

IMF SMART EDUCATION

Trabajo Fin Grado **Aplicación Web en Blockchain**



Grado Superior en Desarrollo de Aplicaciones Multiplataforma

Autor: Alejandro Vindel Ares

Tutor: Carlos Rufiángel García

Junio 2022

DEDICATORIA

Me gustaría dedicar este TFG a toda mi familia por estar siempre a mi lado y apoyarme en todas las circunstancias que se han presentado y que se presentarán.

ÍNDICES

Tabla de contenido

RESUMEN	6
MOTIVACIÓN.....	8
INTRODUCCIÓN.....	9
OBJETIVOS.....	10
Diagrama RFTP del proyecto:.....	11
DESCRIPCIÓN.....	13
Diagrama de casos de uso.....	13
Diagrama caso de uso, usuario no conectado.	13
Diagrama de caso de uso, usuario conectado.	15
DISEÑO	18
Diagrama de clases del proyecto.	18
Diagrama de clases de la aplicación:.....	18
Diagrama de clases del entorno blockchain.	20
Diagrama de clases de la aplicación.....	21
Diagrama de sistemas.	22
Aplicación web - Aplicación blockchain.	22
Aplicación blockchain – Red blockchain.	22
Aplicación web – API Bing News Search.	22
Aplicación web – API CoinRanking.	22
Aplicación web – API Giphy.....	23
DESARROLLO DEL PROYECTO.....	24
Preparación del entorno.	24
1. Diseño de interfaz de la aplicación web.	24
1.1. Explicación de diseño de la aplicación web.	25
1.1.1. Clase Navbar.	25
1.1.2. Clase Welcome.....	26
1.1.3. Clase Estadísticas.	27
1.1.4. Clase News.	28
1.1.5. Clase services.	29
1.1.6. Clase Transactions.....	30
1.1.7. Clase Footer.	30
2. Desarrollo del entorno blockchain.....	31
2.1. Testeo y deploy del Smart contract.	32

2.2.	Implementación del entorno blockchain en la aplicación web.	33
2.3.	Realización de transacciones y completo funcionamiento aplicación-blockchain.	36
3.	Implementación de blockchain en aplicación web.	45
	Formulario de envío de transacciones.	45
	Muestra por pantalla de las transacciones del usuario.	47
	Introducción de sistema de Gifs en la aplicación.	48
4.	Implementación de APIs en la aplicación.	50
4.1.	Implementación de CoinRanking.	50
4.2.	Implementación de Bing News.	57
4.3.	Almacenamiento de los datos de las APIs.	62
5.	Testeo de la aplicación.	63
5.1.	Testeo de la conexión con MetaMask.	63
5.2.	Testeo de transacción.	63
5.3.	Testeo de APIs.	67
5.4.	Testeo de API Giphy.	68
TECNOLOGÍA	69
	Tecnologías utilizadas para el desarrollo de la aplicación:	69
	Tecnologías utilizadas para el desarrollo del entorno blockchain:	69
	Lista de dependencias NodeJs utilizadas:	69
	Tecnologías utilizadas para el desarrollo y testeo del proyecto:	70
	Herramientas utilizadas para el desarrollo del proyecto.	70
	APIs implementadas:	71
METODOLOGÍA	72
	Metodología usada en el proyecto.	72
	Diagrama de Gantt.	72
TRABAJOS FUTUROS	74
CONCLUSIONES	75
REFERENCIAS	76

RESUMEN

Este proyecto consiste en el desarrollo y diseño de una aplicación de transacciones de criptomonedas. El objetivo realizar esta acción de una manera fácil y eficaz el usuario pueda realizar transacciones con la menor complejidad posible por lo cual la aplicación no recoge ningún dato del usuario.

La aplicación dispone de una página web donde el usuario podrá realizar los movimientos necesarios, dispone de un apartado donde podrá revisar todas sus transacciones y movimientos y apartados en los cuales puede ver las últimas novedades, valores de criptomonedas y estadísticas a tiempo real.

La aplicación está subdividida en varios sistemas: interfaz de la aplicación web, desarrollo del entorno blockchain.

A diferencia de otras aplicaciones, esta aplicación busca hacer más simple trabajar/utilizar con estas últimas tecnologías y ofrecer al usuario la cantidad de información necesaria.

Durante años, la elección por parte de los desarrolladores para elaborar una herramienta informática, han sido las aplicaciones de escritorio, pero durante los últimos años, se ha ido avanzando hacia las aplicaciones web, debido a la versatilidad que proporcionan los nuevos lenguajes de programación y la mejora de la conectividad.

La aplicación está desarrollada en una aplicación web desarrollada en React, Css, JavaScript en el frontend, y en Solidity y JavaScript para el backend.

A lo largo de esta memoria, se irá destallando desde la explicación de la necesidad de realizar este desarrollo, pasando por la fase de diseño, desarrollo y evaluación del software desarrollado en este Trabajo Fin de Grado.

Palabras Clave

Criptomonedas, Aplicación Web, Transacciones.

Abstract

This project consists of the development and design of a cryptocurrency transaction application. The objective is that the user can carry out transactions with the least possible complexity in an easy and efficient way, for which the application does not collect any data from the user.

The application has a web page where the user can make the necessary movements, it has a section where they can review all their transactions and movements and sections in which they can see the latest news, cryptocurrency values and statistics in real time.

The application is subdivided into several systems: web application interface, development of the blockchain environment.

Unlike other applications, this application seeks to make it easier to work/use with these latest technologies and to offer the user the necessary amount of information.

During all these years, the choice made by developers to develop a computer tool has been desktop applications, but in recent years, there has been progress towards web applications, due to the versatility provided by new programming languages and improving connectivity.

The application is developed in a web application developed in React, Css, JavaScript in the frontend, and in Solidity and JavaScript for the backend.

Throughout this memory, it will be detailed from the explanation of the need to carry out his development, going through the phase of design, development and evaluation of the software developed in this Final Degree Project.

Key words

Cryptocurrencies, Web Application, Transactions.

MOTIVACIÓN

Este proyecto nació con la idea de aprender nuevas tecnologías e implementarlas de una manera eficaz. En la realización de este proyecto no solo se ha buscado la perfección a la hora de realizar el proyecto, sino que se ha buscado el aprendizaje de nuevas tecnologías mientras se ha realizado el mismo. Nace tras ver muchas páginas de criptomonedas y la cantidad de datos personales que solicitan y lo complicado que puede ser realizar hasta la operación más básica. En este proyecto se trabaja de una manera fácil con transacciones de Ethereum(criptomonedas) introduciendo tan solo datos en un formulario con los datos precisos y necesarios.

El ecosistema de las criptomonedas es un entorno muy volátil y sujeto a especulaciones, en este proyecto se ha incorporado tecnologías para que el usuario pueda estar en las últimas novedades, últimos valores y pueda informarse correctamente.

Se han incorporado de manera efectiva el trabajo con APIs externas a la aplicación, tecnologías que son de gran uso y para recopilar una gran cantidad de datos. Este proyecto recoge datos de dos APIs y es capaz de mostrarle al usuario los últimos datos, noticias cada vez que acceda a la aplicación web.

Desde hace unos cuantos años llegaron estas nuevas tecnologías y están en auge, pero de cierta manera son desconocidas y no se adquiere el conocimiento necesario para poder trabajar con ellas si no te dispones a realizar un proyecto extenso. Este proyecto busca ofrecer al usuario una manera sencilla de interactuar en el mundo de las criptomonedas.

Este tipo de aplicaciones por lo general suele pedir al usuario que introduzca una gran cantidad de datos personales, este proyecto se ha realizado con la visión de que el usuario posee su manera de conectarse con MetaMask, lo cual es bastante fácil y sin necesidad de introducir una gran cantidad de datos personales. La aplicación ofrece al usuario una manera sencilla para conectarse y para realizar cambios de cuentas. Visualizar el saldo de su cartera y también podrá ver las interacciones con el Smart contract y las comisiones que cobran al realizar una transacción.

El diseño de las interfaces de usuario se ha diseñado buscando un diseño bonito e intuitivo. Se ha buscado un diseño moderno pensando en el ecosistema de las criptomonedas.

INTRODUCCIÓN

Este proyecto ofrece un abanico de opciones al usuario final. De manera que si el usuario desea realizar transacciones o interactuar con sus criptomonedas será capaz de realizar transacciones de una manera rápida y eficaz.

El usuario que no desee realizar transacciones en ese momento podrá utilizar la aplicación de una manera informativa ya que ofrece muchos datos sobre las novedades actuales, valores y noticias del mundo de las criptomonedas.

Uno de los mayores problemas que suelen tener los usuarios con las páginas de criptomonedas es la cantidad de datos que se le pide al usuario. Teniendo que enviar todos los datos personales, incluyendo documento de identidad, documentos de residencia, etc. Esta aplicación ofrece al usuario realizar transacciones con sus criptomonedas sin necesidad de tener que introducir ningún dato personal, esto es gracias a que la aplicación se apoya en MetaMask. MetaMask es una plataforma que se ha implementado en el proyecto que permite a los usuarios mediante su página web o la extensión de Google crear carteras de criptomonedas y ver su saldo, etc. Esta aplicación trabaja con MetaMask como portal para poder realizar transacciones mediante el contrato inteligente desarrollado.

Las principales funciones de este proyecto podríamos definir las como:

Realizar transacciones de criptomonedas de una manera rápida y eficaz. Una aplicación con un diseño intuitivo, elegante y poco saturada. Facilidad para ofrecer al usuario las herramientas necesarias para estar siempre atento en las últimas novedades, cambios. Este último apartado es importante ya que hay mucho desconocimiento sobre el mundo de las criptomonedas y ofrecer al usuario una lista de noticias y datos de último momento le ofrece nuevas oportunidades.

La aplicación trabaja sobre una red de pruebas de Ethereum la cual se llama red de pruebas Ropsten. Tras realizar un gran número de transacciones y ver que el Smart contract es eficaz y no nos proporciona una gran cantidad de comisión por operaciones, la aplicación es funcional en una red de blockchain totalmente operativa y ser funcional para todo tipo de personas.

Este proyecto está dirigido a cualquier tipo de usuario que interactúe con criptomonedas. El usuario no tiene por qué ser un experto en el mundo para poder realizar transacciones.

