Escuela Politécnica Superior

Diseño y desarrollo de una pasarela de pagos automática con criptomonedas

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Universitat d'Alacant
Universidad de Alicante

Autor:
Tomás Rafael Martínez Sanchis

Tutor/es:
José Vicente Berná Martínez

Julio 2023

Resumen

Aunque las raíces del dinero digital se remontan a iniciativas anteriores a Bitcoin, fue este último quien catalizó la revolución de las criptomonedas, marcando un hito importante en la historia del dinero digital. El creador de Bitcoin, conocido con el seudónimo de Satoshi Nakamoto, fue quien rompió los moldes y trazó nuevos caminos con su sistema de pago descentralizado. El diseño de Bitcoin, basado en criptografía y redes distribuidas, permite transacciones seguras de una persona a otra sin la necesidad de intermediarios y de forma anónima.

A lo largo de los años, se ha hecho evidente la necesidad de un control seguro y personal de sobre estos activos digitales. Los exchanges centralizados de intercambio de criptomonedas como Mt.Gox y FTX han sufrido quiebras y hackeos, mostrando los riesgos asociados que hay con el almacenamiento y custodia de las criptomonedas por terceros. Estas situaciones han enfatizado la importancia de mantener las criptomonedas en un lugar seguro bajo nuestro propio control.

En la actualidad, hay disponibles muchas empresas que ofrecen pasarelas de pago con criptomonedas, pero presentan una serie de limitaciones, como son las comisiones por cada transacción, limitación de criptomonedas y redes disponibles con costes elevados. Además, al estar ubicadas en paraísos fiscales y opacos, corres el riesgo de que en cualquier momento tu cuenta sea bloqueada.

Es aquí donde entra la pasarela de pagos propia para garantizar un nivel de seguridad superior y poseer en todo momento la propiedad de las criptomonedas sin tener intermediarios. Esta pasarela brinda dos opciones a la hora de realizar pagos: interactuando con billeteras como Metamask y Smart Contracts para asegurar la validez de las transacciones, o realizando pagos de forma manual desde exchanges, en cuyo caso los datos se comprueban directamente en la blockchain de la criptomoneda correspondiente. Esta solución, permite al usuario elegir entre un abanico de criptomonedas y redes distintas para pagar la menor cantidad de comisiones de red por cada transacción.

La idea es incorporar esta pasarela de pago a un sitio web propio de venta de productos o servicios digitales relacionado con las inversiones y las criptomonedas.

Motivación, justificación y objetivo general

La idea de crear una pasarela de pagos mediante criptomonedas surgió una tarde en la que había quedado con un amigo, el cual hacía tiempo que no había visto, y en la conversación tratamos diversos temas, pero uno en especial fueron las inversiones, ya que se aproximaba una posible subida de tipos de interés a raíz de la inflación de los últimos meses.

Comentamos que habíamos pasado por una época en la que todo tipo de inversiones habían tenido un desempeño muy positivo después del COVID-19. Pero había mucha gente que independientemente de los conocimientos que tuviera, al encontrarnos al final de un ciclo económico alcista, habían tenido un buen rendimiento.

Fue aquí donde tuvimos la idea de lanzar un proyecto junto con un amigo más, en el cual pudiéramos explicar y demostrar todos nuestros conocimientos relacionados con la bolsa, criptomonedas y el análisis técnico.

Estos conocimientos serian publicados en varias redes sociales como YouTube, Twitter, Telegram (un grupo privado), en una página web a modo de artículos o guías y diferentes cursos avanzados de forma online. En las dos primeras, compartiríamos contenido didáctico de forma gratuita, mientras que en las dos últimas sería de pago.

Mirando otras personas y negocios que hacían algo parecido, me di cuenta que los métodos que utilizaban eran poco amigables con los usuarios y, además, llevaban mucho trabajo el control de los pagos y acceso a los cursos y grupos privados. Ya que tenían que comprobar el pago uno a uno con una probabilidad muy alta de cometer algún error.

Aquí me di cuenta de la posibilidad de automatizar todo el proceso de pago con criptomonedas, desde el control e ingreso de los usuarios a Telegram, hasta su acceso a la plataforma propia de cursos online sin la necesidad de comprobarlo manualmente.

El objetivo principal es crear una pasarela de pagos mediante diferentes criptomonedas sin la necesidad de utilizar un servicio de terceros que conlleve un gasto de gestión y coste por transacción. Además de que normalmente, todas estas plataformas suelen tener inconvenientes y al desarrollarlo a mi gusto, me permitiría conectarlo con mi página web de forma sencilla.

Al tratarse de una mejora para un proyecto personal, es una idea que claramente me motiva, por una parte, para facilitar a los clientes que quieran pagar con criptomonedas sin utilizar

tarjetas de crédito y por otra, de aumentar mis conocimientos en la materia como son la implementación de funcionalidades descentralizadas con criptomonedas.

Agradecimientos

Quiero aprovechar este apartado del trabajo para agradecer a todas las personas que me han dedicado parte de su tiempo en esta etapa de mi vida.

En primer lugar, a mi tutor de TFG y ABP, José Vicente Berná Martínez. Por todas las ideas y consejos que nos ha aportado durante el último curso del proyecto ABP. Por toda motivación para seguir mejorando y especialmente, por darme la oportunidad de hacer este último trabajo.

En segundo lugar, a mis profesores que me han acompañado durante el curso. A todos los compañeros que he conocido durante la carrera, especialmente en los últimos años. Gracias por hacer tan fácil trabajar en equipo.

A mis amigos, por animarme y disfrutar de todos esos momentos que hemos vivido.

A mi pareja, por apoyarme, animarme y entenderme cada día en esta etapa que no ha sido fácil.

Por último, quiero agradecer a mi familia el esfuerzo que han hecho en estos años no solo económicamente sino también por alentarme en los duros momentos. A mi madre, por poner mis intereses por encima de todo. A mi padre, por enseñarme que las cosas cuestan y hay que sacrificarse para conseguirlas. Y, sobre todo, a mi abuela, por todo lo que ha hecho por mí. Muchísimas gracias por estar siempre ahí, espero haceros sentir orgullosos.

Citas

"Si algo es lo suficientemente importante, incluso si las probabilidades están en tu contra, debes seguir intentándolo."

Elon Musk

"No es necesario hacer cosas extraordinarias para conseguir resultados extraordinarios".

Warren Buffett

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Las criptomonedas son un medio de intercambio completamente digital que, para proteger sus transacciones, controlar la creación de nuevas unidades y verificarlas, utiliza métodos criptográficos. Desde hace dos años, las criptomonedas están ganando mucha popularidad llegando incluso a una valoración de alrededor de 3 billones de dólares en noviembre de 2021 como se puede apreciar en la *Figura 1*.

Figura 1. Capitalización del mercado de las criptomonedas.
(Fuente propia)

Una de las criptomonedas más populares es Bitcoin, creada por Satoshi Nakamoto[1] en 2009 como un sistema de pagos *Peer to Peer*, es decir, compartir archivos o información sin necesidad de intermediarios. Es por ello, que podemos considerarla como una alternativa descentralizada a las monedas convencionales y centralizadas emitidas por una entidad o banco.

El éxito de las criptomonedas ha llevado a la aparición de muchas empresas que ofrecen pasarelas de pagos. Coinbase, Bitpay, CoinPayments, entre otras, son algunas de las empresas que ofrecen estas soluciones de pago. El sistema permite que los usuarios puedan pagar con criptomonedas y las convierte automáticamente en la moneda fiduciaria deseada por el comerciante además de transferir el dinero a su cuenta.

Todas estas pasarelas de pago para criptomonedas demuestran el creciente interés de las empresas por esta tecnología y su potencial para revolucionar los sistemas de pago. Con el tiempo, es posible que las criptomonedas se conviertan en una forma de pago común y aceptada en todo el mundo.

[1] Satoshi Nakamoto es el pseudónimo bajo el cual se oculta el creador de Bitcoin, se desconoce si es una persona o un grupo de personas.

Y no solo empresas, si no también países, el 7 de septiembre de 2021, El Salvador fue el primer país en adoptar la criptomoneda Bitcoin como moneda curso legal. Más tarde, el 27 de abril lo hace la República Centro Africana [1].

Según un estudio realizado por la plataforma de intercambio de criptomonedas Crypto.com, a pesar de las caídas de precio en 2022, el ecosistema no ha parado de crecer, como se puede apreciar en la *Figura 2* el número de usuarios poseedores de criptomonedas ha aumentado un 39%, pasando de 306 millones de usuarios en enero a los 425 millones a finales de año [2].

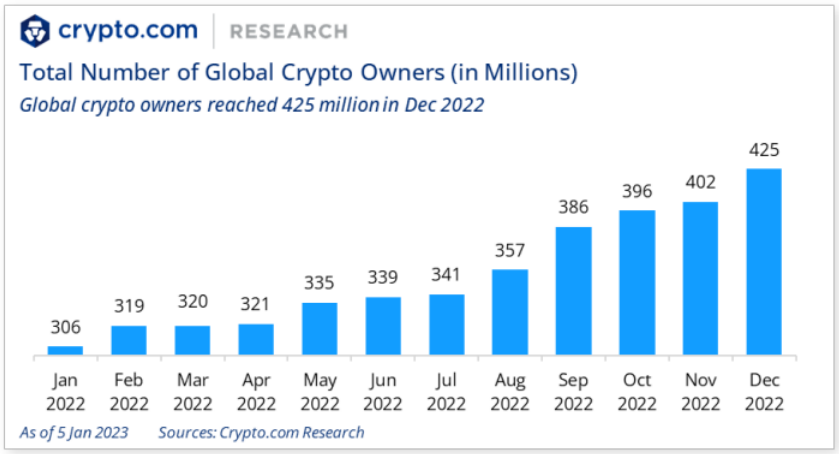


Figura 2. Número de personas que poseen criptomonedas (Fuente: https://content-hub-static.crypto.com/wp-content/uploads/2023/01/Cryptodotcom_Crypto_Market_Sizing_Jan2023-1.pdf)

Sin embargo, dentro del sector de las criptomonedas, también ha llevado a la creación de un sector conocido como DeFi[2], o finanzas descentralizadas. Este sector utiliza la tecnología blockchain y las criptomonedas para crear servicios financieros descentralizados, que no dependen de intermediarios como bancos, donde podemos encontrar préstamos entre pares, los pools de liquidez y los intercambios descentralizados.

Debido a todo esto, he querido desarrollar una pasarela de pagos mediante *smart contracts*[3] para un proyecto propio relacionado con la temática y, entre otras cosas, que imparta justicia a su verdadero origen, realizar pagos entre dos personas sin la necesidad de un intermediario. Además, al cliente le proporcionas más opciones para pagar y mejorar su experiencia.

La pasarela de pagos contará con todas las funcionalidades necesarias para administrar los pagos de un usuario, desde la elección de diferentes criptomonedas según la necesidad y

[2] Decentralized Finance

[3] Un Smart contract, o contrato inteligente, es un programa informático que se ejecuta automáticamente cuando se cumplen ciertas condiciones preestablecidas, permitiendo la realización de transacciones y acuerdos de manera autónoma y segura.

disponibilidad de la red, hasta la gestión automatizada para acceder al servicio contratado de la forma más rápida y adecuada posible.

2. Estudio de viabilidad

Antes de comenzar con el proyecto, es importante identificar los objetivos, analizar si son viables o incluso necesarios, para llevarlos a cabo. Además, es importante indicar los riesgos y los planes de contingencia.

En primer lugar, para hacer el estudio de viabilidad, vamos hacer uso del análisis DAFO, que nos ayudará a identificar las estrategias que permitirán alcanzar los objetivos, seguidamente un Lean Canvas para analizar el proyecto desde diferentes puntos de vista y tener una visión clara de los objetivos a realizar.

Por último, se tendrá en cuenta los posibles planes de previsión y contingencia para evitar riesgos.

2.1. Análisis DAFO

El análisis DAFO es una metodología de estudio o herramienta de planificación utilizada para evaluar la situación de un proyecto. DAFO es el acrónimo de Debilidades, Amenazas, Fortalezas y Oportunidades. Esta metodología se divide en cuatro categorías analizando sus características internas (Debilidades y Fortalezas) y su situación externa (Amenazas y Oportunidades) en una matriz cuadrada como Figura muestra la *Figura 3*.

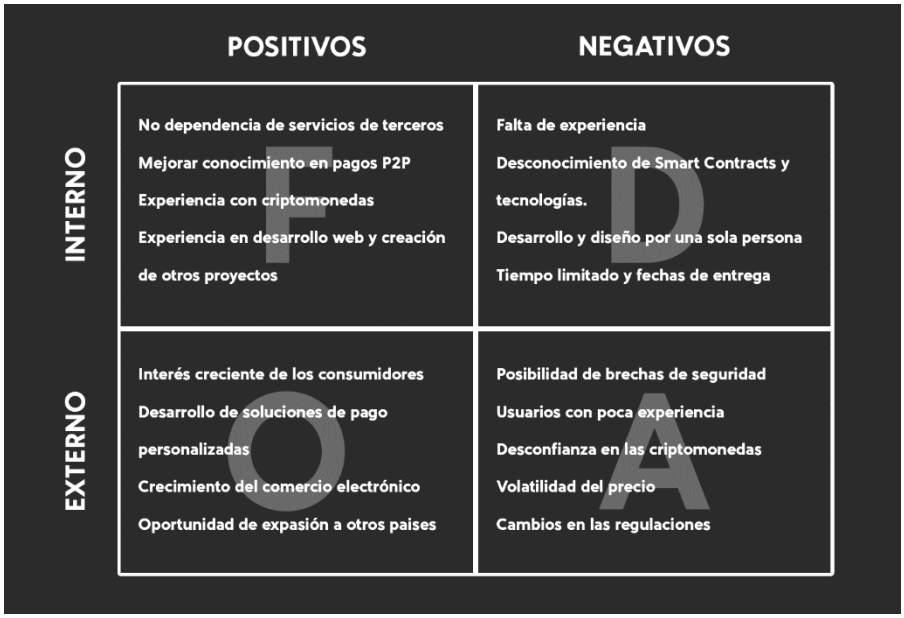


Figura 3. Esquema de un análisis DAFO.
(Fuente propia)

Para empezar, dentro de las características internas, encontramos

Debilidades:

- Falta de experiencia en la creación de pasarelas de pago con criptomonedas.
- Desconocimiento en el desarrollo de Smart Contracts y algunas de las tecnologías a utilizar.
- Desarrollo y diseño del proyecto recae en una sola persona.
- Tiempo limitado y condicionado por fechas de entrega.

Amenazas:

- Posibilidad de brechas de seguridad y robos.
- Usuarios con poca experiencia y conocimiento.
- Desconfianza y preocupación por la seguridad de las criptomonedas.
- Volatilidad de los precios de las criptomonedas.
- Cambios en las regulaciones y normativas del mercado de criptomonedas.

Fortalezas:

- No depender de servicios de terceros que se ubican en países opacos y/o no operan en ciertos países.
- Mejorar mi conocimiento en el funcionamiento de pagos p2p.
- Experiencia en el mundo de las criptomonedas y, por tanto, conocer la mentalidad de los posibles usuarios.
- Experiencia en desarrollo web y creación de otros proyectos.

Oportunidades:

- Interés creciente de los consumidores en las criptomonedas y su adopción.
- Desarrollo de soluciones de pago personalizadas.
- Crecimiento del comercio electrónico y la demanda de soluciones de pago con criptomonedas.
- Oportunidad para expandirse a todos los países.

2.2. Lean Canvas

Lean Canvas es otra de las herramientas que pueden ser idóneas para analizar un producto que está siendo creado y que tiene carácter innovador, ya sea porque es una solución nueva a un problema ya existente o bien porque es una mejora sobre otras soluciones que ya existen. La idea de Lean Canvas es analizar el proyecto desde diferentes perspectivas de interés como muestra la *Figura 4*.

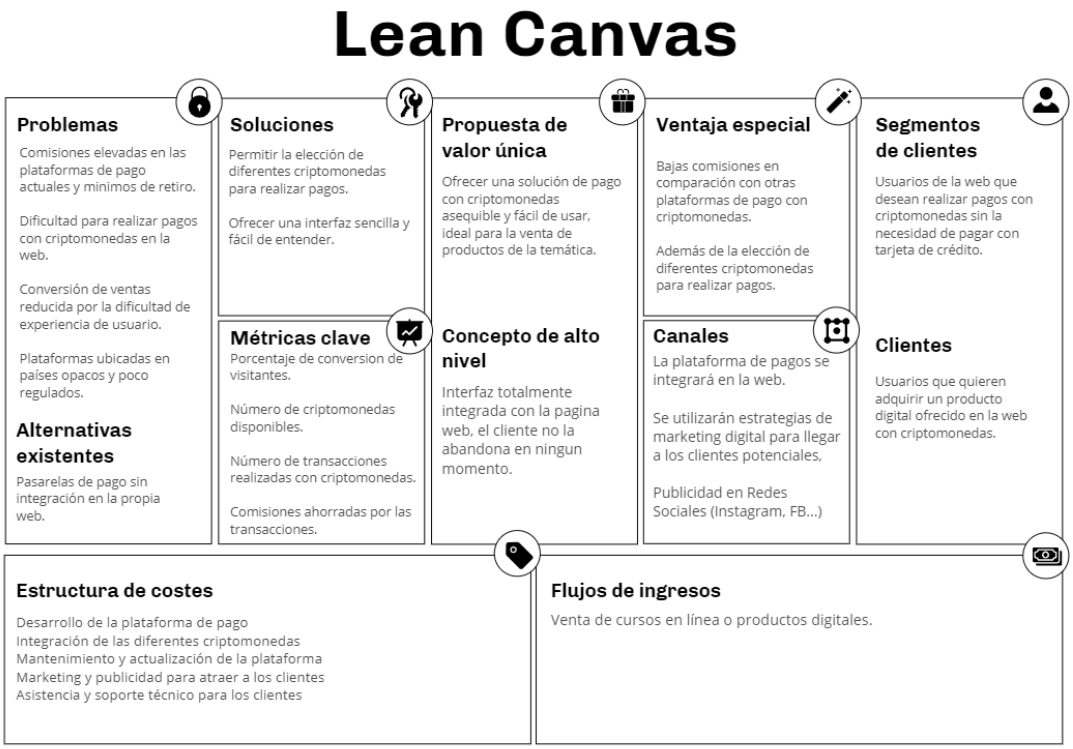


Figura 4. Cuadro para el análisis Lean Canvas
(Fuente propia)

2.2.1. Problemas

Para empezar, necesitamos detectar los problemas que hay en el sector con las pasarelas de pagos con criptomonedas, para posteriormente encontrar las soluciones de cara al desarrollo óptimo del proyecto.

Las empresas que ofrecen opciones de pago con criptomonedas tienen comisiones elevadas que recaen al cliente o al usuario a la hora de pagar, ya sea por la volatilidad o la dificultad de

gestionar las criptomonedas al cambiarlas por dinero fiduciario[4], además suelen tener mínimos para retirar fondos.

Estas plataformas suelen estar ubicadas en países opacos y poco regulados, por lo que hay preocupación por la seguridad de los fondos. Por ejemplo, FTX, el tercer mayor exchange[5] de criptomonedas por volumen de dinero, con sede en las Bahamas, utilizaba el dinero de sus clientes para inversiones y como colateral para préstamos. Se declaró en bancarrota y todos los usuarios perdieron sus inversiones [3].

La mayoría de plataformas redirigen al usuario a una página externa para completar la transacción, lo que reduce la conversión de los clientes al no conocer la plataforma por estar en otro idioma o sea totalmente diferente y, por tanto, la experiencia de usuario se ve mermada.

A todo esto, hay que sumarle que las transacciones se suelen hacer enviando las criptomonedas (una cantidad determinada) a una dirección y en un tiempo límite, en caso de que no se realice correctamente suelen haber problemas, como la confusión de redes en una misma criptomoneda.

2.2.2. Solución

Para solucionar los problemas anteriores, la plataforma ofrecerá una pasarela con los siguientes puntos:

- Los usuarios podrán elegir entre diferentes criptomonedas populares para realizar sus pagos, lo que les brindará una mayor libertad, además de poder elegir monedas estables que replican el precio del dólar.
- Las comisiones serán únicamente las necesarias para realizar la transacción en la cadena de bloques de la criptomoneda elegida, es lo que se conoce con el nombre de gas.
- La interfaz será sencilla y fácil de entender, para que la experiencia de usuario sea la más satisfactoria posible, estará en el idioma del usuario y no abandonará la página web.

2.2.3. Propuesta de valor única

La plataforma de pago combina la comodidad de las criptomonedas con una experiencia de usuario fácil y sin comisiones de terceros. Este enfoque es ideal para la venta de productos digitales y en línea como los cursos o infoproductos, y está diseñado para facilitar a un público que desea realizar pagos seguros con criptomonedas.

[4] Dinero emitido por un banco central, como podría ser el euro o el dólar.
[5] Plataforma de intercambio de criptomonedas.

2.2.4. Ventaja especial

El módulo de pagos con criptomonedas ofrece una serie de ventajas únicas que la distinguen de otras soluciones. Al ser un proyecto propio, se eliminan intermediarios, lo que permite ofrecer las comisiones más bajas del mercado, ya que sólo se aplican aquellas correspondientes a las transferencias entre billeteras. Además, la plataforma brinda a los usuarios la posibilidad de seleccionar redes de criptomonedas populares, como BSC, en lugar de limitarse únicamente a Ethereum, lo cual reduce significativamente las comisiones de red y los tiempos de espera. La integración del módulo se ve facilitada al estar diseñado y desarrollado internamente, garantizando una mayor adaptabilidad y eficiencia en la implementación.

2.2.5. Canales

Para llegar a nuestros clientes potenciales, se utilizarán varios canales de marketing como los videos propios de la temática de criptomonedas en YouTube, noticias e hilos informativos en Twitter y vídeos cortos en Instagram y TikTok.

Por otro lado, se harán diferentes campañas de anuncios, entre ellas de remarketing, es decir, para aquellos usuarios que habían entrado anteriormente a la web. Las plataformas a utilizar serán: Google Ads, para aparecer en páginas de la misma temática; Instagram y Facebook, para publicaciones y redes sociales; y por último email marketing, para aquellos que se hayan unido a la Newsletter[6].

2.2.6. Métricas claves

Las métricas clave son imprescindibles para medir la información y ajustar la estrategia en caso de ser necesario. Aquí están las métricas más importantes que se deberían tener en cuenta:

- Porcentaje de conversión de los visitantes: este indicador mide la cantidad de visitantes que realmente realizan una transacción a través de esta opción de pago. Este número será importante para identificar problemas con la experiencia de usuario.
- Número de criptomonedas disponibles y uso: este número será importante para medir la aceptación de cada criptomoneda y determinar si es necesario ampliar la cantidad de monedas aceptadas e incluso otros tokens de la misma blockchain.
- Número de transacciones realizadas que repiten con criptomonedas: este número será importante para medir el éxito en general y para determinar si se está alcanzando el objetivo de ser una solución de pago con criptomonedas fácil de usar.

[6] Boletín informativo a través de correo electrónico.

- Comisiones ahorradas por las transacciones: este dato será importante para medir el éxito de la propuesta de valor única de la plataforma, ofrecer bajas comisiones en comparación con otras plataformas de pago con criptomonedas.

2.2.7. Segmento de clientes

Personas que desean realizar pagos con criptomonedas sin la necesidad de utilizar tarjetas de crédito. Estos usuarios pueden ser propietarios de pequeñas empresas en línea, vendedores individuales o simplemente personas que desean proteger su privacidad y seguridad financiera en línea.

Arquetipos:

- Juan, joven de 25 años recién graduado con un interés en la tecnología y las inversiones. Juan está buscando aprender sobre las criptomonedas y cómo pueden ser una oportunidad de inversión a largo plazo. Tiene ya conocimientos sobre las criptomonedas, sin embargo, está dispuesto a invertir tiempo y dinero en mejorar sus conocimientos.
- Agustín, un hombre de 45 años que ha sido tradicionalmente un inversor agresivo, su primera inversión en criptomonedas fue en Bitcoin hace 4 años. Agustín ha estado siguiendo de cerca el mercado de las criptomonedas y está indagando en ideas para invertir. Está buscando una formación clara y accesible para comprender mejor el mundo DeFi de las criptomonedas.

2.2.8. Estructura de costes

En este caso, al tratarse de un Trabajo de Fin de Grado, en cuanto al coste hablamos del sueldo de una persona que se encargaría del diseño y desarrollo de la plataforma de pago además de la integración de las diferentes criptomonedas.

No obstante, a medio plazo habrá costes como el mantenimiento y actualización de la plataforma, publicidad, además de asistencia y soporte técnico para los clientes.

2.2.9. Flujo de ingresos

Se trata de un proyecto académico, y aunque el objetivo no sea lucrarse con el resultado final, sí que ayudará al usuario y facilitará que el número de ventas de cursos en línea o productos digitales aumente.

2.3. Análisis de riesgos

En este apartado, se pretende identificar los desafíos u obstáculos que podrían presentarse durante la realización del Trabajo Final de Grado. Dado que este proyecto equivale a 300 horas, es importante tener en cuenta cualquier amenaza que pueda afectar negativamente su desarrollo. En caso de que se de alguna incidencia se recurrirá a las opciones dadas.

Los tipos de riesgos contemplados son: tecnología, personal, herramientas, requerimientos, estimación y otros.

En la columna de probabilidad, cada nivel está asociado con las veces que puede ocurrir dicho riesgo.

Tabla 1. Leyenda probabilidades.

Probabilidad	Muy baja	Baja	Moderada	Alta	Muy alta
Descripción	Una vez cada 5 años	Una vez al año	Una vez cada semestre	Una vez al mes	Una vez a la semana

Por otro lado, los efectos los clasificaremos en: catastrófico, serio, tolerable o insignificante.

Tabla 2. Leyenda efectos.

Efecto	Tolerable	Serio	Catastrófico
Descripción	No hay retrasos ni alteraciones	Pueden ocurrir retrasos de horas o días	Pueden ocurrir retrasos de semanas o meses.

Tabla 3. Análisis de riesgos tecnológicos.

Tipo de riesgo	Posible riesgo	Probabilidad	Efectos
Tecnología	Avería grave del ordenador o hardware	Baja	Serio
	Pérdida de datos	Baja	Serio
	Problemas en la implementación	Moderado	Tolerable
	Servicios de terceros (API) dejen de funcionar	Baja	Serio

Tabla 4. Análisis de riesgos personales.

Tipo de riesgo	Posible riesgo	Probabilidad	Efectos

Personal	Abandono del proyecto	Muy baja	Catastrófico
	Falta de experiencia o conocimientos sobre el tema	Moderado	Serio
	Enfermedad del desarrollador	Moderado	Serio
	Enfermedad de un familiar	Baja	Serio
	Falta de constancia por problemas familiares (fallecimiento, enfermedad, asuntos importantes…)	Baja	Serio

Tabla 5. Análisis de riesgos de estimación.

Tipo de riesgo	Posible riesgo	Probabilidad	Efectos
Estimación	Sobrevalorar y subestimar el tiempo para la redacción del TFG	Baja	Catastrófico
	Infraestimar o sobreestimar el tiempo necesario para el desarrollo de la pasarela de pago	Moderada	Serio
	Coste del proyecto superior a lo previsto	Moderada	Tolerable
	Error en la estimación de horas para añadir criptomonedas disponibles	Moderada	Serio

Tabla 6.Análisis de riesgos de herramientas.

Tipo de riesgo	Posible riesgo	Probabilidad	Efectos
Herramientas	Pérdida de datos y archivos del TFG	Muy baja	Catastrófico
	Herramientas como Clockify o Trello no funcionen o tengan fallos	Baja	Tolerable
	Prohibición de criptomonedas o pagos p2p	Muy baja	Catastrófico
	Información escasa sobre el tema a desarrollar	Muy baja	Serio
	Encarecimiento de las API de precios de las criptomonedas	Baja	Tolerable

Tabla 7.Análisis de riesgos de organización.

Tipo de riesgo	Posible riesgo	Probabilidad	Efectos
Organización	Necesidad de añadir nuevas tareas	Alta	Tolerable
	Acumular tareas sin realizar	Moderada	Serio

1. Estrategias

En esta sección, nos basaremos en el análisis previo para proponer una serie de estrategias que permitan prevenir, reducir o, en su defecto, establecer un plan de contingencia para hacer frente a los riesgos potenciales.

Tabla 8. Estrategias de riesgos.

Tipo de riesgo	Riesgo	Estrategia
Tecnológico	Avería grave del ordenador o hardware	**Prevención:** Realizar un mantenimiento de forma periódica al hardware del ordenador para evitar posibles problemas. **Plan de contingencia:** Tener un ordenador de respaldo preparado en caso de que ocurra una avería grave.
	Pérdida de datos	**Prevención:** Realizar copias de seguridad frecuentes y guardarlas en una ubicación segura y externa al equipo. **Plan de contingencia:** Tener una copia de seguridad actualizada de los datos para restaurarlos en caso de pérdida.
	Problemas en la implementación	**Prevención:** Realizar pruebas frecuentes y de estrés para identificar posibles fallos. **Plan de contingencia:** Consultar tutoriales, foros o documentación, en último lugar al tutor.
	Servicios de terceros (API) dejen de funcionar	**Prevención:** Realizar pruebas de integración de los servicios de terceros antes de su implementación además de contar con alternativas. **Plan de contingencia:** Consultar soporte técnico y documentación. Encontrar una alternativa.
Personal	Abandono del proyecto	**Prevención:** Establecer plazos realistas y alcanzables. **Plan de contingencia:** No corresponde.
	Falta de experiencia o conocimientos sobre el tema	**Prevención:** Investigar e informarse con tutoriales o artículos. **Plan de contingencia:** Consultar en foros o en último lugar al tutor.

	Enfermedad del desarrollador	**Prevención:** Cuidarse y descansar lo suficiente **Plan de contingencia:** Seguir recomendaciones médicas hasta la recuperación.
	Fallecimiento o problema grave	**Prevención:** Contar con tiempo adicional por si ocurre algún problema. **Plan de contingencia:** No corresponde.
	Falta de constancia por problemas familiares (fallecimiento, enfermedad, asuntos importantes…)	**Prevención:** Tener una visión preventiva de los acontecimientos y anticiparse. **Plan de contingencia:** Atender problemas familiares y retomar el proyecto lo antes posible.
Estimación	Sobrevalorar y subestimar el tiempo para la redacción del TFG	**Prevención:** Estimar el tiempo requerido para cada sección y revisar los progresos para asegurarse de que se está avanzando según lo previsto. **Plan de contingencia:** En caso de que el tiempo estimado no sea suficiente, trabajar horas extra para cumplir con los plazos. Si es excesivo, revisar la planificación para ajustar.
	Infraestimar o sobreestimar el tiempo necesario para el desarrollo de la pasarela de pago	**Prevención:** Realizar una investigación previa y establecer una planificación realista. **Plan de contingencia:** Trabajar horas extra para cumplir en caso de quedarse corto de tiempo, si es excesivo, revisar la planificación para ajustar y mejorar otras secciones.
	Coste del proyecto superior a lo previsto	**Prevención:** Realizar una investigación previa sobre los costes asociados al proyecto y contemplar una reserva extra por si surgen imprevistos. **Plan de contingencia:** Identificar áreas del proyecto donde se pueden reducir costes.
	Error en la estimación de horas para añadir criptomonedas disponibles	**Prevención:** Realizar una investigación previa sobre las tecnologías de cada criptomoneda y hacer una planificación realista acorde. **Plan de contingencia:** Finalizar el proyecto con las criptomonedas disponibles.
Herramientas	Pérdida de datos y archivos del TFG	**Prevención:** Realizar copias en la nube y en dispositivos de los avances realizados. **Plan de contingencia:** Utilizar la última copia disponible o posponer la entrega a la siguiente convocatoria.

	Herramientas como Clockify o Trello no funcionen o tengan fallos	**Prevención:** Revisar y realizar pruebas antes de su uso. **Plan de contingencia:** Buscar herramientas alternativas de forma temporal o seguir sin ellas.
	Prohibición de criptomonedas o pagos p2p	**Prevención:** Investigar e informarse sobre las leyes y regulaciones sobre criptomonedas y pagos p2p **Plan de contingencia:** Intentar cumplir las leyes.
	Información escasa sobre el tema a desarrollar	**Prevención:** Realizar una investigación en profundidad sobre el tema además de consultar a expertos en el área. **Plan de contingencia:** Ajustar el alcance del proyecto para adaptarlo a la información disponible
	Encarecimiento de las API de precios de las criptomonedas	**Prevención:** Investigar e identificar varias opciones de API y realizar un análisis de beneficios-desventajas. **Plan de contingencia:** Ajustar el presupuesto o pasar a otra alternativa.
Organización	Necesidad de añadir nuevas tareas	**Prevención:** Establecer un plan de trabajo y un calendario con fechas límite para cada tarea. **Plan de contingencia:** Evaluar la importancia de las nuevas tareas y su impacto en el proyecto.
	Acumular tareas sin realizar	**Prevención:** Establecer fechas límite para cada tarea y asignar el tiempo necesario para completar cada tarea. **Plan de contingencia:** En caso de considerar que son importantes para el proyecto, realizarlas con prioridad al resto.

3. Planificación

Es importante que antes de comenzar el proyecto, se dedique tiempo a planificar correctamente para así tener mejores resultados y, sobre todo, ayuda a que en caso de que aparezca algún imprevisto, se pueda actuar rápidamente para que no perjudique al resultado final.

Esta planificación nos permitirá realizar un análisis posterior del proyecto, comparando los resultados finales con los concebidos en un principio y, por tanto, si se ha infravalorado el tiempo para cada uno de los contenidos o por el contrario se ha sobrevalorado.

En la Tabla 1 se ha realizado una planificación inicial para el proyecto, donde se encuentran los contenidos y una estimación de tiempo dedicado junto con una fecha límite.

