurso • **description**: Descripción del concurso, puede incluir detalle sobre la temática buscada • **initTimestamp**: Fecha y hora de inicio del concurso, no puede ser anterior al momento actual. • **endTimestamp**: Fecha/hora de finalización del concurso Solo será posible modificar concursos que están pendientes de iniciar (status = ‘pending’). El servicio retornará los nuevos datos del concurso: • **id**: Identificador del concurso • **title**: Título del concurso • **description**: Descripción del concurso, puede incluir detalle sobre la temática buscada • **initTimestamp**: Fecha y hora de inicio del concurso, no puede ser anterior al momento actual. • **endTimestamp**: Fecha y hora de finalización

GET	**/api/contest/{contest_id}**
	
	Servicio de acceso restringido, requiere un JWT válido.
	Obtiene los datos de un concurso para el id contest_id
	
	El servicio retornará los datos del concurso:
	• **id**: Identificador del concurso
	• **title**: Título del concurso
	• **description**: Descripción del concurso, puede incluir detalle sobre la temática buscada
	• **initTimestamp**: Fecha y hora de inicio del concurso, no puede ser anterior al momento actual.
	• **status**: Estado del concurso: PENDING, ACTIVE, CLOSED
	• **endTimestamp**: Fecha y hora de finalización (init + duración)
	• **photos**: Lista de fotos inscritas con datos básicos:
	  ○ **contestPhotoId:** ID de la foto
	  ○ **photoKey:** Clave de la foto en el repositorio S3
	  ○ **title:** Título de la foto
	  ○ **size:** Tamaño de la foto

	○ **price:** Valor de venta ○ **photoId:** ID de la foto ○ **ownVotes:** Número de votos propios
POST	**`/api/contest/{contest_id}/addphoto`** Servicio de acceso restringido, requiere un JWT válido. Si la foto indicada no existe en el sistema, crea a partir de foto una meta-transacción que generará el token NFT e inscribirá la foto en el concurso indicado por contest_id, si la foto ya existe, la asocia directamente al concurso. Recibirá los siguientes datos: • **photoKey:** Clave de la foto • **salePrice:** Precio de venta • **signedMessage:** Mensaje original a firmar con la correspondiente firma realizada por el usuario El servicio retornará los datos de la foto en el concurso: • **photoId:** Identificador de la foto en BBDD • **contestPhotoId:** Identificador de la foto en el concurso • **title:** Título de la foto • **size:** Tamaño de la foto • **price:** Precio de venta • **photoKey:** Clave de la foto
POST	**`/api/contest/{contest_id}/vote`** Servicio de acceso restringido, requiere un JWT válido. Añade votos del usuario conectado a una foto en el concurso contest_id. Recibirá los siguientes datos: • **photoId**: Identificador de la foto • **votes**: Número de votos a añadir a la foto en el concurso. Si el usuario no dispone de votos suficientes en su cuenta se lanzará un error y no se aplicará ningún voto sobre la foto. • **wantBuy**: Determina si el votante de la foto está dispuesto a comprarla. El servicio retornará los datos: • **contestPhotoId**: identificador de la foto en el concurso • **remainigVotes**: Votos restantes del usuario • **logVoteId**: Identificador del registro de votos realizado • **votes**: Total de votos añadidos

Servicios relacionados con las operaciones

Método	URI y descripción del servicio
GET	**/api/operation** Servicio de acceso restringido, requiere un JWT válido. Retorna la lista de operaciones pendientes asociadas al usuario conectado y las operaciones ejecutadas o rechazadas recientemente. El servicio retornará la lista de operaciones con estos datos: • **id**: Identificador de la operación • **type**: Tipo de operación ('accept_prize', 'notification', 'buy', 'sell') • **contestPhotoId**: Identificador de la foto en el concurso sobre la que se realizará la operación. • **status**: Estado de la operación. • **message**: Texto asociado a la operación • **expirationTimestamp**: Fecha de expiración de la operación • **executionTimestamp**: Fecha de ejecución si existe • **rejectionTimestamp**: Fecha de rechazo si existe • **rejectionReason**: Motivo del rechazo si procede • **photoId**: Identificador de la foto • **salePrice**: Valor de venta • **title**: Título de la foto • **photoKey**: Identificador de la foto en S3
GET	**/api/operation/{operation_id}/data_to_sign** Servicio de acceso restringido, requiere un JWT válido. Retorna los datos en formato JSON que el usuario conectado deberá firmar con su wallet Metamask para iniciar la meta-transacción • **messageToSign:** Mensaje que será enviado al wallet para su firma • **domain**: Dominio asociado al SmartContract
POST	**/api/operation/{operation_id}/signature** Servicio de acceso restringido, requiere un JWT válido. Se ejecuta la operación (venta (sell) o aceptación del premio del concurso (accept_prize)) para ello se envía la firma del mensaje recibido en /data_to_sign para poder iniciar la meta-transacción. • **signature**: Firma en formato hexadecimal (string) El resultado del servicio retorna el nuevo estado de la operación