OBJETIVOS

Objetivos principales del proyecto:

Objetivo 1	Realización de transacciones a través de la página.
Tipo	Obligatorio.
Descripción	El proyecto debe de ser capaz de realizar transacciones con la criptomoneda Ethereum, creando el correspondiente Smart Contract e implementando MetaMask como medio para que ese contrato se suba a la red de Ethereum.

Objetivo 2	Diseño y desarrollo del entorno blockchain.
Tipo	Obligatorio
Descripción	El proyecto a la hora de realizar las transacciones lo realiza mediante un Smart contract diseñado y desarrollado.

Objetivo 3	Integrar la aplicación en una red de Ethereum
Tipo	Obligatorio
Descripción	La aplicación a la hora de realizar las transacciones deberá de trabajar sobre una red de pruebas.

Objetivo 4	Importación de datos a la aplicación mediante APIs.
Tipo	Opcional
Descripción	El proyecto debe de ser capaz de coger datos en tiempo real de APIs, de manera que se pueda mostrar lo último en cuanto a noticias y datos que se muestren en la página.

Objetivo 4	Desarrollo y diseño de la aplicación web.
Tipo	Obligatorio
Descripción	El proyecto deberá de contener una interfaz gráfica de usuario funcional y con buen diseño.

Diagrama RFTP del proyecto:

R01- Interfaz básica del usuario.

- R01F1- Diseño y desarrollo de la interfaz.
 - R01F1T1- Botón de conectar wallet del usuario.
 - R01F1T2- Validación de si el usuario ha conectado la wallet.
- R01F2- Implementación API Giphy.
 - R01F2T1- Implementación de gifs al mostrar las transacciones.
- R01F3- Formulario de transacciones.
 - R01F3T1- Validación del formulario.
 - R01F3T2- Animación de “cargando” mientras se realiza la transacción.
- R01F4- Validación de datos del usuario.
 - R01F4T1- Mostrar transacciones del usuario.
 - R01F4T2- Indicar al usuario que debe de conectar su cuenta para obtener las transacciones.

R02- Desarrollo entorno blockchain.

- R02F1- Diseño y desarrollo de smart contract.
- R02F2- Implementación de entorno blockchain en aplicación.
 - R02F2T1- Testeo de funciones blockchain desde la interfaz de usuario.
- R02F3- Testeo de smart contract.
 - R02F3T1- Deploy del smart contract en la red de Ethereum.
- R02F4- Trabajo sobre red de testeo Ropsten.
 - R02F4T1- Implementación del smart contract en una red de testeo.

R03- Interfaz de usuario en el entorno de blockchain.

- R02F1- Estadísticas de la criptomoneda.
 - R02F2T1- Implementación API CoinRanking.
 - R02F2T2- Estadísticas criptomonedas.
 - R02F2T3- Top criptomonedas ordenado en cuanto a su valor.
- R02F2- Transacciones de criptomoneda.
 - R03F3T1- Transacciones con Ethereum, comisiones.
 - R03F3T2- Transacción a cuentas indicada por el usuario.
 - R03F3T3- Testeo de las transacciones.
- R02F4- Implementación de la plataforma MetaMask.
 - R02F4T1- Asociación de la wallet.
 - R02F4T2- Visualización de comisiones, costes, notificación del estado de las notificaciones.
- R02F5- Visualizar valores de criptomonedas.

R04- Trabajo con APIs.

- R03F1- API CoinRanking.
 - R03F1T1- Importación de la API y realizar peticiones.
 - R03F1T2- Tratado de los datos que se recogen de la API.
- R03F2- API Bing Search.
 - R03F2T1- Importación de la API y realizar peticiones.
 - R03F2T2- Tratado de los datos que se recogen de la API.

R05- Datos de Criptomonedas.

- R04F1- Mostrar los últimos datos de valores pedidos al momento.
- R04F2- Mostrar las criptomonedas con mayor valor y mostrarlas.

R06- Interfaz de visualización de noticias.

- R05F1- API Bing News Search.
- R05F2- Visualización de las últimas novedades.
- R05F3- Acceso a las noticias.

R07- Soporte de ayuda.

- R06F1- Enviar correos de ayuda al servicio técnico.
- R06F1- Aplicación subida a un host online.

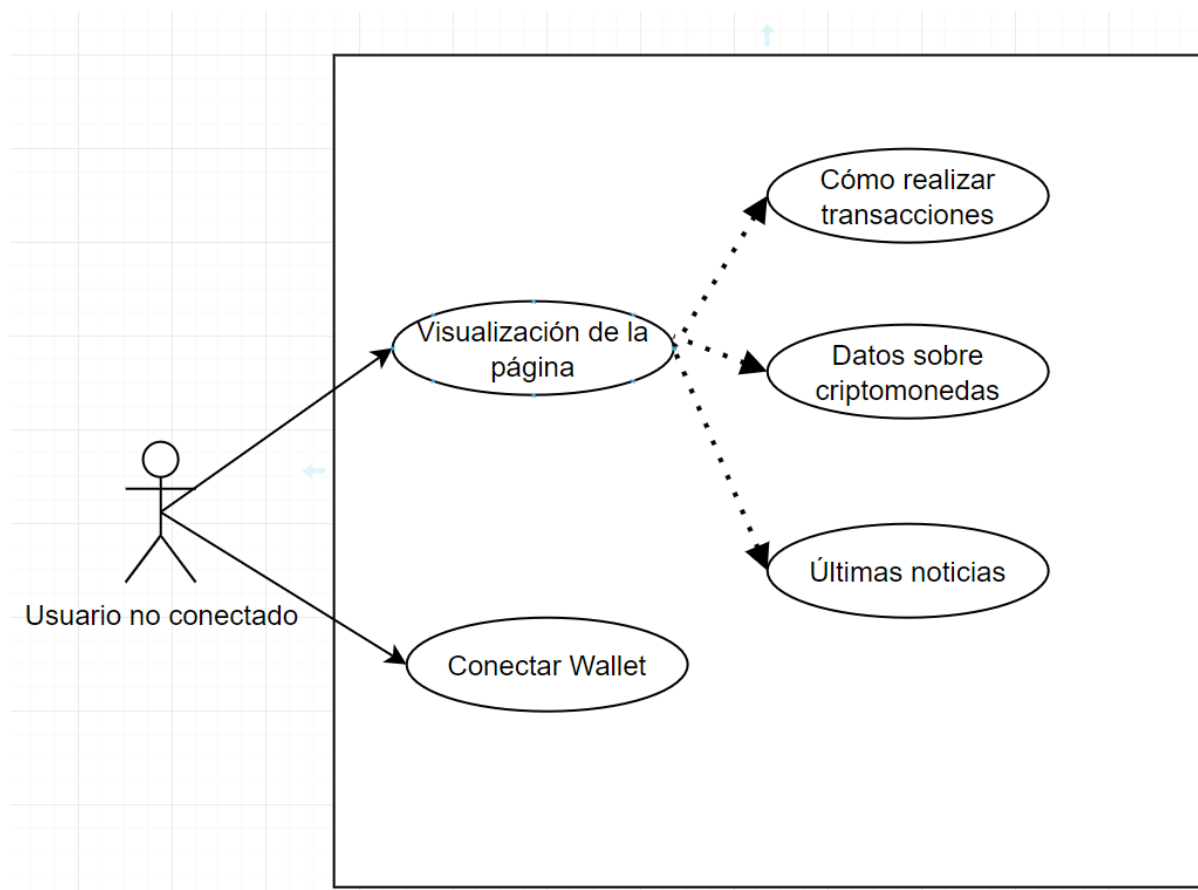
DESCRIPCIÓN

Diagrama de casos de uso.

Actores. Este proyecto cuenta con dos tipos de usuarios finales, el usuario que conecta su cuenta para poder realizar transacciones y el usuario que solo visualiza la página.

Diagrama caso de uso, usuario no conectado.

En la siguiente figura se muestra la interacción del usuario a la hora de interactuar con la aplicación y las posibilidades que esta le ofrece.



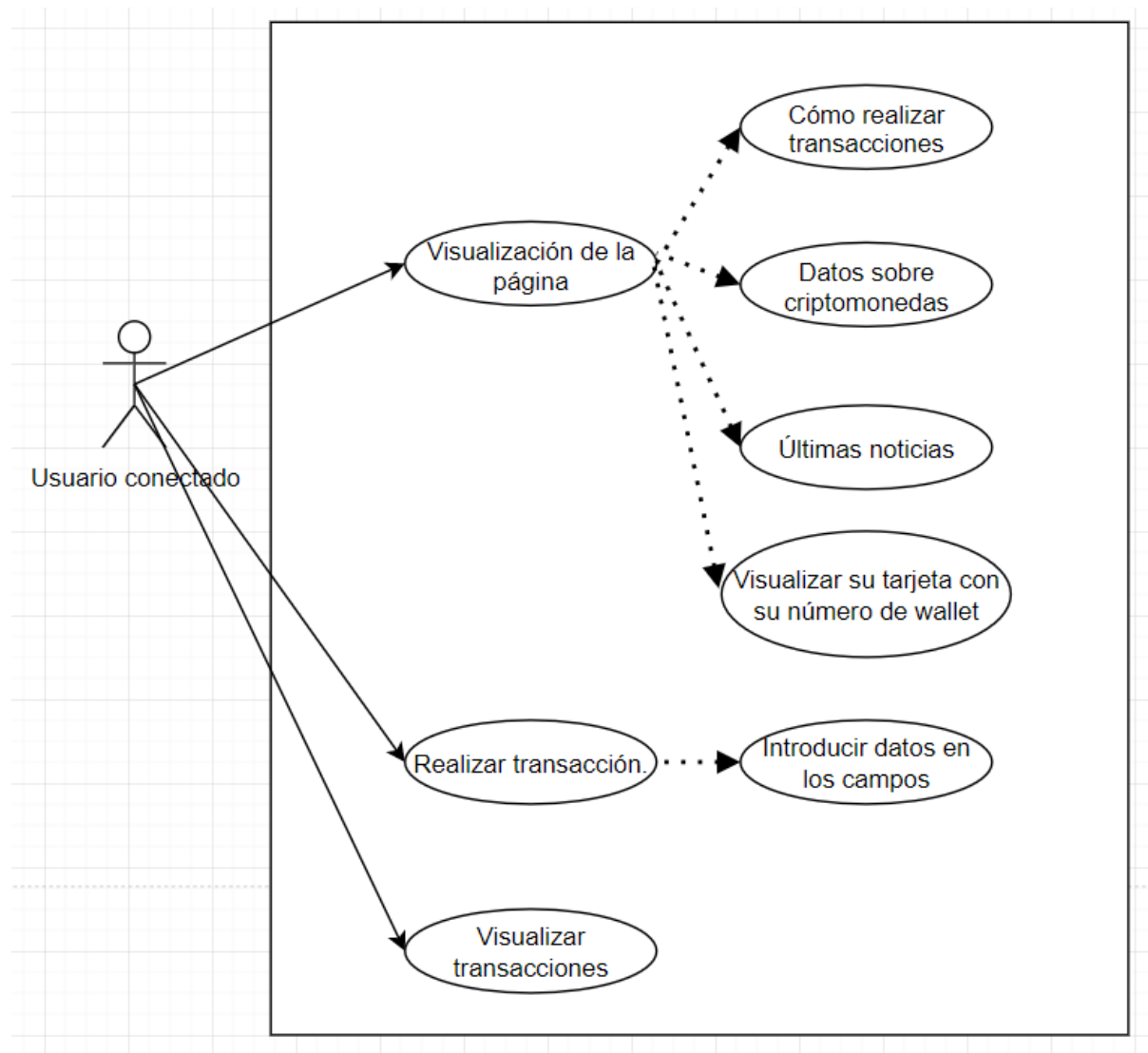
Como podemos observar en la imagen anterior, el usuario tiene la posibilidad de visualizar la aplicación completa y la posibilidad de poder conectar su wallet (posibilidad de convertirse en un usuario conectado y capaz de realizar y ver sus transacciones). Podemos observar las funciones directas de la aplicación (líneas continuas) y a su vez las que se realizan de una manera interna pero que pertenecen a la misma (líneas discontinuas). En la siguiente tabla podremos observar las precondiciones, postcondiciones, los datos de entrada (los