Tabla 9. Planificación temporal TFG

Contenidos	Tiempo total	Fecha límite fin
Motivación, justificación, objetivo general Introducción Estudio de viabilidad Planificación	3 semanas	20 febrero
Estado del arte Objetivos Metodología Análisis y especificación Presupuesto, estimaciones	4 semanas	20 marzo
Diseño	3 semanas	10 abril
Implementación	3 semanas	1 mayo
Pruebas y validación Resultados Conclusiones y trabajo futuro Referencias, bibliografía y apéndices Agradecimientos, citas, índices	2 semanas	15 mayo

Como todas las asignaturas del grado han sido superadas, se dispondrá de lunes a viernes hasta justo antes de la fecha de entrega, es decir, aproximadamente cuatro meses, lo que supone 16 semanas. Eso corresponde a una media de más de 3 horas y 45 minutos al día.

Si entre semana no se avanza lo suficiente, en caso de ser necesario se utilizarán los fines de semana para avanzar y recuperar el tiempo perdido.

4. Estado del arte.

El estado del arte es una sección esencial para cualquier trabajo o proyecto, de hecho, hay que realizarlo antes de diseñar cualquier solución, ya que permite al autor conocer más información sobre el problema y las posibles soluciones que hay hoy en día.

El objetivo del estado del arte es recopilar toda la información existente relevante para el desarrollo del proyecto, además de analizar las tecnologías que podrían utilizarse.

Desde la creación del Bitcoin en el año 2009, el mundo de las criptomonedas ha experimentado un crecimiento exponencial. La adopción de criptomonedas como forma de pago se ha extendido más allá de los círculos de entusiastas y se ha convertido en una tendencia mundial en el último año. Actualmente, algunas de las empresas más grandes del mundo, como Microsoft, PayPal, Starbucks, entre otras, aceptan pagos con ellas.

Por lo tanto, el presente Trabajo de Fin de Grado tiene como objetivo crear un módulo que permita la admisión de pagos con criptomonedas y distintas redes blockchain en una plataforma web de un proyecto propio.

En este contexto, es importante destacar que existen diversas pasarelas de pago disponibles en el mercado, y que cada una presenta características y funcionalidades específicas. Por esta razón, se llevará a cabo un análisis comparativo de las principales pasarelas, con el fin de identificar las fortalezas y debilidades de cada una de ellas.

Para realizar este análisis, se establecerán una serie de características que serán tenidas en cuenta para cada una de las pasarelas de pago analizadas. Estas características incluyen aspectos como: la cantidad de criptomonedas, tipos, el coste de las comisiones, tiempo de cobro, la usabilidad de la plataforma, entre otros.

Una vez efectuado el análisis, se realizará una tabla que permitirá obtener información valiosa acerca de la situación actual de las pasarelas de pagos con criptomonedas, lo que servirá de base para el diseño y desarrollo de una solución o módulo integrado en la web.

Seguidamente, analizaremos las billeteras y blockchains más utilizadas para poder crear un módulo que cubra las necesidades actuales de los usuarios.

4.1. Soluciones existentes

Actualmente, a pesar de que el mercado de las criptomonedas es relativamente muy joven, existen muchas soluciones para pagar con ellas. Sin embargo, cada una puede ser muy diferente al resto, es por ello que se ha realizado una investigación y estudio para comprobar que ventajas tienen y su funcionamiento de cara a mejorar el diseño y desarrollo del proyecto.

Para ello analizaremos las pasarelas de pago más populares, para poder comparar posteriormente que opciones permiten: billeteras, que criptomonedas y cantidad, redes utilizadas, tiempo, necesidad de registros, datos que se solicitan, entre otras.

4.1.1. Coinbase

La primera plataforma que vamos a analizar y posibilita agregar una pasarela de pagos con criptomonedas es *Coinbase*. Esta empresa nos permite integrar su pasarela de pago mediante un botón azul de *Pagar con Coinbase* como se puede ver en la Figura 5, no obstante, necesitas registrarte o ser usuario para poder seleccionar las criptomonedas que tienes en tu cuenta. Desde hace poco permiten pagar con hot wallets[7], es decir, billeteras calientes como Metamask o *Coinbase Wallet*, esta última es una opción que a diferencia de lo que ofrecían en un principio, no es necesario registrarse en su plataforma.

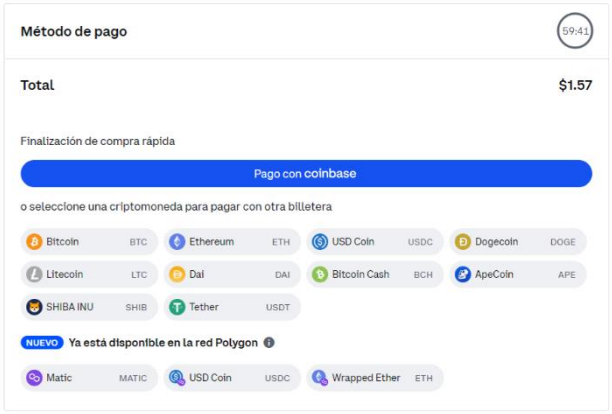


Figura 5. Elección de pago a través de Coinbase (Fuente propia)

Como hemos podido ver, ambas opciones aparecen a la hora de pagar en un negocio que tenga la pasarela de pago de *Coinbase*, sin embargo, si pulsamos el botón azul mencionado anteriormente como podemos ver en la Figura 6, nos pedirá que iniciemos sesión en la plataforma.

[7] Billeteras que están conectadas a internet.

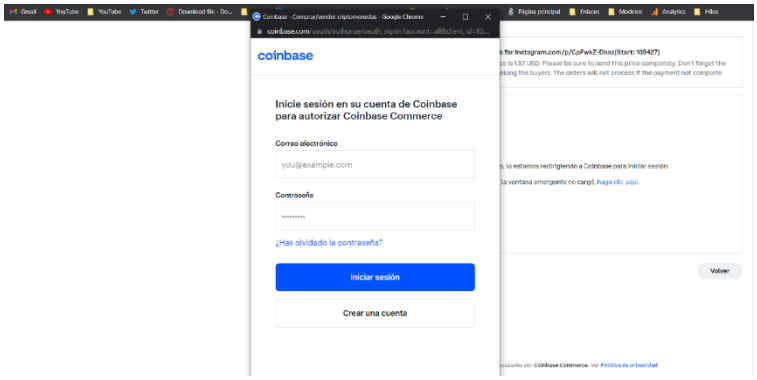


Figura 6. Inicio se sesión con usuario de Coinbase
(Fuente propia)

Por el contrario, si pulsamos en una criptomoneda en caso de que sea compatible con las billeteras calientes más populares, se abrirá para confirmar la transacción. Pero si no es compatible, como es el caso de *Bitcoin* o *Litecoin*, nos mostrará un código QR o la opción de copiar al portapapeles la dirección para hacer el envío manualmente en cualquier otra billetera o desde un Exchange como se puede ver en la Figura 7.

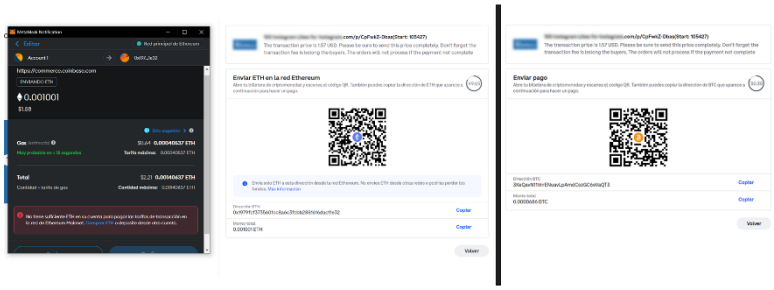


Figura 7. Elección de pago con Ethereum en la parte izquierda y Bitcoin a la derecha
(Fuente propia)

Esta solución tiene el inconveniente de que la gran mayoría de monedas con las que se pueden pagar son de la red Ethereum y, por tanto, hablamos entre 2$ y 5$ como mínimo de coste extra de transacción de la red en la mayor parte del día. Incluso podría sumarse horas de retraso si la red está colapsada.

4.1.2. BitPay

BitPay es una de las plataformas más grandes dentro del sector, permite comprar, vender e incluso gastar las criptomonedas con una tarjeta de crédito propia. Por ejemplo, para los negocios tiene un amplio uso para: pagos en línea, aceptar donaciones, entre otras.

Esta plataforma permite al usuario en primer lugar, elegir entre un gran abanico de billeteras, para posteriormente seleccionar algunas criptomonedas que están disponibles. Como se puede ver en la Figura 8, a la hora de seleccionar una criptomoneda, nos permite en algunos casos la

posibilidad de elegir la red para pagar (por ejemplo, *Ethereum* y *Polygon*), algo importante porque la diferencia de comisiones puede llegar a ser del más del 400% entre diferentes redes.

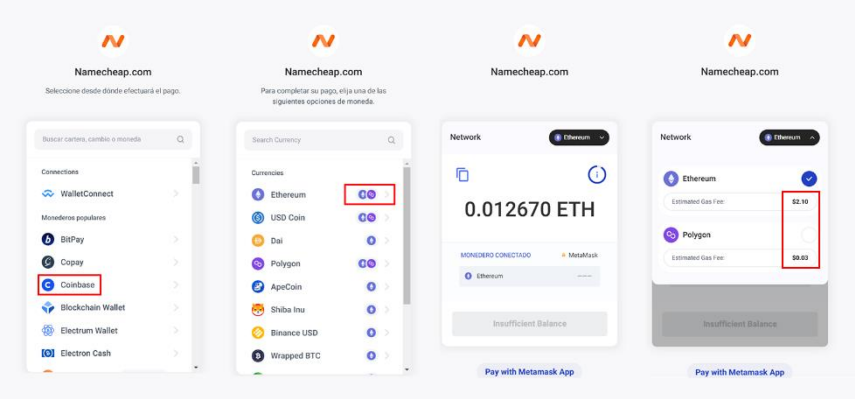


Figura 8. Pasos para pagar con Bitpay, elección de billetera, redes y costes (Fuente propia)

A pesar de tener una gran variedad de billeteras y muchas monedas en cada una de ellas, se echa en falta que acepten criptomonedas de la red *BSC*[8], ya que tienen comisiones muy bajas de apenas 0,20$ por transacción y es la segunda red más utilizada en los ecosistemas *DeFi* según *DefiLlama* [4].

Esta opción también tiene integrada la alternativa de *Coinbase* en caso de querer iniciar sesión con un usuario y clave, además también tiene la opción de *Coinbase Wallet* que no necesita registro y se ha analizado en el apartado anterior.

4.1.3. NowPayments

Es una plataforma que permite añadir una opción de pago con criptomonedas, al seleccionar esta opción de pago te redirigirá a su página propia web para que puedas seleccionar la criptomoneda entre una amplia variedad de opciones con un simple selector como se puede apreciar en la Figura 9.

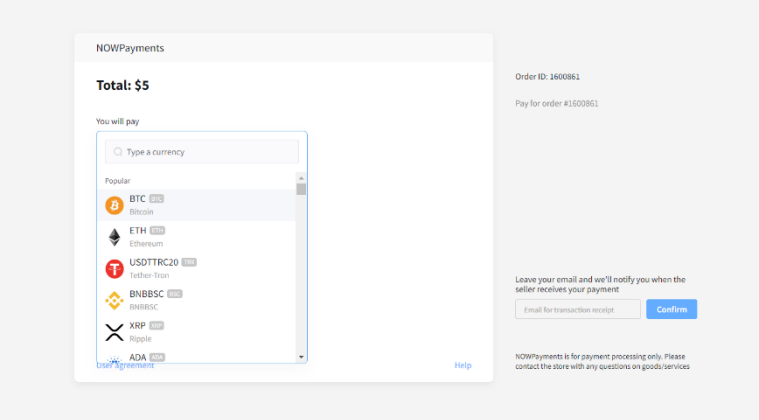


Figura 9. Selector de criptomonedas al pagar con NowPayments

[8] BNB Smart Chain (antes conocida como Binance Smart Chain)

(Fuente propia)

A pesar de que puedes elegir entre criptomonedas de distintas redes (*Bitcoin*, *Solana*, *Ethereum*, *Tron*, *BNB*...), una vez selecciona la criptomoneda y la red que soporta, tendremos que esperar entre 20 y 30 segundos a que se genere una billetera única para realizar el pago.

A pesar de que todos los pagos se deben de realizar de forma manual, copiando y pegando la dirección en tu billetera, tiene una opción algo oculta como se puede ver en la Figura 10, llamada *More payment options* que al pulsarlo desbloquea un enlace para poder hacer el pago mediante una billetera compatible con *WalletConnect*.

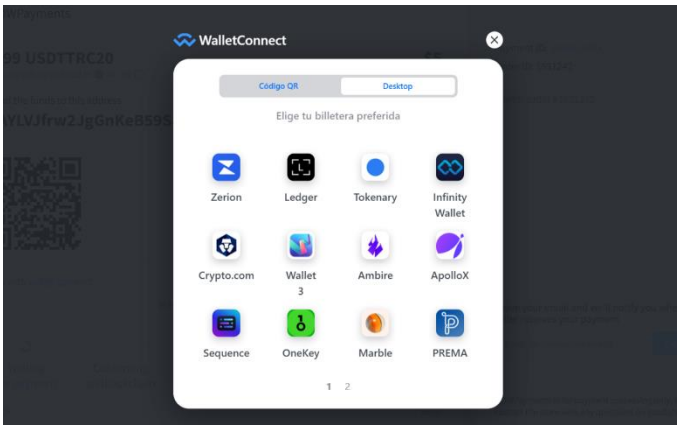


Figura 10. Wallets compatibles con WalletConnect
(Fuente propia)

Una vez hecho el envío, habrá que esperar a que se confirme la transacción en la cadena de bloques para que la plataforma la cuente como realizada.

4.1.4. Conclusión

En este apartado haremos una comparativa entre las pasarelas de pago que se han elegido, en este caso son: *Coinbase*, *Bitpay* y *Nowpayments*. De esta forma no ayudará para orientarnos en el desarrollo de la pasarela de pagos.

	Coinbase	Bitpay	Nowpayments
Características			
Criptomonedas aceptadas distintas	11	16	160
Redes distintas aceptadas (total)	BTC, LTC, ETH, DOGE, BCH, MATIC (6)	BTC*, LTC, ETH, DOGE, BCH, MATIC (7)	BTC, ETH, TRX, BSC, XRP, ADA, SOL, DOGE, MATIC,

			CCHAIN, XMR, DGB, BCH, ZEC, XVG (+41)
Posibilidad de elegir redes distintas con una misma criptomoneda	Sí, Ethereum y Polygon	Sí, Ethereum y Polygon	Sí, Tron, Ethereum y BSC
Número de billeteras disponibles	2 (Coinbase Wallet Y Metamask)	+100 (Incluyendo Metamask, Coinbase Wallet y WalletConnect)	22 (Billeteras compatibles con WalletConnect)
Comisión de la plataforma por cada transacción**	1% por transacción + comisión por retiro o conversión	2% + 0,25$ por transacción	0,5% transacción + 0,5% cambio
Tiempo para realizar el pago	1 hora	15 minutos	20 minutos

Tabla 10. Comparativa entre plataformas de pago.
(Fuente propia)

* Incluye también Bitcoin Lightning[9]
**Datos obtenidos de las páginas webs para transacciones menores a 400.000 dólares al mes.

Como se ha podido observar, las tres pasarelas de pago tienen muchas características en común, por ejemplo, aceptan prácticamente las mismas criptomonedas. Hay que destacar que algunas de ellas apenas tienen movimientos bien sea porque los proyectos no tienen una comunidad fuerte que las respalde o bien porque las comisiones en cadena son muy elevadas y, por tanto, los usuarios con poco poder adquisitivo huyan de estas en busca de algunas redes más económicas y populares como es la *BNB Smart Chain*.

Por otro lado, a pesar de que *Nowpayments* acepta 160 criptomonedas distintas [5], más de 40 redes diferentes y 22 billeteras, no es compatible con *Metamask*, la hot wallet por excelencia.

A raíz de analizar e investigar cada pasarela de pago, se ha podido extraer varios datos muy interesantes para el desarrollo del proyecto, ya que se podrán aprovechar los puntos débiles en favor de los usuarios.

[9] Es una segunda capa agregada a la cadena de bloques de Bitcoin para transacciones fuera de la cadena.

4.2. Billeteras

En este apartado, se analizarán algunas de las billeteras más utilizadas, además de las redes y criptomonedas que permiten. Hay que destacar que cada criptomoneda tiene su propio funcionamiento de Smart contracts (o incluso no tienen, como es el caso de Bitcoin) por lo que es común que una billetera solo sea compatible con una serie de redes, concretamente aquellas que permiten conectarse con la máquina virtual de *Ethereum*, conocida como *EVM*, que posibilitan que los contratos inteligentes se ejecuten en un entorno seguro. Algunas redes compatibles son, por ejemplo, *BNB Smart Chain*, *Avalanche*, *Polygon*, entre otras.

En el caso de *Bitcoin*, el lenguaje de script es muy simple, por poner un ejemplo, es utilizado para realizar operaciones sencillas como hacer una transacción de una billetera a otra, pero no está diseñado para cosas más complejas. Sin embargo, *Ethereum*, fue creada como una plataforma específicamente para contratos inteligentes y con lógica completa.

Por todo esto, no encontraremos herramientas como *Metamask* que tengan la principal criptomoneda por excelencia y solo billeteras manuales para hacer pagos con Bitcoin o Litecoin.

4.2.1. Metamask

La billetera de *Metamask* es una de las más utilizadas por los usuarios para conectarse a las dApps e interactuar con los ecosistemas blockchains. *Metamask* permite conectar una infinidad de cadenas de forma muy sencilla a través de páginas como *Chainlist*, donde simplemente tienes que seleccionar la red que quieres añadir y aceptar la notificación, esta página web permite reducir el proceso de añadir de forma manual los datos técnicos como son: nombre de la red, dirección URL de la red, identificador de cadena, símbolo de la moneda, entre otros.

Metamask es una wallet que permite conectar todas las redes que son compatibles con la *Ethereum Virtual Machine*, además de añadir sus respectivos tokens mediante la introducción de la dirección de contrato correspondiente.

4.2.2. Trust Wallet

Trust Wallet es una billetera muy conocida ya que pertenece a Binance, el mayor exchange de criptomonedas. Esta billetera posee una integración de navegador Web3 que te permite conectarte con dApps compatibles con redes *EVM*, además posee una aplicación móvil compatible con *iOS* y *Android* en la que aceptan 65 redes distintas [6].

La integración con *Web3* de esta billetera se hace a través del protocolo de comunicación *WalletConnect*, que como hemos podido observar en el apartado de soluciones existentes, es

aceptada por *BitPay* y *NowPayments*, ya que te permite una gran compatibilidad con billeteras, además de utilizar un cifrado end-to-end.

4.3. Redes y blockchains

Una vez examinadas las soluciones y billeteras existentes en los apartados anteriores, en este se estudiarán cada una de las redes utilizadas que aceptan esas billeteras, para ver cuáles son las más utilizadas por los usuarios e incluso que permitan la integración de una mayor cantidad de criptomonedas populares.

Como podemos apreciar en la Figura 11, según la aplicación de *DefiLlama* que permite analizar todo tipo de métricas *on-chain* de todas las redes de criptomonedas a través de sus respetivas cadenas de bloques, las siete redes más utilizas a fecha 1/03/2023 dentro del ecosistema de las criptomonedas además de tener el 93% del volumen de dinero son: *Ethereum*, *BSC*, *Arbitrum*, *Polygon*, *Optimism*, *Avalanche* y *Solana*.

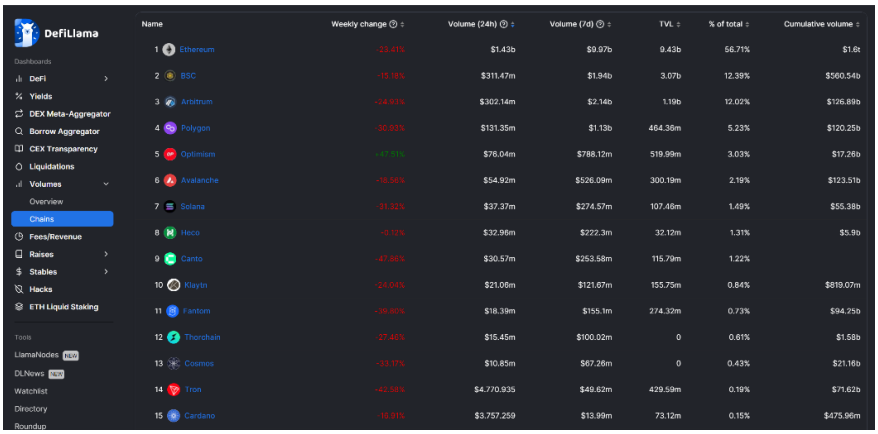


Figura 11. Volumen en dólares de las últimas 24 horas según Defillama (Fuente propia)

Por otro lado, es interesante conocer aquellas redes blockchain que tienen un mayor uso de monedas estables o *stablecoins*, es decir, aquellas que replican el precio del dólar u otros valores, ya que muchos usuarios pueden estar protegidos en estos activos cuando el mercado está muy volátil. Incluso opten por estas monedas porque sean más fáciles de administrar y contabilizar.

Como podemos ver en la Figura 12, estas redes según *DefiLlama* son: *Ethereum* (58,22%), *Tron* (29,29%), *BSC* (5,21%), *Polygon* (1,3%) y *Solana* (1,25%). Sin embargo, comentaremos solo las tres primeras, ya que suman aproximadamente el 95% del volumen de monedas estables.

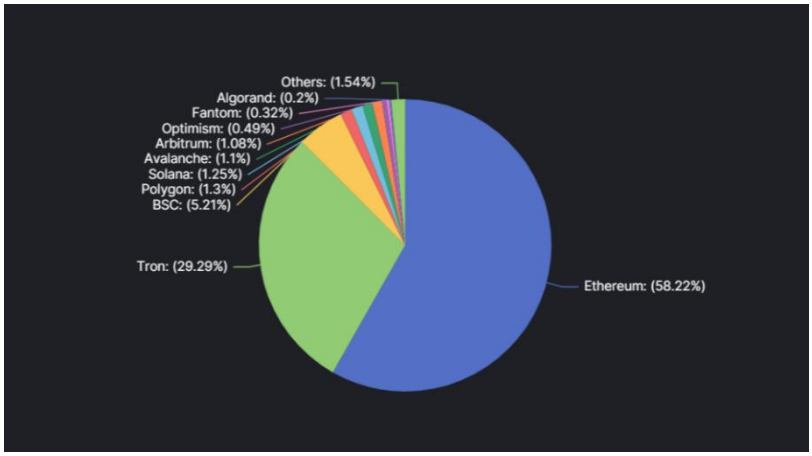


Figura 12. Porcentaje de cuota de monedas estables según las redes (Fuente propia)

4.3.1. Red Ethereum

La red de *Ethereum* es la más popular de todas con más de la mitad de la cuota de mercado, ya que ha sido una de las primeras en permitir los contratos inteligentes para poder crear *dApps* e interaccionar con ellas además de favorecer la demanda del sector. Por otro lado, también posibilita una mayor cantidad de transacciones y, por tanto, una mayor escalabilidad, lo que proporciona un excelente caldo de cultivo para que otros proyectos se desarrollen en él.

4.3.2. Red Tron

En segundo lugar, tenemos una red que no es compatible con la Ethereum Virtual Machine, pero sus bajas comisiones y su relativa velocidad han hecho que sea una opción muy popular para hacer transacciones con monedas estables entre dos exchanges centralizados.

Por ejemplo, con la red Tron puedes enviar una cantidad concreta de dólares por menos de 1 dólar, no obstante, con Ethereum serían entre 2-5 dólares como mínimo.

4.3.3. Red BNB Smart Chain

En tercer lugar, tenemos la red *BSC* de la criptomoneda emitida por el *Exchange Binance*, que durante estos dos últimos años ha tenido un movimiento explosivo colocándose como segunda posición en cuanto a volumen acumulado de casi 600 billones de dólares.

A diferencia de la de red *Tron*, esta si es compatible con *EVM* y junto con unas comisiones bajas mientras otras redes se veían damnificadas por el colapso de la red, han permitido que su

moneda principal, *BNB*, se encuentre entre una de las cinco primeras criptomonedas con mayor capitalización.

4.4. Tecnologías para el desarrollo

En esta sección analizaremos las diferentes herramientas y tecnologías que se pueden utilizar para la implementación de un módulo de pagos con criptomonedas en la aplicación web. En este sentido, es importante tener en cuenta que la elección de las tecnologías adecuadas es un factor clave para el éxito del proyecto.

En este caso, se destaca la elección de Next.js como *framework* de React[10]. Next.js es un *framework* que nos permite crear aplicaciones web con React de manera sencilla y eficiente, ya que permite la posibilidad de generar páginas estáticas y dinámicas de forma automática [7]. Además, ofrece la capacidad de actualizar la interfaz de usuario de forma dinámica sin necesidad de recargar la página, lo que reduce los tiempos de carga y mejora del rendimiento.

Una de las principales ventajas de utilizar *Next.js* es que nos permite renderizar contenido en el servidor, lo que nos garantiza que el contenido esté disponible para todos los motores de búsqueda.

Por poner un ejemplo, los robots de *crawling*[11] de Microsoft revisan la web para ver el contenido, con React tienen problemas para leerlo porque no ejecutan el *Javascript* (como si lo hace Google desde hace unos años). Por esta razón, se optó por el uso de Next.js para ejecutar el código en el servidor y así garantizar que el contenido se indexe correctamente en los motores de búsqueda.

Es importante tener en cuenta que, aunque la utilización de *Next.js* en el servidor para renderizar contenido tiene ventajas importantes en términos de *SEO*[12], también presenta algunas desventajas. En primer lugar, el proceso de renderización del servidor puede ser más lento que la carga de contenido directamente en el cliente. Además, la carga en el servidor puede generar una sobrecarga en él, lo que podría afectar el rendimiento de la web.

[10] Librería de JavaScript para la construcción de interfaces, desarrollado por Facebook y de código abierto.
[11] Método que utilizan los motores de búsqueda para clasificar e indexar el contenido de una web.
[12] Search Engine Optimization (Optimización para motores de búsqueda)

Sin embargo, en general, las ventajas de utilizar *Next.js* superan ampliamente estos inconvenientes, especialmente en proyectos en los que la indexación del contenido es un factor muy importante para el éxito no solo del proyecto, sino también del negocio.

Por otro lado, se utilizará *Nginx*, un servidor web de alto rendimiento que es utilizado para servir contenido estático y dinámico en aplicaciones web. *Nginx* ofrece una serie de ventajas interesantes como la posibilidad de escalar la aplicación a medida que aumenta el tráfico, así como la capacidad de manejar grandes cantidades de solicitudes de forma eficiente.

A pesar de las ventajas, *Nginx* tiene una curva de aprendizaje mayor que si se utilizara Apache, ya que esta última es más intuitiva y cuenta con muchos más módulos y extensiones [8].

Por último, se utilizará *MongoDB*, una base de datos *NoSQL* de código abierto. Algunas de las ventajas por las que se utilizará esta base de datos es principalmente por la flexibilidad en la estructura de datos, ya que facilita la adaptación a los cambios en los esquemas de datos. Por ejemplo, al añadir nuevas criptomonedas o redes.

4.5. APIS

4.5.1. Coingeko

Uno de los requisitos más importantes para una pasarela de pago que va incluir diferentes criptomonedas, es tener el precio de cada una de ellas en tiempo real a la solicitud del pago, ya que se trata de un mercado muy volátil y en apenas 1h podemos ver variaciones en el precio muy grandes.

Hay muchas plataformas en el mercado que ofrecen precios en tiempo real de las criptomonedas. Pero después de compararlas, se va a utilizar la API de Coingecko ya que tiene una versión gratuita que ofrece entre 10 y 30 llamadas por minuto. Esta opción incluye para comprobar el precio de más de 12.000 criptomonedas [9]. Además, para tener acceso a esta versión, solo se necesitaría hacer una atribución de que los precios de estas criptomonedas son obtenidos de esta plataforma, en caso de pagar una versión superior, no será necesario.

5. Objetivos

Es crucial definir objetivos claros y concisos que destaquen el objetivo de un proyecto, no solo para tener una comprensión clara de nuestra propia idea, sino también para comunicarla de manera efectiva a los demás. En lugar de tratar de enumerar todas las posibles funcionalidades de nuestra aplicación, debemos enfocarnos en lo que realmente queremos lograr con nuestro trabajo. Esto nos permitirá definir un camino claro y directo para conseguir nuestros objetivos y metas además de mantener el enfoque necesario para alcanzarlos.

Para hacerlo, utilizaremos la estrategia SMART[13] propuesta por Doran [10], que nos permite tener en cuenta una serie de características esenciales a la hora de definir nuestros objetivos.

Como objetivo principal, queremos desarrollar una pasarela de pagos con criptomonedas que permita a los usuarios realizar transacciones de manera segura y confiable, utilizando una amplia variedad de criptomonedas. Los subobjetivos relacionados con la aplicación que podemos enumerar son los siguientes:

- Que la pasarela de pago sea intuitiva y fácil de usar, si no transmite confianza es posible que los usuarios se echen atrás a la hora de hacer el pago.
- Que el usuario pueda elegir entre varias billeteras de criptomonedas y redes de forma sencilla.
- Que la pasarela tenga un diseño adaptable para todo tipo de pantallas, tanto ordenadores de escritorio como dispositivos móviles.
- Poner en producción la pasarela de pago para probar que todo funciona correctamente y con el lanzamiento de productos los usuarios tengan la opción de pagar con ella.

También es interesante considerar subobjetivos que sean más abstractos y personales, pero que sean igualmente útiles, ya que se convierten en metas a alcanzar durante todo el proceso de desarrollo del proyecto.

- Aprender a crear una pasarela con herramientas del sector web3 y criptomonedas para el proyecto personal.
- Aprender a utilizar Smart Contracts en diferentes cadenas de bloques compatibles.

[13] Sus siglas en ingles significan: (Específico, Medible, Alcanzable, Realista y Oportuno)

6. Metodología

En el siguiente apartado, se abordará la planificación y organización del Trabajo de Fin de Grado, detallando las metodologías seleccionadas y las razones que las respaldan. Asimismo, se expondrán las herramientas que serán utilizadas en la implementación de dichas metodologías para asegurar el éxito del proyecto.

A pesar de que existen muchas metodologías para el desarrollo de un proyecto, en nuestro caso se trata de un trabajo individual, por tanto, para la redacción de esta memoria se utilizará una metodología Kanban, ya que la tarea no sufrirá muchos cambios y no se realizarán iteraciones en este proceso.

Sin embargo, para la implementación del módulo de la pasarela, se utilizará una combinación del Proceso Unificado de Desarrollo y la metodología Scrum, ya que son más dinámicas y permiten dividir el trabajo en un mayor número de tareas, mejorando así el producto final poco a poco en cada sprint o iteración.

El Proceso Unificado de Desarrollo de Software (RUP) es un enfoque de desarrollo iterativo e incremental centrado en la arquitectura, el análisis de riesgos y la planificación a largo plazo [11]. Por otro lado, Scrum es una metodología de trabajo ágil que promueve el desarrollo iterativo y el enfoque en el cliente [12].

Como se puede observar en la Figura 13 para combinar RUP y Scrum seguiremos los siguientes pasos:

En primer lugar, una fase de inicio, donde estableceremos la visión del proyecto, los objetivos y el alcance. Utilizaremos las prácticas de RUP para definir la arquitectura y realizar un análisis de riesgos preliminar. También estableceremos un Backlog inicial utilizando técnicas de Scrum.

En segundo lugar, una fase de elaboración, donde se refinará la arquitectura y gestión de riesgos mientras que se implementan esprints para desarrollar incrementos de software que se ajusten a la arquitectura.

En tercer lugar, una fase de construcción, donde se completará el producto utilizando esprints, además se seguirán monitorizando la arquitectura y los riesgos. También, se irán completando tareas de software del Backlog.

Por último, una fase de transición, donde el producto se prepara para su lanzamiento y se realiza la capacitación y la documentación necesaria. Aquí se realizará una transición sin problemas al

entorno de producción. Igualmente, se recopilarán comentarios de los usuarios para aprender de la experiencia y así mejorar el proceso de desarrollo y el producto en sí.

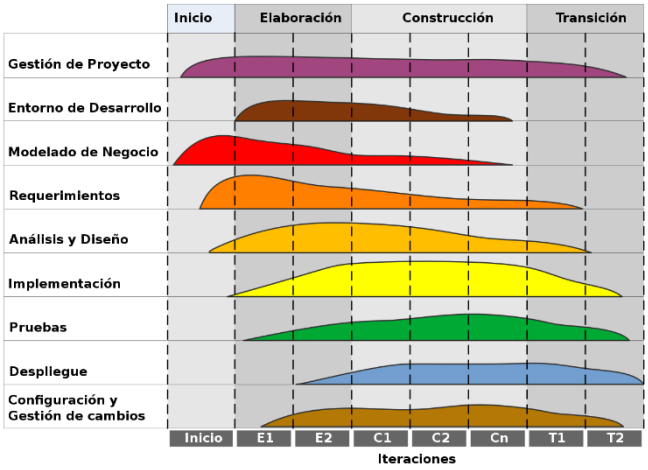


Figura 13. Diagrama de fases del Proceso Unificado de Desarrollo
(Fuente: Wikipedia https://es.wikipedia.org/wiki/Proceso_unificado)

Para todo esto utilizaremos Trello [13], que nos permite organizar las tareas por columnas tanto para la metodología Kanban como la Scrum y RUP, de forma que se irán moviendo en función del estado en el que se encuentre dicha tarea.