	• **id:** Identificador de la operación • **type:** Tipo de operación • **status:** Estado resultante de la operación • **executionTimestamp:** Fecha de la ejecución
POST	**`/api/operation/{operation_id}/execute`** Servicio de acceso restringido, requiere un JWT válido. Ejecuta la operación (tipos: 'buy' o 'notification') identificada por operation_id por parte del usuario conectado. Retorna la operación con el estado EXECUTED: • **id**: Identificador de la operación • **status**: EXECUTED
POST	**`/api/operation/{operation_id}/reject`** Servicio de acceso restringido, requiere un JWT válido. Rechaza la ejecución de la operación identificada por del operation_id por parte del usuario conectado. Recibe como parámetro opcional el motivo de rechazo: • **rejectionReason**: Motivo de rechazo de la operación. Retorna la operación con el estado REJECTED: • **id**: Identificador de la operación • **status**: REJECTED

Servicios sobre fotos

Método	**URI y descripción del servicio**
POST	**`/api/photo/prepare`** Servicio de acceso restringido, requiere un JWT válido. Sube una nueva foto al sistema provisionalmente. Recibirá los siguientes datos: • **photo**: Datos binarios de la foto • **title**: Título de la foto El servicio retornará los datos de la nueva foto: • **photoKey**: Clave de la foto en S3

	• **messageToSign**: Mensaje con los datos de la transacción para ser firmados por el wallet del usuario • **domain**: Dominio asociado al SmartContract
DELETE	**/api/photo/{photo_key}** Servicio de acceso restringido que requiere un token JWT válido. Borra la foto en el repositorio S3 siempre que no haya sido almacenada en BBDD. Si existe borra también el token URI, el cual se calculará a partir del photoKey. El servicio no retornará datos
GET	**/api/photo/{photo_key}** Servicio que retorna la propia foto Se utilizará como URL para la generación del NFT de la foto, también retornará el fichero json con los metadatos de la foto en la blockchain.

Otros servicios

Método	URI y descripción del servicio
GET	**/api/config/votes_pricing** Servicio de acceso restringido, requiere un JWT válido. Retorna la lista de precios de los lotes de votos: El servicio retornará un listado con los siguientes campos en cada elemento: • **amount**: Cantidad de votos en el lote • **price**: Precio del lote
GET	**/api/config/fees** Servicio de acceso restringido, requiere un JWT válido. Retorna la lista de tarifas para participar en un concurso. El servicio retornará un objeto con los siguientes campos: • **CONTEST**: Tasa por participar con una foto existente en el sistema (ha sido utilizada en otro concurso previamente) • **CONTEST_NEW_PHOTO**: Tasa por participar con una foto nueva
GET	**/api** Servicio de acceso público Retorna el nombre y versión del API

- **name**: Nombre de la aplicación
- **version**: Versión actual desplegada

7.3 Anexo III. Meta-transacción por usuario

Se ha definido en detalle en secciones anteriores qué es una meta-transacción a nivel conceptual y a nivel técnico, en este apartado describiremos visualmente cómo interactúa el usuario con la aplicación para realizar la meta-transacción, en particular en el proceso de añadir una foto a un concurso y como se acaba creando el token NFT sobre la misma.

En primer lugar, el usuario accede a la aplicación indicando sus credenciales y la cuenta que tiene actualmente activa en su wallet.