cuáles serán pedidos a APIs externas ya sea CoinRanking o Bing News), los datos de salida, las clases con las que interactúa y las interfaces(entenderemos como interfaz el frontend, entenderemos como clase el contenido backend aparte de contener contenido visual en alguna de ellas).

Precondiciones: <ul style="list-style-type: none"> - Entrar a la página web. 	Postcondiciones: <ul style="list-style-type: none"> - Conectar wallet.
Datos de entrada: API CoinRanking, estadísticas: <ul style="list-style-type: none"> - Total de criptomonedas. - Total de exchanges. - Market Cup total. - Total en 24 horas. - Mercados totales. API CoinRanking, criptomonedas(datos que se recogen por cada criptomoneda): <ul style="list-style-type: none"> - Id. - Nombre. - Precio. - URL del logo. - Market Cup. - Change. API Bing News, datos que se recogen por cada noticia: <ul style="list-style-type: none"> - Nombre. - URL de la imagen de la portada. - Descripción. - URL de la imagen del proveedor de la noticia. - Nombre del proveedor. - URL de la noticia. 	Datos de salida:
Interfaces: Welcome.jsx Estadísticas.jsx News.jsx Navbar.jsx Loader.jsx Services.jsx Transactions.jsx	Clases: Main.jsx App.jsx Index.js CryptoApi.js CryptoNewsApi.js

Diagrama de caso de uso, usuario conectado.

En la siguiente figura se puede observar la interacción completa del usuario con la aplicación final, el usuario tendrá la posibilidad de poder realizar transacciones y además de poder visualizar las transacciones previas que se hayan realizado.



Como podemos observar en la figura anterior el usuario tiene todas las posibilidades que ofrece la aplicación. El usuario tiene la posibilidad de visualizar la página completa y si ha realizado alguna transacción podrá ver el registro total de cada una. El usuario podrá realizar transacciones rellenando los campos necesarios y mediante el entorno de MetaMask podrá observar las comisiones y costes que genera el mismo(la transacción se realiza a partir de un contrato realizado). Entenderemos como interfaz las clases frontend, y

entenderemos como clases las que aportan funcionalidad y el trabajo backend de la aplicación.

<p>Precondiciones:</p> <ul style="list-style-type: none"> - Entrar a la página web. - Conectar Wallet 	<p>Postcondiciones:</p> <ul style="list-style-type: none"> - Realizar transacción.
<p>Datos de entrada:</p> <p>Wallet del usuario(datos que se recogerán de la wallet del usuario):</p> <ul style="list-style-type: none"> - Dirección. - Red a la que esté conectado el usuario. <p>Transacciones del usuario(datos de las transacciones previas del usuario que se recogen):</p> <ul style="list-style-type: none"> - Dirección de usuario. - Dirección de destino. - Cantidad. - Palabra clave. - Mensaje. - Fecha de la transacción. - Comisión. - Tiempo estimado que puede tardar en realizarse la transacción. <p>API CoinRanking, estadísticas:</p> <ul style="list-style-type: none"> - Total de criptomonedas. - Total de exchanges. - Market Cup total. - Total en 24 horas. - Mercados totales. <p>API CoinRanking, criptomonedas(datos que se recogen por cada criptomoneda):</p> <ul style="list-style-type: none"> - Id. - Nombre. - Precio. - URL del logo. - Market Cup. - Change. 	<p>Datos de salida:</p> <p>Realización de una transacción:</p> <ul style="list-style-type: none"> - Dirección de destino. - Cantidad. - Palabra clave. - Mensaje.

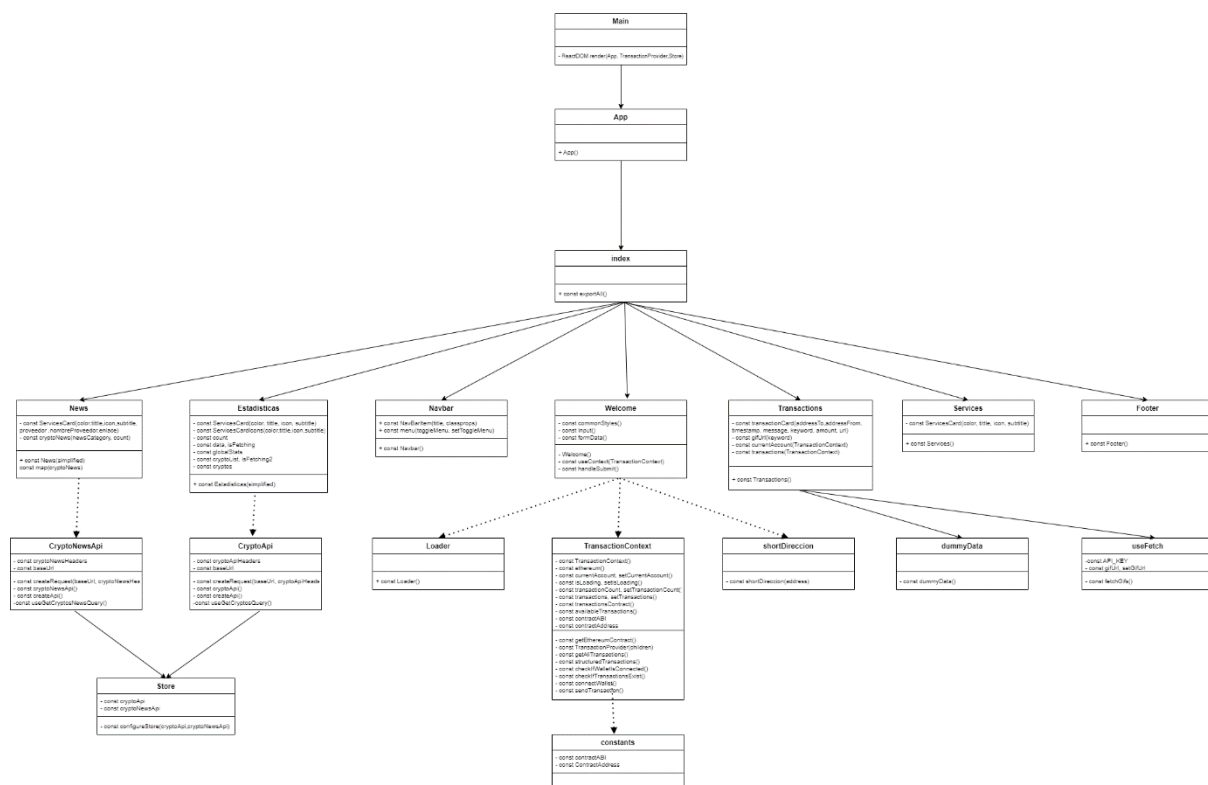
<p>API Bing News, datos que se recogen por cada noticia:</p> <ul style="list-style-type: none"> - Nombre. - URL de la imagen de la portada. - Descripción. - URL de la imagen del proveedor de la noticia. - Nombre del proveedor. - URL de la noticia. 	
<p>Interfaces:</p> <p>Welcome.jsx</p> <p>Estadísticas.jsx</p> <p>News.jsx</p> <p>Navbar.jsx</p> <p>Loader.jsx</p> <p>Services.jsx</p> <p>Transactions.jsx</p> <p>Footer.jsx</p> <p>Interfaz visual de MetaMask para confirmar la transacción.</p>	<p>Clases:</p> <p>Main.jsx</p> <p>App.jsx</p> <p>Index.js</p> <p>CryptoApi.js</p> <p>CryptoNewsApi.js</p> <p>useFetch.jsx</p> <p>store.js</p> <p>constants.js</p> <p>dummyData.js</p> <p>shortDireccion.js</p> <p>Transactions.json</p> <p>Transactions.sol</p>

DISEÑO

Diagrama de clases del proyecto.

Para la creación de la aplicación se ha necesitado un total de 21 clases. Las cuales se dividirán en dos grupos, primero las clases de la aplicación, segundo las clases del entorno blockchain y por último podremos ver como se unen los dos entornos de manera que la aplicación interactúa con el entorno blockchain. Cada una con su explicación y su funcionamiento. En este caso indicaremos las clases del proyecto con los atributos que pueda contener cada una de ellas y sus métodos. Las líneas continuas indicarán las clases que tienen una relación directa y las líneas discontinuas indicarán las clases que solo se utilizarán a la hora de realizar operaciones con la aplicación.