Como podemos apreciar en la Figura 14, en las primeras columnas tendremos el "Backlog" que nos servirá para colocar la lista de ideas y tareas de forma ordenada por cada sección del trabajo.

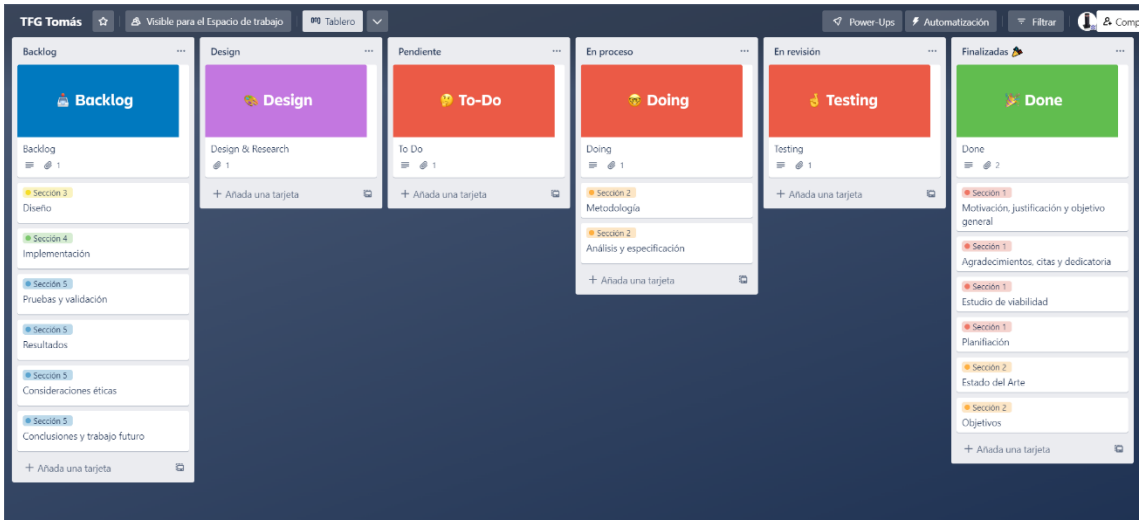


Figura 14. Tablero de Trello
(Fuente propia)

Por otro lado, en la columna naranja "To-Do" encontraremos las tareas para realizar, en "Doing" las que se están realizan en ese momento y en "Testing" aquellas que se están comprobando y

están a la espera del visto bueno por parte del tutor. Por último, en la columna “Done” tendremos las tareas finalizadas. En caso de RUP se utilizará en la fase de construcción.

Por otra parte, una herramienta que puede ser de gran utilidad para controlar el tiempo invertido en un proyecto es Clockify [14]. Esta aplicación realiza un seguimiento automático de las actividades realizadas en el ordenador, lo que nos permitirá conocer en detalle cómo estamos utilizando nuestro tiempo durante el trabajo en el desarrollo del proyecto. De esta manera, podremos identificar y eliminar distracciones innecesarias y optimizar nuestro tiempo de trabajo para ser más eficientes y productivos en el desarrollo del TFG.

Para poder visualizar cuanto tiempo se ha empleado en cada tarea y sección, se utilizarán etiquetas y se organizará en secciones. De esta forma se podrá visualizar de forma individual como en la Figura 15.

Figura 15. Registro de horas en ClockiFy
(Fuente propia)

7. Análisis y especificación

En esta sección se aborda el problema a resolver y se define el alcance de la solución a desarrollar, el cual dependerá de diversos factores como los usuarios que lo utilizarán. Para asegurar una definición sistemática de los aspectos a considerar en el diseño de la solución software nos apoyaremos del estándar IEEE 830 [15], sin embargo, nos centraremos en los requisitos.

7.1. Requisitos

Se recogerán a continuación todos los requerimientos del proyecto, los cuales definirán las funcionalidades necesarias para cumplir con los objetivos. Cada requerimiento contará con un identificador único (RF-XX para los requisitos funcionales y RNF-XX para los no funcionales), especificando el tipo de usuario al que está relacionado y una breve descripción.

Además, se distinguirán entre aquellos requisitos que deben ser implementados de forma imprescindible (básico) y los opcionales, los cuales se desarrollarán en caso de disponer de tiempo suficiente. En este caso, el tipo de usuario se limita al usuario cliente, es decir, aquel que hace uso de la pasarela de pago.

7.1.1. Requisitos funcionales

A continuación, detallaremos los requisitos funcionales, concretamente los servicios y funcionalidades necesarios que la pasarela de pago debe ofrecer al usuario para cumplir con su objetivo.

Tabla 11. Requisitos funcionales.

Identificador	RF-01
Usuario	Usuarios
Tipo	Básico
Descripción	Realizar un pago con criptomonedas

Identificador	RF-02
Usuario	Usuarios
Tipo	Básico
Descripción	Buscar y seleccionar una criptomoneda entre varias posibilidades

Identificador	RF-03
Usuario	Usuarios
Tipo	Básico
Descripción	Poder elegir entre varias redes blockchain una misma criptomoneda (Ethereum, BSC, Polygon, Arbitrum, Optimism…)

Identificador	RF-04
Usuario	Usuarios
Tipo	Básico
Descripción	Seleccionar una billetera con la que realizar el pago de la criptomoneda elegida.

Identificador	RF-05
Usuario	Usuarios
Tipo	Secundario
Descripción	Integrar la pasarela de pagos en la web

Identificador	RF-06
Usuario	Usuarios
Tipo	Secundario
Descripción	Permitir descargar o enviar la factura de compra.

7.1.2. Requisitos no funcionales

Los requisitos no funcionales se refieren al desempeño y las limitaciones del sistema en sí mismo. En este caso, la categoría "Tipo" muestra el ámbito al que están asociados.

Tabla 12. Requisitos no funcionales.

Identificador	RNF-01
Usuario	Sistema
Tipo	Usabilidad
Descripción	La pasarela de pago tendrá una interfaz fácil de usar, diseñada para que cualquier persona familiarizada con las aplicaciones descentralizadas de criptomonedas pueda utilizarla sin requerir un extenso proceso de aprendizaje.

Identificador	RNF-02
Usuario	Sistema
Tipo	Accesibilidad
Descripción	La pasarela de pagos tendrá un diseño accesible que considerará elementos como el contraste de colores, la facilidad de lectura y el mínimo de clicks necesarios para completar una acción.

Identificador	RNF-03
Usuario	Sistema
Tipo	Disponibilidad
Descripción	Al ser un método de pago, la pasarela estará disponible en todos los dispositivos, sin embargo, dependerá de la billetera elegida si es compatible con el dispositivo.

Identificador	RNF-04
Usuario	Sistema
Tipo	Rendimiento
Descripción	El sistema deberá ser ágil en cuanto a su tiempo de respuesta, de tal forma que el usuario obtenga la información que solicita en un plazo máximo de 3 segundos tras realizar una acción.

Identificador	RNF-05
Usuario	Sistema
Tipo	Confidencialidad
Descripción	La información privada de los usuarios que se guarden en el sistema solo será accesible para los administradores de la aplicación y no estará disponible públicamente.

Identificador	RNF-06
Usuario	Sistema
Tipo	Legalidad
Descripción	Se garantizará que todas las herramientas utilizadas para desarrollar e implementar la aplicación cuenten con las licencias correspondientes. Además, la aplicación cumplirá con el Reglamento General de Protección de Datos y ofrecerá a los usuarios acceso a sus derechos.

8. Diseño

En esta sección del TFG nos centraremos en el diseño de la solución propuesta, el cual es el aspecto más crítico e importante del proyecto. El diseño es el núcleo principal del TFG y debe abordar todos los requerimientos funcionales y no funcionales que se han establecido previamente, haciendo referencia a los identificadores correspondientes de cada requerimiento en el proceso.

8.1. Diseño de la persistencia

En esta sección, abordaremos el diseño de la persistencia de datos para nuestro proyecto. La persistencia de datos es un aspecto crucial en cualquier aplicación, ya que permite almacenar y gestionar información de manera eficiente y segura. Para lograr esto, es muy importante elegir el enfoque y la tecnología adecuados que se ajusten a las necesidades y requisitos del proyecto.

8.1.1. Almacenamiento de datos

Para el almacenamiento de datos, hemos optado por una base de datos NoSQL, MongoDB, debido a su flexibilidad, escalabilidad y rendimiento.

Antes de diseñar el modelo de datos es buena idea pensar los objetivos que va a tener nuestra aplicación. En nuestro caso, la pasarela de pagos tendrá: usuarios, criptomonedas, redes, billeteras, transacciones y facturas.

A lo largo de este apartado, describiremos las colecciones y documentos que conforman nuestra base de datos, así como las relaciones entre ellos. También veremos cómo se guardarán y gestionarán los datos de las criptomonedas, billeteras compatibles y transacciones realizadas por los usuarios.

Como hemos mencionado en apartados anteriores, MongoDB es una base de datos NoSQL basada en documentos, por tanto, no utiliza tablas y filas como en las bases de datos relacionales. MongoDB almacena los datos en documentos flexibles, que se organizan en colecciones. Estos documentos tienen un formato llamado BSON, es decir, una representación binaria de JSON [16].

Los documentos son la unidad básica de almacenamiento de datos en MongoDB, contiene pares clave-valor muy parecidos a los objetos JSON. Además, un documento puede contener

subdocumentos, lo que permite representar datos jerárquicos. Las colecciones agrupan documentos relacionados entre sí, y a diferencia de las tablas de bases de datos relacionales, pueden tener diferentes estructuras y campos.

Tras este análisis inicial, es esencial examinar cómo optimizar la organización de los datos para llevar a cabo operaciones CRUD[14] de manera eficiente en MongoDB.

Por ejemplo, un aspecto importante de MongoDB es que los documentos se recuperan por completo, y, por tanto, no se puede obtener información parcial de un documento. Dependiendo de cómo se empleará la información, será conveniente mantenerla agrupada o separada.

En nuestro caso particular, se podría considerar que, en cuanto a la modificación de un dato en este tipo de bases de datos, el proceso puede resultar más costoso. Esto se debe a que la información no se almacena en una ubicación específica como en las bases de datos relacionales. Por lo tanto, es necesario buscar el dato en toda la base de datos antes de realizar cualquier cambio. Sin embargo, en nuestra aplicación se realizarán principalmente consultas, en lugar de modificaciones, lo que minimiza el impacto de este aspecto.

Tras explicar cómo funcionará la base de datos, procedemos a definir el modelo de datos.

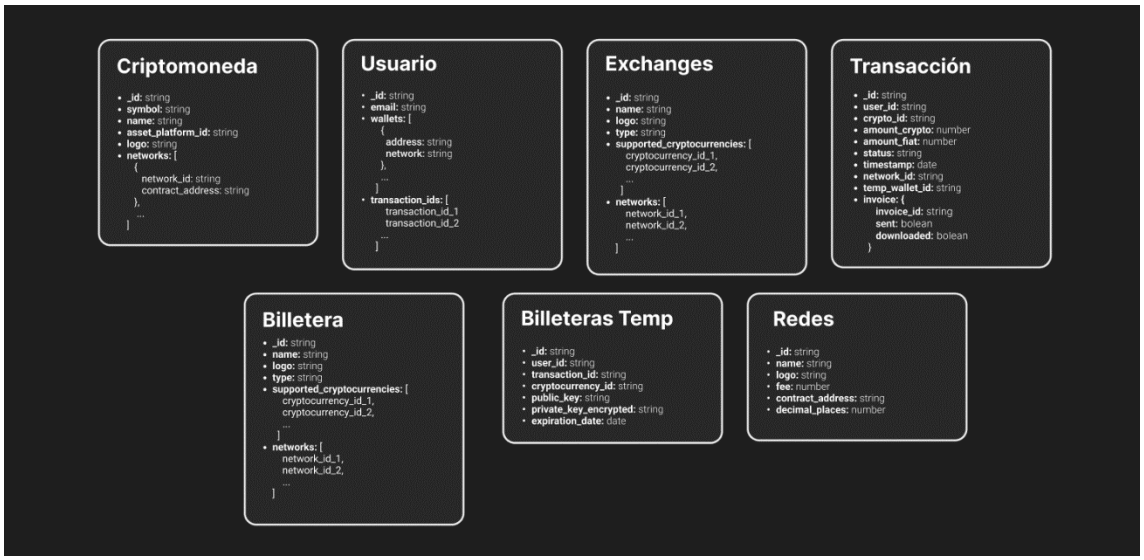


Figura 16. Diseño de BD
(Fuente propia)

[14] CRUD es un acrónimo que se refiere a las cuatro operaciones básicas en la gestión de datos: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar).

A continuación, examinaremos en detalle los componentes que conforman el modelo de datos, ya que no se pueden apreciar claramente a simple vista.

En primer lugar, tenemos una colección llamada Criptomoneda, que va a ser nuestra tabla principal y contendrá toda la información básica de ellas. Seguidamente, se expondrán los campos que integran esta colección:

- _id: Este campo es un identificador único de tipo cadena (string) que representa a cada criptomoneda en la base de datos. Utilizaremos este campo para comprobar con el precio en tiempo real proporcionado por la API de datos de CoinGecko.
- symbol: Es una cadena (string) que representa el símbolo de la criptomoneda, como "ETH" para Ethereum o "MATIC" para Polygon.
- name: Este campo es una cadena (string) que contiene el nombre completo de la criptomoneda, como "Bitcoin", esto facilitará la búsqueda por nombre.
- asset_platform_id: Es una cadena (string) que almacena el identificador de la plataforma de activos asociada a la criptomoneda. Por ejemplo, en el caso de tokens ERC-20 en la red Ethereum, este campo contendría el identificador de la red Ethereum. Nos permitirá filtrar y buscar criptomonedas según la plataforma en la que se encuentren.
- logo: Es una cadena (string) para almacenar la URL de la imagen del logotipo de cada criptomoneda. Esto nos permitirá mostrar visualmente la criptomoneda en la interfaz de usuario.
- networks: Es un array de objetos, este campo contendrá una lista de objetos que representan las redes compatibles con la criptomoneda. Cada objeto de la lista incluirá un *network_id* que hace referencia a la tabla Redes y un *contract_address* que contiene la dirección del contrato del token (si corresponde) en la red específica.

Posteriormente la colección Usuario tendrá los siguientes campos:

- _id: Es una cadena (string) que contiene un identificador único para cada uno de los usuarios de nuestra base de datos. Este identificador se genera automáticamente al crear un nuevo registro en la base de datos.
- email: Es una cadena (string) para almacenar el correo electrónico del usuario, nos permitirá enviarle la factura o comunicarnos con él.
- wallets: Es un array de objetos que contendrá las billeteras asociadas al usuario, cada objeto contendrá una *address* que contiene la dirección de la billetera del usuario y una *network* que hace referencia al nombre de la red en la que se encuentra la dirección de la billetera.

- transaction_ids: Es un array de Strings, este campo contendrá una lista de ids de transacciones asociadas con el usuario. Cada identificador en la lista hace referencia a un registro en la colección Transacción.

Toda esta estructura nos permitirá almacenar y gestionar la información relevante sobre los usuarios, además hay que recordar que un mismo usuario puede utilizar una sola billetera en todos sus pagos o para cada pago una distinta, por otro lado, guardaremos las transacciones asociadas a los pagos que haya realizado.

Seguidamente tenemos las colecciones Exchange y Billetera, que contendrán información muy similar entre ellas, primero veremos Exchange que tiene como campos:

- _id: Es una cadena (string) que contiene el identificador único para cada exchange en la base de datos.
- name: Es una cadena (string), este campo almacenará el nombre del exchange. Nos permitirá mostrar el nombre del exchange en la interfaz de usuario y facilitar la identificación de los exchanges por parte de los usuarios.
- logo: Es una cadena (string) que tendrá la URL del logotipo del exchange para mostrar en la interfaz de usuario y mejorar la experiencia de este.
- supported_cryptocurrencies: Este array de strings contendrá una lista de identificadores de criptomonedas que el Exchange admite y son compatibles con la pasarela. Cada identificador en la lista hace referencia a un registro en la colección Criptomoneda.
- network: Es un array de strings, este campo contendrá una lista de identificadores de redes compatibles con el exchange. Cada identificador en la lista hace referencia a un registro en la tabla Red.

En el caso de la colección Billetera, es muy similar a la colección Exchange, ya que ambos almacenan información sobre entidades compatibles con la pasarela de pagos. La principal diferencia entre las dos colecciones es el campo "type" en la colección Billetera, que permite clasificar las billeteras en el caso de que haya muchas según su tipo, como "web", "móvil" o "EVM" por ejemplo.

La colección Redes tendrá la siguiente estructura:

- _id: Es una cadena (string) que contiene el identificador único para cada red en la base de datos.
- name: Es una cadena (string), este campo almacenará el nombre de la red. Nos permitirá buscar y mostrar el nombre de la red.

- logo: Es una cadena (string) que tendrá la URL del logotipo de la criptomoneda con la que se pagan las comisiones de esa red, y así mostrarla en la interfaz para mejorar la experiencia del usuario.
- fee: Es un campo que representa la tarifa asociada con la realización de transacciones en la red. Nos permite informar a los usuarios sobre el coste de las tarifas para ayudarles a tomar una decisión.

La colección Transacción almacena información sobre las transacciones realizadas a través de la pasarela de pagos y tiene la siguiente estructura:

- _id: Es una cadena de tipo string para identificar cada transacción en la base de datos.
- user_id: Es una cadena (string), identifica el usuario que realiza la transacción.
- crypto_id: Es una cadena (string) contiene el identificador de la criptomoneda utilizada en la transacción.
- amount_crypto: Es un campo (number) para la cantidad de criptomoneda involucrada en la transacción.
- amount_fiat: Es un campo (number) con el valor en moneda fiduciaria de la cantidad de criptomoneda en la transacción.
- status: Es una cadena (string) con el estado actual de la transacción (por ejemplo, pendiente, confirmada, etc.). En el caso de implementar pagos con criptomonedas que no tienen Smart Contracts y si, o en el caso de que se hagan desde billeteras de exchanges directamente, utilizaremos los siguientes estados:
    1. pending: El pago está pendiente y aún no se ha confirmado en la red blockchain. Este estado se aplica a ambos tipos de transacciones (contratos inteligentes y pagos directos).
    2. waiting_for_confirmations: Este estado es específico para pagos directos (por ejemplo, Bitcoin) y se utiliza cuando la transacción se ha detectado en la red, pero aún no ha alcanzado el número mínimo de confirmaciones requerido.
    3. confirmed: El pago ha sido confirmado en la red blockchain y ha alcanzado el número mínimo de confirmaciones requerido. En el caso de contratos inteligentes, esto indica que la transacción del contrato inteligente se ha realizado correctamente. Para pagos directos, indica que el pago ha sido recibido en la dirección de la billetera temporal y ha alcanzado el número mínimo de confirmaciones.

4. underpaid: Este estado se aplica a pagos directos cuando la cantidad recibida en la dirección temporal es inferior a la cantidad requerida.
5. overpaid: Este estado se aplica a pagos directos cuando la cantidad recibida en la dirección temporal es superior al monto requerido.
6. expired: La transacción ha expirado, lo que puede ocurrir si no se realiza el pago dentro del tiempo límite establecido ya que el precio podría haber cambiado considerablemente.
7. failed: La transacción ha fallado debido a un error en el contrato inteligente o un problema en la red blockchain.

- timestamp: Es un campo (date) que contiene la fecha y hora en que se creó la transacción.
- network_id: Es un campo (string) con el identificador de la red blockchain en la que se realiza la transacción.
- temp_wallet_id: Es un campo (string) con la dirección temporal de la billetera a la que el usuario envía el pago.
- invoice: Es un campo de tipo object que contiene información sobre la factura asociada a la transacción.
    - invoice_id: Es un campo (string) con el identificador único de la factura.
    - sent: Es un campo (boolean) que indica si la factura ha sido enviada al usuario.
    - downloaded: Es un campo (boolean) para indicar si la factura ha sido descargada por el usuario.

Por último, se procederá a crear una colección denominada Billetera Temporal. Esta medida resulta necesaria en caso de que se implementen pagos con criptomonedas que no dispongan de Smart Contracts, o bien si se desea pagar directamente desde un exchange. La función de esta colección consistirá en diferenciar los pagos entre dos usuarios que pretendan pagar el mismo producto con idéntico precio y en el mismo momento. Sin ella, no sería posible determinar qué usuario ha realizado el pago, puesto que ambos lo efectuarían a la misma dirección. La colección estará compuesta por los siguientes campos:

- _id: Es una cadena de tipo string para identificar cada billetera generada de forma temporal en la base de datos.
- user_id: Es una cadena (string), identifica el usuario que realiza el pago a la dirección.
- transaction_id: Es una cadena (string) contiene el identificador de la transacción.
- cryptocurrency_id: Es una cadena (string) contiene el identificador de la criptomoneda con la que se ha realizado el pago.

- public_key: Es una cadena (string) con la dirección de la billetera a la que se realiza el pago.
- private_key_encrypted: Es una cadena (string) con la clave privada de la dirección publica, esta información estará cifrada por seguridad y con acceso solo por parte del administrador para retirar los fondos.
- expiration_date: Es una cadena (date) con la fecha de vencimiento de la dirección. Después de esta fecha, la billetera ya no es válida para realizar pagos.

8.2. Diseño arquitectura conceptual

En este apartado, se esbozan de forma simple los módulos y bloques funcionales que conformarán el sistema, proporcionando una descripción de cómo se desarrollará la solución y estableciendo las bases que condicionarán las tecnologías que se utilizarán más adelante.

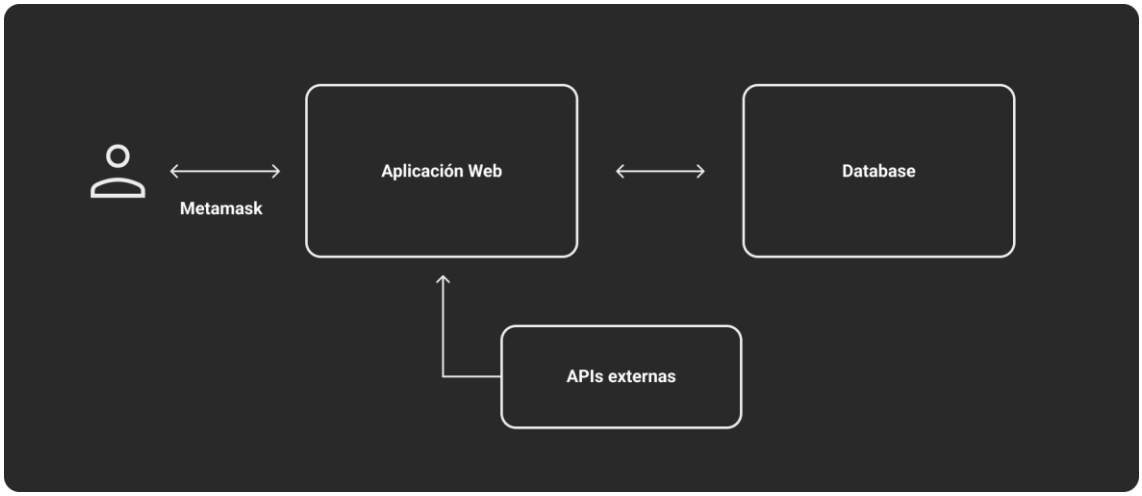


Figura 17. Diagrama de la arquitectura conceptual
(Fuente propia)

En primer lugar, el usuario seleccionará una criptomoneda y la billetera Metamask para realizar el pago, la lógica de la aplicación web se encargará de realizar las peticiones a la base de datos para ofrecerle la información necesaria.

Por otro lado, la aplicación web obtendrá de la API externa los precios y datos necesarios para poder calcular el coste en la criptomoneda elegida.

8.3. Diseño arquitectura tecnológica Front/Back-end

Para el diseño de la arquitectura tecnológica, tomaremos como punto de partida el diagrama de arquitectura conceptual previamente establecido. El objetivo es definir y concretar las tecnologías específicas que se emplearán en cada bloque de la solución propuesta.

Durante el estudio del arte, se investigaron diversas tecnologías óptimas para abordar el problema planteado en el proyecto de desarrollo de una pasarela de pagos con criptomonedas, teniendo en cuenta las necesidades y características específicas de la solución. En este caso, al tratarse de un módulo enfocado en la gestión de transacciones con criptomonedas, se ha considerado el siguiente stack tecnológico web en el diseño de la arquitectura tecnológica:

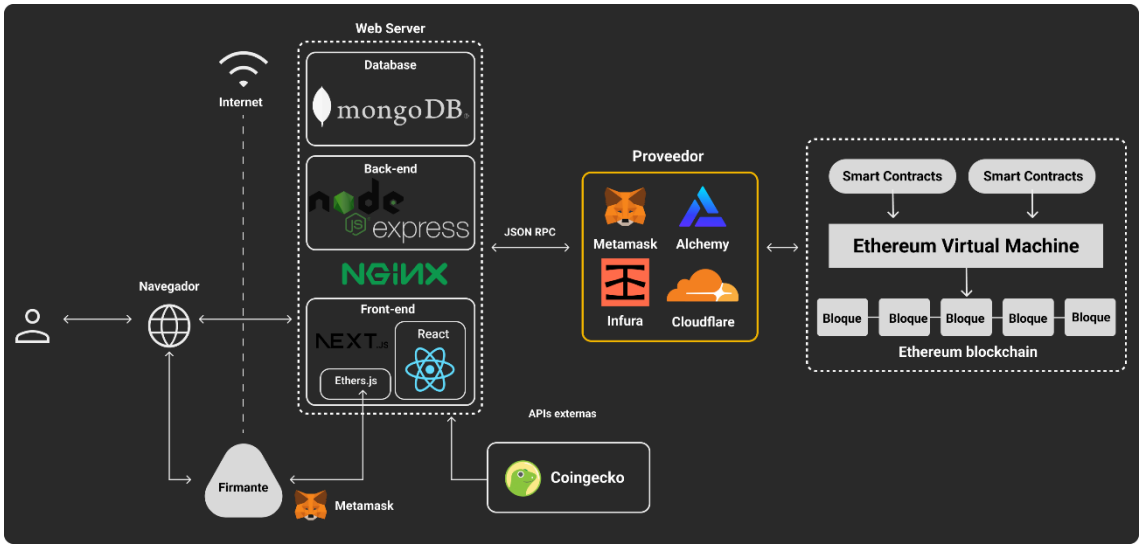


Figura 18. Arquitectura tecnológica
(Fuente propia)

Para la implementación del Front-end utilizaremos React y Next.js, este último recordemos que nos permite el renderizado del contenido de la aplicación en el servidor antes de enviarlo al navegador del usuario, lo que mejora la experiencia del usuario, el rendimiento y la optimización para motores de búsqueda (SEO). Como parte de la pila tecnológica se utilizará TypeScript, SCSS (Sass)[15], y HTML5.

Utilizaremos Ether.js que es una biblioteca JS de código abierto para facilitar la interacción con la red Etherum, ya que ofrece una serie de funciones y herramientas para trabajar con contratos inteligentes, así como para gestionar y manipular datos relacionados con Ethereum [17]. Ether.js

[15] Sass (Syntactically Awesome Style Sheets) es un preprocesador de CSS que permite a los desarrolladores escribir CSS de una manera más estructurada y modular.

hará de intermediario entre la aplicación web (el front-end) y *Metamask*, que se encargará de firmar transacciones y autentificar al usuario.

Para el Back-end, utilizaremos *Node.js* y *Express*. Node.js es un entorno de ejecución que permite ejecutar código JavaScript en el servidor. Express es un framework de Node.js que nos proporciona una serie de características para desarrollar aplicaciones web. Esta combinación nos permite construir un back-end para administrar las solicitudes de los clientes.

Además, se utilizará *Nginx*, un servidor web y proxy reverso, que puede servir contenido estático y enrutar las solicitudes entrantes a diferentes servicios de back-end.

Por último, para la Database, *MongoDB*, una base de datos NoSQL orientada a documentos que permite esquemas flexibles y dinámicos, es decir, permite modificar fácilmente la estructura de los datos sin tener que cambiar el esquema de la base de datos.

Como APIs utilizaremos *Coingecko* para obtener información y guardar en nuestra base de datos como el nombre de Exchanges, criptomonedas, logos y redes. Además de consultar los precios en tiempo real de cada uno de los activos cuando sea pertinente.

Además, como también podemos ver representado en la arquitectura tecnológica, utilizaremos algunos proveedores como *Metamask*, *Alchemy*[18] o *Infura*[19] para conectar la aplicación web y la red Ethereum mediante el protocolo JSON-RPC. Los proveedores se encargan de enviar o recibir solicitudes desde y hacia la red Ethereum. Estos nos permiten interactuar con los Smart Contracts para enviar transacciones o consultar datos como el saldo de la cuenta. A pesar de no ser obligatorio este servicio, lo utilizaremos para evitar la necesidad de ejecutar un nodo propio en la red, aunque veremos reducida nuestra privacidad.

8.4. Diseño Interacción o Experiencia de Usuario - UX

La calidad de la experiencia del usuario es un aspecto muy importante en el éxito de cualquier producto, ya que si tiene un mal diseño puede llevar a los usuarios a buscar otras alternativas en el mercado. La satisfacción de un usuario no se basa únicamente en la estética del producto, sino en cómo satisface sus necesidades o resuelve problemas específicos.

Hay que tener en cuenta que los usuarios no utilizan un producto o servicio por muy bonito que sea, sino porque éste satisface realmente sus necesidades. Por poner un ejemplo, hace unos años las personas solían descargar música, ya fuera comprando descargando de forma pirata a

través de sitios web de intercambio de archivos. Sin embargo, con el lanzamiento de plataformas de música online como Spotify, cambió la forma en que los usuarios consumen la música.

Las plataformas de streaming de música mejoran la experiencia del usuario al proporcionar un acceso instantáneo a una gran variedad de canciones y artistas, sin necesidad de descargar, e incluyendo la posibilidad de crear listas de reproducción personalizadas.

El aumento de los servicios de música en streaming y el descenso de las descargas de canciones, nos muestran un claro ejemplo de cómo la experiencia del usuario mejorada y enfocada en el propio cliente puede marcar la diferencia en el éxito de un producto.

En este apartado, se ha llevado a cabo una investigación para garantizar un diseño de interacción y experiencia de usuario óptimos. De esta forma, se busca aumentar el interés de la mayor cantidad de usuarios posible para que utilicen esta opción de pago en la web, ofreciendo una experiencia agradable y eficiente.

8.4.1. Diseño de Personas

En este apartado vamos a representar personas mediante arquetipos de hipotéticos usuarios que utilicen la pasarela de pagos, así podremos tomar mejores decisiones para el diseño.

En este caso, se han creado 3, cada una de estas personas tiene un perfil distinto al anterior e incluye una fotografía, datos demográficos, motivaciones, restricciones y patrones de comportamiento específicos que nos permiten comprender mejor las necesidades y preferencias de cada persona. Con esta información, podemos ofrecer soluciones personalizadas y adaptadas a las necesidades de cada perfil.

Persona 1

Nombre: Oscar Moreno
Edad: 62 años
Estado: Casado
Ocupación: Gerente de presupuestos
Ubicación: Valencia

Información básica

Oscar Moreno, 60 años, casado y con una hijo. Trabaja en una empresa constructora desde hace 35 años. Vive en el centro de Valencia. Además, tiene un nivel económico medio-alto.

Motivación y objetivos

Hace 2 años que le han ascendido y tiene unos ahorros importantes. Le gustaría poder comprarse el coche de su sueños y el resto seguir invirtiendolo para la jubilación y dejarle una parte como herencia a su hijo.

Restricciones

Es una persona con pocos conocimietos sobre inversiones, a lo largo de su vida ha invertido en fondos que su banco le ha recomendado.

Patrones de comportamiento

Oscar utiliza el móvil lo menos posible, solo para recibir y hacer llamadas. No tiene redes sociales. A pesar de no ser experto, se desenvuelve bien con su ordenador, ya que a menudo utiliza aplicaciones bancarias para hacer pagos.

Lleva 2 años comprando y acumulando Bitcoin de forma semanal por recomendación de su hijo.

Figura 19. Persona 1
(Fuente propia)

Persona 2

Nombre: Pedro García
Edad: 23 años
Estado: Soltero
Ocupación: Estudiante
Ubicación: Alicante

Información básica

Pedro García, 23 años, soltero. Estudiante de ingeniería multimedia. Vive en un piso con sus padres cerca de la playa de Alicante.

Motivación y objetivos

Pedro quiere aumentar sus ahorros y tenerminar el grado universitario. Le gustaría poder comprarse o alquilar un piso en Madrid, su ciudad favorita. Además quiere especializarse en el sector de las criptomonedas y web3.

Restricciones

A Pedro no le gusta invertir en proyectos que no conoce. Los fines de semana ayuda a sus padres con el negocio familiar. Tiene poco tiempo y dinero para investigar y aprender.

Patrones de comportamiento

Con frecuencia, consulta desde su telefono móvil foros y redes sociales como Twitter y YouTube para mantenerse al día sobre las criptomonedas y sus inversiones.

Cuando tiene algo ahorrado, suele comprar productos y cursos online para mejorar su conocimiento sobre el sector.

Utiliza todo tipo de billeteras de criptomonedas para interactuar con los protocolos web3 y así recibir recompensas.

Figura 20. Persona 2

(Fuente propia)

Figura 21. Persona 3
(Fuente propia)

A pesar de las diferencias entre Oscar, Pedro y Sofía, todos ellos tienen ciertos patrones de comportamiento que podrían ayudar al proyecto.

Oscar quiere asegurar su futuro financiero y el de su hijo, mientras que Pedro busca ampliar sus conocimientos en criptomonedas y web3, y Sofía busca diversificar sus inversiones y estar al tanto de las tendencias tecnológicas.