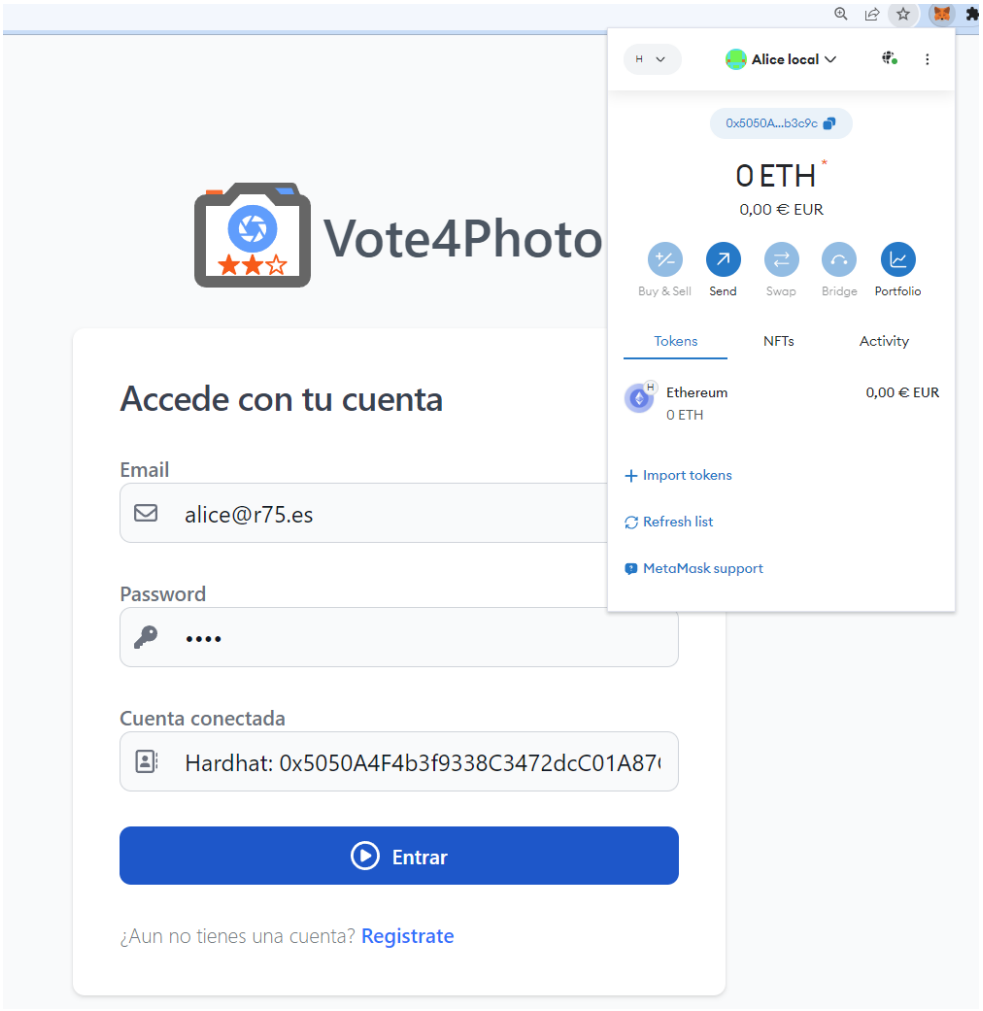

Figura 11: Acceso a la aplicación con una cuenta con 0 ETH

Como podemos apreciar en la figura anterior, la cuenta conectada (0x5050A…) no dispone de fondos (ETH).
A continuación, buscamos un concurso para subir una foto, en el ejemplo, elegimos el concurso “Mascotas” y subimos una foto de “Willy”