Diagrama de clases de la aplicación:

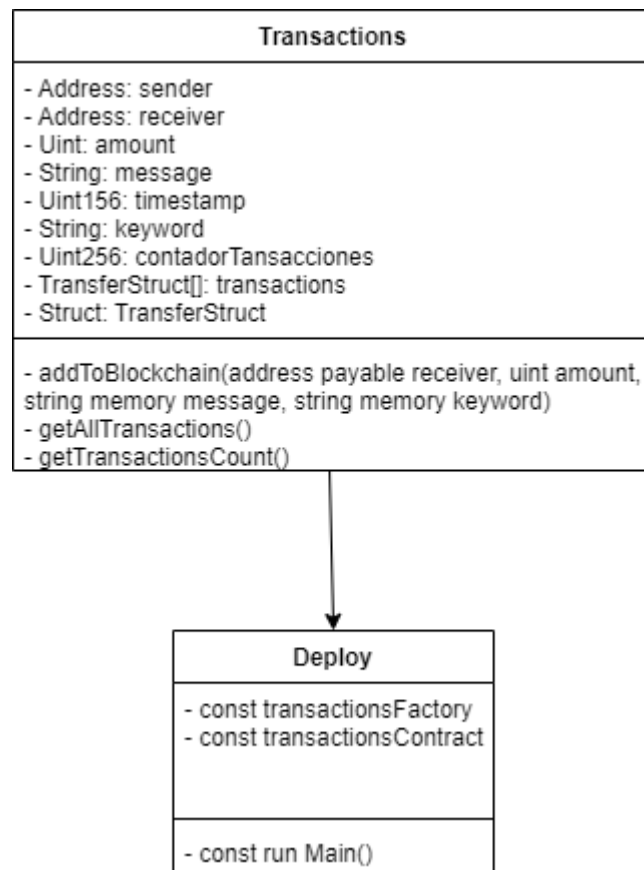


- **Main:** Ejecuta la aplicación.
- **App:** Importa todas las interfaces que se muestran en la aplicación. Envía al main lo que debe de mostrar a la hora de cargar la página.

- **Navbar:** Barra de navegación superior de la aplicación. Desarrollada y diseñada de una manera responsive. Será accesible desde dispositivos pequeños como grandes.
- **Welcome:** Apartado de inicio de la aplicación, desde donde se pueden realizar transacciones (por lo cual es la clase de la aplicación que se encarga de recoger los datos para realizarlas).
- **Loader:** Clase que se utilizará cada vez que la aplicación esté cargando o solicitando unos datos (animación de spinner indicando al usuario de que está cargando).
- **TransactionContext:** Clase que se encarga de establecer un puente entre la aplicación web y el entorno blockchain. Esta clase se encarga de interactuar a la hora de pedir datos al blockchain y realizar transacciones.
- **Constants:** Recoge los datos que se generan por primera vez al compilar el Smart contract de blockchain.
- **ShortDireccion:** Clase que se encarga de coger la dirección de la cartera del usuario y reducirla en tamaño para mostrarla en las transacciones y en la tarjeta.
- **Footer:** Footer de la aplicación. Contiene datos sobre la aplicación y manera de contactar.
- **Services:** Clase de contenido visual en la cual se muestran los servicios que ofrece la aplicación.
- **Estadísticas:** Su función es recoger los datos de CryptoApi y mostrarlos por pantalla. Se muestran los últimos valores de las criptomonedas y cómo está el mercado. Muestra un top de cinco criptomonedas con sus valores correspondientes.
- **CryptoApi:** Su función es hacer una petición a la API CoinRanking y recoger los datos. Datos que más tarde se mostrarán en la página mediante la clase Estadísticas.
- **News:** Su función es recoger los datos de CryptoNewsApi, tratar los datos recibidos de la misma y mostrar los datos deseados al usuario.
- **CryptoNewsApi:** Realiza una petición a la Api Bing Search y recopila los datos. Datos que se mostrarán mediante la clase News.
- **Transactions:** Valida y si dispone de los datos los recoge y muestra las últimas transacciones del usuario tratando los datos. Si el usuario tiene su cartera conectada está mostrará las últimas transacciones, en caso de no ser así pedirá al usuario que se conecte para poder mostrarlas.
- **DummyData:** Indica a la aplicación el formato en el cual se deberán de mostrar las transacciones, indicados en un formato JSON.

- **UseFetch:** Su función es hacer una petición a Giphy(página para obtener gifs). Ya que el usuario al introducir un keyword en su transacción este se mostrará como un GIF.

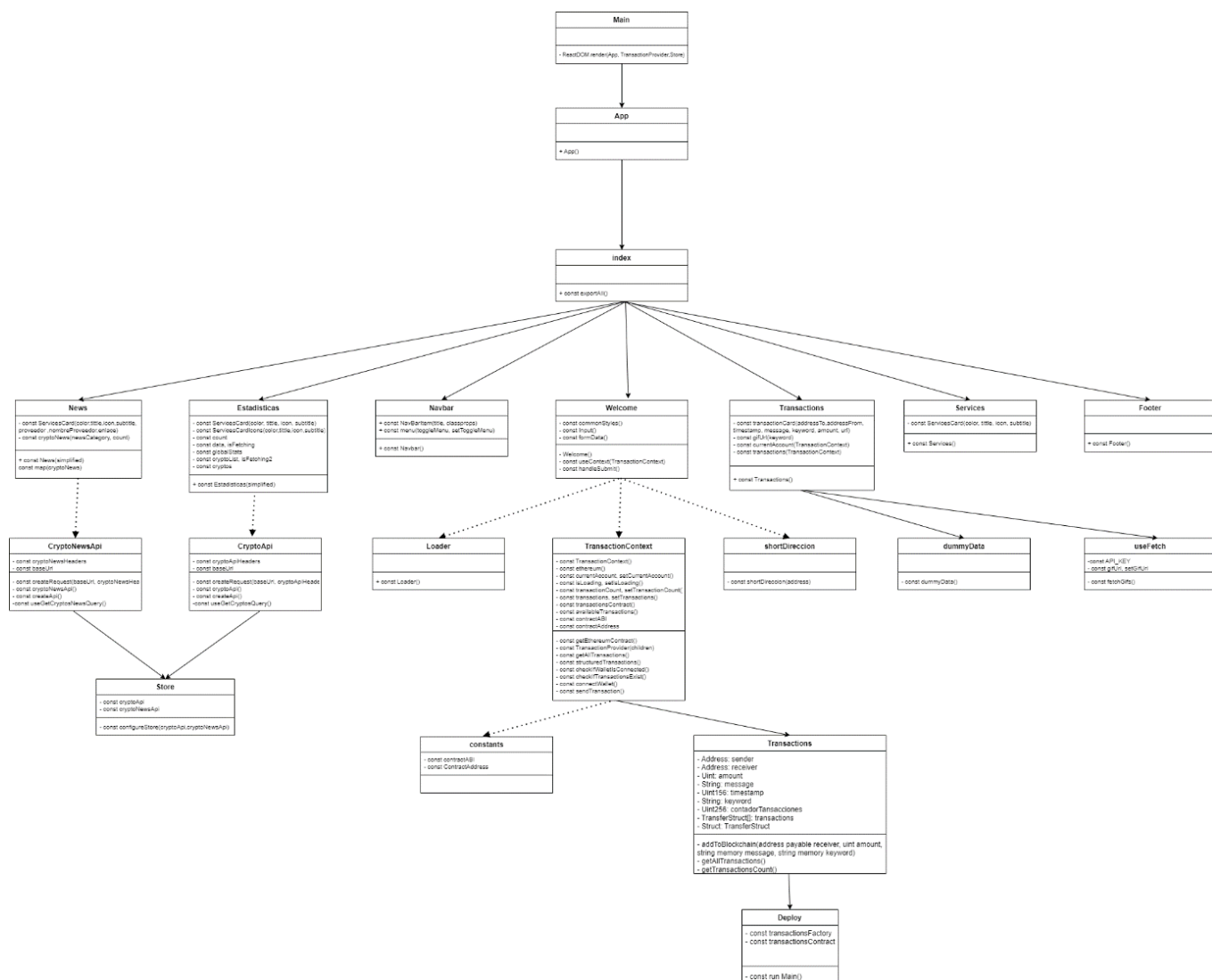
Diagrama de clases del entorno blockchain.



- **Transactions:** Es el Smart contract donde se realizan las transacciones, en él están definidas las pautas y los datos que deberá de llevar cada transacción. Al implementar el contrato se implementa en la red de blockchain.
- **Deploy:** Clase que sirve para subir el contrato a la red de blockchain. Esta clase sólo se utiliza cuando el Smart contract está finalizado y se quiere probar a subir a una red de blockchain. De esta manera se ha testeado el Smart contract antes de su implementación en una red blockchain.

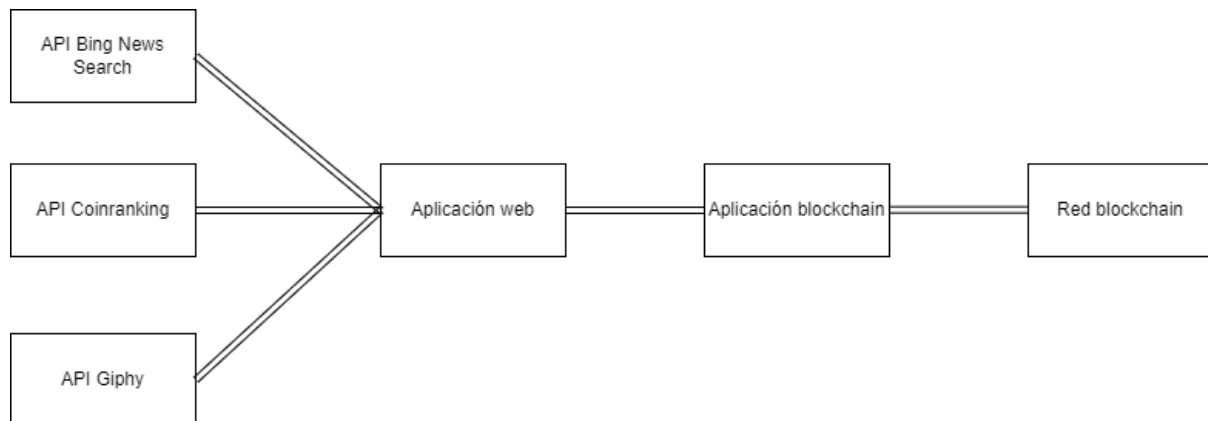
Diagrama de clases de la aplicación.