En términos generales, esta pasarela de pagos debe permitir a los usuarios realizar transacciones con criptomonedas de manera segura, facilitar la interacción con diferentes billeteras y protocolos. Además, la pasarela debe ser intuitiva y fácil de usar, especialmente para usuarios como Oscar y Sofía, quienes no tienen mucha experiencia con criptomonedas.

Es importante que la pasarela de pagos se adapte tanto a dispositivos móviles como a ordenadores, ya que esto permitirá a los usuarios como ellos acceder y utilizar la plataforma de manera cómoda y eficiente desde el dispositivo que prefieran.

Este análisis de perfiles también ayuda a identificar requisitos clave para el proyecto. Todas las necesidades mencionadas anteriormente ya se han tenido en cuenta y se han incluido en sus

respectivas secciones del proyecto. Con esta información en mente, se puede desarrollar una solución que mejore su experiencia en el ámbito de las criptomonedas.

8.4.2. User Journey Maps

En este apartado se van a utilizar las personas para ver como reaccionarían a los casos de uso. Esto se conoce como “User Journey Maps”, es decir, representar gráficamente las etapas que experimenta un usuario al interactuar con uno de los servicios y registrar la emoción asociada a cada una de esas etapas [20].

En este caso realizaremos dos User Journey Map, ya que hay criptomonedas que son compatibles con la EVM como hemos visto en apartados anteriores y otras como Bitcoin, no tienen Smart Contracts y el funcionamiento es muy distinto. Además, utilizaremos las aplicaciones que mejor se adaptan al resultado final.

El primero será mediante el pago con una billetera como Metamask, donde las etapas para el proceso de realizar un pago serán las siguientes: darle a pagar, seleccionar la billetera Metamask, seleccionar USDT o ETH en el caso de que no esté disponible, seleccionar la red Ethereum compatible si procede, realizar el pago. Finalmente esperar confirmación para continuar.

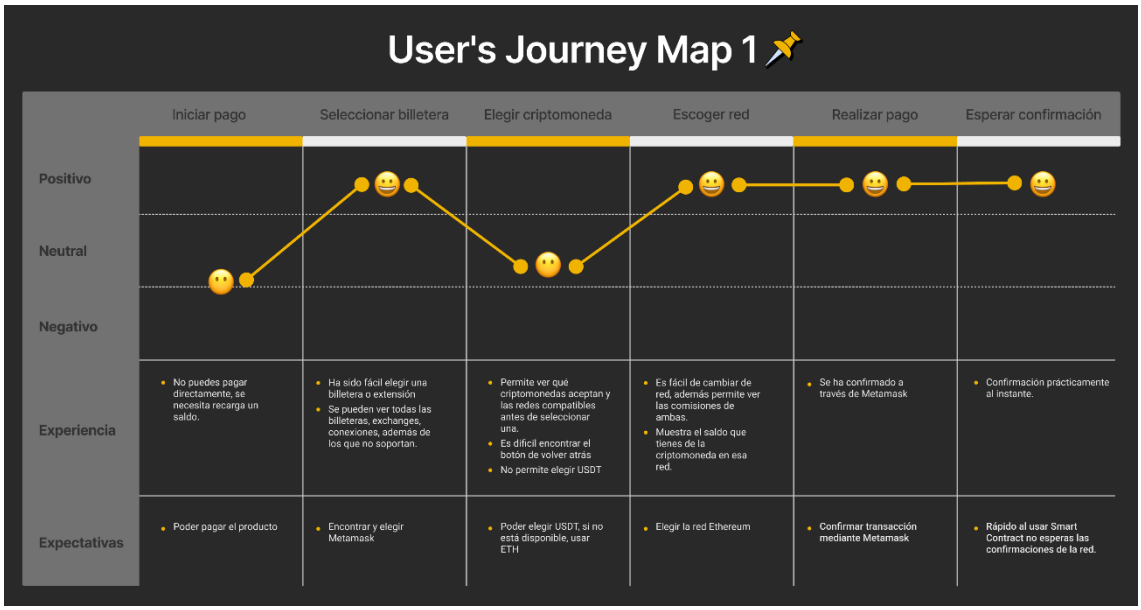


Figura 22. User's Journey Map 1
(Fuente propia)

El segundo consiste en realizar un pago sin interactuar con una billetera como Metamask, es decir, el pago se realizará de forma manual sin un Smart Contract de por medio. Este proceso

consiste en: darle a pagar, seleccionar Bitcoin como método de pago, copiar dirección y monto, realizar pago y finalmente esperar confirmación.

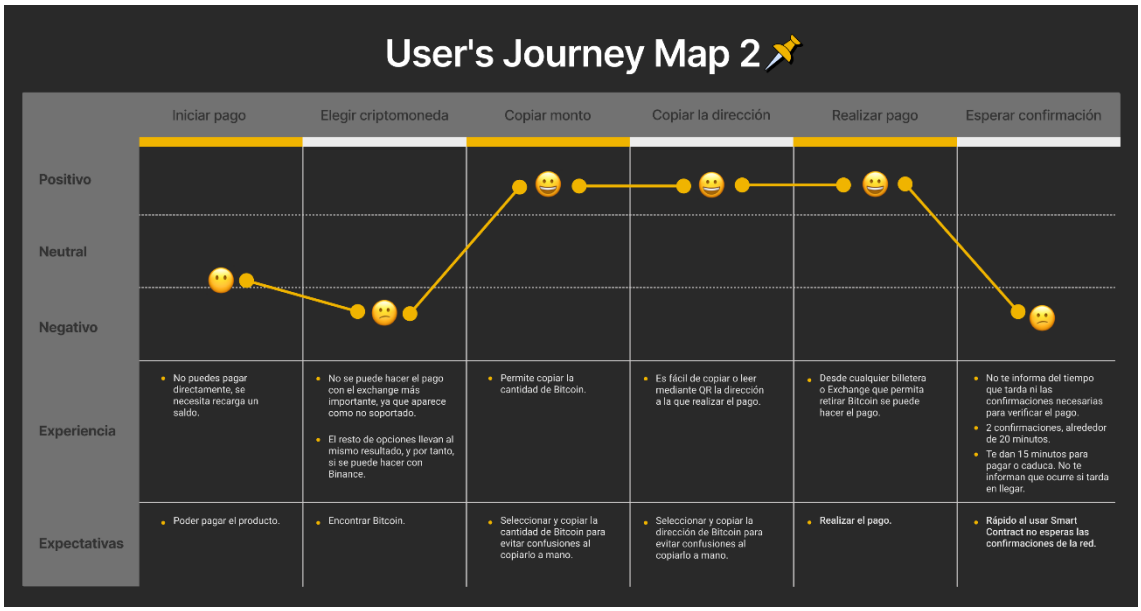


Figura 23. User's Journey Map 2
(Fuente propia)

A partir de estos diagramas, podemos inferir las emociones experimentadas por los usuarios y su experiencia al realizar diversas acciones en las aplicaciones. Estas experiencias se han categorizado en tres tipos: positivo, neutral y negativo.

Las experiencias positivas en el primer mapa se han dado al elegir una billetera específica, ya que la interfaz facilitó su localización y selección. Además, al permitirte elegir la red y realizar el pago mediante un Smart Contract y Metamask, la experiencia de usuario resultó muy satisfactoria, dado que el pago se ejecutó y confirmó casi de inmediato. Sin embargo, también se observó un inconveniente al seleccionar una criptomoneda que no estaba disponible: el botón de volver atrás se encontraba en la esquina superior izquierda, alejado del menú, lo que dificultaba su acceso y utilización en pantallas grandes.

Por otro lado, en el segundo mapa, sucedió todo lo contrario: cuando se intentó efectuar un pago con Bitcoin, se experimentó incertidumbre y frustración en dos etapas. En la primera, debido a la falta de una opción clara para pagar, ya que a pesar de que sí se podía hacer a través de cualquier monedero o Exchange, posteriormente avisaba de que no era posible. Y en la última etapa, al tener que esperar 20 minutos sin recibir ninguna confirmación o aviso. Cabe mencionar que se requieren al menos dos confirmaciones de seis bloques dentro de la cadena blockchain de Bitcoin (aproximadamente 10 minutos por bloque) para confirmar un pago de forma segura.

En resumen, a partir de lo analizado, resulta esencial centrarse en mejorar la experiencia de usuario, procurando diseñar una interfaz intuitiva y limpia que contribuya a reducir al mínimo las experiencias negativas. También es importante que cuando un usuario realice una tarea que implique esperar la confirmación del pago, se le muestre un mensaje del tiempo de espera y las confirmaciones hasta que se realice correctamente, ya que se evitará que el usuario se sienta confundido y garantizaremos una experiencia positiva.

8.5. Guías de estilos

Al tratarse el trabajo de un módulo concreto de la página web, no se diseñará un logo para esta. Sin embargo, para poder tener un desarrollo más rápido, se definirán una paleta de colores para utilizar, las tipografías, otros elementos e incluso plantillas. De esta forma conseguiremos un resultado final más coherente e integrado en la web.

8.5.1. Colores corporativos

En la selección de colores corporativos para este proyecto, se ha optado por una paleta que incluye el negro, el gris oscuro, el blanco y el amarillo anaranjado.

Figura 24. Colores corporativos
(Fuente propia)

El negro y el gris oscuro se emplearán principalmente como base y fondo de la página web, transmitiendo una sensación de poder, sofisticación, prestigio y valor a través del negro, mientras que el gris aportará sencillez y autoridad.

Por otro lado, el blanco será el color elegido para el texto y los títulos, ya que crea contraste con los colores oscuros del fondo, facilitando la legibilidad y proporcionando claridad visual.

Por último, el amarillo anaranjado se utilizará en botones y elementos destacados para llamar la atención del usuario. Este color simboliza energía y dinamismo, logrando un efecto para destacar las partes más importantes de la interfaz y así atraer a los visitantes.

8.5.2. Tipografía

La tipografía utilizada es “Roboto” (ver Figura 25. Tipografía. Fuente Roboto), diseñada por Christian Robertson. Además, se trata de una tipografía Sans Serif, este tipo de tipografía se caracteriza por no tener adornos en los extremos de sus trazos, lo que la hace más limpia y favorece la legibilidad en pantalla [21]. También se asocian con estilos modernos y profesionales. Se puede encontrar de forma gratuita en Google Fonts [22].

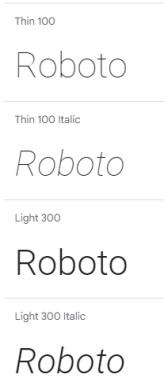


Figura 25. Tipografía. Fuente Roboto (Fuente Google Fonts)

8.6. Diseño Interfaces - UI

En el apartado de diseño de interfaces, se aborda la materialización de los Requerimientos Funcionales (RF) y Requerimientos No Funcionales (RNF) a través del desarrollo de elementos físicos que permiten la interacción de los usuarios con el sistema.

Esta fase es muy importante y, por tanto, requerirá de una inversión considerable de tiempo. A mayor número de interfaces diseñadas con precisión, mayor será el tiempo ahorrado en la siguiente fase de implementación. Ya que evita tener que tomar decisiones de diseño en ese momento y nos permite centrarnos plenamente en el desarrollo.

El diseño de interfaces presenta varios niveles de detalle y exactitud. En primer lugar, tenemos los wireframes, que a pesar de falta de precisión respecto al diseño que habrá al final, cumplen con la función de mostrar de manera sencilla cómo y dónde irán los elementos. Estos bocetos nos ayudarán a refinar más adelante.

En el siguiente nivel de exactitud encontramos los mockups, que son representaciones de interfaces que reflejan con exactitud la ubicación y apariencia final de los elementos (incluyendo colores, formas e imágenes).

Por último, encontramos los prototipos, que, además de replicar la apariencia de mockups, simulan también su interactividad. Este diseño es prácticamente una réplica de lo que será el diseño final [23].

En este caso se ha decido crear wireframes y mockups. Los wireframes han sido diseñados utilizando Balsamiq [24], esto nos ha permitido visualizar rápidamente los elementos que se necesitan para las funcionalidades del sistema de una forma muy poco refinada como se pueden ver en la Figura 26. Se han creado un total de 14 interfaces, donde se incluyen los requisitos funcionales como realizar un pago, seleccionar una criptomoneda, cambiar entre diferentes redes, descargar la factura, entre otros.

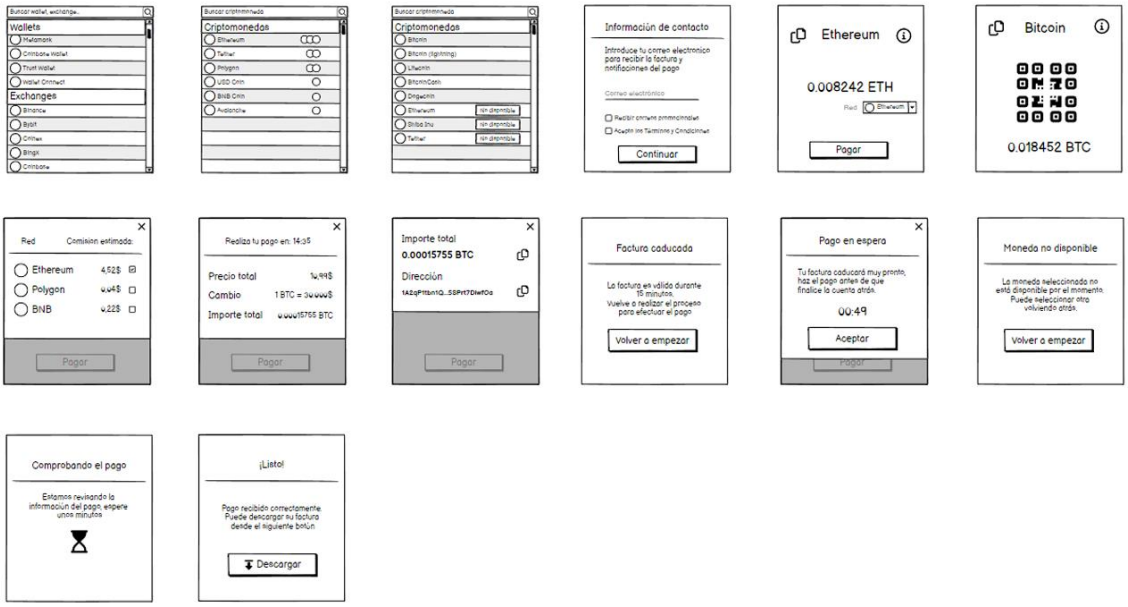


Figura 26. Diseños de wireframes
(Fuente propia)

Posteriormente se ha utilizado Figma [25] para crear mockups que representan con precisión cada uno de los elementos dentro de la interfaz de la pasarela de pagos. No se ha desarrollado un prototipo ya que su creación requeriría demasiado tiempo.

A continuación, en la Figura 27 se muestra una imagen panorámica con todas las interfaces diseñadas. Son un total de 27, y, aunque algunas interfaces puedan tener un ligero cambio durante la etapa de implementación (o incluso la necesidad de diseñar alguna más), se han creado la mayoría y las más importantes para el desarrollo del trabajo.

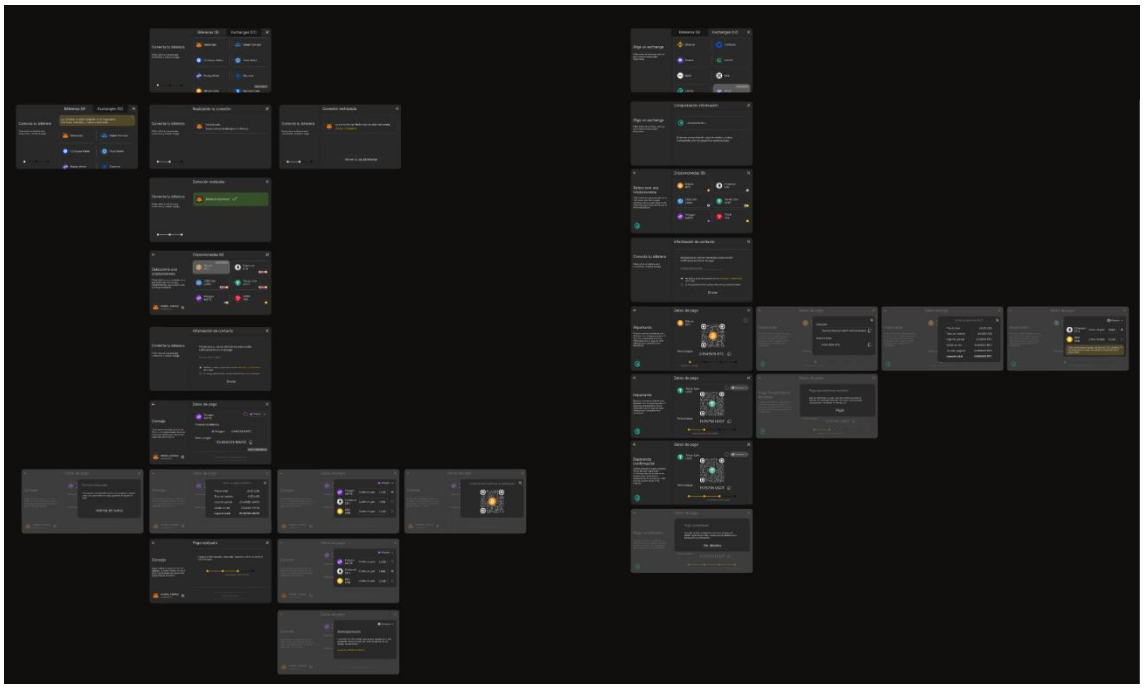


Figura 27. Vista panorámica de las interfaces
(Fuente propia)

A continuación, se abordarán los puntos clave de cada una de las interfaces, empezaremos con el proceso de pago mediante una billetera y posteriormente el pago directo con una billetera custodiada por una empresa (Exchange centralizado). Además, en los títulos de cada figura, se identificará la interfaz con un número en caso de que haga falta, lo que permitirá hacer referencia a ellas.

8.6.1. Interfaces para Billeteras

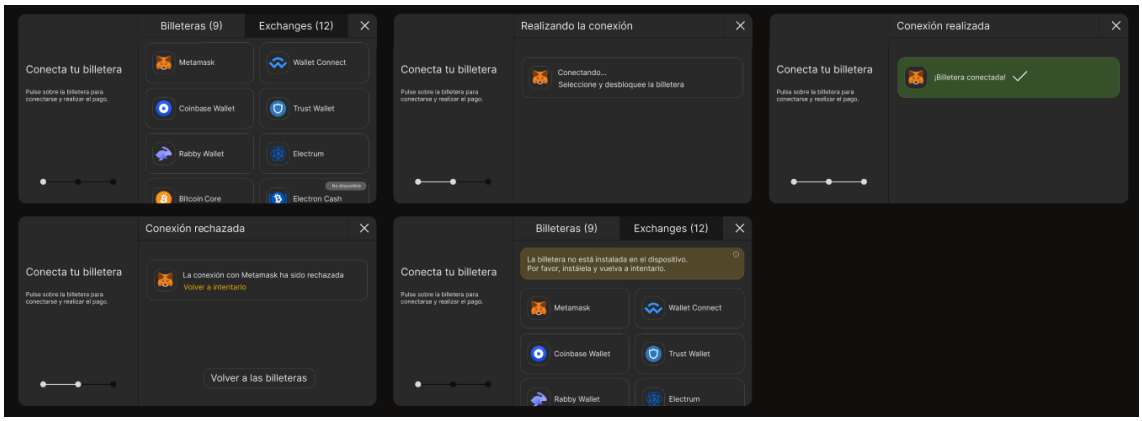


Figura 28. Interfaces elección de billeteras (1-5)
(Fuente propia)

En primer lugar, como ya hemos comentado, veremos primero las interfaces relacionas con el pago a través de una billetera como Metamask, donde la interfaz 1 de la Figura 28 corresponde

a las opciones de las billeteras que se le dan al usuario al elegir el método de pago con criptomonedas.

Cuando el usuario pulse sobre una opción (en este caso Metamask) pasará a una interfaz (la número 2) en la que se quedará la aplicación a la espera de que se desbloquee la billetera mediante la clave del propio usuario y posteriormente acepte la conexión (número 3).

Si el usuario rechaza la conexión por cualquier motivo, aparecerá un mensaje para avisarle de volver a intentarlo o bien volver a elegir otra opción (interfaz 4). En caso de que no tenga esa billetera instalada en el navegador, se le informará de que realice primero la instalación y luego vuelva a intentarlo (número 5).

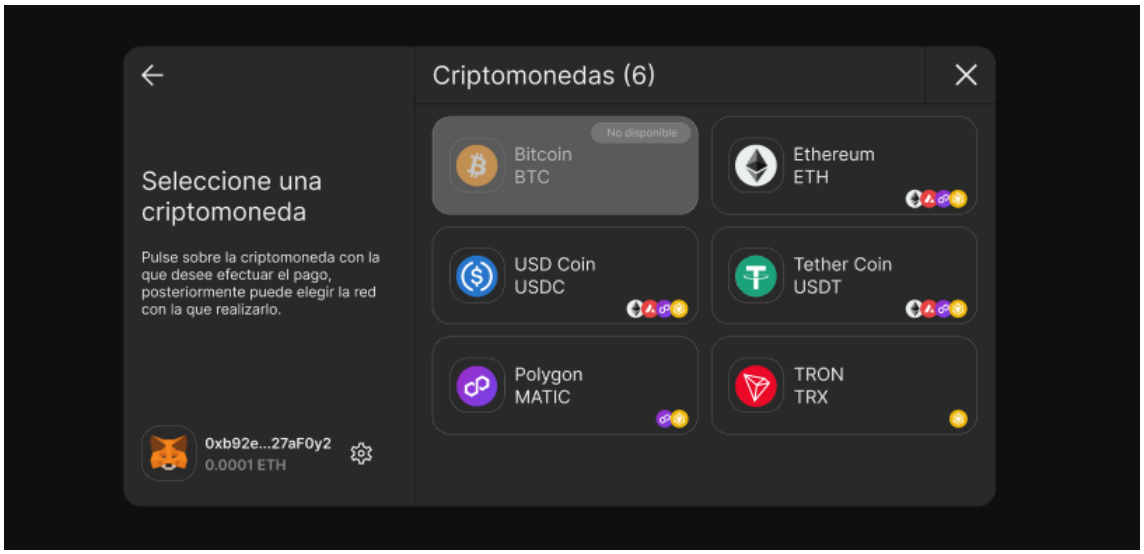


Figura 29. Interfaz 6, selector de criptomonedas
(Fuente propia)

En la Figura 29 se pueden ver las criptomonedas disponibles que son compatibles con esa billetera, además de las distintas redes. Se muestra también en la esquina inferior derecha la dirección de la billetera utilizada y la cantidad de la criptomoneda que se usa pagar las tarifas de red (en este caso Ethereum).

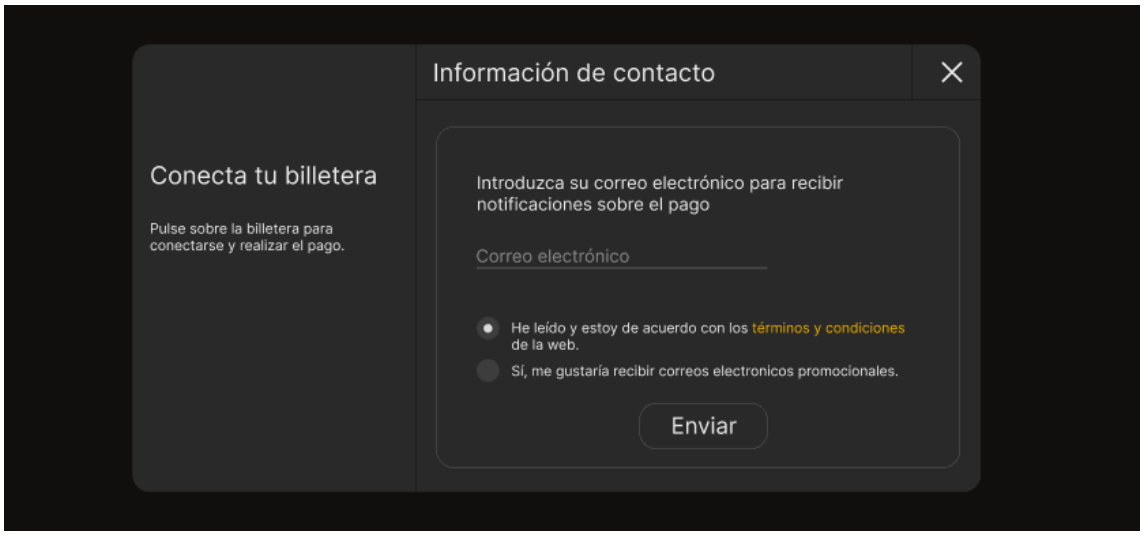


Figura 30. Interfaz información de contacto
(Fuente propia)

Seguidamente, antes de crear la factura y una vez seleccionada una criptomoneda, se le solicitará al usuario un correo electrónico para identificarlo y así poderle mandar notificaciones del pago o incluso promociones en el caso de que haya aceptado, ver Figura 30.

En la Figura 31, se muestran los datos de pago, es una de las interfaces más complejas, ya que aparece mucha información (parte de ella repartida en otras interfaces). En esta interfaz se puede ver la criptomoneda elegida, la red por defecto, la cantidad de la criptomoneda que tiene el usuario en su billetera y justo debajo, la cantidad que tiene que pagar.

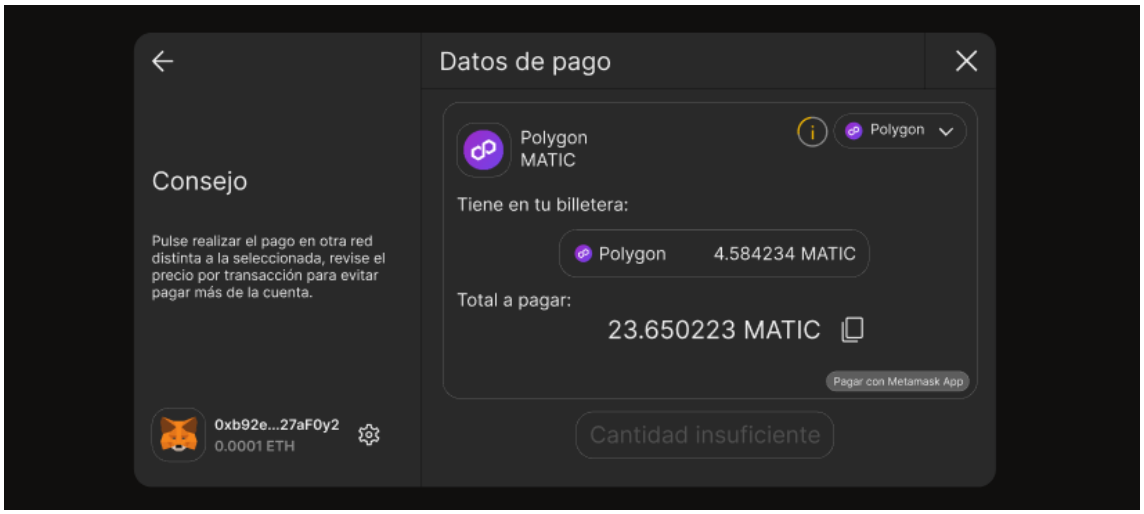


Figura 31. Interfaz de datos de pago
(Fuente propia)

Además, también se ha añadido una “i” de información que muestra gráficamente el tiempo que queda para que la factura caduque. En caso de que el usuario pulse sobre ella, accederá a otra interfaz como la que se puede ver más abajo.

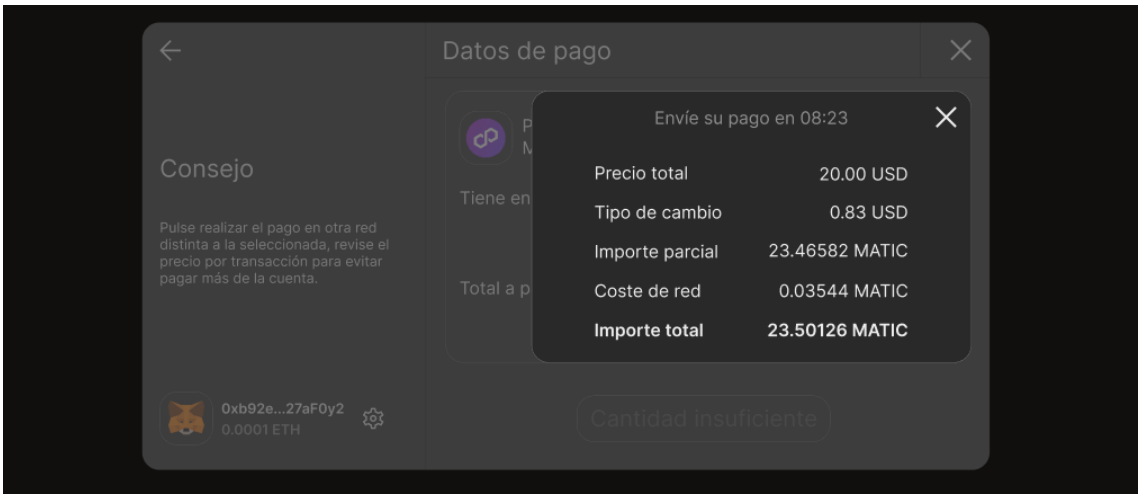


Figura 32. Interfaz información pago
(Fuente propia)

En ella se puede observar la cuenta atrás y el desglose del pago que se va a realizar. Se calcula en el momento que se pulsa sobre una criptomoneda en los pasos anteriores y se actualiza si cambia de red.

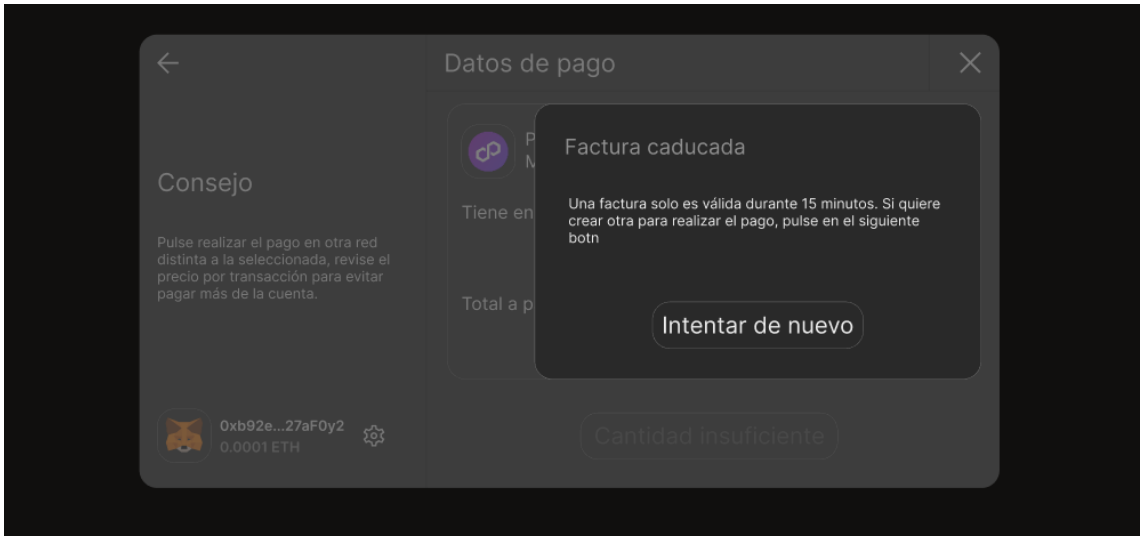


Figura 33. Interfaz factura caducada
(Fuente propia)

Si el usuario tarda más del tiempo estipulado, la factura caducará y se le mandará volver a iniciar una nueva como se puede ver en la Figura 33.

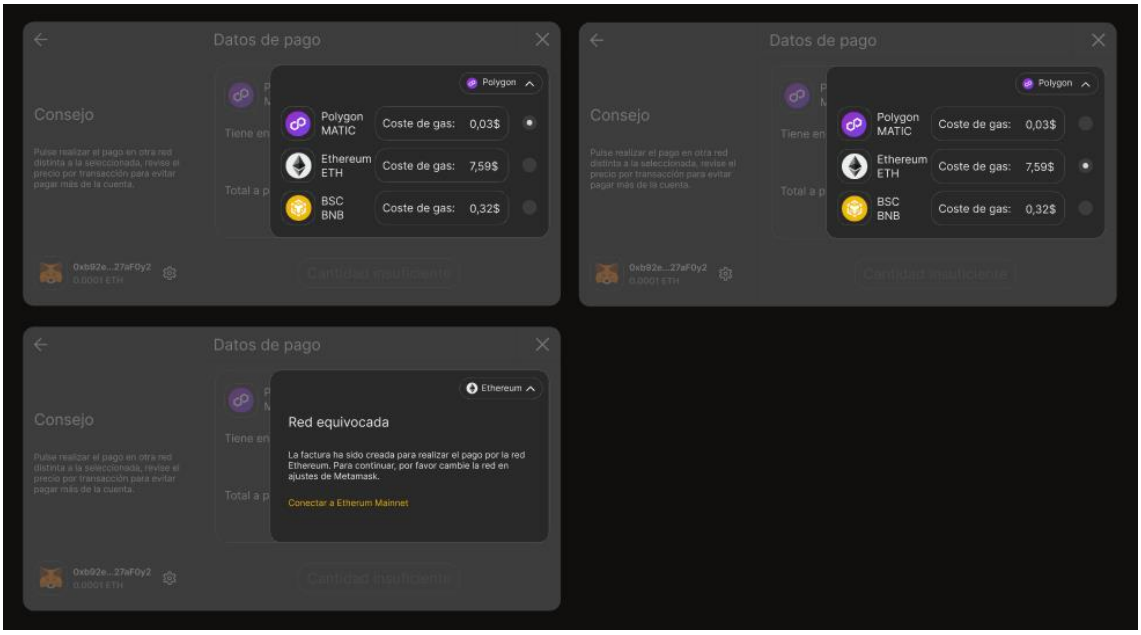


Figura 34. Interfaz cambio de redes (1-3 de izquierda a derecha, de arriba a abajo)
(Fuente propia)

En caso de que el usuario decida cambiar de red y sea posible, aparecerá la interfaz 3 de la Figura 34 para poder hacerlo. En ella, se mostrarán las redes junto con el coste de gas por transacción.