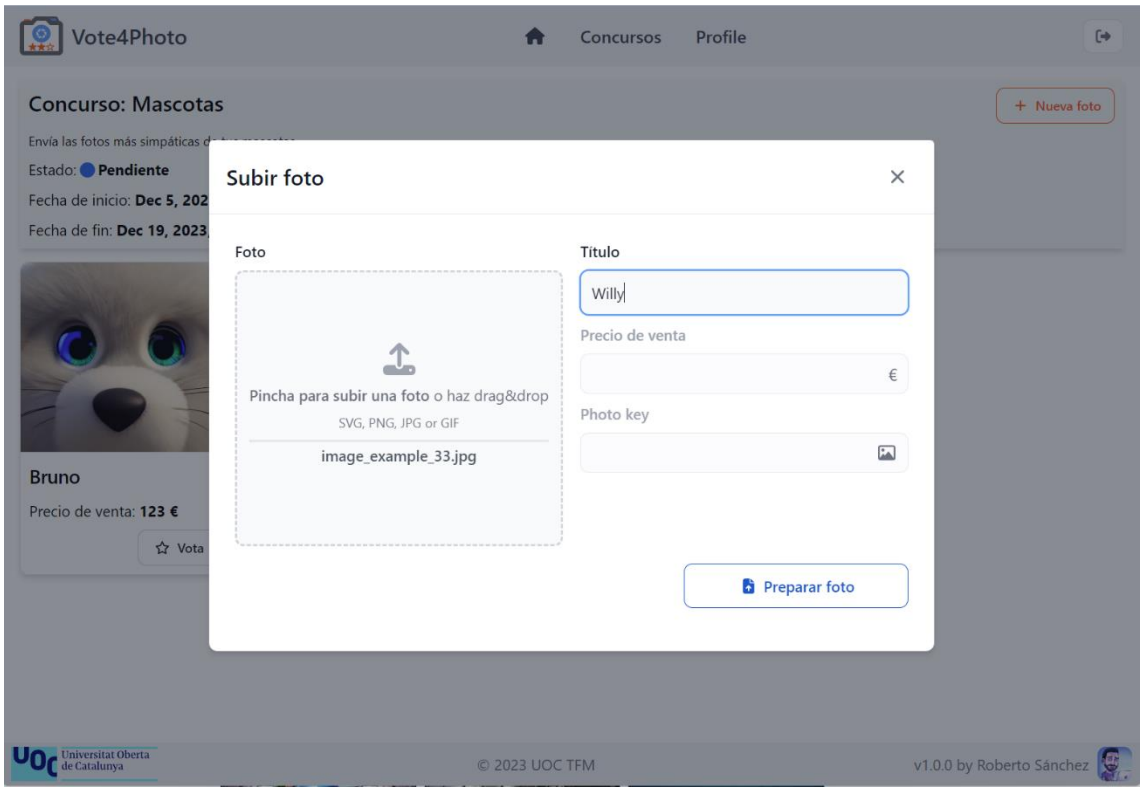

Figura 12: Formulario para subir una foto

Al enviar la foto, el sistema la almacena en el repositorio S3 (la foto mostrada se carga desde el repositorio) y genera el mensaje para firmar que servirá en el siguiente paso para lanzar la meta-transacción.