En la siguiente figura podemos observar el diagrama de clases completo uniendo la aplicación con el entorno blockchain:



En la anterior figura podemos observar cómo se conectan la aplicación web y el entorno blockchain. Esto se realiza mediante la clase TransactionContext, la cual es la encargada de recoger los datos de la red de blockchain y en trabajar con el Smart contract.

Diagrama de sistemas.



En la anterior imagen podemos observar el diagrama de sistemas del proyecto. Las conexiones entre las diferentes partes del sistema podemos observar que se han realizado con una línea doble. Esto significa que las conexiones y transferencias de datos se realiza de una manera mutua.

Aplicación web - Aplicación blockchain.

La relación entre estos dos sistemas de la aplicación es fundamental para el funcionamiento de la página. Ya que la aplicación web precisará del smart contract para realizar la transacción.

Aplicación blockchain – Red blockchain.

La aplicación necesitará conectarse a una red de blockchain para poder realizar las transacciones y subir la transacción y el smart contract a la red de blockchain.

Aplicación web – API Bing News Search.

La aplicación realizará una petición a esta API para pedirle los datos de las noticias que se mostrarán en la página. La API devolverá los datos solicitados a la aplicación web.

Aplicación web – API CoinRanking.

La aplicación realizará una petición a esta API para pedirle los datos de las criptomonedas que se mostrarán en la página. La API devolverá los datos solicitados a la aplicación web.

Aplicación web – API Giphy.

La aplicación realizará una petición a esta API cada vez que se realice una transacción para relacionar el keyword introducido en el formulario con un gif de la biblioteca. La aplicación realizará una petición y la API devolverá el enlace al Gif correspondiente para poder mostrarlo en la aplicación web.

DESARROLLO DEL PROYECTO

Preparación del entorno.

Lo primero para empezar a desarrollar el proyecto será definir varios conceptos:

- IDE que se utilizará para el desarrollo, Visual Studio Code.
- Servidor local para correr la aplicación web, Vite.
- Servidor de blockchain para el deploy y testeo del Smart contract, HardHat.
- Lenguajes de programación con los que se trabajará en el proyecto, JavaScript, Solidity, CSS.
- Gestor de dependencias y dependencias, NodeJS y npm.

Una vez con el entorno preparado para poder empezar a desarrollar el proyecto se realiza un plan sobre lo que se empezará a desarrollar e implementar. De manera que se podrá seguir un plan a la hora de realizar el desarrollo.

1. Diseño de interfaz de la aplicación web.

Para comenzar a tener una aplicación web deberemos de empezar a crear la página, con esto marcaremos el rumbo hacia dónde queremos que se dirija la página en cuanto a diseño general.

Al trabajar con React tendremos una gran librería de iconos y demás cosas que utilizaremos para el diseño de la página, además del código CSS.

Mientras se desarrolla la interfaz en las clases las cuales tengan funcionalidad y no sean diseño, se irán adaptando para recibir, enviar y trabajar con los datos que se llevarán a cabo al final del proyecto.

La aplicación web es responsive, esto significa que según se vaya desarrollando y diseñando se prepara la aplicación. Se realizan cambios entre una pantalla más pequeña(teléfono) a las pantallas de grande. De esta manera se cuida que la página tenga un diseño funcional e interactivo en todo tipo de dispositivos.

Una vez con el diseño principal de la aplicación, responsive y preparada con el formulario y botones correspondientes podremos pasar a la funcionalidad.

1.1. Explicación de diseño de la aplicación web.

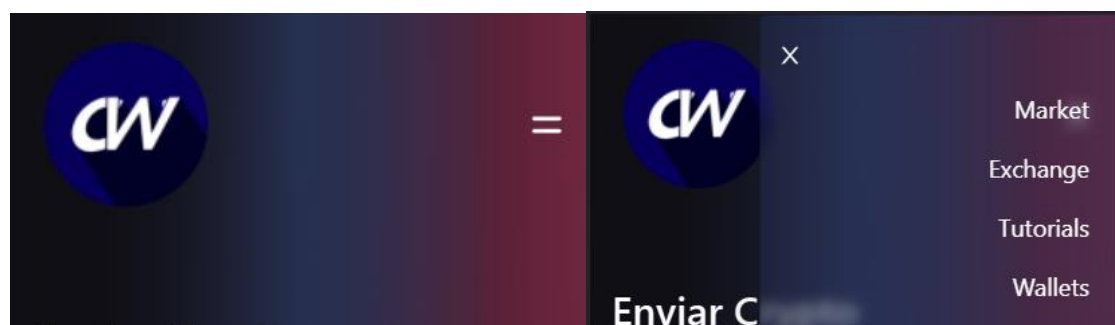
La aplicación web trabaja con siete clases diferentes en las cuáles se define el diseño de la página y el funcionamiento de esta. En este apartado explicaremos las clases en cuanto a su diseño, el funcionamiento lo observaremos de manera más detallada en los siguientes puntos.

La aplicación web es totalmente responsive, lo que significa que se podrá visualizar desde cualquier tipo de dispositivo que su vista será correcta y tendrá un diseño personalizado.

1.1.1. Clase Navbar.



En la anterior imagen podemos observar el navbar de la aplicación web desde un ordenador.



En las anteriores imágenes podemos observar el diseño de la aplicación web observado desde un teléfono móvil(dispositivo pequeño). Se puede observar que el menú se ha introducido en un botón(bocadillo) y se expandirá si el usuario interactúa con él. Esto facilita el acceso desde dispositivos pequeños.

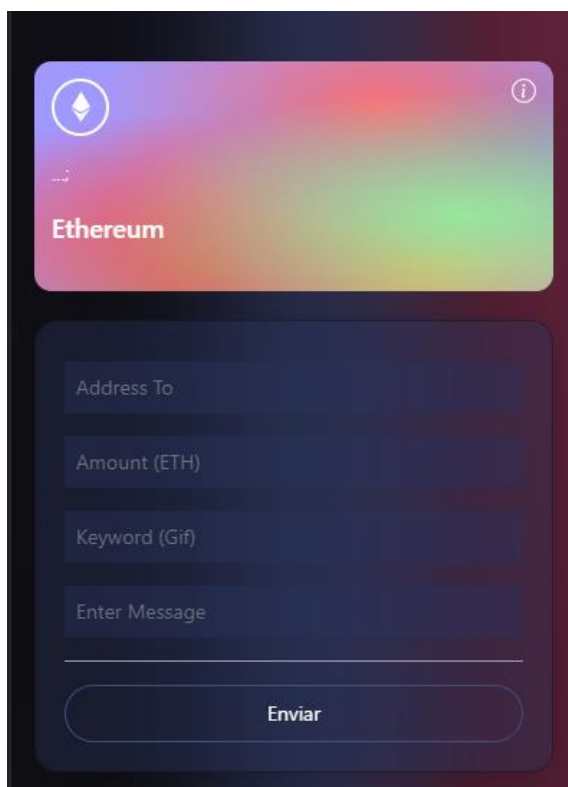
1.1.2. Clase Welcome.

En esta clase podremos observar además del diseño y datos de información, el botón para conectar la wallet del usuario, la tarjeta del usuario y el formulario a rellenar para realizar una petición.



En la anterior imagen podemos observar como se ve en un dispositivo grande.





En las anteriores imágenes podemos observar(en sentido descendente) la manera en la que se ve la aplicación web desde un dispositivo pequeño.

1.1.3. Clase Estadísticas.

En esta clase podemos observar las estadísticas y el top de las criptomonedas.