Si decide cambiarla por otra (por ejemplo, más económica) aparecerá un mensaje de "Red equivocada" y se le solicitará a la billetera utilizada por el usuario que le cambie a la red seleccionada. No hay que olvidar que aquí se volverá a recalcular el precio final de la factura (incluyendo así los costes de red para mover el capital a una billetera principal).

En la Figura 31 que hemos visto anteriormente, tambien había en la parte inferior derecha una opción para abrir la billetera en la aplicación de Metamask para dispositivos móviles. Esta interfaz consta de un código QR con los datos correspondientes para ser escaneada por la aplicación y asi poder realizar estos pasos desde ella, ver Figura 35.

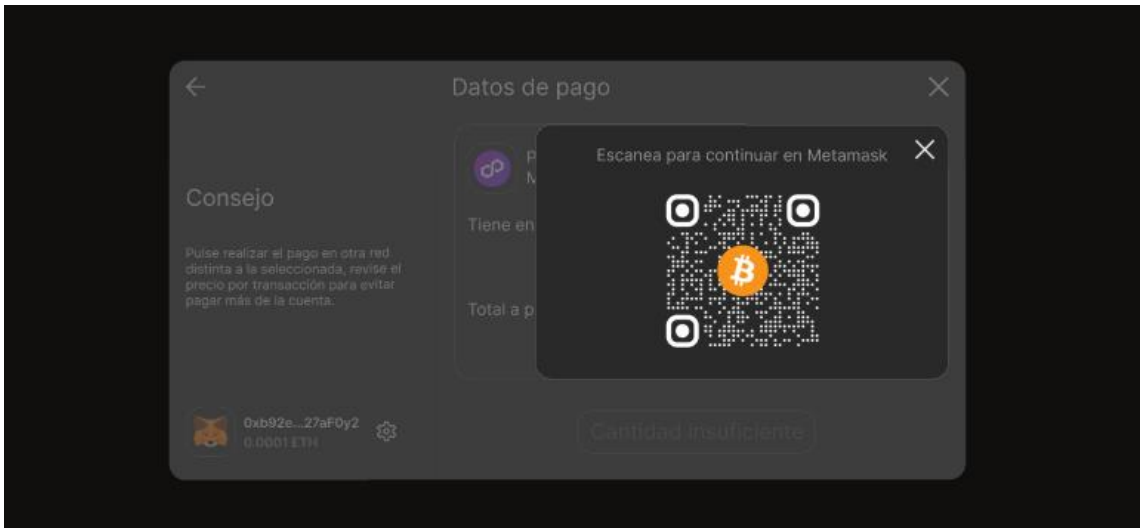


Figura 35. Interfaz Escanear QR Metamask
(Fuente propia)

Cuando el usuario tenga la cantidad suficiente de la criptomoneda seleccionada, se habilitará el botón que mandará la señal a su billetera para que esta le muestre el mensaje que deberá aceptar para efectuar el pago. Una vez realizado, al usuario le aparecerá la siguiente interfaz de la Figura 36.

Figura 36. Interfaz pago realizado
(Fuente propia)

En ella se le mostrará un pequeño esquema de los pasos que tendrá que esperar (Esperando confirmación en la red, Transacción detectada, Confirmaciones y Pago completado) para proceder con la factura y descarga de ella.

8.6.2. Interfaces para Exchanges

El diseño de interfaces para los Exchanges será muy similar al que hemos visto de las billeteras, sin embargo, en este caso los pagos no se realizarán mediante un *Smart Contract*, por lo que tendrá ligeros cambios.

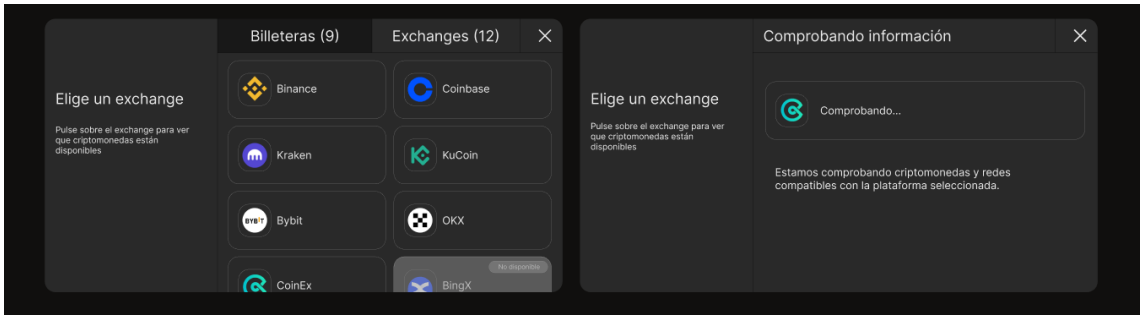


Figura 37. Interfaces elección de Exchanges
(Fuente propia)

Como se puede ver en la interfaz de la izquierda de la Figura 37, aparecerán los exchanges que tienen criptomonedas y redes compatibles con la pasarela de pagos. Al pulsar sobre uno, se comprobarán las criptomonedas y sus redes.

Posteriormente, se mostrarán, y una vez se haya elegido una criptomoneda, aparecerá la interfaz de información de contacto para generar la factura y con ello la cantidad a pagar.

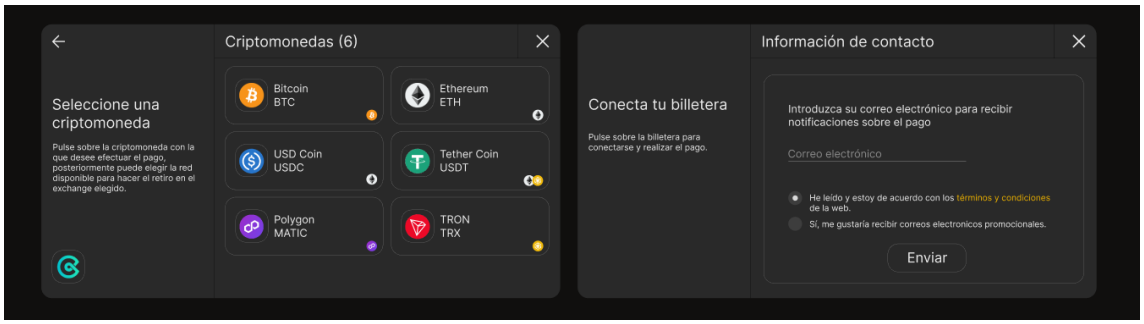


Figura 38. Interfaces selección de criptomonedas e información pago
(Fuente propia)

En la siguiente imagen se pueden observar varios cambios con respecto a los anteriores diseños del aparatado Interfaces para billeteras.

La primera es que se mostrará directamente el código QR para escanearlo desde la propia app del Exchange si estamos desde un dispositivo móvil, y en el caso de que el pago se haga desde el propio navegador del ordenador, podremos copiar la dirección pulsado sobre la imagen.

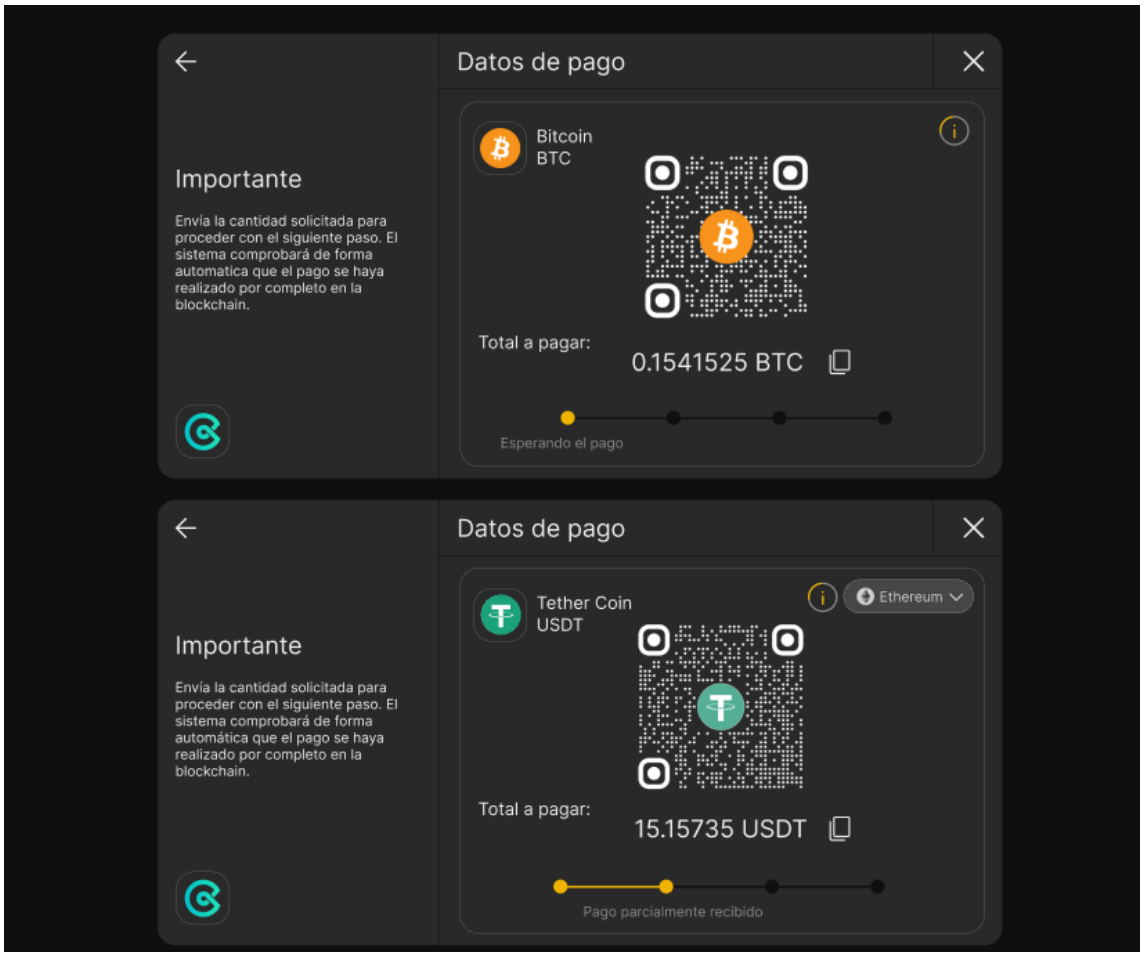
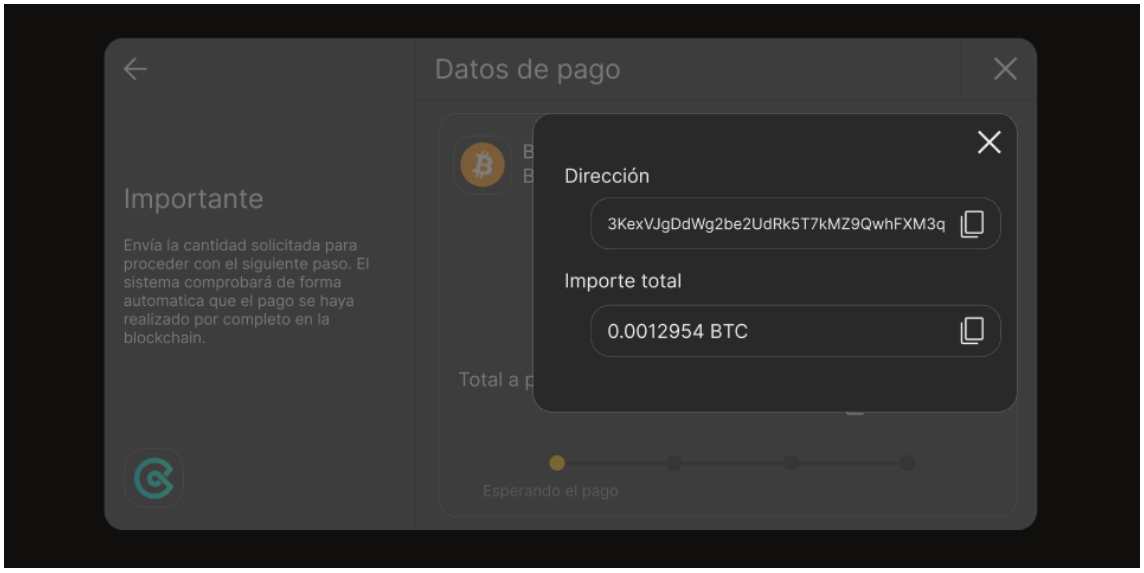


Figura 39. Interfaces datos de pago (sin y con opción de redes)
(Fuente propia)

En segundo lugar, se mostrará el estado del pago con los siguientes títulos: Esperando el pago, Pago parcialmente recibido, Esperando confirmación en la red y Pago completado.

Figura 40. Interfaz pago parcialmente recibido
(Fuente propia)

En este proceso a diferencia de las billeteras, puede que el pago no se realice con la cantidad acordada y, por tanto, el pago recibido sea menor a la cantidad indicada. En este caso se mostrará un aviso como se puede ver en la Figura 40. Solo en este caso, la cantidad a pagar se reducirá en función de lo detectado en la cadena y se bloqueará el cambio de red (si estaba disponible) para evitar confusiones.



Si pulsa sobre el icono de *copiar* a la derecha de *total a pagar*, se mostrará la dirección y el importe total para poder copiar y pegar en la sección de retiro del exchange elegido como se puede ver en la Figura 41.

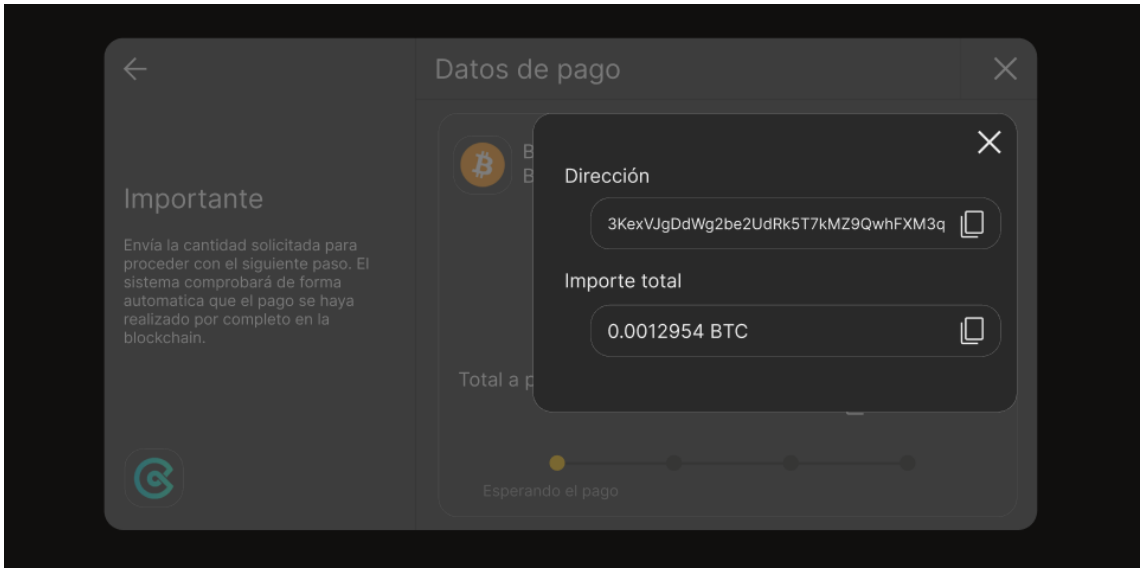


Figura 41. Interfaz dirección e importe
(Fuente propia)

Como también se ha mostrado anteriormente en el apartado de billeteras, si el usuario pulsa sobre la "i" de información, se mostrará el tiempo faltante para que caduque la factura y el

desglose. Sin embargo, se añadirá también la cantidad que haya sido pagada (si se da el caso que lo hace en varias transacciones), si no, aparecerá como “0.000000”.

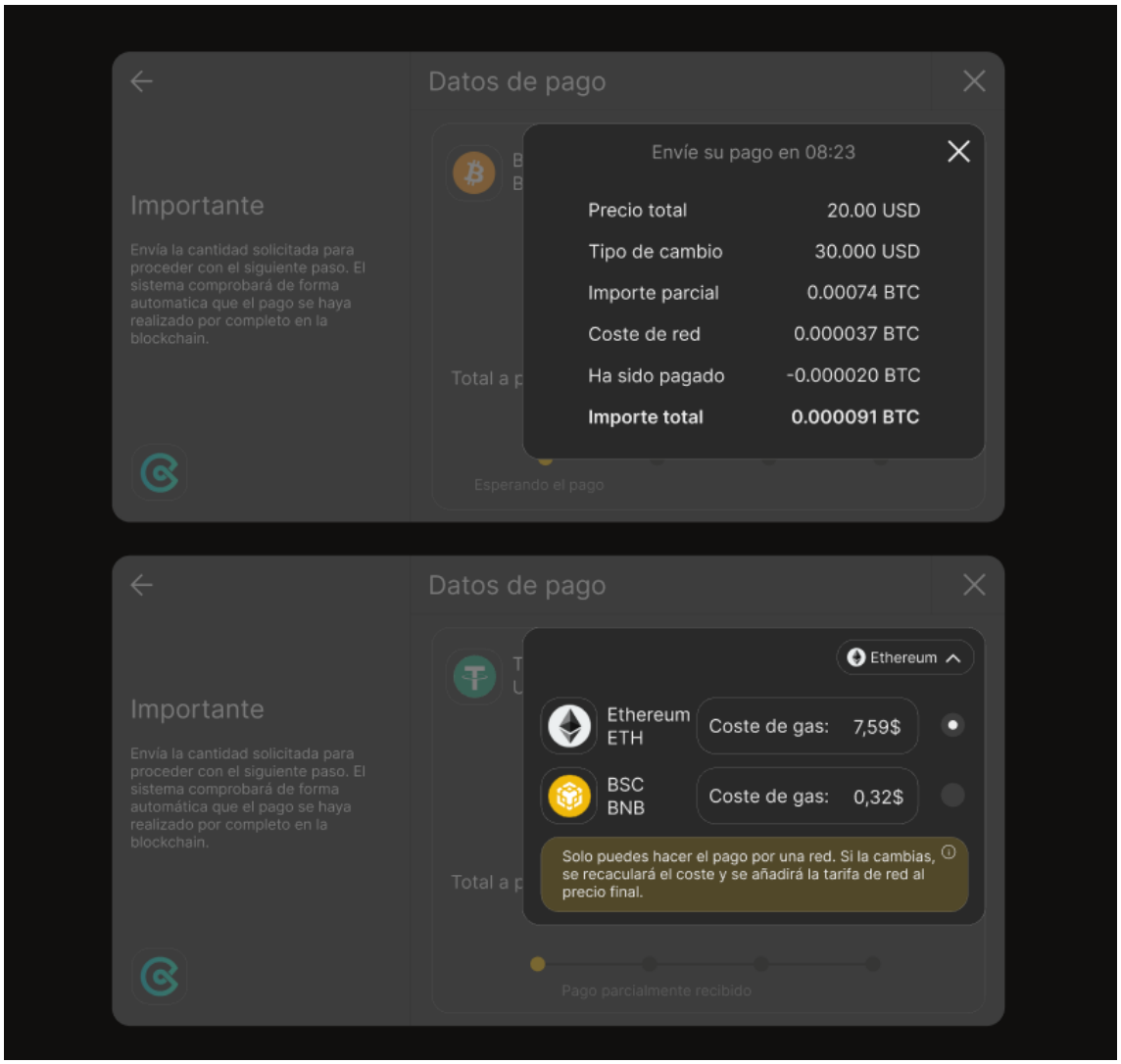


Figura 42. Interfaces de tiempo restante y cambio de redes
(Fuente propia)

Si se hace un cambio de red, se avisará al usuario de que se recalculará el precio y se añadirá el coste de transacción.

Una vez realizado el pago (con la cantidad total), el sistema irá actualizando el estado del pago, pasando por “Esperando confirmación” para confirmar que la transacción se ha confirmado en la cantidad mínima (normalmente dos) de bloques de la cadena para que la transacción sea segura e irreversible.

Por último, una vez pasado este estado, como se puede ver en la Figura 43 se mostrará un mensaje de pago completado y un botón para ver los detalles (factura y descarga de esta).

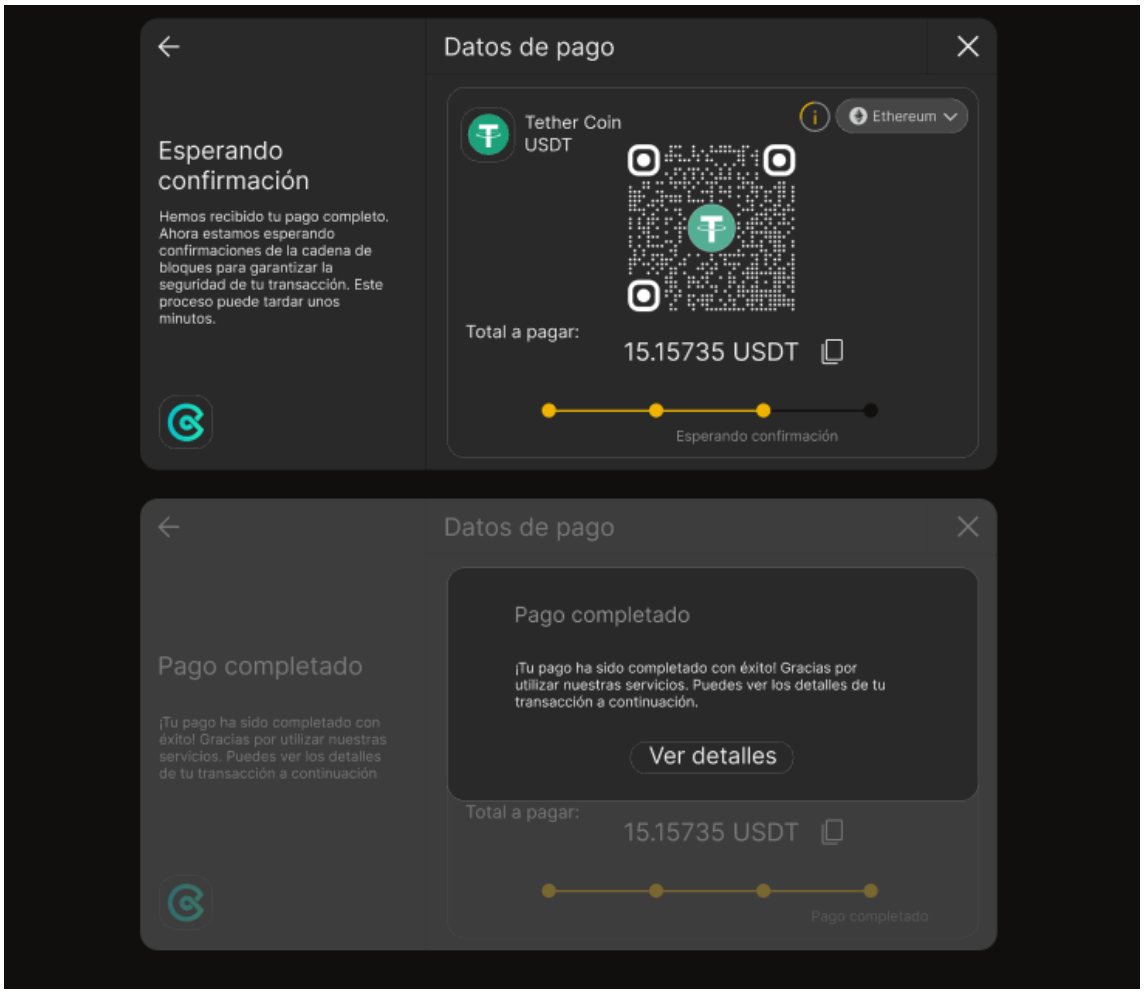


Figura 43. Interfaces datos de pago y completado
(Fuente propia)

8.7. Diseño de pruebas y validación

En este apartado, en el contexto de un Trabajo de Fin de Grado, el foco principal es comprobar si los objetivos planteados inicialmente se han alcanzado. El diseño de pruebas es vital para verificar el funcionamiento total del proyecto y así determinar las áreas que necesitan alguna corrección o mejora.

El primer método que se utilizará y el más básico, es verificar si se han desarrollado y cumplido todos los requisitos especificados. Aunque puede parecer innecesario, es posible que algún requisito no se cumpla por completo o se olvide, por lo que esta comprobación es fundamental.

Por otro lado, se implementará un testing propio para buscar posibles fallos ya que es esencial para asegurar la calidad y seguridad de la pasarela de pagos, esto se realizará con redes de pruebas como la red Ethereum Goerli, que nos permite probar y experimentar con contratos inteligentes y otras funcionalidades sin utilizar la red principal, donde las transacciones tienen un coste real en criptomonedas [26].

Esta red utiliza un token llamado GÖETH que no tiene ningún valor real, esto nos permitirá experimentar con un entorno muy parecido a la red principal sin coste alguno, este token se conseguirá a través de Faucets [27]. Del mismo modo utilizaremos estas redes de pruebas en otras blockchains.

Para complementar el feedback que obtendremos de estas pruebas, realizaremos encuestas dirigidas a una comunidad de criptomonedas en Telegram y Twitter. Esto nos permitirá recopilar un gran volumen de datos y opiniones sobre la funcionalidad de la aplicación desde varios dispositivos y países.

En caso de necesitarse, también se recurrirá a personas específicas de nuestro entorno como amigos y familiares, para así obtener información de un rango más amplio de usuarios potenciales. Buscaremos una variedad diversa de perfiles, incluyendo a jóvenes, adultos mayores y personas con diferentes niveles de experiencia con criptomonedas.

Para recopilar la información de manera más detallada y efectiva, utilizaremos Google Forms [28]. Ya que nos permite generar encuestas online y redactar un formulario con todo tipo de preguntas. El objetivo es que el formulario no sea demasiado largo y pueda ser completado de manera sencilla y rápida por los usuarios.

Este feedback nos permitirá pulir la aplicación basándonos en las opiniones de aquellos que no están directamente involucrados en el proyecto. Este método no solo nos brinda diferentes perspectivas, sino que también nos ayuda a mejorar de manera significativa la versión final de la aplicación.

A continuación, se presenta las preguntas que aparecerán en el formulario:

Número	Pregunta	Respuestas
1	Nombre (opcional)	(Texto corto)
2	Edad	(Texto corto)
3	Experiencia con criptomonedas	• Novato (3 meses) • Intermedio (4 meses - 1 año) • Avanzado (1-2 años) • Experto (+2 años)
4	¿Con qué frecuencia usas criptomonedas para realizar pagos?	• Diariamente • Semanalmente • Mensualmente • Raramente
5	¿Qué billetera de criptomonedas usas más a menudo?	(Opción múltiple)
6	¿Qué criptomoneda prefieres usar para realizar pagos?	(Opción múltiple)
7	¿Qué exchange utilizas más a menudo?	(Opción múltiple)
8	Cuando haces pagos con criptomonedas, ¿qué usas más, billeteras o pagos desde exchange?	Opción 1 o 2
9	Solo en el caso de que hayas utilizado una billetera, puntúa del 1-10 lo fácil que te ha resultado los pasos a seguir.	(Rango del 1 al 10)
10	Solo en el caso de que hayas utilizado un exchange directamente para hacer el pago, puntúa del 1-10 lo fácil que te ha resultado los pasos a seguir.	(Rango del 1 al 10)
11	¿Ha habido algún paso en el que has tenido problemas? Si la respuesta es sí, detalla el problema que has encontrado	(Opción múltiple y texto largo)

Tabla 13. Preguntas formulario feedback

Se trata de un formulario corto y muy fácil de completar, el cual consta de todo tipo de preguntas sobre la pasarela de pagos con criptomonedas. A través de este formulario podemos recoger la perspectiva de los usuarios sobre aspectos específicos de esta. Esto nos ayudará para focalizar el esfuerzo en optimizar y perfeccionar el módulo, con el objetivo de alcanzar un producto final con mayor calidad.

9. Implementación

Finalizadas las secciones de análisis y diseño de nuestro proyecto, es momento de empezar la etapa de implementación. Esta fase englobará todas las acciones y tareas necesarias para la creación de la pasarela de pagos además de la documentación de los obstáculos encontrados en el proceso y las soluciones que se han aplicado para poder superarlos.

Siguiendo la metodología elegida al principio, implementaremos una adaptación del Proceso Unificado de Desarrollo y la metodología *Scrum*. Por tanto, se presentarán progresos a través de *sprints* donde cada uno de ellos tendrá pequeñas iteraciones (desarrollo, implementación y pruebas), permitiéndonos así evaluar gradualmente los logros alcanzados y definir metas claras para cada etapa.

Realmente, la Fase de inicio (análisis de riegos, funcionalidad y objetivos) se ha completado en los aparatados anteriores, por lo que estos *sprints* estarán enfocados en las fases de elaboración y construcción. Una vez finalizados, quedaría la fase de transición donde se lanzaría el producto al entorno de producción y se recopilarán los comentarios de los usuarios. Se estima que cada uno de los *sprint* serán unas 25 horas aproximadamente.

9.1. Sprint 1: Preparación del entorno de desarrollo

Durante el primer sprint del proyecto, no se implementó ninguna funcionalidad especifica, si no que el objetivo era establecer un entorno de desarrollo y preparar todo lo necesario para que el proceso de desarrollo sea lo más eficiente posible. Este periodo de preparación es muy importante para que todas las herramientas y programas funcionen correctamente antes de empezar con la implementación real.

En primer lugar, para el desarrollo del código se ha utilizado *Visual Studio Code* [29], un editor de código gratuito y de código abierto desarrollado por *Microsoft*. Este editor permite una amplia cantidad de extensiones para soportar una variedad de lenguajes de programación.

En cuanto al control de versiones, se ha elegido *GitHub* [30], que permite el seguimiento de cambios en el código y además permite gestionar eficientemente el código junto con mejores prácticas de desarrollo. Esto es necesario ya que el proyecto será llevado a cabo con dos ordenadores y esto junto con *Node* permitirá actualizar e instalar las librerías necesarias en cualquier momento sin tener problemas por compatibilidad de versiones.

Centrándonos más en el sprint, la mayor parte del tiempo se ha dedicado a revisar los fundamentos de las librerías y tecnologías que se iban a utilizar. Haciendo pequeñas pruebas para entender cómo funcionan y así posteriormente implementarlo a gran escala. Realmente ha sido muy necesario para garantizar el hecho de estar preparado para el desarrollo del proyecto.

Después de revisar varios tutoriales en YouTube, guías en blogs y documentación de varias webs se empezó a crear el proyecto base. Para ello se instaló *React* y *Next.js*, donde *React* es una biblioteca de *JavaScript* para construir interfaces de usuarios y *Next.js* es un marco de trabajo que permite la generación estática de sitios y renderización del lado del servidor.

Además, también se han analizado varios proyectos web3 como *Sushi* [31] y *PancakeSwap* [32] para obtener una visión más amplia de las tecnologías que se están utilizando en el ecosistema de las criptomonedas. Es aquí donde se descubrió *Wagmi* [33], una colección de *Hooks*[16] de *React* que permite la conexión de la billetera, consultar información de esta e interactuar con contratos inteligentes de Ethereum.

Es probable que se utilice la librería *Viem* [34], una interfaz de *TypeScript* para Ethereum que proporciona primitivas sin estado de bajo nivel para interactuar con Ethereum. A pesar de ser desarrollado por los creados de *Wagmi*, tiene una ventaja en los bloques de construcción modulares ya que hacen que sean mucho más flexibles.

Además, también se plantea utilizar *ethers*, una biblioteca muy conocida en el sector que proporciona una serie de utilidades para trabajar con Ethereum.

Recapitulando, en este primer sprint se ha llevado a cabo pruebas preliminares además de analizar en profundidad las tecnologías que se van a utilizar para preparar el entorno de implementación del proyecto. Esto permite establecer una base sólida en la cual desarrollar el proyecto.

9.2. Sprint 2: Diseño del sistema e implementación del Smart Contract

Para este segundo sprint, se ha tenido como objetivo principal el diseño del sistema y la implementación del Smart Contract en varias redes.

[16] Los Hooks son una característica introducida en React 16.8 que permiten usar el estado y otras características de React sin tener que escribir una clase.

En primer lugar, se comenzó con la creación del diseño de sistema que incluye la arquitectura del sistema, la definición de los componentes clave y como interactuarían entre sí. El objetivo es modularizar al máximo posible todos aquellos bloques y componentes que pueda reutilizarse en otras pantallas y así evitar duplicación de código.

Por tanto, el proyecto incluirá las siguientes carpetas:

- **components**: esta carpeta contiene todos los componentes de *React* que se utilizan en la aplicación, es decir, son piezas independientes y reutilizables de código que controlan una parte de la interfaz de usuario. Por ejemplo, contendrá formularios o mensajes de ayuda.
- **pages**: esta carpeta contiene los archivos *TypeScript* que a su vez se convierten automáticamente en una ruta en la aplicación web.
- modules: en esta carpeta estarán los archivos de código que exportan funciones que se utilizarán en varios componentes o páginas.
- **services**: en esta carpeta encontraremos módulos que realizan llamadas a la API Rest y módulos que interaccionan con la base de datos.
- **scripts**: esta carpeta contiene los scripts para ejecutar los Smart Contracts

Paralelamente, se comenzó a trabajar en la implementación del Smart Contract. Utilizando *Solidity* [35] que es el lenguaje de programación para la creación de contratos inteligentes en Ethereum. En este caso, se ha creado un contrato básico en el que se podía recibir criptomonedas y almacenarlas hasta que el dueño de este contrato (la persona que hace el *deploid* en la blockchain) ejecute una función para retirar el saldo acumulado de los pagos que se hayan realizado.