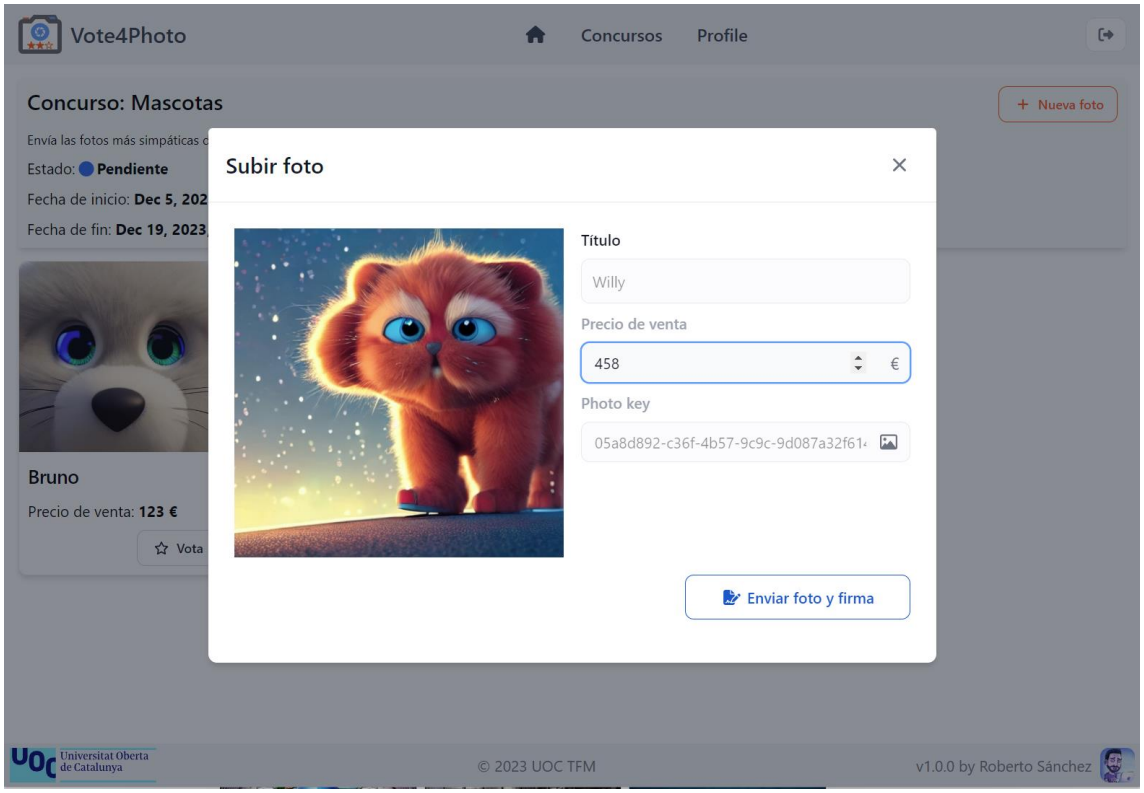

Figura 13: Formulario para firmar el mensaje y generar el NFT

Al pulsar en [Enviar foto y firmar] se lanza una petición de firma al Wallet del navegador, Metamask, que será firmado con la cuenta que hemos utilizado para acceder al sistema, 0x5050A…, la firma generada se envía al servidor que utilizará el mensaje firmado para enviar la meta-transacción al contrato Forwarder que, a su vez, la enviará al contrato de los tokens ERC721.

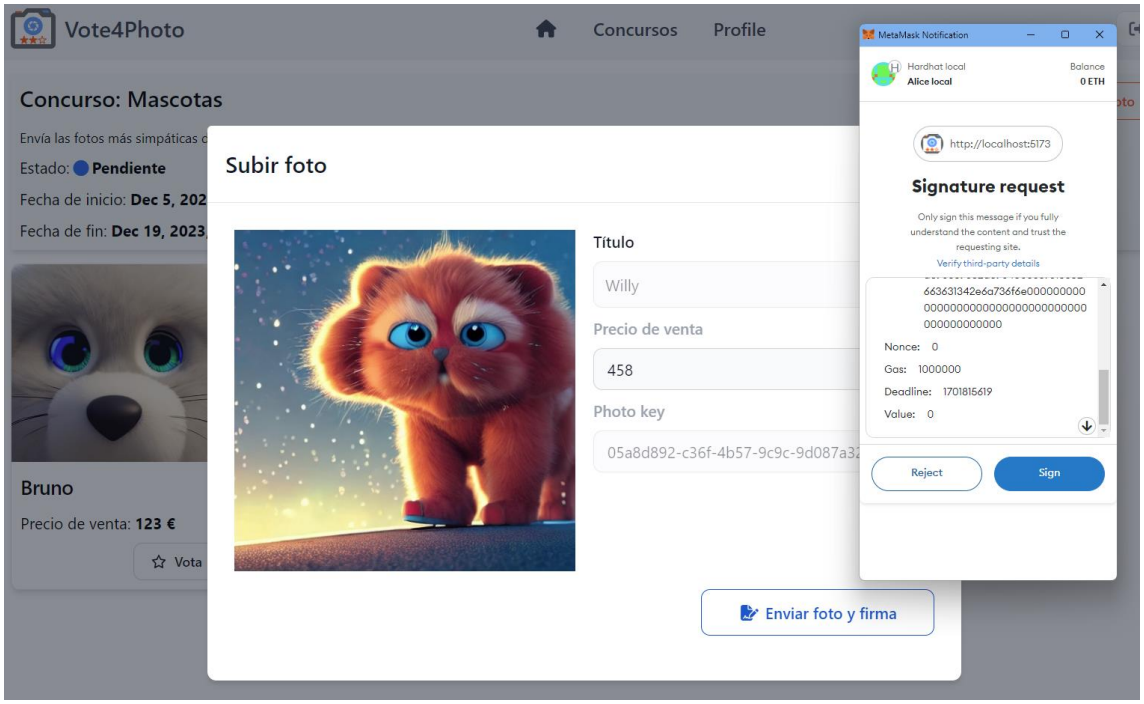

Figura 14: Firma del mensaje para la meta-transacción de creación del NFT

El NFT se genera y la foto queda inscrita en el concurso, tal y como se puede apreciar en la siguiente imagen:

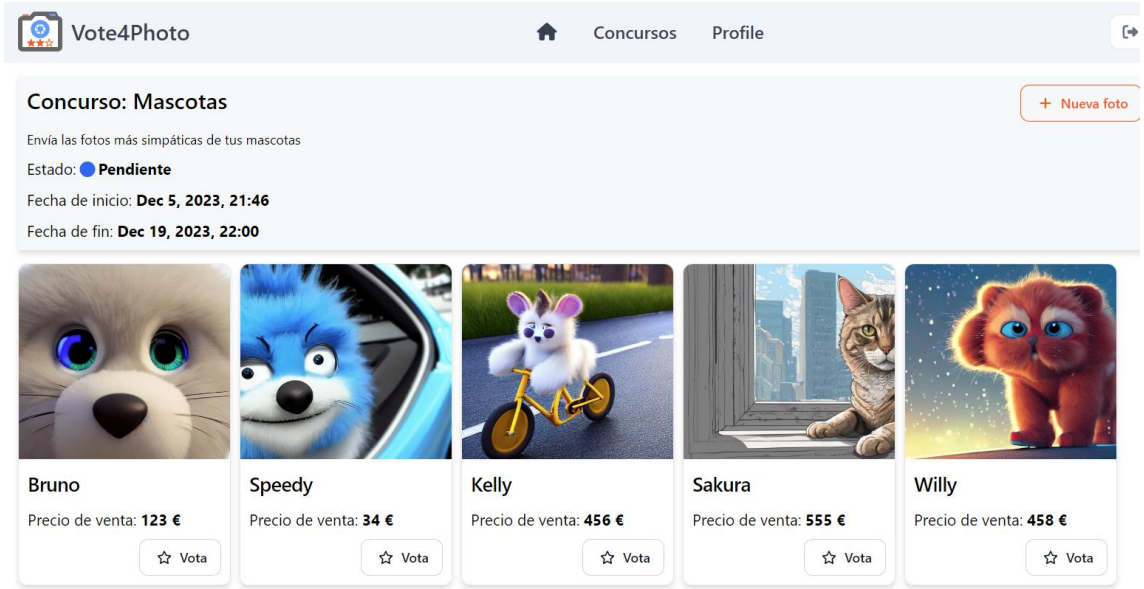

Figura 15: Foto inscrita en el concurso

Por último, en el perfil del usuario podemos ver que la foto le pertenece con datos como el token ID generado:

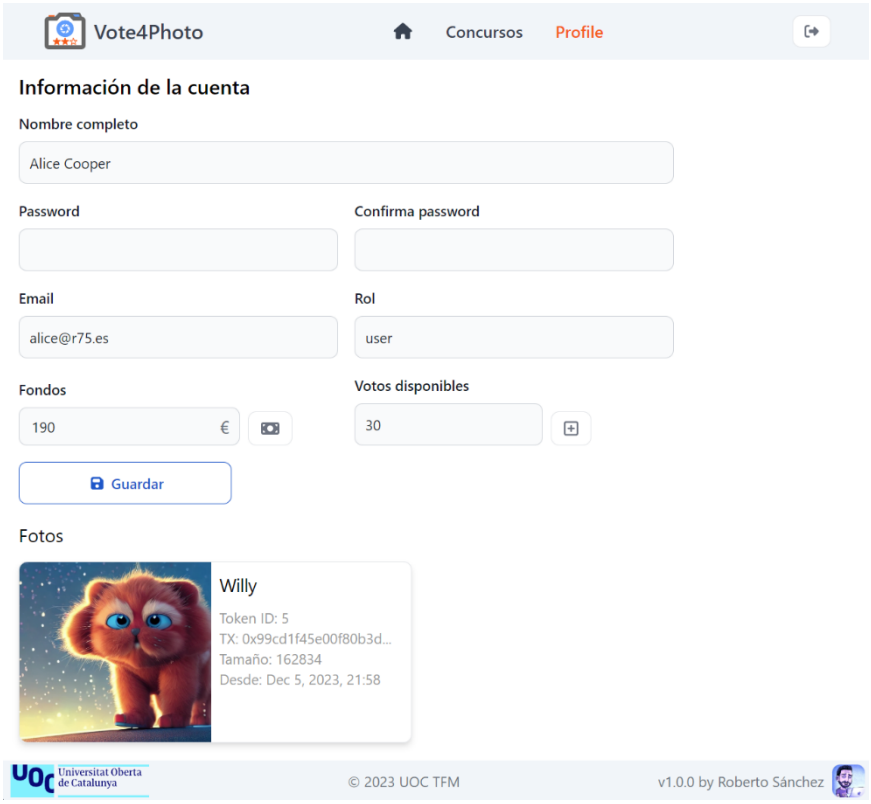

Figura 16: Foto asociada al usuario

Como se ha podido apreciar, todo el proceso es realmente sencillo para el usuario, el cual no ha precisado de fondos previos en ETH en su cuenta del wallet, además el proceso de firma es semi-automático, solo necesita de un “click” aceptando los datos enviados.