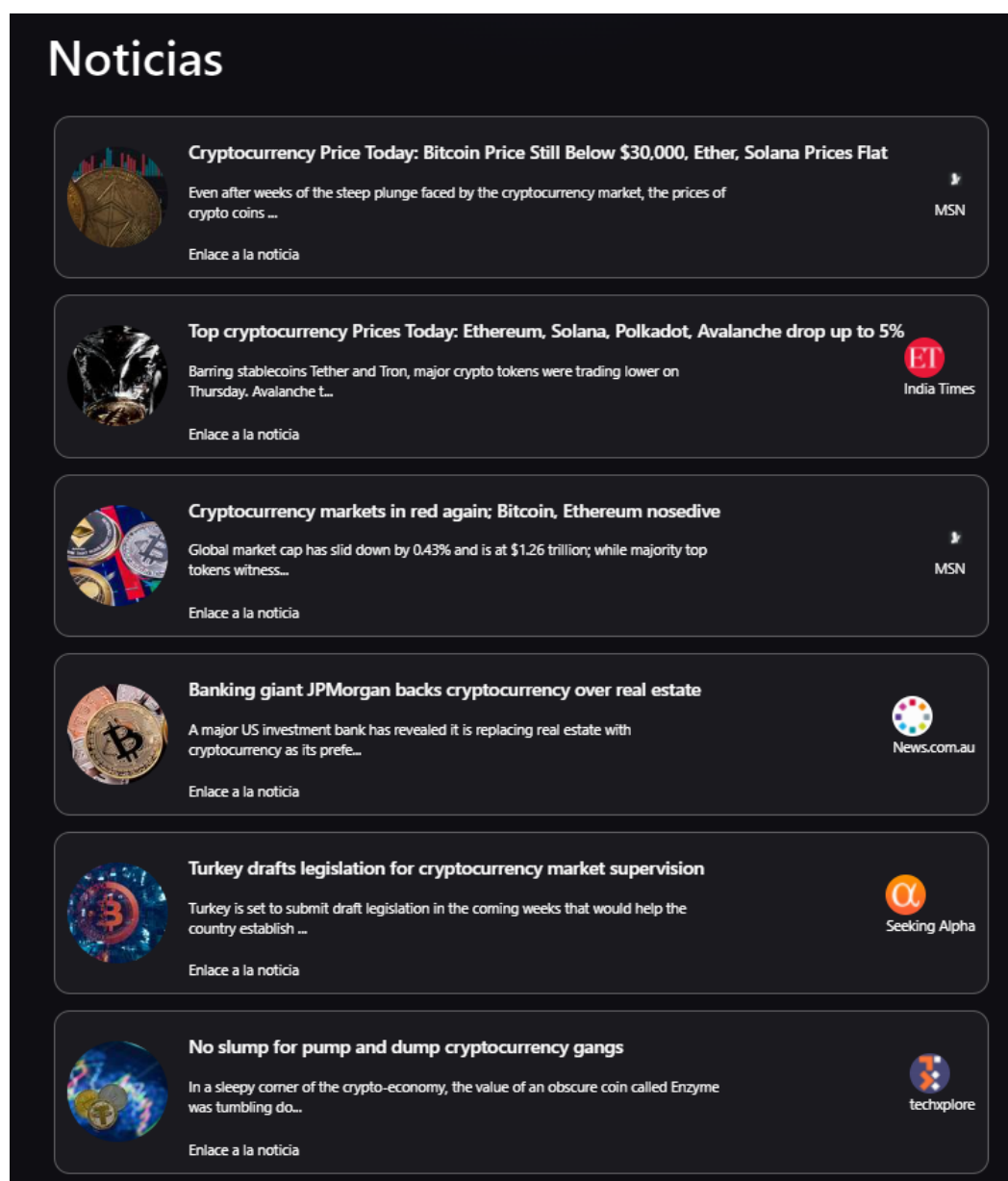


En las anteriores imágenes podemos observar el diseño de las estadísticas y del top de la aplicación web.

En este caso desde un dispositivo pequeño se vería exactamente igual pero adaptado a dispositivos más pequeños. La manera en la que se muestran los datos es la misma.

1.1.4. Clase News.

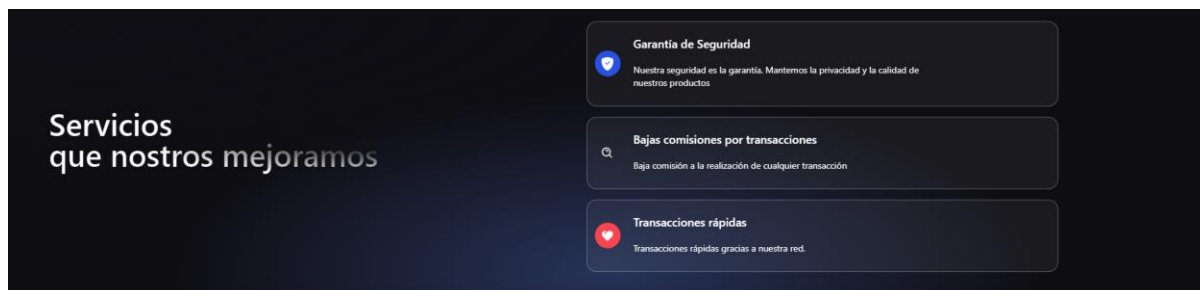
En esta clase podremos observar el apartado de las noticias de nuestra aplicación web. La manera en la que se muestran los datos es igual en dispositivos de cualquier tamaño.



En la anterior imagen podemos observar el apartado de noticias.

1.1.5. Clase services.

En esta clase podremos observar una lista de datos en la cual indicamos lo que nuestra aplicación mejora en referencia a las demás aplicaciones del mercado.



En la anterior imagen podemos observar como se visualizaría en un dispositivo de tamaño grande.

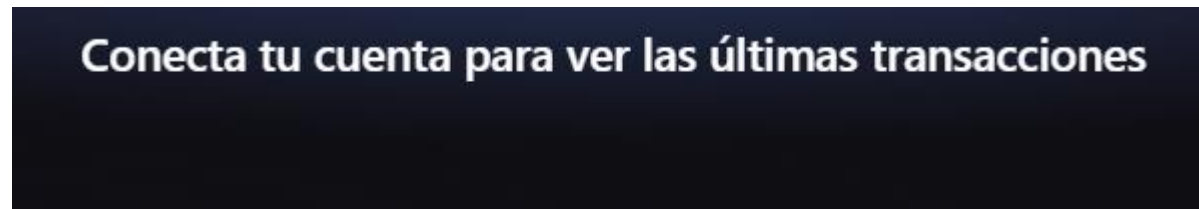


En la anterior imagen podemos observar como se vería el apartado de servicios desde un dispositivo pequeño.

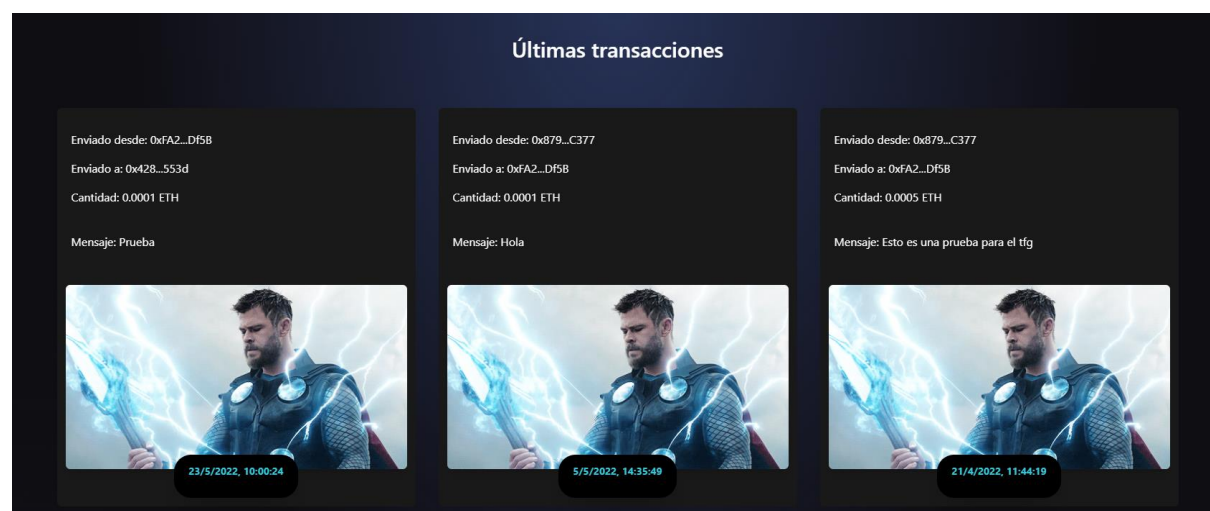
1.1.6. Clase Transactions.

En esta clase podremos observar dos tipos de diseño:

- Diseño de usuario que no tiene su wallet conectada. La página le indicará que conecte su wallet para observar sus transacciones:



- Diseño de usuario que tiene su wallet conectada:



En la anterior imagen podemos observar como un usuario con su wallet conectada podrá visualizar sus transacciones. En el caso de que se visualice desde un dispositivo pequeño las transacciones se mostrarán una a una en orden descendente (captura no posible de realizar).

1.1.7. Clase Footer.

Esta clase será la clase final de la página web y contendrá los datos y logo de la aplicación.



En la imagen anterior podemos observar el footer de la aplicación web.

2. Desarrollo del entorno blockchain.

Pudiendo apoyarnos en una aplicación visible, se empieza a desarrollar el entorno blockchain.

Empezaremos desarrollando el Smart Contract:

Para desarrollar un Smart Contract debemos de tener claro con que finalidad estamos desarrollando este contrato y que funcionalidad queremos darle. Debemos de asignarle los atributos correspondientes y los métodos de este.

Los atributos de este Smart Contract son los que definen una transacción y son los siguientes:

- Vendedor.
- Recibidor.
- Cantidad.
- Mensaje.
- Fecha.
- Palabra clave.

En la siguiente figura podemos observar como se define un objeto de una transacción:

```
//Objeto de una transferencia
struct TransferStruct {
    address sender;
    address receiver;
    uint amount;
    string message;
    uint256 timestamp;
    string keyword;
}
```

Sin alguno de estos atributos a la hora de realizar una transacción, la transacción no sucederá.

Los métodos que contiene el Smart contract son los siguientes:

- addToBlockchain, este método nos permite subir la transferencia a la red de blockchain, es el método encargado de validar que todos los datos que contiene la transacción son correctos.
- getAllTransactions, método que nos permite recoger todas las transacciones realizadas con este Smart Contract. Es una manera fácil de recoger las

transacciones aunque en una red de blockchain las transacciones no se pueden borrar y siempre quedarán registradas.

- `getTransactionCount`, con este método nos devolverá el número de transacciones realizados con este Smart contract.

El contrato llevará asignada una variable numérica que se autoincrementa cada vez que se realiza una operación con el mismo.

2.1. Testeo y deploy del Smart contract.

Teniendo ya el Smart contract podremos realizar su testeo y después su deploy en la red de blockchain.

Empezaremos el testeo del Smart contract decidiendo en que red vamos a realizar el deploy del mismo. En este proyecto se ha realizado en una red de pruebas de Ethereum Ropsten.

Realizaremos el testeo con la ayuda de HardHat(entorno de desarrollo de Ethereum), el Smart contract se compilará y nos devolverá la dirección del contrato. La dirección del contrato la usaremos más tarde por lo cual deberá de ser guardada.