Para las pruebas, se utilizó la librería *Hardhat* [36]. *HardHat* es un entorno de desarrollo de Ethereum que facilita tareas como la compilación de contratos inteligentes, la ejecución de pruebas y la implementación de contratos. También proporciona una red Ethereum local con varias direcciones y saldos de Ethereum ficticios que son muy útiles para realizar pruebas de las funciones de los contratos.

Siguiendo los pasos de la documentación, se lanzó el contrato en la red de pruebas *Goerli* de Ethereum y de esta manera se obtuvo la dirección del contrato para poder interactuar con él. Sin embargo, para poder realizar todo esto, se necesitó una cantidad mínima de ETH Goerli. Para ello se utilizó la *faucet* gratuita de *Alchemy* [37] que permite recibir 0.02 ETH cada día.

Por otro lado, tambien se creó una cuenta en la plataforma *Alchemy* para obtener acceso a su API de proveedor. Esta API proporciona una conexión entre la cadena de bloques Goerli y el servidor local, de esta manera se accede a ella mediante una clave y un enlace de la RPC.

Una vez que el contrato fue lanzado, se hizo público y, por tanto, era ya visible en Etherscan (de la red Goerli) para comprobarlo en la cadena de bloques. Incluso se puede ver la dirección de la billetera a la que pertenece.

Se ha tenido que generar con la librería *dotenv* un archivo (.env) para almacenar el enlace de la RPC y su clave API, junto con la clave de Metamask, algo que es muy cómodo para poder compilar y lanzar los Smart Contracts en la cadena de bloques pero que tiene un inconveniente y es que, si se hace público por error, con esa clave de Metamask podría no solo tener acceso a la billetera, si no también, al contrato y los fondos que contiene. De esta forma se podría firmar cualquier contrato o enviar las criptomonedas, perdiendo así todo el dinero.

Es por ello que, en caso de que no haya problemas, para próximas pruebas y lanzamientos de contratos en las cadenas de bloques, se utilizará la herramienta *Remix Proyect* [38]. *Remix* es un entorno de desarrollo de Ethereum basado en la web que facilita la escritura, la prueba y la implementación de contratos inteligentes en *Solidity*. Además, nos permite de forma muy sencilla sin necesidad de tener acceso a la clave de Metamask ni tampoco a la API de *Alchemy*, hacer el *deploid* en la red que nos interese eligiéndola desde Metamask como se puede ver en la Figura 44.

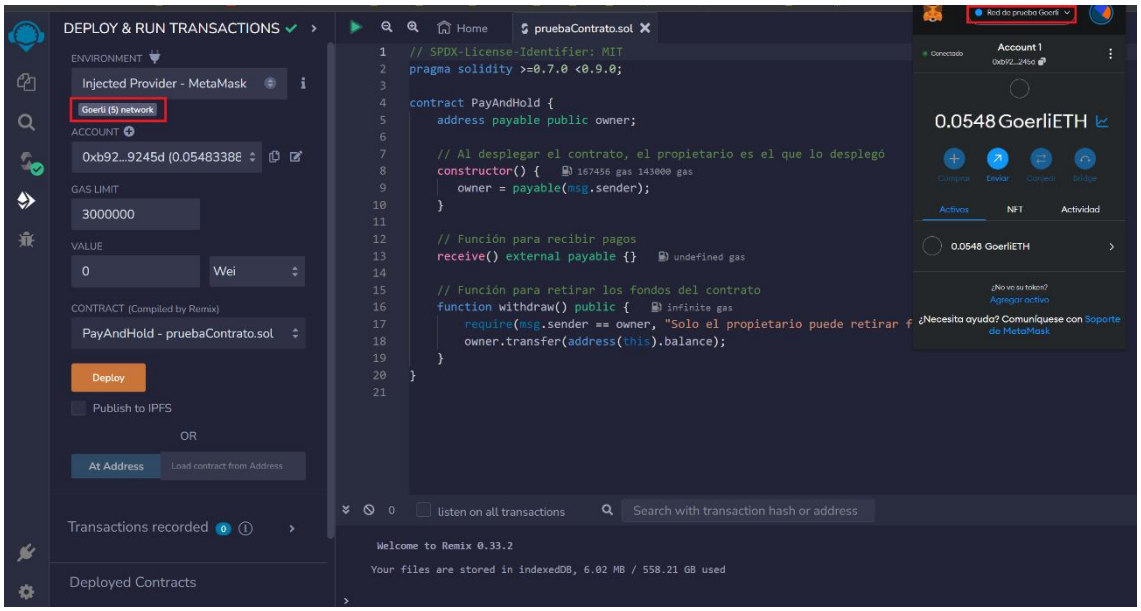


Figura 44. Red elegida para el deploy
(Fuente propia)

Otra ventaja que nos permite *Remix Proyect* es elegir fácilmente la versión del compilador de Solidty, hacer pruebas unitarias, compilarlo y ejecutar el *script* antes de hacer el *deploid* en la red. Esta ventaja ha sido muy necesaria, ya que el mismo Smart Contract puede ser lanzado en otras redes con apenas un par de cambios (elegir otra red y tener suficientes fondos con la criptomoneda principal para poder lanzarlo).

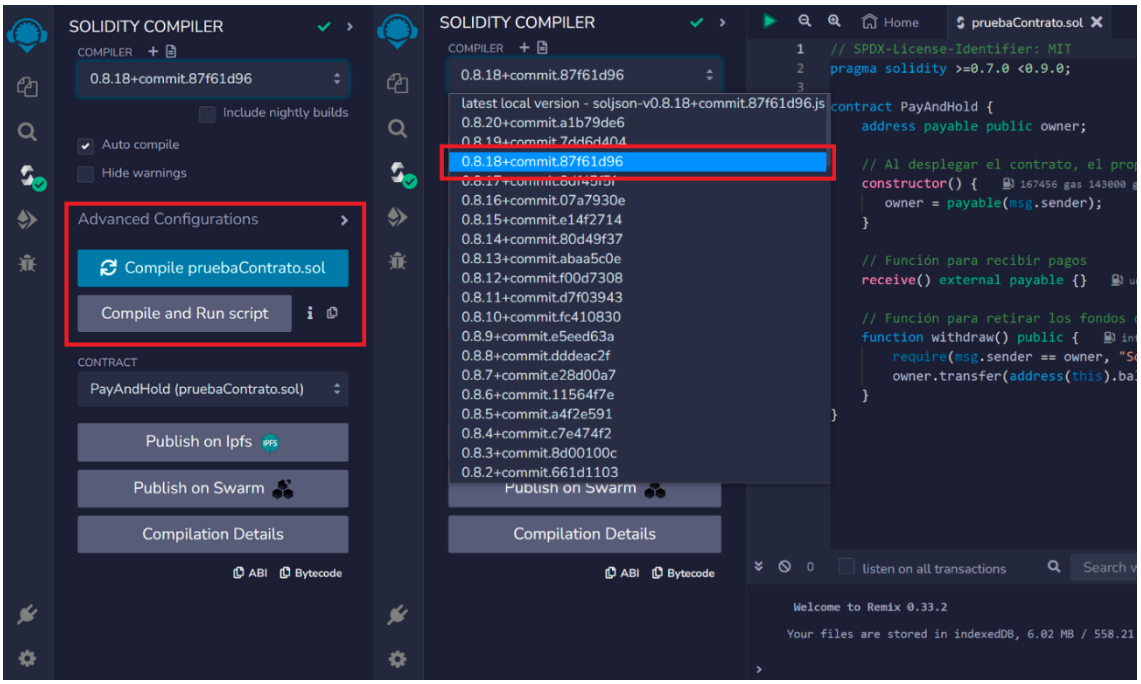


Figura 45. Versiones del compilador de Solidity
(Fuente propia)

Posteriormente se ha lanzado el mismo contrato, pero utilizando esta herramienta para comprobar que no haya problemas con la versión *"v0.8.18+commit.87f61d96"* como se puede ver en la Figura 45. Sin embargo, una vez lanzado hay que revisar el contrato en la cadena de bloques para confirmarlo y así cualquier usuario con una billetera pueda interaccionar con él. Además, esto nos permite conseguir el código *ABI*[17] que nos hará falta más adelante.

Seguidamente, en un segundo test se ha utilizado la red *Mumbai* (de pruebas) de *Polygon* que también es *EVM* compatible, se ha repetido el mismo proceso, pero en este caso el compilador no llegaba a ejecutarse por completo, por lo que he tenido que utilizar otra versión, concretamente la *"v0.8.9+commit.e5eed63a"*.

Una vez que tenemos los contratos desplegados en ambas redes podemos empezar a probar si los métodos que hemos creado funcionan como queremos. Para ello utilizaremos de nuevo la herramienta *Scan* de la cadena de bloques correspondiente colocando la dirección del contrato

[17] Application Binary Interface

que hemos obtenido como resultado de la compilación en *Remix Proyect*. Esta opción se encuentra en el apartado "*Contract*" una vez que hayamos colocado la dirección en el buscador.

En este paso tendremos que conectar una billetera para poder darle a los botones que corresponden a las funciones que tiene el Smart Contract. Es importante saber, como se puede ver en la Figura 46, que cualquier usuario puede hacer uso de las funciones, sin embargo, el método para retirar solo funcionará en el caso de que el usuario sea el que haya desplegado dicho contrato.

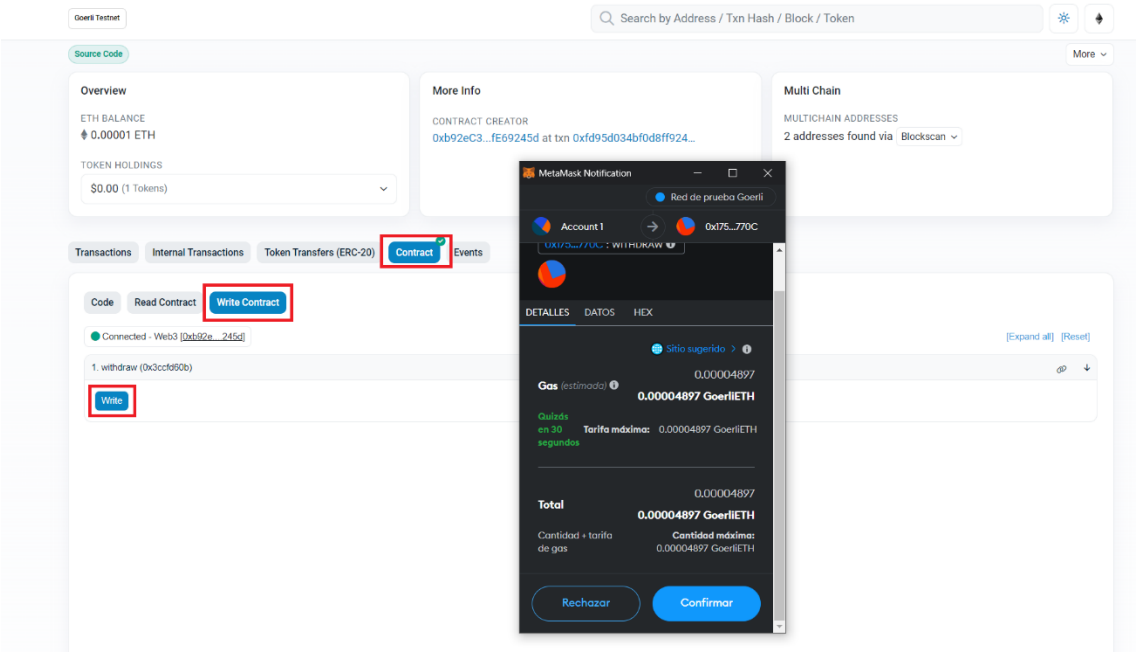


Figura 46. Prueba de la función de retiro
(Fuente propia)

Visto todo esto, en el apartado de "*Code*" se encuentra el código implementado de nuestro contrato y una sección muy interesante llamada Contract ABI que se ha mencionado anteriormente. Pues bien, este código que aparece es una representación JSON del contrato que permite a las aplicaciones cliente interactuar con él. Básicamente es una especificación de cómo codificar y decodificar los datos para que sean leídos y escritos en la *blockchain* [39]. En pocas palabras, detalla las funciones y variables del contrato, incluyendo nombres y tipos de datos que necesita y si son de lectura o escritura.

Como se puede ver en la Figura 47, se hizo una primera prueba enviando 0.01 ETH, una segunda usando la función de escritura para retirar los fondos y por último un envío de 0.00001 ETH.

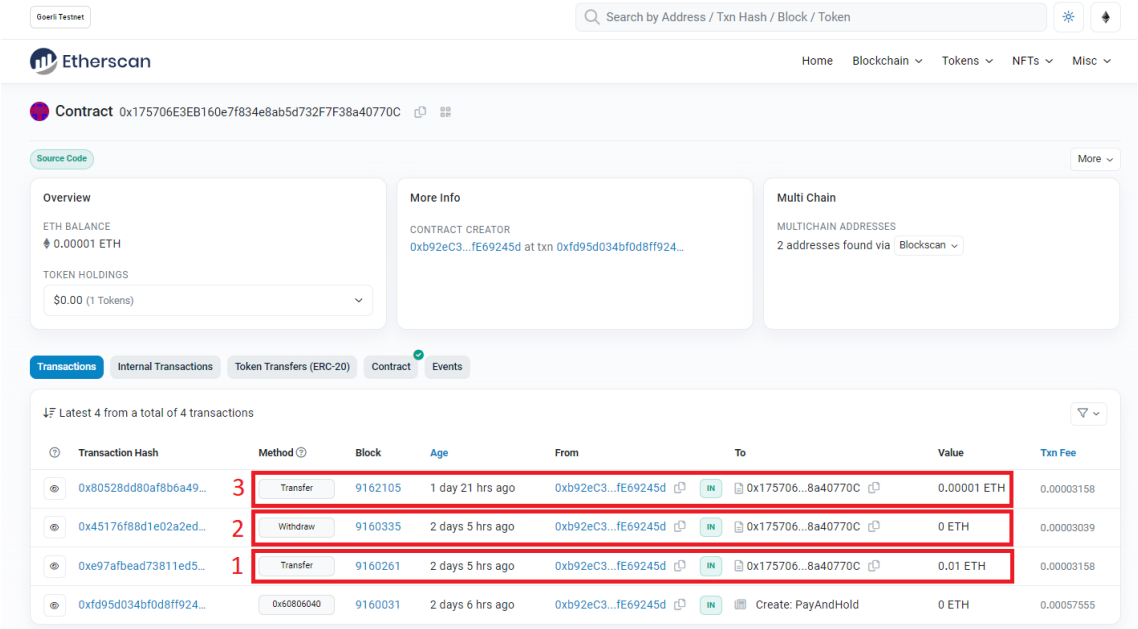


Figura 47. Pruebas contracto red Ethereum Goerli
(Fuente propia)

Posteriormente se repitió el proceso en la red de pruebas de Polygon, *Mumbai*, no obstante, se hizo la prueba de enviar a esa misma dirección del contrato de ETH en la red de Polygon una pequeña cantidad de MATIC. Lamentablemente no es posible interactuar con ese contrato ya que cada red tiene el suyo propio y, por tanto, esos fondos están perdidos.

En la Figura 48 se pueden ver las 3 pruebas realizadas en el segundo contrato, dos transferencias de 0.00001 MATIC cada una y la tercera es un retiro de todos los fondos (0.00002 MATIC).

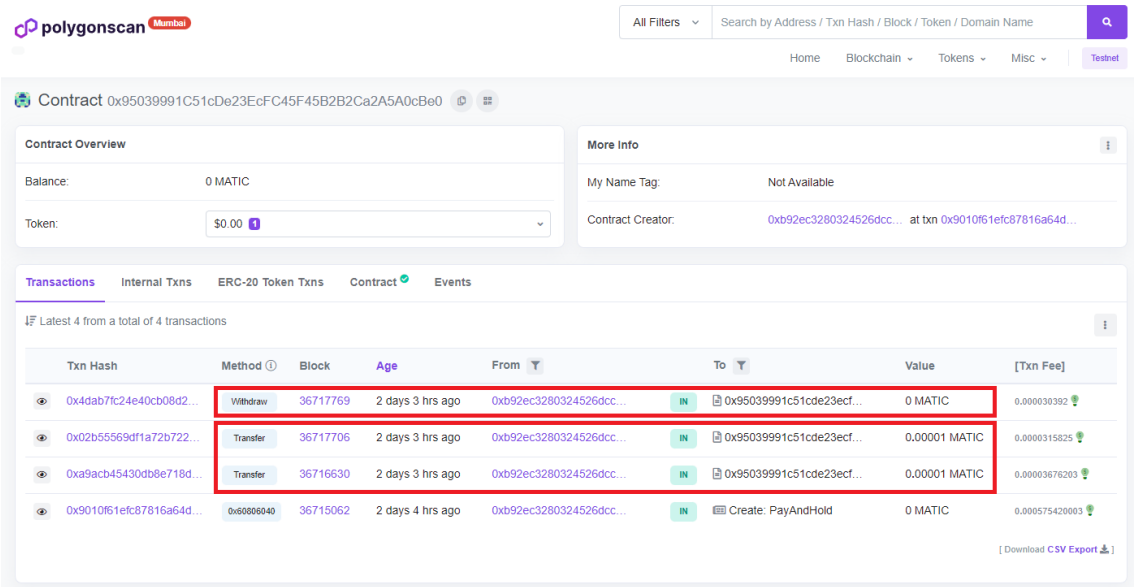


Figura 48. Pruebas en el contrato red Mumbai Polygon
(Fuente propia)

Llegados a este punto, y comprobado el funcionamiento, se tuvo la idea de probar con otras criptomonedas en una misma red. Es aquí donde se detectó un fallo en el Smart Contract que no se conocía hasta este punto. Los contratos implementados solo servían para la moneda principal de la cadena de bloques correspondiente, para el resto no era funcionales porque se tratan de tokens *ERC-20*, es decir, siguen un estándar especifico diferente.

En este caso, ver Figura 49, se envió por la red Mumbai dos transacciones cada una de ellas de 1 USDC (criptomoneda que replica el precio del dólar y es un token *ERC-20*). Una de las características que tiene este tipo de blockchains es que es inmutable, es decir, cuando una transacción ha sido confirmada no puede ser revertida, en este caso, como el contrato no tenía esta función, no tiene forma de interactuar con ellos para moverlos y es por ello que se han perdido. Realmente no es un problema, ya que estas redes están hechas para probar todo esto y evitarlo con dinero real.

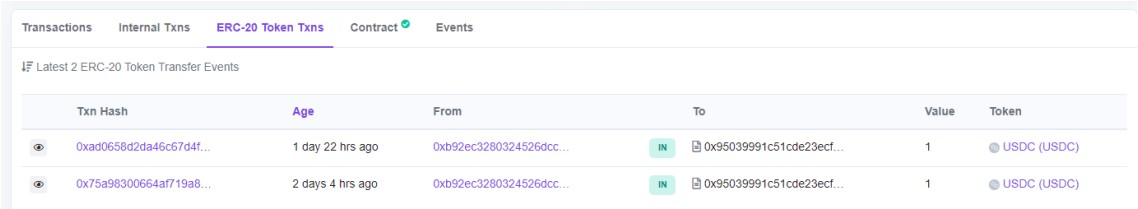


Figura 49. Prueba de envío tokens ERC-20
(Fuente propia)

Para poder resolver este inconveniente, hay que saber que estos tokens *ERC-20* son Smart Contracts en sí mismos y, por tanto, tienen su propia lógica. En este caso al enviar tokens ERC-20 a un contrato, para poder retirarlos se tiene que llamar a la función "*transfer*" del propio contrato del token *ERC-20*. Es por ello, que si se desea recibir y retirar este tipo de tokens se debe de agregar la siguiente función (a falta de comprobar si se desea retirar una cantidad o todo el saldo, además de devolver un aviso si el saldo no es mayor a cero):

```
21    // Función para retirar Tokens ERC-20
22    function withdrawToken(address token, uint256 amount) public {    infinite gas
23        require(msg.sender == owner, "Solo el propietario puede retirar tokens");
24        IERC20(token).transfer(owner, amount);
25    }
```

Figura 50. Función para retirar tokens ERC-20
(Fuente propia)

De esta forma, mediante esta función que solo podrá ejecutar por completo el dueño del contrato, enviará a la dirección del contrato de ese token *ERC-20* en concreto, la cantidad que tiene que enviar del token a la dirección del dueño, realizándose así el retiro.

Llegados a este punto se ha completado con éxito el segundo *sprint* de implementación de la pasarela de pagos.

9.3. Sprint 3: Implementación de la interfaz de usuario y la integración de Metamask

En este tercer sprint nos centraremos en la implementación de la primera parte de la interfaz de usuario, que corresponde a la elección de billeteras. Para comprobar el funcionamiento utilizaremos Metamask que es la wallet más usada por la comunidad.

En este caso se utilizó Wagmi para importar el conector de Metamask y así posteriormente configurarlo e incluso añadir las redes (Avalanche, BSC, Mainnet, Goerli y Polygon Mumbai) para hacer pruebas en los próximos sprints. Para las comprobaciones que tengan un coste (enviar una transacción) se utilizarán las dos últimas mencionadas. Igualmente, esto nos permitirá comprobar saldos y que la conexión funciona en las redes que hemos elegido.

Por otro lado, necesitaremos un proveedor para conectarnos a cada cadena de bloques. Aquí entra en juego Alchemy, Infura y otros proveedores vía API, sin embargo, no soportan todas las redes, por lo que deberemos de realizar una configuración para evitar errores. La solución ha sido ordenar los proveedores de RPC por prioridad, de modo que, si una blockchain no es compatible con un proveedor, el sistema pasa al siguiente. Si no se encuentra ningún proveedor compatible, se utiliza un proveedor público como última opción. Como hemos visto anteriormente, esto solo nos reducirá prestaciones como la latencia y la privacidad.

Como dato curioso, el uso de una RPC privada ofrece una ventaja notable. En situaciones donde la red está congestionada debido a la alta demanda de direcciones públicas, podemos disfrutar de una experiencia de usuario mejorada al evitar errores y el uso de más gas, que nos perjudicaría con costes más altos para llevar a cabo la transacción.

Por otro lado, para completar correctamente este sprint, se ha tomado la decisión de crear tres componentes en la interfaz de usuario. El primero de ellos, llamado *selectorWalletExchanges*, se ubicará en el lado derecho de la interfaz y mostrará el listado disponible de billeteras y exchanges como se puede ver en la

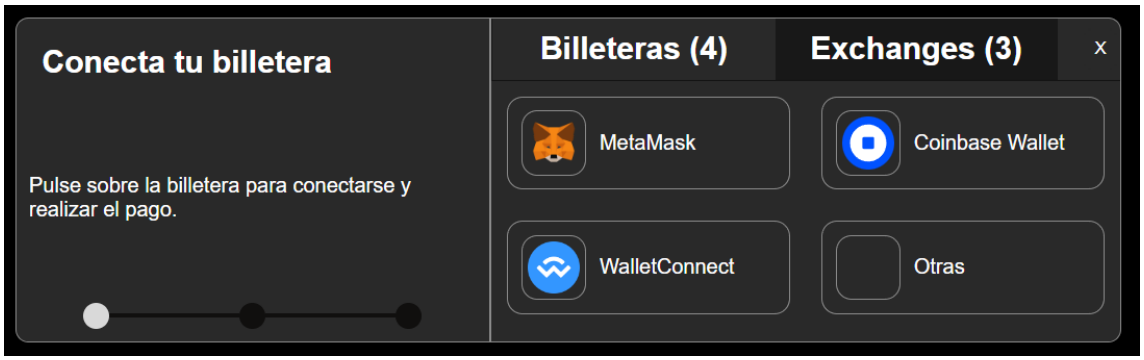


Figura 51.

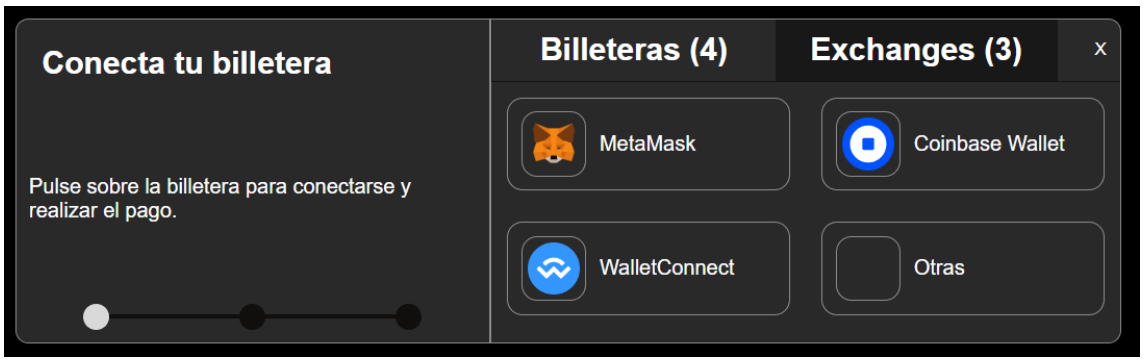


Figura 51. Interfaz de selección de billeteras
(Fuente propia)

Al interpretar las Wallets y los exchanges de manera similar, se ha creado un archivo *global.d.ts* como *definitions*. En este archivo, la interfaz *IBaseWalletExchanges* establece propiedades comunes como el identificador (id), el nombre (name), la imagen del logotipo (logoImg) y las redes (networks) asociadas. Por otro lado, la interfaz Network define la estructura de una red, incluyendo un identificador (id) y un nombre (name), esto último nos será muy necesario para el próximo sprint.

Posteriormente, las interfaces *Wallet* y *Exchange* extienden *IBaseWalletExchanges*, lo que significa que heredan las propiedades de dicha interfaz. La interfaz *Wallet* no agrega nuevas propiedades, mientras que la interfaz *Exchange* incluye una propiedad adicional llamada *nameExchange* para diferenciar los exchanges específicamente.

En segundo lugar, se ha creado un componente llamado *stepper* que permitirá mostrar en qué paso de la conexión de la billetera nos encontramos, y estará ubicado en el lado izquierdo de la interfaz.

Por último, *pendingConnection*, nos permitirá mostrar en el lado derecho la interfaz de los estados de conexión: *pendiente (1)*, *rechazado (2)*, *conectado (3)* o *no encontrado (4)*. Como podemos ver en la imagen 2 de la Figura 52, el estado *rechazado* dará al usuario la opción de intentarlo de nuevo o volver al listado de billeteras.

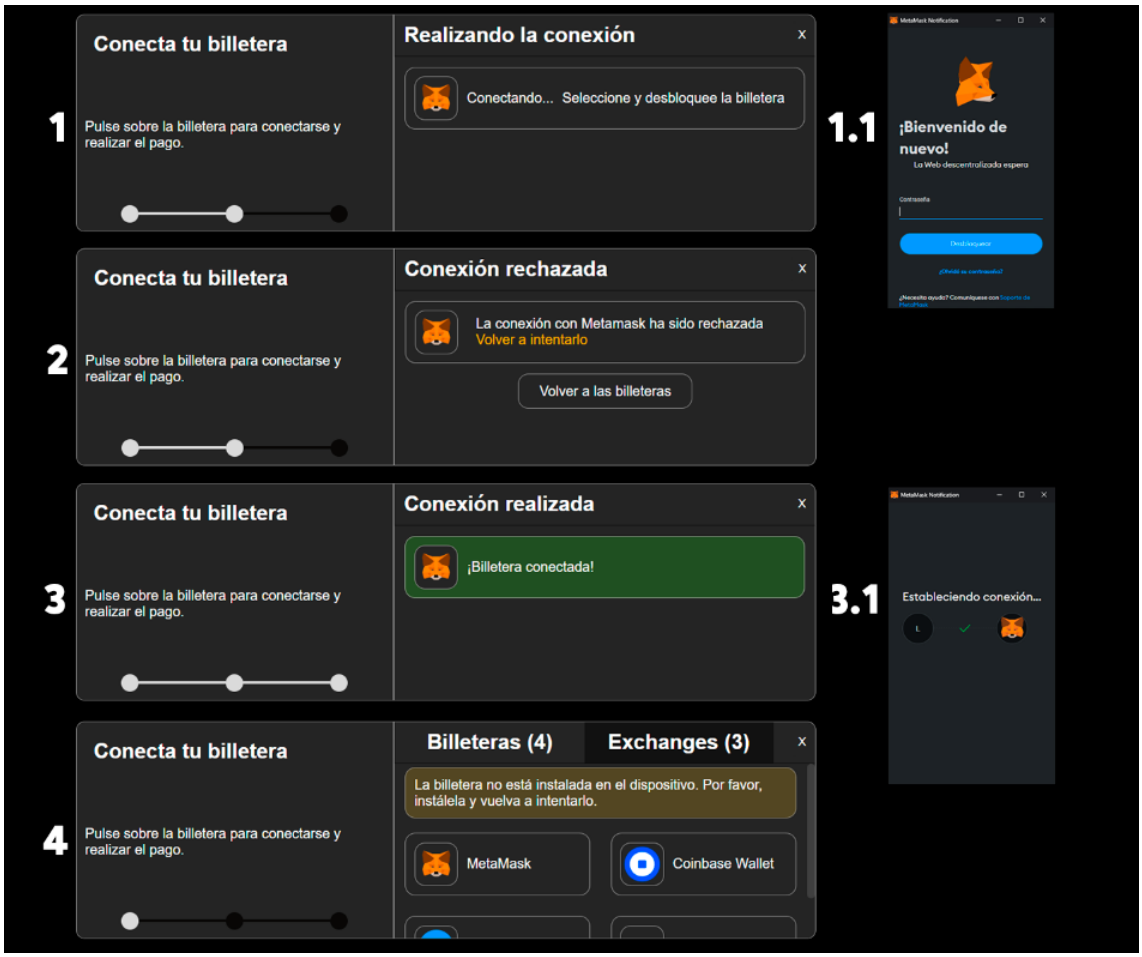


Figura 52. Interfaces de progresos de conexión y estados
(Fuente propia)

En caso de que no haya sido posible conectarse a la billetera elegida porque no está instalada, se ha añadido un aviso para elegir otra o intentarlo más tarde (imagen 4).

Además, se ha implementado una funcionalidad para detectar el cambio de cuentas en Metamask. Metamask permite a los usuarios tener múltiples cuentas o "direcciones" en una misma billetera. Con la propiedad *shimDisconnect* de *Wagmi*, puedo revisar que, si un usuario cambia de cuenta en Metamask, la aplicación también actualiza la cuenta mostrada. Esto es importante para mantener la coherencia entre los datos que muestra la billetera del usuario y la aplicación.

También se han implementado las billeteras *Coinbase Wallet* y *WalletConnect*, que a pesar de no ser un objetivo de este Sprint ni del TFG por limitación de tiempo, las configuraciones son muy parecidas. En este caso si pulsamos sobre ellas y no tenemos las extensiones instaladas en el navegador, nos mostrará el mensaje por defecto de instalar o conectar, ver Figura 53.

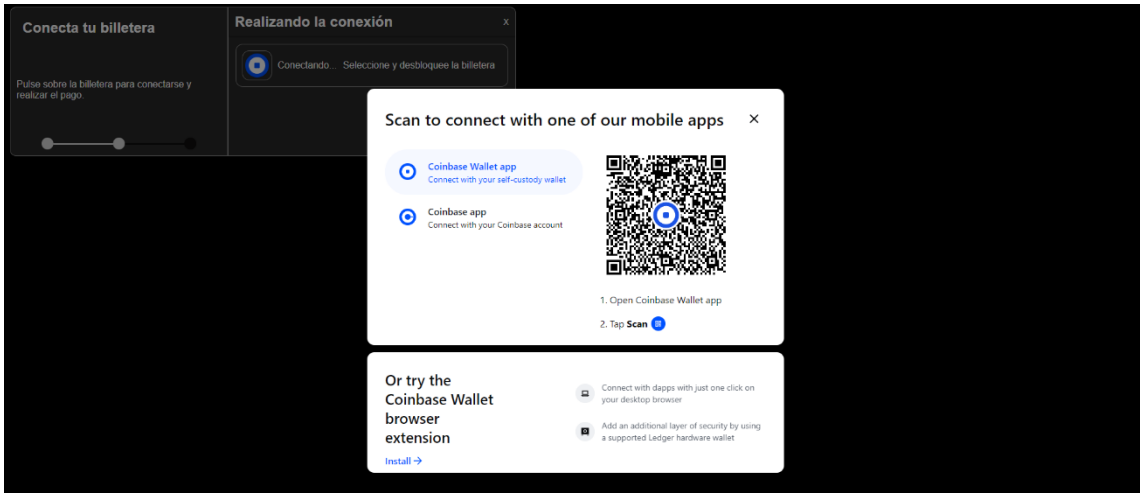


Figura 53. Mensaje para instalar billetera
(Fuente propia)

En último lugar, mostraremos la dirección de la billetera con los 4 primeros y últimos dígitos para preservar la privacidad, asimismo se ha añadido un botón de desconexión para volver al estado inicial y desconectar por completo la billetera de nuestra aplicación.

En resumen, en este tercer sprint, implementamos la interfaz de selección de billeteras utilizando Metamask. Configuramos proveedores RPC, priorizando la privacidad y reduciendo costes. Además, creamos los componentes para mostrar el listado de billeteras, el progreso de conexión y el estado de conexión.

9.4. Sprint 4: Implementación de la selección de criptomonedas y redes

El objetivo de este sprint es crear una lista de selección de criptomonedas. Para cada criptomoneda se mostrará: el logo, el nombre, el ticker y, en caso de tener redes compatibles, el logo de estas.

En este sprint se ha implementado el selector de criptomonedas mediante un nuevo componente llamado *selectorCrypto*, la función de este componente es diferenciar y filtrar las criptomonedas y redes que soporta la billetera elegida, de forma que las criptomonedas que no

sean compatibles con la billetera aparecerán con la etiqueta “no disponible” (no tienen redes que soporte la billetera) y, por tanto, no serán accesibles.

Si recordamos, en el pasado sprint, implementamos la billetera Metamask, esta billetera solo permite criptomonedas con redes EVM, por tanto, como podemos ver en la Figura 54, tanto Bitcoin como Litecoin están inaccesibles. En cambio, para el resto se pueden ver los logos de las redes compatibles en el siguiente orden: Ethereum, BSC, Polygon Mumbai, Ethereum Goerli y Avalanche.

Figura 54. Elección de criptomonedas y redes de la billetera Metamask
(Fuente propia)

Para poder probar el método creado de filtrado, que también diferencia entre Exchange y Billetera, se ha creado un fichero llamado *CryptoData.ts* donde se ha definido las interfaces y constantes relacionadas con criptomonedas y redes blockchain. Esto solo se utilizará en este sprint y en el siguiente (en el que se incluirá otros datos como la dirección de contrato de cada criptomoneda en su red y el código ABI correspondiente), ya que en el sexto sprint se implementará la base de datos.

Además, en este sprint se ha integrado la parte de elección de exchanges (ver Figura 55), de forma que cuando se seleccione un Exchange se mostrarán las criptomonedas que soporta, del mismo modo se comprueba las criptomonedas y las redes que tiene disponible para retirar desde la plataforma. De momento, para el ejemplo se ha decidido crear un fichero *exchangesData* con la interfaz de exchanges, y sean definido tres exchanges de prueba con una configuración distinta cada uno.

Figura 55. Elección de exchanges
(Fuente propia)

En la Figura 56, se puede observar un ejemplo del Exchange 1, el cual acepta únicamente dos criptomonedas, ya que son las únicas compatibles con la red Ethereum (esto implica que el Exchange solo tiene habilitados los retiros a través de dicha red).

Figura 56. Elección de criptomonedas del Exchange 1
(Fuente propia)

También se ha añadido el componente *invoice* para mostrar la información de contacto, y una vez el usuario elija una criptomoneda, le aparecerá el formulario que se muestra en la Figura 57 para introducir su correo electrónico. A partir de aquí, se creará la factura, en esta pantalla se incluyen dos checks que devolverán true o false en función de si están marcados. Esto por ahora solo será visual, en el sprint 6 se finalizará para que la factura se cree y almacene en la base de datos una vez el usuario pulse sobre el botón “Enviar”.

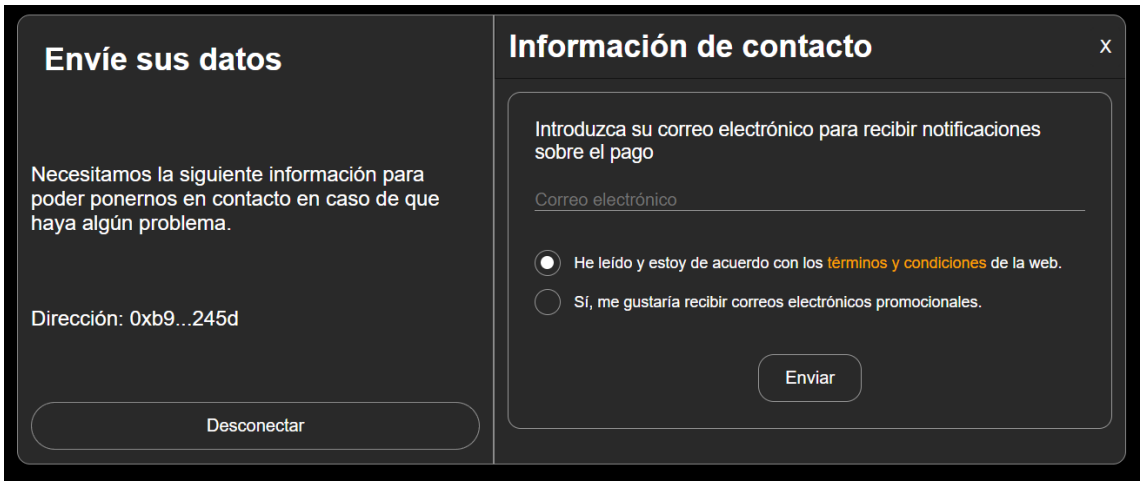


Figura 57. Interfaz de Información de Contacto
(Fuente propia)

Para resumir, en este cuarto sprint se creado el selector de criptomonedas en función de las redes que soporta la Billetera o Exchange elegido. También se ha implementado la elección de exchanges y el componente de factura, que muestra información de contacto y permitirá la creación de facturas más adelante.

A pesar de que este sprint parece más corto, es muy importante, ya que tendremos las bases para el próximo sprint que es con diferencia el que más información se muestra por pantalla y más detalles se tienen que tener en cuenta.

9.5. Sprint 5: Implementación de la visualización de datos de pago y gestión de fees

Llegamos a uno de los sprints más importantes y posiblemente con mayor cantidad de módulos y detalles a tener en cuenta, por lo que el tiempo estimado superará con creces las 25 horas aproximadas que se había establecido por sprint.

El objetivo de este sprint es la implementación de la interfaz donde se muestran todos los datos de pago, es decir, la cantidad a pagar, la cantidad que tiene el usuario en su billetera, el tiempo que falta para que caduque la factura y el selector de cambio de redes con sus correspondientes fees. Además, esta pantalla se reutilizará en gran medida para los pagos a través de exchanges, donde se añadirá algunos elementos como el código *QR* para escanear desde la propia aplicación del Exchange, un nuevo *stepper* de 4 pasos y, por último, varios mensajes modales para informar al usuario.

En primer lugar, se ha creado el componente *paymentData* que será aquí donde se implemente todo lo necesario para mostrar la información, con el *Props* se le enviará toda la información necesaria de los componentes anteriores, especialmente si el usuario ha elegido una billetera o un Exchange ya que esto marcará que métodos se utilizarán y en última instancia, cambiará el diseño de lo que el usuario verá por pantalla.

En los primeros días se ha trabajo en el selector de redes y los datos de las fees, para ello se había creado un selector básico con opciones, pero no permitía personalizarlo por completo para mostrar imágenes y otros datos. Se optó por buscar otra solución al importar un *Select* de *React*, el cual incluía mayor personalización para lograr el objetivo de ser lo más parecido a los diseños iniciales.

Para mostrar los costes de red, se ha utilizado la *API* de *Owlracle* [40] que ofrece 15 redes distintas para obtener los precios de realizar un envío en la cadena de bloques correspondiente. Una de las ventajas de esta API es que devuelve el coste de enviar una transacción en diferentes formatos: el precio en la criptomoneda principal de la red, en *Gwei* (antes de multiplicar por el precio de Ethereum en dólares y dividir el resultado entre el número de decimales del token) y en dólares, por lo que no será necesario hacer conversiones.

Esta API nos ofrece también distintos precios en función del porcentaje de probabilidad de que la transacción sea aceptada, es decir, cuando realizamos una transacción hay que pagar un coste para añadirla al bloque y sea validada por los nodos validadores. Si la comisión no es suficientemente alta, la transacción puede ser cancelada dando un error en este proceso, por lo que se ha decidido elegir la opción con mayor coste para que sea aproximadamente el 100% de éxito, lo que incluso permite que la transacción sea efectúe más rápidamente.

Sin embargo, lamentablemente una de las redes de pruebas como es Polygon Mumbai, que se está utilizando en el proyecto para poder comprobar el funcionamiento correcto, no es compatible, por lo que se ha buscado otra opción, en este caso se ha elegido en concreto la API oficial del proyecto, pero no ha sido tan fácil, ya que el dato que recibimos es en formato *GWei*, y por tanto, tendremos que hacer una petición a la *API* de *Coingecko* para conocer el precio de Ethereum en ese momento y hacer la operación de conversión.

Como podemos ver en la Figura 58 se ha conseguido un resultado final muy cercano a los diseños mostrados en la sección anterior del proyecto.

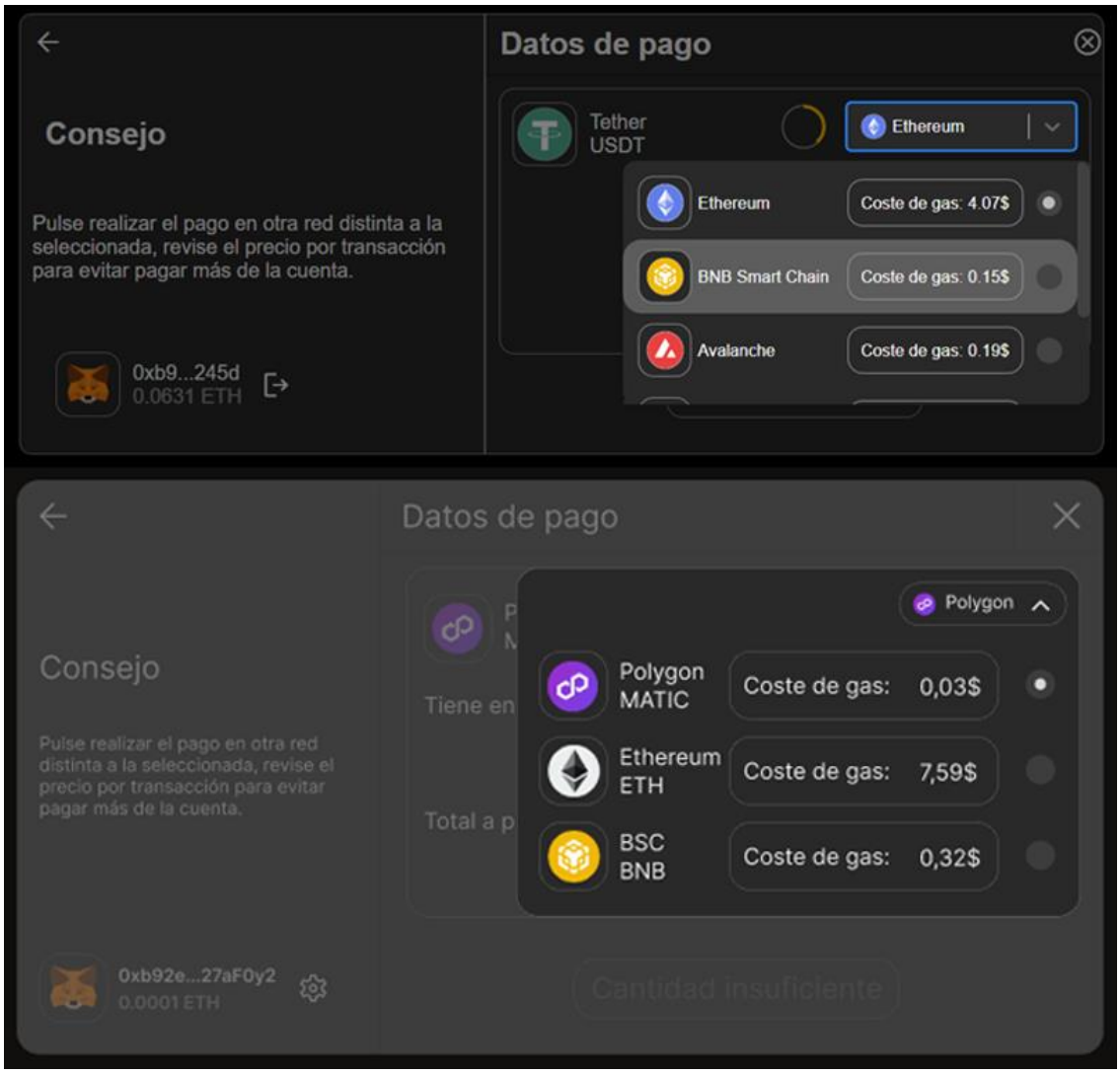


Figura 58. Comparativa entre resultado obtenido y diseño inicial (Fuente propia)

En cuanto a los códigos *QR* personalizados para la opción de exchanges, lamentablemente no se ha podido conseguir la versión gratuita de prueba de *qrcode-monkey* (ha sido solicitada, pero no se ha recibido respuesta por parte del equipo desarrollador), por lo que, para mantener el diseño se ha decidido pagar la suscripción mensual de 9$/mes en *Rapidapi* [41], un servicio de gestión de *APIs* con precios muy competitivos. El único inconveniente es la limitación de 200 peticiones al mes de este plan. Al ser solo un proyecto académico por ahora, será suficiente y no será necesario ampliar el plan.

Por todo esto, una vez implementado la parte correspondiente a la generación de los *QR*, se comentarán las líneas necesarias de código para evitar realizar peticiones mientras se desarrolla e implementa el resto del proyecto. En este caso se mostrará una animación de cargando.

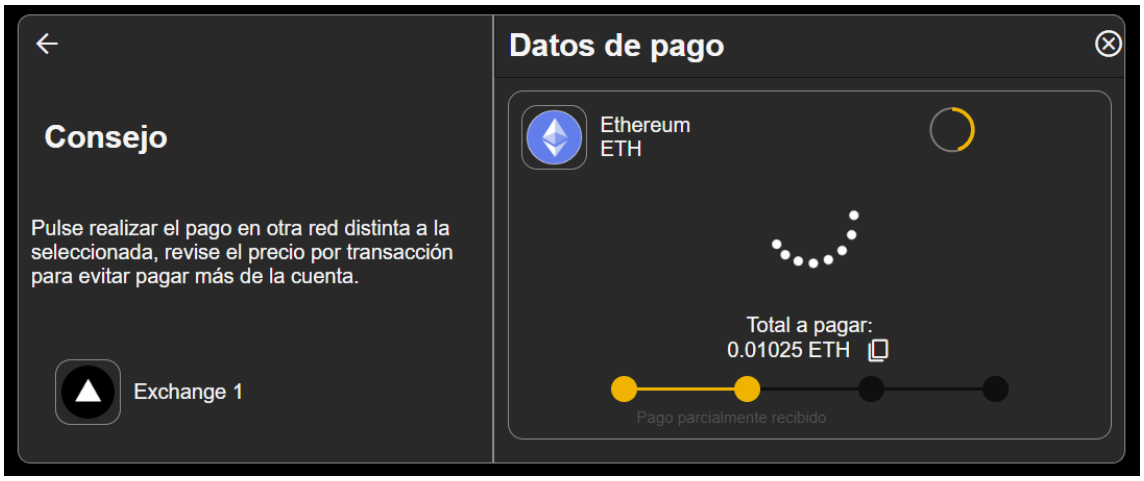


Figura 59. Interfaz de pago con animación en la zona del QR
(Fuente propia)

La principal razón para seleccionar esta API es que permite la personalización completa del código QR para integrarse con la pasarela y también incluir una imagen central de la criptomoneda seleccionada (ver Figura 60). La respuesta que recibamos de esta API necesitará ser procesada a través de un método que la convierta en base 64 para poder visualizarla correctamente.

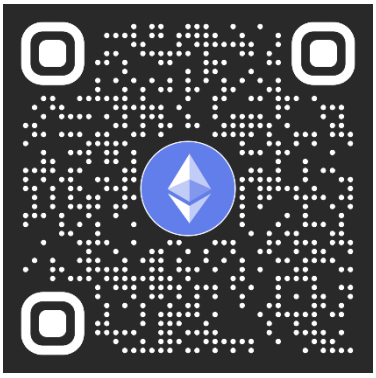


Figura 60. Ejemplo código QR de la billetera de Ethereum
(Fuente propia)

Como se ha comentado al inicio, se ha añadido también un *stepper* de cuatro pasos para mostrar en qué estado se sitúa el pago: esperando el pago, pago parcialmente recibido, pago completo detectado y, por último, pago confirmado. Para comprobar el funcionamiento y evitar estar haciendo transacciones continuamente, se ha creado unos métodos de prueba para generar una simulación y ver que todo el diseño y los componentes estaban donde corresponde.

Los pasos de este *stepper* funcionan gracias a la información que se obtiene mediante la API de *Etherscan* [42]. Esta API nos permite hacer consultas a la cadena de bloques de Ethereum y enviarle unos argumentos (especialmente la dirección pública de la billetera) para poder verificar los pagos que se han recibido. Para ello, deberemos de pasarle la dirección de la billetera que se ha generado previamente. Con esto, se recibe como respuesta un array de las

transacciones que ha realizado la dirección de la billetera junto con la información detallada de cada transacción.

De toda esta información, nos quedaremos con: la cantidad enviada, el contrato de la criptomoneda que es único y nos permite diferenciar entre otras que puedan tener el mismo nombre (así evitamos intentos de estafas), los decimales para poder posteriormente hacer el cálculo correcto, la dirección desde dónde viene el pago para almacenarlo en la base de datos y, por último, la cantidad de confirmaciones posteriores que ha tenido esa transacción una vez se ha añadido a la cadena de bloques.

Por motivos de limitación de tiempo, y dado que se trata de un proyecto académico, solo se hará funcional para la red de pruebas de Ethereum, aunque para añadir más redes se tendría que añadir nuevas *APIs* correspondientes a cada red, ya que no se ha encontrado una misma *API* que incorpore varias opciones al mismo tiempo.

Casi ya para finalizar este componente, solo para la sección de exchanges, recordemos que necesitábamos generar una billetera única para poder diferenciar usuarios que desean pagar un mismo producto simultáneamente. Para ello, utilizamos la librería de *ethereumjs-wallet* que nos permite generar una clave pública y privada en cada ocasión, la clave publica se utilizará para la generación de los códigos *QR* y en el mensaje modal para poder copiarla, mientras que la privada se añadirá cifrada a la base de datos (necesaria para posteriormente realizar los retiros).

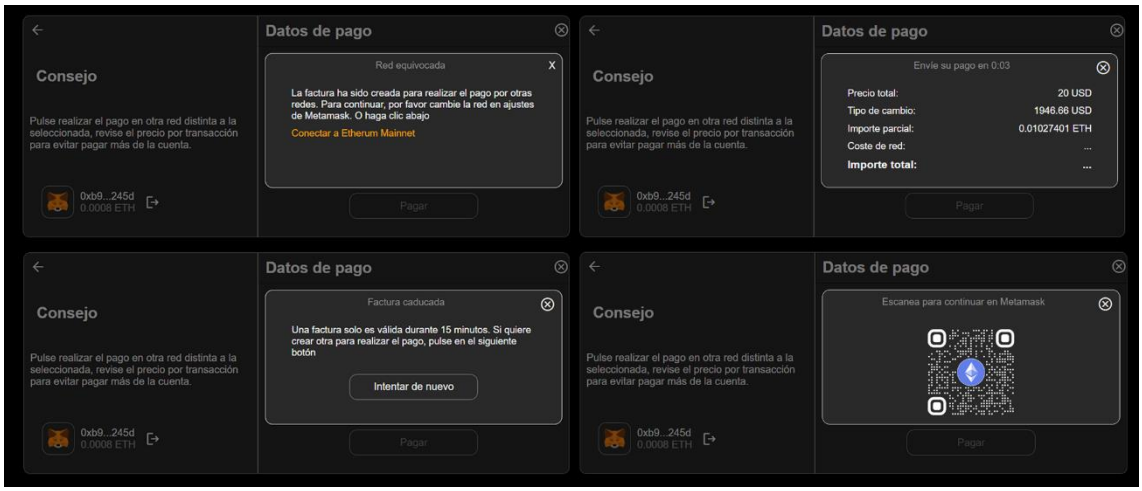


Figura 61. Resultados finales mensajes modales
(Fuente propia)

Finalmente, se han implementado los mensajes modales de la Figura 61. El primero se activa cuando se selecciona incorrectamente una red, es decir, si el usuario cambia de red en su propia billetera a una que no es compatible con la pasarela, se mostrará un mensaje de "red equivocada" y solo se permitirá cambiar a la principal, que es Ethereum.

El segundo mensaje modal corresponde con los datos de información, donde se muestra el tiempo restante para que la factura caduque y los precios de conversión que se han utilizado para calcular el importe final. Para la sección de exchanges, será el mismo modal, pero incluyendo un extra (cantidad pendiente si el pago no se completa en una sola transacción).

Si caduca la factura, se mostrará el tercer mensaje modal, con un botón para volver a realizar los pasos y crear una nueva.

En caso de que pulsemos el botón de pagar con “Metamask App” se abrirá el cuarto mensaje modal, está función no se ha desarrollado por completo porque realmente era compleja e innecesaria, ya que para ello sería mejor utilizar o incluir esta opción en exchanges ya que no se usaría un *Smart Contract* sino un pago de forma manual.

Los últimos tres mensajes modales de ayuda se pueden ver en la Figura 62:

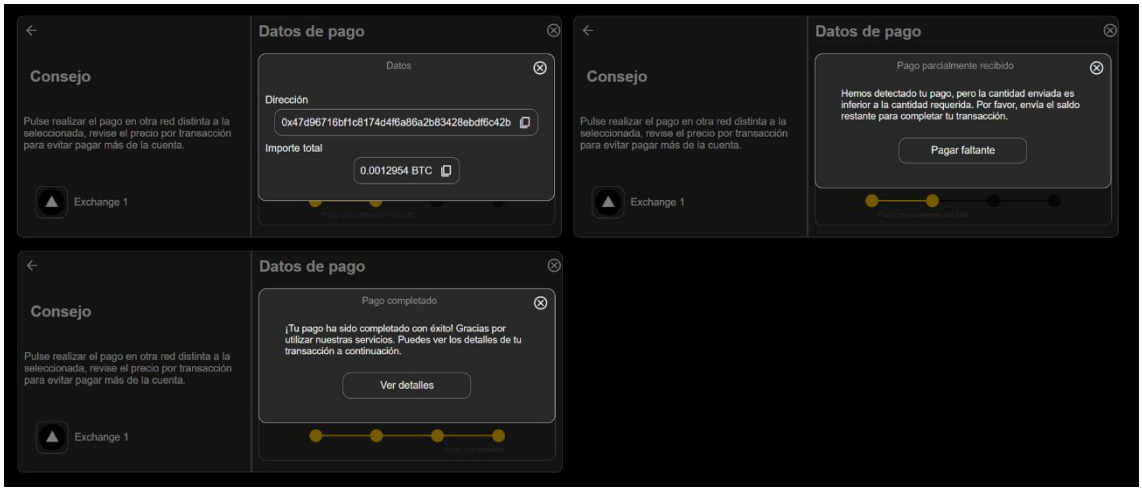


Figura 62. Resultado final mensajes modales sección Exchange
(Fuente propia)

El primero se muestra al darle copiar al “Total a pagar”, en ese modal se muestran los dos datos para poder copiarlos e introducirlos manualmente en la plataforma elegida.

El segundo mensaje modal se mostrará cuando el pago recibido sea menor a la cantidad total, ya sea porque el usuario se haya equivocado o simplemente quiera hacer el pago desde dos o más direcciones distintas. Cuando la pasarela detecte más pagos, hará la suma hasta que esta supere el importe total.

En último lugar, este mensaje se mostrará cuando el pago se haya completado en su totalidad y haya sido confirmado al menos dos veces en la cadena de bloques, lo que significa que es irreversible la transacción.

Antes de acabar con este sprint se ha implementado la funcionalidad de realizar el pago en la sección de billeteras (ya que la de exchanges está finalizada a falta de algún cambio final o pruebas más estrictas). El funcionamiento de los *Smart Contract* es bastante curioso, ya que dependiendo de si la criptomoneda elegida para hacer el pago es la *Coin* principal o un token *ERC-20*, la interacción con este, será distinta.

Por ejemplo, para la moneda *ETH* dentro de la red Ethereum, que es la que se utiliza para pagar las comisiones de red, utilizaremos el Hook de Wagmi *useSendTransaction*. Para utilizarlo, solo le tendremos que pasar la dirección de nuestro contrato en la red correspondiente y la cantidad a enviar.

No obstante, para los tokens *ERC-20*, utilizaremos *useContractWrite*. La razón de esto es que para interactuar con un contrato y enviar un token a una dirección necesitaremos: la dirección del Smart Contract de esa criptomoneda, el código ABI que define cómo se llaman a las funciones de un SC y se codifican, la función que utilizaremos del contracto (en nuestro caso "transfer") y, finalmente, los argumentos que serían la dirección de la billetera que recibe los tokens (en nuestro caso la dirección de SC de la red) y la cantidad.

Es por ello, que, en el siguiente sprint, cuando se lleve a cabo la integración de la base de datos, los diseños de la colección Criptomoneda, deberán tener unas modificaciones para almacenar el código *ABI*, ya que al principio del proyecto se desconocía la necesidad de este dato, lo cual justifica esta modificación futura.

Dicho todo lo anterior, en caso de que el usuario no tenga la cantidad suficiente de la criptomoneda elegida, el botón de pagar estará deshabilitado como se puede ver en la Figura 63.

Figura 63. Interfaz de pago de la sección de billeteras.
(Fuente propia)

Para finalizar, lo cierto es que este sprint ha sido crucial, encontrándose con varios desafíos técnicos como la limitación en el uso de las API y la necesidad de ajustar los esquemas de diseño. Pero pese a todos estos problemas, se ha logrado solucionar y hacer un progreso significativo en el proyecto. Consiguiendo así la interfaz multifuncional de pago y la gestión de las transacciones con criptomonedas tanto en la sección de Billeteras (con SC) como de Exchanges (manualmente).

9.6. Sprint 6: Integración con la base de datos e implementación de las transacciones

En este sprint el objetivo es implementar la integración con la base de datos para almacenar y recuperar datos de las transacciones realizadas y de los usuarios. Todo lo visto en los sprints anteriores estaba relacionado con el front-end, sin embargo, en este sprint se llevará a cabo el desarrollo del back-end.

Lo primero que se ha realizado es la instalación de MongoDB para ello se ha descargado e instalado *MongoDB Community Server (v 6.0.7)* y la herramienta de *MongoDB Compass (v 1.38.2)*.

Posteriormente se ha creado una carpeta back-end, donde se ha generado un archivo server.js que será el punto de entrada a la aplicación back-end. Este archivo inicializa y configura el servidor para poner en marcha el manejo y respuesta de las solicitudes *HTTP* que llegarán de los usuarios.

Principalmente se encarga de cinco aspectos:

En primer lugar, se encarga de la inicialización del servidor Express que nos permitirá manejar las solicitudes y respuestas *HTTP*.

En segundo lugar, de la conexión de la base de datos mediante Mongoose, conecta la aplicación a una base de datos para almacenar y recuperar datos.

En tercer lugar, de la configuración del *Middleware* que se encargará de procesar las solicitudes y respuestas de la aplicación, concretamente hay dos, el primero para *parsear* los cuerpos de las solicitudes entrantes en un formato que la aplicación pueda utilizar. El segundo para que el servidor pueda aceptar solicitudes de diferentes orígenes ya que es posible que en un futuro el front-end y back-end estén separados.

En cuarto lugar, nos permitirá la configuración de las rutas, esto es necesario para que las rutas definan los endpoints de la API que los usuarios utilizarán. En este caso todas las rutas irán con el prefijo “/api”.

Por último, se encarga de la inicialización del servidor para que escuche en el puerto *3001*.

Posteriormente se ha creado una carpeta *models* con archivo *models.js* donde se encuentran los modelos en Mongoose que sirven para definir la estructura de los documentos en las colecciones de la base de datos de MongoDB. Es decir, que tipo de datos son y algunas validaciones de si son requeridos, se inicialicen a “false”, entre otros.

Se han mantenido los diseños de las colecciones de la sección 8, con algún ligero cambio en Redes, donde se ha añadido un campo “contract_ABI” para de cada contrato inteligente, y un “contract_pay” con la dirección del Smart Contract al que se hace el pago.

Por otro lado, en la colección *Transaccion,* se ha añadido el campo hash, que es un array de *String* en el que se almacenará los hashes de las transacciones realizadas, en caso de que sea necesario consultarlas en la cadena de bloques.

También se ha creado una carpeta de *Routes* que contiene *routes.js* y *routes.rest*; en el primero encontraremos las definiciones de cómo responderá la aplicación a las solicitudes *HTTP* que llegan a las rutas específicas. Por poner un ejemplo, para usuario tendremos las siguientes rutas:

- “router.post('/usuario', createUsuario)” es una ruta POST a “/usuario” y cuando una solicitud llega a “/usuario”, la aplicación ejecutará la función *createUsuario* que creará un usuario en la base de datos.
- “router.get('/usuarios', findAllUsuarios)” es una ruta GET que devuelve la lista de todos los usuarios.
- “router.get('/usuario/:email', findOneUsuario)” es una ruta GET que incluye el parámetro email, devuelve el usuario con ese correo.
- “router.put('/usuario/:email', updateUsuario)” es una ruta PUT que incluye el parámetro email y actualiza los detalles de ese usuario mediane la función *updateUsuario*.
- “router.delete('/usuario/:email', deleteUsuario)” es una ruta DELETE y ejecuta la función *deleteUsuario* que elimina el usuario de la base de datos.

De una manera similar, se definen las demás rutas para Billetera, BilleteraTemp, Transaccion, Criptomoneda, Red y Exchange.

Seguidamente, en el fichero *routes.rest* se ha descrito como usar las rutas de la API en el back-end. Para cada una de las colecciones se han creado las operaciones *CRUD* (Crear, Leer, Actualizar y Eliminar) utilizando el protocolo *HTTP*. En cada ejemplo encontramos la URL a la que hacer la solicitud, el tipo de solicitud *HTTP* (POST, GET, PUR y DELETE) y los datos que deben incluirse en el cuerpo de la solicitud.

En este caso utilizaremos este archivo junto con la extensión de *Visual Studio Code*, *REST Client* para hacer pruebas e introducir en la base de datos los datos necesarios de Exchanges, Billeteras, Redes y Criptomonedas. Este archivo tambien puede ser utilizado por aplicaciones como *Postman*.

Por último, encontramos la carpeta *Controllers*, donde se sitúan los controladores de cada colección, es decir, los controladores manejan la lógica. Cuando el cliente envía una petición a un *endpoint* en particular, es el controlador el que procesa esa petición e interactúa con el modelo para obtener o modificar los datos y en última instancia, devuelve la respuesta al cliente.

Una vez finalizado con el back-end necesitamos conectarlo con el front-end, para ello se ha creado una carpeta de *Stores*, donde encontraremos los módulos que proporcionan funciones para interactuar con la *API* y realizar las operaciones *CRUD*. Esto permite que las funciones estén agrupadas y, por tanto, el código sea más modular y sencillo de mantener.

En primer lugar, dentro de la carpeta *Stores* encontramos el *basicStore.ts*, que contiene la función *fetchAPI* que envuelve a la función *fetch*. Esta función la utilizaremos para realizar solicitudes a la *API*. En pocas palabras, esta función recibe un *path*, es decir, la ruta a la que se realiza la solicitud y opcionalmente un objeto *options* que contiene los métodos, encabezados y el cuerpo de la solicitud *HTTP*.

Posteriormente, se concatena la *URL* de la *API* con el *path* para formar la *URL* completa a la que se realiza la solicitud. Si la respuesta de la solicitud no es exitosa se lanza un error, pero si es exitosa, devuelve un objeto *JSON*.

Seguidamente encontramos los *stores* individuales para cada colección donde están definidas las funciones para realizar operaciones *CRUD*, cada una de las funciones utiliza la función *fetchAPI* definida en el *basicStore*.

Para concluir este sprint, desde el front-end se han importado las funciones de los stores correspondientes (findAllCriptomonedas, getExchanges, y getAllBilleteras) para recuperar las criptomonedas, los exchanges y las billeteras respectivamente.

A continuación, se ha utilizado el *hook* de React *useState* para crear tres estados, cada uno contiene el array de objetos de cada tipo. Así podremos mostrarlos cuando sea necesario remplazándolo por los datos estáticos creados en los anteriores sprints.

Del mismo modo, en *invoice*, se crearán nuevos usuarios en la base de datos tras completar el formulario y pulsar el botón de "Enviar". Si el usuario no existe se creará, si por el contrario ya se encuentra en la base de datos, se actualizará la información.

En *paymentData*, implementado este componente en el sprint anterior, en el momento que se crea la billetera temporal (si es el caso), se creará y almacenará en la base de datos. En cuanto a las transacciones, cuando se lleve a cabo en la sección de billeteras, se crearán introduciendo todos los datos (hash, criptomoneda, red…) y la transacción tendrá el estado de "*confirmed*". En el caso de la sección de Exchange, la transacción tendrá diferentes estados y, por tanto, se creará la transacción en el estado "*pending*" e irá pasando por los demás estados actualizándose este campo en la base de datos. Cuando llegue al último paso, se establece como "*confirmed*" y se añade el hash de la transacción. Finalmente, se guardará el *_id* de la transacción para almacenarlo en el campo correspondiente del usuario.

Al llegar a este punto, se ha completado con éxito el sprint, donde se instaló MongoDB para establecer un servidor con *Express* y *Mongoose*. Se definieron modelos de datos, rutas de la API y sus controladores, Además, se utilizaron *stores* para la interacción con el front-end/API y el remplazo de los datos estáticos por datos reales.

9.7. Sprint 7: Pruebas finales, ajustes y Lanzamiento a Producción

Llegados al último sprint de la implementación, antes de llevar el producto a un entorno de producción, el objetivo es realizar algunas pruebas finales para encontrar errores y ajustar la interfaz correctamente.

Al seleccionar una moneda en la sección de exchanges, ocurría que a veces el valor de la criptomoneda salía como "Infinity", esto se debe a que al realizar los cálculos a veces ocurría que el divisor era cero. Se ha corregido comprobándolo y cambiando por la variable correcta.

En el mensaje modal de información de pago, donde se muestra la billetera temporal y la cantidad para copiar pulsado el botón, se ha remplazado la cantidad por el valor correspondiente a cada pago. También se ha modificado para que las comprobaciones de pago se realicen en las billeteras temporales y no en la billetera de ejemplo.

Se ha detectado que, cuando el usuario realiza un pago inferior al solicitado, es útil mostrar la cantidad pendiente en el mensaje modal, para así poder copiarlo y hacer el pago de la cantidad restante.