En la siguiente imagen se puede observar la dirección del Smart contract guardada en una variable:

```
export const contractAddress = '0x23e2F2BB2E930E3Daf2415b96081FF5DD8C45F1D';
```

Al realizar la compilación del Smart contract, este nos devolverá un archivo JSON que tendremos que introducir en nuestra aplicación web(este paso es de suma importancia porque define como funciona el Smart contract). En la aplicación web recogeremos el ABI de nuestro JSON. ABI, es el modo estándar de interactuar con contratos en el ecosistema Ethereum, lo podemos utilizar en una red blockchain(como es en este proyecto) o interacciones contrato-contrato.


```
"abi": [  
  {  
    "anonymous": false,  
    "inputs": [  
      {  
        "indexed": false,  
        "internalType": "address",  
        "name": "from",  
        "type": "address"  
      },  
    ],  
  },  
],
```

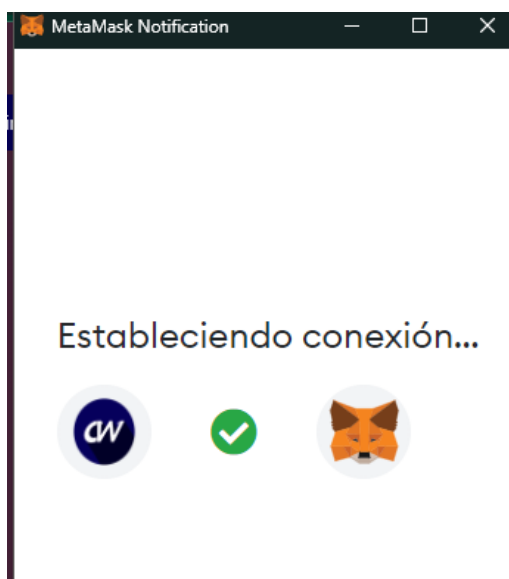
En la anterior figura podemos observar un ejemplo de como este JSON en ABI dicta las normas a la hora de cumplir y trabajar con el Smart contract.

2.2. Implementación del entorno blockchain en la aplicación web.

Realizaremos la implementación del entorno blockchain en la aplicación web mediante una nueva clase a la que llamamos Transaction Context, esta clase se encargará de recoger todos los datos previamente mostrados (ABI del contrato, dirección del contrato) y contendrá los métodos para trabajar con el Smart contract en la red de blockchain.

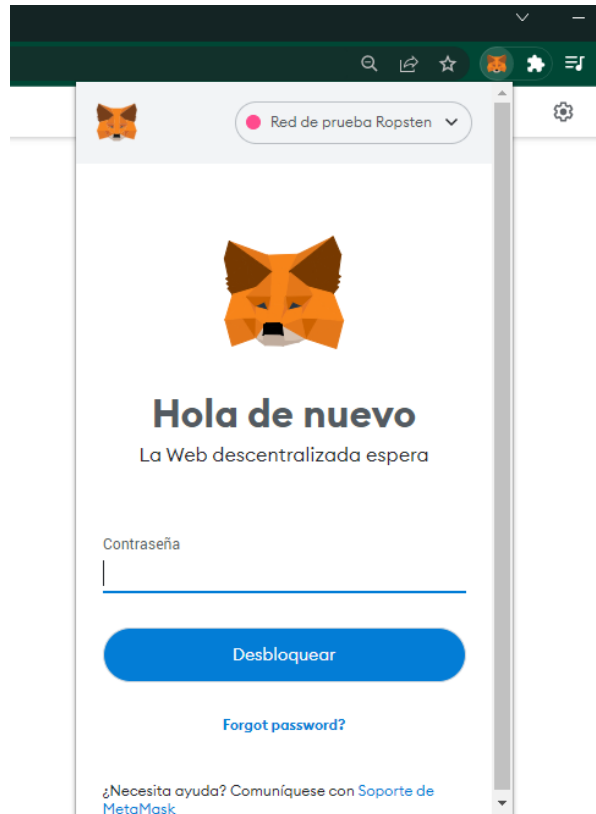
2.2.1. Implementación de MetaMask en la aplicación.

MetaMask, es un software de Criptomoneda el cual se instala como extensión de un navegador, se utiliza para interactuar con la plataforma de blockchain de Ethereum.

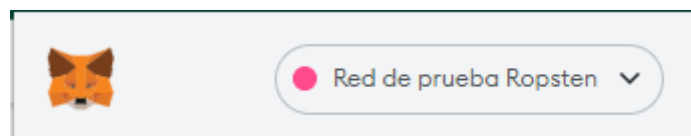


En la anterior imagen podemos observar como se realiza la conexión entre la aplicación web y MetaMask.

En la siguiente figura podemos observar la extensión de MetaMask:



En la anterior imagen podemos observar la extensión de MetaMask, también podemos observar en que red se encuentra actualmente (la red la seleccionamos nosotros previamente).



En la anterior imagen podemos observar que estamos trabajando en una red de pruebas Ropsten.

¿Cómo relacionamos MetaMask con nuestra aplicación?

El primer paso será realizar los pasos previos de creación de wallet que nos ofrece esta plataforma pero en los que no entraremos en detalle.

MetaMask, una vez que estamos conectados en una cuenta y con nuestra aplicación funcionando en un servidor local o en un host externo, este nos devolverá un objeto con todos los datos de la cuenta.

Nuestra función será recoger los datos para trabajar con ellos y mostrarle los datos precisos al usuario final.

```
const { ethereum } = window;
```

En la anterior figura se puede ver cómo recogen los datos que nos proporciona MetaMask del usuario una vez haya iniciado sesión.

¿Qué se realiza con estos datos recogidos del usuario?

Una vez recogidos estos datos del usuario, procederemos a crear la transacción del contrato. Conocemos ya los datos del usuario que desea realizar una transacción mediante nuestra aplicación.

```
const getEthereumContract = () => {  
  //Le pasamos por parámetro el objeto ethereum.  
  const provider = new ethers.providers.Web3Provider(ethereum);  
  const signer = provider.getSigner();  
  const transactionsContract = new ethers.Contract(  
    contractAddress,  
    contractABI,  
    signer  
  );  
  
  return transactionsContract;  
};
```

En la anterior imagen podemos observar como se realiza el relleno de datos del usuario para poder realizar la transacción y cumplir con el Smart contract.

La aplicación ya contiene cómo trabajar con MetaMask(en caso de que el usuario precise de instalar MetaMask, al intentar conectar su wallet se le notificará) y contiene los datos de la wallet del usuario que esté conectado en la página.

Estos datos solo se recogen de una manera temporal para realizar la transacción, una vez se realice la transacción los datos quedarán guardados en la red de blockchain.

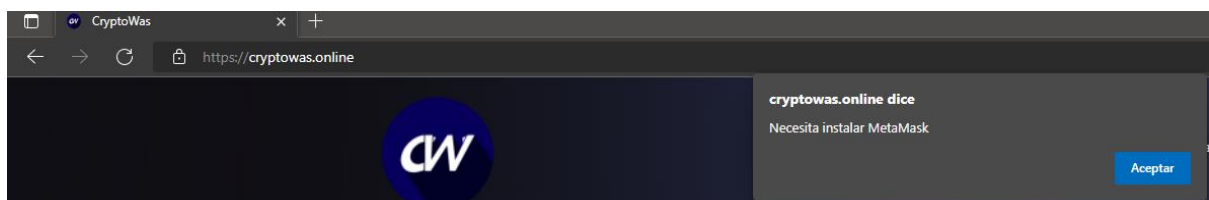
2.3. Realización de transacciones y completo funcionamiento aplicación-blockchain.

Hemos recogido los datos del Smart contract (ABI, dirección del contrato) y tenemos los datos del usuario cuando inicia sesión en MetaMask.

Se validará cada vez que se intente realizar una operación si el usuario está conectado con MetaMask y si tiene la extensión activada.

```
if (!ethereum) return alert("Necesita instalar MetaMask");
```

En la anterior figura podemos observar la validación en el caso de que el usuario no tenga la extensión MetaMask.



En la anterior imagen podemos observar como la aplicación indica de una manera informativa que necesita instalar MetaMask si quiere conectar su wallet.



En la anterior imagen podemos observar cómo el botón de conectar wallet desaparece una vez que el usuario haya conectado su wallet.

Asociación de aplicación web a blockchain.

Se trabajará creando un contexto dentro de la aplicación(un contexto trabajando con React es una manera de crear variables globales con las que podremos trabajar en toda la aplicación). Se recogerán los datos previamente mostrados.

Procederemos a realizar los métodos y validaciones necesarias para el correcto funcionamiento del mismo. Se validará si el usuario precisa cambiar de wallet, si el usuario tiene transacciones previas para mostrarlas en la aplicación y se realizarán los métodos que envían las transacciones.

Variables.

```
//Comprobamos si tiene una cuenta conectada y validamos que pueda cambiar de cuenta  
const [currentAccount, setCurrentAccount] = useState("");
```

En la anterior imagen podemos observar un estado de usuario de React(useState se utiliza para almacenar los datos del usuario y se guarden mientras dure la sesión):

- currentAccount, cuenta conectada del usuario al conectarse a MetaMask.
- setCurrentAccount, cambiamos la cuenta del usuario en caso de que quiera cambiarla o para el primer inicio de sesión.

```
const [transactions, setTransactions] = useState([]);
```

En la anterior imagen podemos observar donde se almacenan las transacciones. Las transacciones serán almacenadas en una array.

- Transactions, almacenamos las transacciones del usuario.
- setTransaction, guardamos en transactions las transacciones del usuario o incluimos una nueva si se acaba de realizar.

```
//Contador de transacciones que almacenamos en la memoria para que no se reinicie
const [transactionCount, setTransactionCount] = useState(
  localStorage.getItem("transactionCount")
);
```

En la anterior imagen podemos observar como almacenamos el contador de transacciones. El contador de transacciones está definido en el Smart contract y se guardará siempre en la memoria de la aplicación. El contador sumará +1 cada vez que se realice una transacción, cada vez que se inicie la aplicación este recogerá el contador y en caso de nueva actualización esta se realizará.

- transactionCount, contador de transacciones.
- setTransactionCount, actualiza el contador de transacciones si se realiza una nueva transacción.

Métodos.

Conectar Wallet.

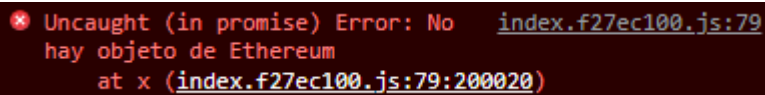
```
//Funcion para conectar a MetaMask
const connectWallet = async () => {
  try {
    if (!ethereum) return alert("Necesita instalar MetaMask");

    const accounts = await ethereum.request({
      method: "eth_requestAccounts",
    });

    setCurrentAccount(accounts[0]);
  } catch (error) {
    console.log(error);

    throw new Error("No hay objeto de Ethereum");
  }
};
```

En la anterior imagen podemos observar el método para conectar la wallet del usuario. Se valida si el usuario tiene la extensión MetaMask. Una vez validado se guardará en una variable la wallet conectada del usuario y se introducirá en la array de cuentas.