Justo después, se ha observado un error que ocurría pocas veces, cuando copias la cantidad (que tiene ocho decimales) y la pegas en una billetera para hacer el pago manual, la propia billetera redondea a cinco decimales en algunas monedas como *USDT*, ya que el valor es insignificante. Del mismo modo lo hace la *API* de *EtherScan* con el valor recibido de pago, que en este caso devuelve seis, por lo que se han ajustado las comprobaciones a solo cinco cifras para evitar errores o pagos parciales por cantidades insignificantes. Este cambio no produce ningún inconveniente, pues apenas supone un 0,02% en los peores casos que son monedas con precios muy elevados en dólares.

Cuando se realiza una transacción, tanto en la sección de billeteras como exchanges, en el mensaje modal de pago completado, se muestra un botón de "*Ver detalles*". Al pulsar, te lleva a la aplicación de *scan* de la *blockchain* elegida para comprobar el pago. En caso de que el pago se realice en varias transacciones, se mostrará la última transacción y aparecerán debajo de este, botones más pequeños con cada transacción.

Había un pequeño error al darle a desconectar la billetera y posteriormente seleccionar la opción de exchanges, esto se ha conseguido solucionar añadiendo unas comprobaciones y limpiando las variables de *useState* de *React* al retroceder en los pasos o volver al inicio directamente.

Se ha eliminado el botón de cerrar el mensaje de modal cuando la factura ha caducado y se ha remplazado el contador de uno a diez minutos, esto se había añadido para hacer más rápido las comprobaciones, pero en la versión final se busca forzar al usuario para volver a generar un nuevo pago. Del mismo modo, el botón de cerrar del mensaje modal te devuelve al inicio de la pasarela de pagos.

Por último, se han ajustado detalles de *CSS* como es mostrar el icono de "*i*" en el centro del botón del tiempo restante para la factura. Asimismo, se ha ajustado su posición para que siempre salga arriba a la derecha, tanto si hay selector de redes como si no.

Solucionados todos estos problemas que a priori era necesarios para el buen funcionamiento de la pasarela, se dispone a desplegar en un entorno de producción.

Para ello, en primer lugar, se ha adaptado la configuración del proyecto de *next.js* para que la ruta principal del proyecto parta de */pasarela*. Esto ha hecho configurando el fichero

next.config.js. Esta modificación era necesaria puesto que en el servidor donde se ha hecho el *deploy* (*empezarainvertir.com*) tenía otra instancia de *next.js* corriendo.

También, se ha revisado que el código compile sin ningún error y se ha remplazado el puerto *3000* por el *3330* tanto en el *back-end* como en el *front-end* para que no entren en conflicto con los existentes en el servidor.

Seguidamente, se ha accedido a Digital Ocean, donde se posee el servidor virtual en el que se encuentra el proyecto principal y que permite escalar los recursos a medida que el proyecto crece [43]. Aquí, en la máquina donde se va a realizar el deploy, se ha procedido a ajustar Nginx de la siguiente forma:

Primero, la modificación del fichero */etc/nginx/sites-available/default* mediante el comando: *sudo nano /etc/nginx/sites-available/default.*

Segundo, se han añadido dos nuevos *upstream* (ver Figura 64), los cuales hacen referencia a las dos nuevas instancias que habrán corriendo en el servidor (el front-end y back-end).

```
upstream apiPasarela {
        least_conn;
        server localhost:3330;
}

upstream frontPasarela {
        least_conn;
        server localhost:3331;
}
```

Figura 64. Upstream de las dos instancias
(Fuente propia)

Tercero, se han añadido dos nuevas *location* encargadas de redirigir el tráfico a las nuevas instancias:

1. La primera para el *front*, la cual debe ir colocada por encima de la que existía previamente para evitar que la primera instancia de *next* intente dirigirte (ver Figura 65).

```
location /pasarela {
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_pass http://frontPasarela;
}

location / {
```

Figura 65. Location del front-end
(Fuente propia)

2. La segunda para el back-end, no importa su localización, ver Figura 66:

```
location /apiPasarela {
        proxy_http_version 1.1;
        proxy_pass http://apiPasarela;
}
```

Figura 66. Location del back-end
(Fuente propia)

Cuarto, una vez realizados y guardados los cambios, reiniciamos el servicio de Nginx para que estas modificaciones surjan efecto. Para ello usaremos el comando: *sudo systemctl restart mongod* y comprobaremos posteriormente que todo esté correcto mediante el comando: *service mongod status*.

A continuación, elegiremos una localización dentro de algún directorio donde el proceso de *Nginx* tenga permisos de lectura y creamos una nueva carpeta donde se clonarán ambos repositorios de *Github*. Para ello utilizaremos el comando de *git*: *git clone <url_repositorio>.*

Una vez esté todo listo, se necesitan levantar ambas instancias al mismo tiempo, para ello utilizaremos el gestor de procesos avanzada para aplicaciones de producción *Node.js PM2* siguiendo estos pasos:

Iremos a la ruta donde se ha clonado los repositorios mediante el comando: *cd <path>.*

Una vez allí, utilizaremos el comando asociado a PM2: *pm2 start npm --name frontPasarela -- run start.*

Repetiremos este proceso con el back-end: *pm2 start npm --name apiPasarela -- run start*

Una vez tengamos ambas instancias en PM2, comprobaremos que todo haya ido bien mediante dos comandos:

- *pm2 log*, el cual nos imprimirá por consola las últimas líneas que hayan impreso nuestras instancias recién levantadas. Esto se hace para asegurarnos de no tener errores en el proceso de levantar alguna de las instancias.
- *pm2 list*, donde veremos todas las instancias asociadas a PM2, el estado que tienen, cantidad de memoria, tiempo levantado, entre otros (Ver Figura 67).

id	name	mode	↺	status	cpu	memory
2	api	fork	330	online	0%	36.3mb
0	apiPasarela	fork	0	online	0%	30.3mb
1	front	fork	114	online	0%	35.7mb
3	frontPasarela	fork	2	online	0%	30.7mb

Figura 67. Instancias asociadas a PM2
(Fuente propia)

Después de realizar el primer despliegue se han encontrado y corregido los siguientes errores:

Cuando el sistema está esperando a recibir las transacciones para compararlas ocurría un error y se ha solucionado comprobando antes de todo si el array de transacciones está vacío o no.

Por otro lado, se ha arreglado un error que ocurría al mostrar el botón de finalización el pago, ya que faltaba una comprobación de si el pago se ha realizado mediante una billetera o un Exchange.

Para finalizar, se han añadido a la base de datos más criptomonedas como: ChainLink, USD Coin, Arbitrum, Optimism. Siendo un total de trece criptomonedas, de las cuales, siete pueden utilizarse para realizar pagos en la pasarela.

Se ha incorporado una nueva billetera llamada *Ledger* que permite la conexión con *coldwallets*, es decir, billeteras frías que no están conectadas a internet, son reconocidas mayoritariamente como dispositivos USB.

También se han incluido los Exchanges: Bybit, Coinex, Kraken y Bitfinex.

Realizadas todas las correcciones y una vez añadidos los datos a la base de datos, se ha comprobado los pagos de la sección de billeteras (con Smar Contract), y los pagos de la sección de Exchanges (tanto un Coin como un Token *ERC-20*), todas las pruebas han funcionado correctamente.

Durante el último sprint de la implementación, se realizaron pruebas y ajustes finales, se solucionaron errores en cálculos y de visualización. Se mejoró la experiencia de usuario además de ajustar el código para producción y se preparó el despliegue. Finalmente, se corrigieron errores de despliegue y se verificó el correcto funcionamiento. Con todo esto, podemos dar por concluida la implementación de la pasarela de pagos.

10. Pruebas y validación

En esta sección llevaremos a cabo las pruebas y la validación que se han definido en el apartado de diseño. Así se podrá confirmar que el proyecto funciona correctamente y cumple con su propósito, o en su defecto, poder identificar y resolver cualquier problema encontrado.

Antes de empezar con las pruebas, es importante destacar que el proyecto no está completado al 100%, es decir, dada la magnitud de la pasarela de pagos con criptomonedas y debido a las restricciones de tiempo, no todas las características previstas han sido implementadas. Por este motivo, no es posible realizar pruebas con usuarios reales, ya que no se venden productos reales a través de la pasarela.

También, se ha probado la pasarela de pagos con la colección de rutas REST que ha desarrollado para añadir, editar y eliminar datos en la base de datos. Asimismo, se ha probado la realización de pagos utilizando ambas opciones de pago y diferentes criptomonedas y redes. No se ha encontrado ningún problema significativo.

Sin embargo, se ha buscado un grupo reducido de personas para probar la pasarela con distintas criptomonedas y redes de pruebas como Ethereum Goerli y Polygon Mumbai. Esta prueba ha sido mayoritariamente a ciegas, lo único en lo que se les ha guiado a los usuarios ha sido en el reclamo de criptomonedas de prueba en las distintas faucets. En algunos usuarios, para la elección de Exchange, se les comentó que deberían de utilizar Metamask para enviar el pago. De esta forma simularían el retiro desde un Exchange ya que, con criptomonedas en redes de pruebas, no es posible probarlo desde una plataforma real.

Posteriormente, se les ha hecho la encuesta a través de Google Forms diseñada en las secciones anteriores para recopilar comentarios, mejoras y posibles errores. Se han obtenido los siguientes resultados:

Siete personas han completado el cuestionario, la edad de los usuarios se comprende entre los 20-28 años de edad y tienen al menos alguna experiencia previa con las criptomonedas.

La frecuencia con la que realizan pagos con criptomonedas es relativamente baja, ya que la gran mayoría se sitúan en los rangos de “mensualmente” o “raramente”. Esto se debe a que no suelen elegir la opción de pago con criptomonedas en las tiendas online, ya sea porque no está disponible o prefieren otra opción.

La billetera utilizada por los usuarios ha sido Metamask, mientras que los pagos suelen hacerse con Polygon por tener comisiones más bajas.

Solo un usuario tuvo problemas al conectar y sincronizar la billetera, se solucionó cambiando de navegador ya que este al parecer no era compatible o estaba desactualizado.

Otro usuario en el apartado de billeteras le fue confuso identificar la dirección para enviar criptomonedas, ya que en este caso no había porque se realiza a través de un Smart Contract, esto ocurrió por la poca experiencia que tenía el usuario con pagos a través de billeteras directamente.

Para la sección de Exchanges, los usuarios no encontraron errores.

A pesar de que los resultados han sido positivos, será importante probar la pasarela cuando esté completamente finalizada e integrada para que así pueda probarse con un entorno más real. De este modo, se obtendrán resultados más representativos.

11. Resultados

11.1. Producto final

El producto final se ha ido presentando a lo largo de la sección de implementación. En términos del total de interfaces que componen la pasarela de pagos a resultando en un total de 22 interfaces (sin contar las que son repetidas o tienen ligeros cambios). De ese número de interfaces, se ha logrado implementar 21, lo que se traduce en el 95,5%. A lo largo de la implementación se descartó crear un Stepper para los pagos mediante Smart Contract ya que era innecesario. En la siguiente figura, se destacan las interfaces implementadas con un borde naranja:

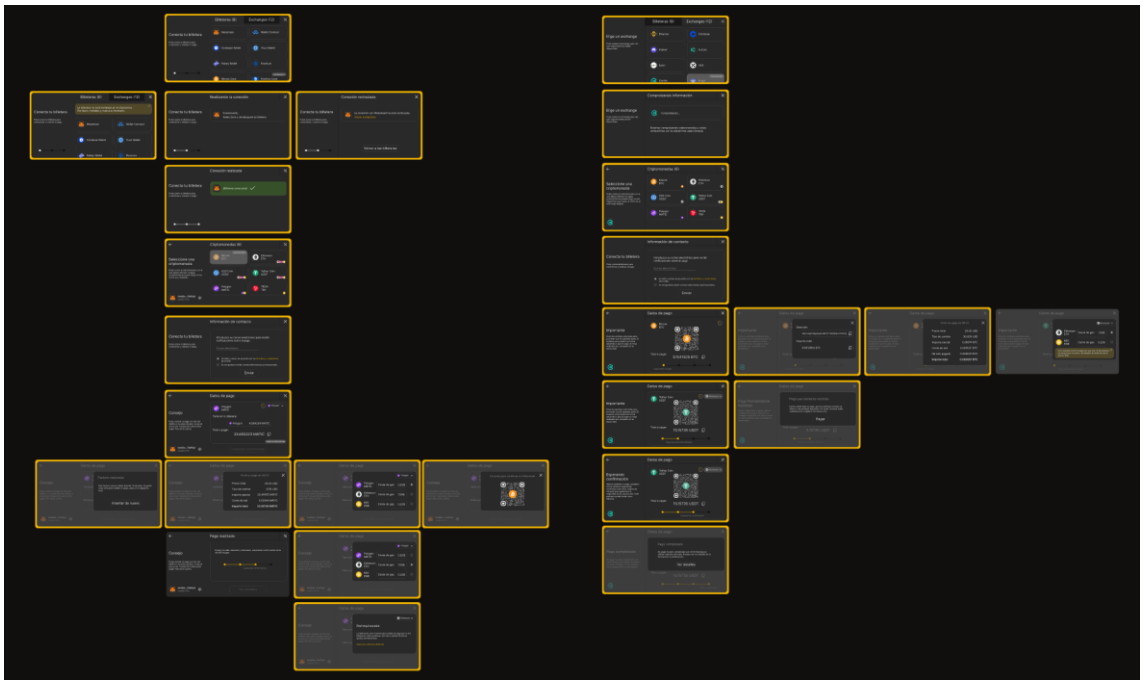


Figura 68. Interfaces implementadas bordes naranjas
(Fuente propia)

Si hablamos de la implementación de la pasarela de pago, se ha completado todo, a excepción de la funcionalidad para abrir en la App de Metamask y la elección de redes en la sección de exchanges.

El primero no se ha llevado a cabo porque, durante la implementación de esta parte, se observó que llevaría demasiado tiempo y la prioridad era completar los sprints previstos. Esto se debe a que se tendría que hacer un rediseño completo y adaptable para el navegador de la aplicación móvil de Metamask. A pesar de esto, al implementarse también la opción de pagos manuales

con exchanges, en esta sección se podría hacer un pago manualmente con Metamask sin interactuar directamente con la billetera y un Smart Contract.

El segundo, es la elección de redes en la sección de exchanges, por la escasez de tiempo, ya que se necesitaba buscar una API nueva por cada red y realizar comprobaciones en los métodos que leen los ficheros devueltos por estos servicios, ya que podría haber ligeros cambios. Sin embargo, el selector sería más sencillo de implementar que el realizado ya en la sección de billeteras, ya que incorporaba muchas conexiones para realizar el cambio de redes, interactuar con la billetera y, además, calcular el precio de las comisiones con distintas APIs.

A pesar de que finalmente no se han implementado estos dos puntos, la pasarela de pagos es funcional, se ha realizado el despliegue en un entorno de producción y admite tanto billeteras como exchanges. Es decir, en el primero interactuamos con aplicaciones web3 y Smart Contracts, mientras que en el segundo lo hacemos mediante APIs y lecturas de las cadenas de bloques correspondientes. Lo que permite comparar y ver el funcionamiento de dos formas distintas de hacer pagos con criptomonedas.

Cuando llegue el momento de defender el Trabajo de Fin de Grado, se realizará, además de una introducción y las conclusiones, una demostración de la pasarela de pagos. Esta demostración se realizará en dos partes: la primera utilizando Metamask y las redes de pruebas Goerli y Mumbai de Ethereum y Polygon respectivamente. Se elegirá una criptomoneda, se creará una factura y en la pantalla del pago se mostrará las funcionalidades del selector de redes antes de realizar la interacción con el Smart Contract para comprobar posteriormente en la blockchain el pago.

En segundo lugar, se realizará una prueba en el apartado de exchanges. Dado que estos no disponen de las criptomonedas ni las redes de pruebas, se utilizará también Metamask simulando un retiro desde un exchange, básicamente un pago de forma manual. Conforme el pago se vaya detectado, se mostrarán los mensajes del stepper.

11.2. Costes temporales

Para llevar un registro del tiempo dedicado al Trabajo de Fin de Grado, se ha utilizado la herramienta Clockfy. A continuación, se muestra la Figura 69 con un resumen de las horas invertidas cada mes desde su comienzo en febrero hasta julio.

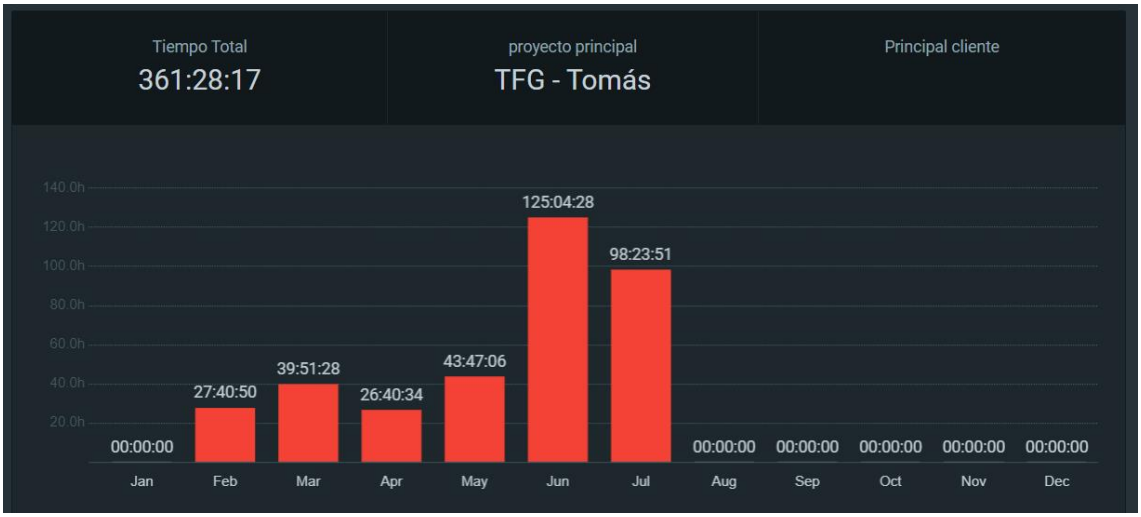


Figura 69. Tiempo en horas dedicadas al TFG
(Fuente propia)

Los apartados que más tiempo han necesitado han sido, por orden: la implementación, el diseño y el estado del arte con 227, 37 y 29 horas respectivamente. Esto supone más del 81% del total del proyecto.

Al inicio del TFG, cuando se propuso el proyecto de la pasarela de pagos con criptomonedas, no se tenía experiencia en la programación y funcionamiento profundo de los Smart Contract. Aquí se infravaloró el tiempo el esfuerzo requerido para llevar a cabo este proyecto. Como consecuencia, en las etapas de implementación fue necesario hacer pequeños ajustes en términos de interfaces y algunas funcionalidades secundarias.

Si se comparan las horas invertidas en el proyecto, sobre todo en las primeras fases de la memoria, ha coincidido extremadamente bien, sin embargo, a partir del diseño y concretamente la implementación, ha habido algunos problemas en los que se ha necesitado más tiempo del estimado por la complejidad, por lo que se debería de haber comenzado con un mayor margen de tiempo.

Por todo esto, no se ha logrado llegar a la fecha prevista de entrega, que era a medidos de mayo, por lo que se ha tenido que revisar y hacer un esfuerzo extra para la última convocatoria en julio. Finalmente, el proyecto ha necesitado un total de 360 horas, superando las 300 horas que,

generalmente, se considerarían suficientes para llevar a cabo un Trabajo de Fin de Grado, que equivale a 12 créditos.

11.3. Asignaturas relacionadas

Este proyecto ha sido influenciado significativamente por los conocimientos adquiridos durante el grado de Ingeniería Multimedia.

Para las primeras secciones del TFG y sobre todo para analizar el problema, elegir las metodologías a utilizar y el estudio de la viabilidad, han sido muy útiles los conocimientos aprendidos en la asignatura Análisis y Especificación de Sistemas Multimedia.

En segundo lugar, por las asignaturas de Diseño de Bases de Datos Multimedia y sobre todo en el ABP por la experiencia obtenida en las bases de datos NoSQL, donde se destaca MongoDB.

Por otra parte, en cuanto a la implementación, por las asignaturas de tercer curso como son Programación del Cliente Web y Desarrollo de Aplicaciones Web.

También, la asignatura de Usabilidad y Accesibilidad para cuidar la experiencia de usuario y hacer que el producto final sea más usable y accesible.

Finalmente, Servicios Multimedia Basados en Internet junto con Servicios Multimedia Avanzados para la seguridad y despliegue en un entorno de producción y el conocimiento adquirido sobre algunos framework, problemáticas y otros detalles de servicios multimedia que están siendo utilizado en la actualidad. En este caso sea utilizado React+Next.js con Node y no Angular como se utilizó en el ABP.

12. Conclusiones y trabajo futuro

Llegados a este punto, hay que hacer una reflexión sobre el trabajo realizado, objetivos conseguidos e incluso mejoras para un futuro. También se incluirán las impresiones personales.

12.1. Comprobación de objetivos

El objetivo principal de este proyecto consistía en la creación de una pasarela de pagos con criptomonedas sin intermediarios para un proyecto personal. El enfoque era en ofrecer una experiencia intuitiva para el usuario y que fomentara la confianza en el sistema.

Además de estos objetivos tangibles también había unos subjetivos más abstractos y personales como son la ampliación de conocimientos para la creación de la pasarela utilizando herramientas del sector web3 y Smart Contracts en distintas cadenas de bloques.

En relación al cumplimiento de los requisitos, tanto funcionales como no funcionales, se ha conseguido satisfacer la mayoría de ellos. A continuación, se detallan los logros alcanzados:

En cuanto a los requisitos funcionales, la pasarela de pagos permite realizar pagos con criptomonedas y elegir entre varias opciones disponibles además de seleccionar entre varias redes blockchain para una misma criptomoneda (Ethereum, BSC, Polygon…). Además, los usuarios pueden seleccionar una billetera con la que llevar a cabo el pago de la criptomoneda seleccionada y se ha logrado realizar el despliegue de la pasarela de pagos en el entorno de producción. A pesar de que no está finalizando algunas funcionalidades como permitir la descarga o el envío de la factura de compra, la infraestructura necesaria ya está en la base de datos.

Por otro lado, en relación a los requisitos no funcionales, se ha mantenido y cuidado el estilo de la aplicación y la facilidad de uso, se ha intentado seguir directrices de usabilidad y accesibilidad asimismo se informa al usuario de todas las secciones con la sección de la izquierda y con mensaje modales para guiarle.

12.2. Trabajo pendiente y posibles mejoras

Hay que reconocer que un proyecto nunca se termina por completo, siempre hay pequeñas mejoras o la posibilidad de añadir nuevas funcionalidades para incrementar el valor. En este

caso la pasarela se seguirá desarrollando ya que se utilizará para recibir pagos en el proyecto que se está desarrollando. Estas mejoras son:

- **Añadir nuevas redes y billeteras no compatibles con EVM** como son Solana, Polkadot o Cardano. A pesar de que el funcionamiento de estas criptomonedas es algo distinto, se podrían añadir sin modificaciones importantes.
- **Adaptar la pasarela de pagos para dispositivos móviles**, de forma que al pulsar sobre una billetera desde el navegador de un smartphone detecte si tiene la App instalada y continue el proceso desde el navegador de la propia aplicación de la billetera.
- **Incluir más criptomonedas y redes en la sección de Exchanges**, como podría ser Bitcoin o Litecoin, que a pesar de no funcionar con Smart Contracts y como está estructura esta sección se necesitaría una librería para generar claves públicas y privadas de billeteras y una API para leer la información de la cadena de bloques.
- **Enlazar la pasarela con la venta de productos en la web**, a pesar de que el proyecto no tiene un servicio o producto que ofrecer, es probable que en un futuro lo haya, por lo que la pasarela se integraría para poder aceptar criptomonedas como método de pago, ya que era el objetivo que se buscaba con el desarrollo.

12.3. Impresiones personales

Ha sido todo un desafío realizar este TFG, ya que no había trabajado en un proyecto de esta magnitud de forma individual, además de que no tenía conocimientos técnicos por falta de experiencia del funcionamiento de los pagos con criptomonedas. No obstante, me ha permitido lograr mi objetivo de añadir una pasarela de pagos con criptomonedas sin intermediarios a un proyecto propio, que, por el momento, tendrá que esperar hasta tener un MVP para poder ofrecer al público.

Los apartados o secciones que más dificultad representaron, fueron el diseño y desarrollo. El primero porque necesitaba conocer e investigar las aplicaciones web3 para ver que se estaba utilizando en el sector y que cualquier usuario con algo de experiencia no tuviera problemas al realizar un pago.

Si hablamos del desarrollo, he tenido que enfrentarme a una nueva tecnología como es React + Next.js, ya que tenía poca experiencia y es algo diferente a lo que había utilizado anteriormente durante el grado. Además, también durante el quinto sprint, tuve algunos problemas con buscar

soluciones y alternativas a las APIS y en la utilización de Smart Contract, ya que nunca antes me había parado a ver como funcionaban.

Para finalizar, debo decir que me siento extremamente orgulloso del resultado que se ha logrado. Cuando elegí este tema, muchos aspectos técnicos eran desconocidos para mí. De hecho, elegir este en particular fue en parte una decisión para salir de mi zona de confort y aprender nuevas tecnologías. A lo largo de este Trabajo de Fin de Grado, no sólo he podido demostrar los conocimientos que he adquirido a lo largo de la carrera, sino también he mejorado mis habilidades como ingeniero. Entre ellas, destacaría la capacidad para solucionar problemas.

13. Código fuente

Para permitir la visualización, descarga y utilización del código fuente de la pasarela desarrollada en este trabajo, se ha compartido públicamente en el repositorio de Git. El acceso a dicho repositorio se encuentra en el enlace proporcionado a continuación:

https://github.com/TFGTomas

Los diseños realizados en Figma se pueden ver en el siguiente enlace:

https://www.figma.com/file/fQWpg0vrQ662kectpHufbP/

Vídeo de la demo:

https://youtu.be/Ak3T8Ap4tJc

Referencias

[1] P. Soler, «La República Centroafricana se une a El Salvador y adopta el bitcoin como moneda de curso legal», *elconfidencial.com*, 27 de abril de 2022. https://www.elconfidencial.com/mercados/2022-04-27/la-republica-centroafricana-se-une-a-el-salvador-y-adopta-el-bitcoin-como-moneda-legal_3415477/ (accedido 6 de julio de 2023).

[2] V. Zapata, «El número de usuarios de criptomonedas aumentó 40% en 2022, según Crypto.com», *BeInCrypto*, 20 de enero de 2023. https://es.beincrypto.com/numero-usuarios-criptomonedas-aumento-40-2022-segun-crypto-com/ (accedido 6 de julio de 2023).

[3] «La plataforma de criptomonedas FTX se declara en bancarrota y amenaza con provocar un efecto contagio | Economía | EL PAÍS». https://elpais.com/economia/2022-11-11/la-plataforma-de-criptomonedas-ftx-se-declara-en-bancarrota-y-amenaza-con-provocar-un-efecto-contagio.html (accedido 6 de julio de 2023).

[4] «DefiLlama», *DefiLlama*. https://defillama.com/chains (accedido 6 de julio de 2023).

[5] «What cryptocurrencies are supported for payments?», *NOWPayments*. https://nowpayments.io/supported-coins (accedido 6 de julio de 2023).

[6] «Billetera Multi Criptomoneda | Billetera Multi Coin | Cripto Billetera», *Trust Wallet*. https://trustwallet.com/es/assets/ (accedido 6 de julio de 2023).

[7] N. Admin, «Beneficios, primeros pasos y SEO con NEXT.js», *Nubaltic*, 14 de enero de 2022. https://nubaltic.com/beneficis-seo-nextjs/ (accedido 6 de julio de 2023).

[8] «Nginx vs Apache: Lucha entre Servidores Web», *Kinsta®*, 3 de julio de 2019. https://kinsta.com/es/blog/nginx-vs-apache/ (accedido 6 de julio de 2023).

[9] «La API de criptomonedas más completa», *CoinGecko*. https://www.coingecko.com/en/api (accedido 6 de julio de 2023).

[10] S. Albares, «Objetivos SMART: Define tus metas sin cometer errores», *BlogsterApp*, 13 de mayo de 2019. https://blogsterapp.com/es/objetivos-smart-definir-metas/ (accedido 6 de julio de 2023).

[11] «Proceso unificado», *Wikipedia, la enciclopedia libre*. 6 de octubre de 2022. Accedido: 6 de julio de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Proceso_unificado&oldid=146409957

[12] «Qué es SCRUM», *Proyectos Ágiles*, 4 de agosto de 2008. https://proyectosagiles.org/que-es-scrum/ (accedido 6 de julio de 2023).

[13] «Manage Your Team's Projects From Anywhere | Trello». https://trello.com/ (accedido 6 de julio de 2023).

[14] COING, «Clockify - Time Tracking Software». https://clockify.me (accedido 6 de julio de 2023).

[15] «ieee830.pdf». Accedido: 6 de julio de 2023. [En línea]. Disponible en: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

[16] «Conceptos Generales de MongoDB | ramoncarrasco.es». https://www.ramoncarrasco.es/es/content/es/kb/128/conceptos-generales-de-mongodb (accedido 6 de julio de 2023).

[17] «Web3.js vs Ether.js: ¿Cuál es la mejor opción para interactuar con la blockchain de Ethereum?» https://es.linkedin.com/pulse/web3js-vs-etherjs-cu%C3%A1l-es-la-mejor-opci%C3%B3n-para-con-de-bravo-cuero (accedido 6 de julio de 2023).

[18] «Alchemy - the web3 development platform». https://www.alchemy.com/ (accedido 6 de julio de 2023).

[19] «Ethereum API | IPFS API & Gateway | ETH Nodes as a Service». https://www.infura.io/ (accedido 6 de julio de 2023).

[20] «Qué es el customer journey map y para qué sirve», *Qualtrics*. https://www.qualtrics.com/es/gestion-de-la-experiencia/cliente/customer-journey-map/ (accedido 6 de julio de 2023).
[21] «Tipografía serif vs sans serif para principiantes | Adobe». https://www.adobe.com/es/creativecloud/design/discover/serif-vs-sans-serif.html (accedido 6 de julio de 2023).
[22] «Browse Fonts - Google Fonts». https://fonts.google.com/ (accedido 6 de julio de 2023).
[23] S. Herranz, «Wireframe, Mockup y Prototipos: en busca de sus diferencias», *Rocket Studio UX*, 9 de marzo de 2018. https://medium.com/rocket-studio-ux/wireframe-mockup-y-prototipos-en-busca-de-sus-diferencias-23a03bcbdb69 (accedido 6 de julio de 2023).
[24] «Balsamiq Cloud». https://balsamiq.cloud/ (accedido 6 de julio de 2023).
[25] «Figma: The Collaborative Interface Design Tool», *Figma*. https://www.figma.com/ (accedido 6 de julio de 2023).
[26] D. / L. J. K. Koopsen Max, «Desarrolladores de Ethereum Quieren Dejar Que la Testnet Goerli Muera Lentamente», *Decrypt*, 27 de febrero de 2023. https://decrypt.co/es/122270/desarrolladores-de-ethereum-quieren-dejar-que-la-testnet-goerli-muera-lentamente/ (accedido 6 de julio de 2023).
[27] G. González, «¿Qué es un faucet de bitcoin y otras criptomonedas?», *CriptoNoticias - Noticias de Bitcoin, Ethereum y criptomonedas*, 24 de junio de 2022. https://www.criptonoticias.com/criptopedia/que-es-faucet-bitcoin-otras-criptomonedas/ (accedido 6 de julio de 2023).
[28] «Formularios de Google». https://docs.google.com/forms/u/0/ (accedido 6 de julio de 2023).
[29] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido 6 de julio de 2023).
[30] «Build software better, together», *GitHub*. https://github.com (accedido 6 de julio de 2023).
[31] «Swap | Sushi». https://app.sushi.com/en/swap (accedido 6 de julio de 2023).
[32] «Home | PancakeSwap». https://pancakeswap.finance/ (accedido 6 de julio de 2023).
[33] «wagmi: React Hooks for Ethereum». https://wagmi.sh/ (accedido 6 de julio de 2023).
[34] «viem · TypeScript Interface for Ethereum». https://viem.sh (accedido 6 de julio de 2023).
[35] «Solidity — Solidity 0.8.20 documentation». https://docs.soliditylang.org/en/v0.8.20/ (accedido 6 de julio de 2023).
[36] «Hardhat | Ethereum development environment for professionals by Nomic Foundation». https://hardhat.org (accedido 6 de julio de 2023).
[37] «Goerli Faucet», *Goerli Faucet*. https://goerlifaucet.com/ (accedido 6 de julio de 2023).
[38] «Remix - Ethereum IDE». https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null (accedido 6 de julio de 2023).
[39] «Especificación de Application Binary Interface — documentación de Solidity - UNKNOWN». https://solidity-es.readthedocs.io/es/latest/abi-spec.html (accedido 6 de julio de 2023).
[40] «polygon Gas Tracker API - Owlracle». https://owlracle.info/poly (accedido 6 de julio de 2023).
[41] «Custom QR Code with Logo API Documentation (qrcode-monkey)», *RapidAPI*. https://rapidapi.com/qrcode-monkey/api/custom-qr-code-with-logo/ (accedido 6 de julio de 2023).
[42] «Etherscan APIs- Ethereum (ETH) API Provider». https://etherscan.io/apis (accedido 6 de julio de 2023).
[43] «Digital Ocean: qué es, cómo usar y qué planes tiene la herramienta», *Rock Content - ES*, 22 de abril de 2020. https://rockcontent.com/es/blog/digital-ocean/ (accedido 13 de julio de 2023).