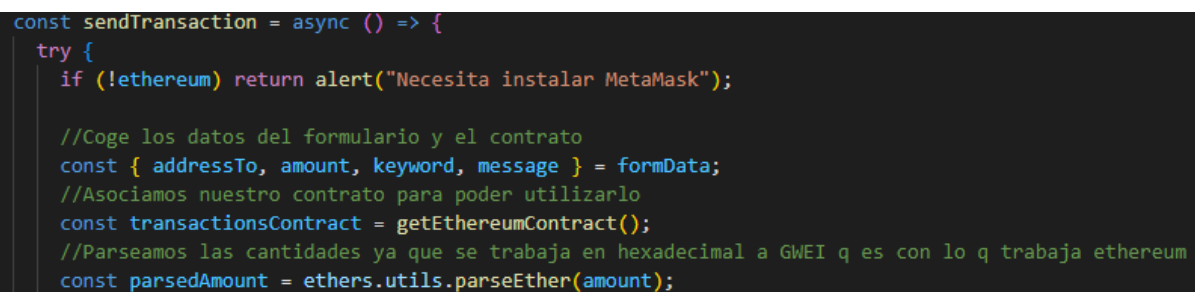
```

* Uncaught (in promise) Error: No hay objeto de Ethereum
    at x (index.f27ec100.js:79:200020)
  
```

En la anterior figura podemos observar como la aplicación nos indica mediante consola que no hay ningún objeto de Ethereum (este “error” desaparecerá una vez el usuario conecte su wallet).

Enviar transacción.

Este método es uno de los métodos principales de la aplicación, ya que es el que se encarga de recoger todos los datos necesarios y realizar la transacción.



```

const sendTransaction = async () => {
  try {
    if (!ethereum) return alert("Necesita instalar MetaMask");

    //Coge los datos del formulario y el contrato
    const { addressTo, amount, keyword, message } = formData;
    //Asociamos nuestro contrato para poder utilizarlo
    const transactionsContract = getEthereumContract();
    //Parseamos las cantidades ya que se trabaja en hexadecimal a GWEI q es con lo q trabaja ethereum
    const parsedAmount = ethers.utils.parseEther(amount);
  }
}
  
```

En la anterior imagen podemos observar como valida que el usuario tenga MetaMask, y recoge todos los datos necesarios para la transacción:

- Recoge de datos del formulario, recoge los datos del formulario y los almacena en variables.
- Recoge de datos de usuario, transactionsContract recoge los datos del usuario con una wallet conectada.
- Recoge y parsea las cantidades con las que se trabaja en Ethereum (cantidades que suelen ser grandes números hexadecimales, se parsea entenderlos mejor).

```
//Enviamos los datos, ponemos la comision y cogemos los datos
await ethereum.request({
  method: "eth_sendTransaction",
  params: [
    {
      from: currentAccount,
      to: addressTo,
      gas: "0x5208",
      value: parsedAmount._hex,
    },
  ],
});
```

En la anterior imagen observamos como se realiza la transacción con los datos necesarios para la misma. Gas, comisión que se aplica a la hora de realizar una transacción, en este Smart contract se trabaja con comisiones bajas.

```
//Guardamos la transaccion en el blockchain
const transactionHash = await transactionsContract.addToBlockchain(
  addressTo,
  parsedAmount,
  message,
  keyword
);
```

En la anterior imagen podemos observar como se añade la transacción a la red de blockchain, esto es indispensable ya que todas las transacciones deben de quedar registradas en la red de blockchain. Este método esta definido en el Smart contract y será donde se han establecido los datos necesarios para realizar la transacción.

```
setTransactionCount(transactionCount.toNumber());
```

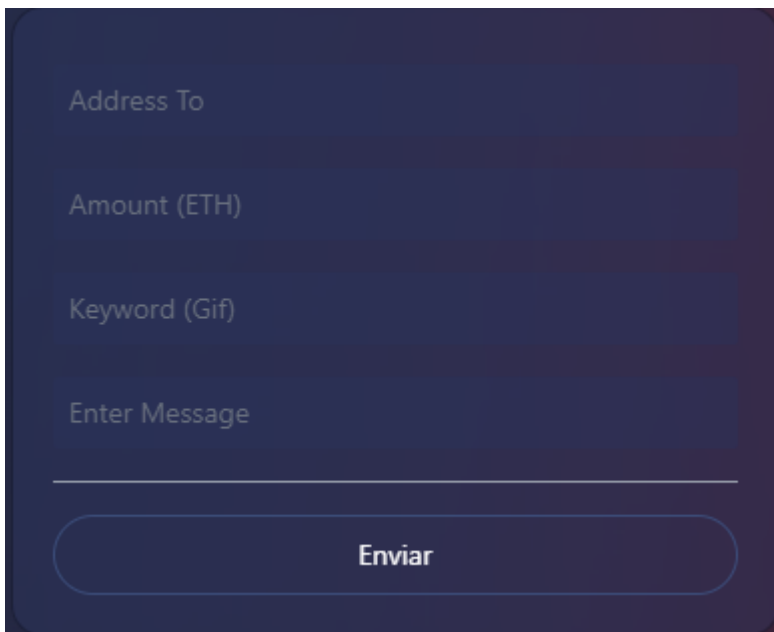
En la anterior imagen podemos observar el último paso para realizar una transacción, aumentar el contador de transacciones.

Almacenamiento de formulario de las transacciones.

```
//Transferimos los datos necesarios para conectarse al blockchain
export const TransactionProvider = ({ children }) => {
  //Cogemos los datos del formulario
  const [formData, setFormData] = useState({
    addressTo: "",
    amount: "",
    keyword: "",
    message: "",
  });
};
```

En la anterior imagen podemos observar un método donde se recogen los datos del formulario(formulario que encontramos en la clase Welcome). Los datos del formulario se validan antes de enviar el formulario, cuando el formulario se envía recogemos los datos(mediante children) y los almacenamos.

- formData, almacena los datos del formulario.
- setFormData, cuando se envía el formulario guarda los datos en la variable formData.

A screenshot of a web form with a dark blue background. It contains four text input fields stacked vertically: 'Address To', 'Amount (ETH)', 'Keyword (Gif)', and 'Enter Message'. Below these fields is a rounded rectangular button with the text 'Enviar' in white.

En la anterior imagen observamos el formulario de datos a rellenar para realizar una transacción.

Validación de transacciones.

```
//Comprobamos que existan transacciones para mostrarlas
const checkIfTransactionsExist = async () => {
  try {
    //Cogemos el contract
    const transactionsContract = getEthereumContract();
    const transactionCount = await transactionsContract.getTransactionCount();

    window.localStorage.setItem("transactionCount", transactionCount);
  } catch (error) {
    console.log(error);

    throw new Error("No hay objeto de Ethereum");
  }
};
```

En la anterior figura podemos observar como se valida si existen transacciones realizadas por el usuario. En el caso de que el usuario no conecte su wallet esta nos mostrará un error por consola indicando que “No hay objeto de Ethereum” lo cual indica que el usuario no ha conectado su wallet. Es una función asíncrona (se utiliza la función asíncrona para pedir y devolver una promesa, cuando esta promesa le llegue entonces el método empezará a funcionar).

Almacén de transacciones.

```
const getAllTransactions = async () => {
  try {
    if (!ethereum) return alert("Necesita instalar MetaMask");

    const transactionsContract = getEthereumContract();

    //Llamamos a la funcion de devolver todas las transacciones de nuestro smart contract
    const availableTransactions =
      await transactionsContract.getAllTransactions();

    //Realizamos la estructura de las transacciones para despues mostrarlas.
    const structuredTransactions = availableTransactions.map(
      (transaction) => ({
        addressTo: transaction.receiver,
        addressFrom: transaction.sender,
        timestamp: new Date(
          transaction.timestamp.toNumber() * 1000
        ).toLocaleString(),
        message: transaction.message,
        keyword: transaction.keyword,
        amount: parseInt(transaction.amount._hex) / 10 ** 18,
      })
    );

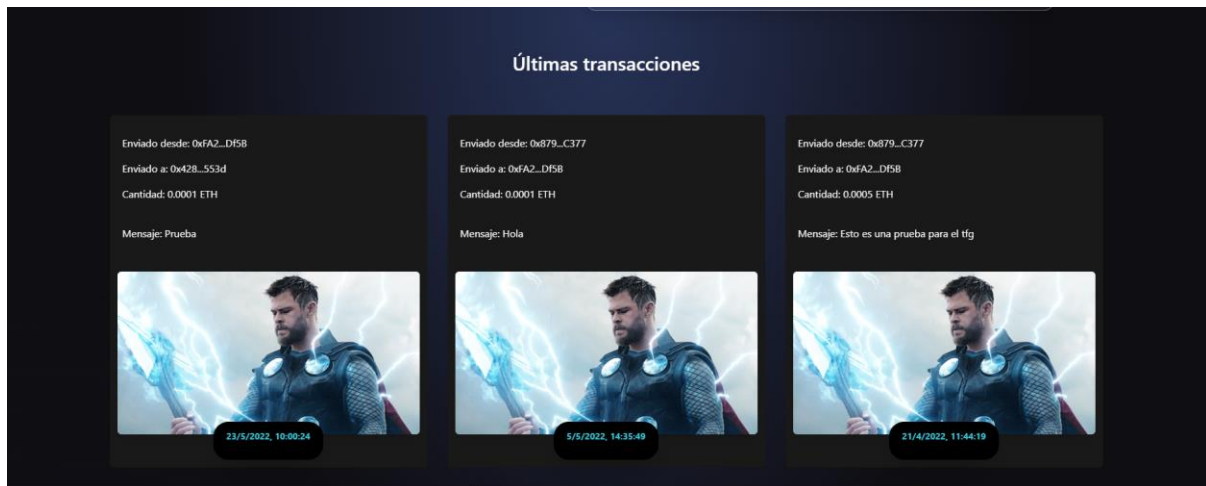
    console.log(structuredTransactions);
    setTransactions(structuredTransactions);
  } catch (error) {
    console.log(error);
  }
};
```

En la anterior figura podemos observar como se realiza la recogida de las transacciones, en esta caso cuando el usuario conecte su wallet con MetaMask este método recogerá todas las transacciones que se hayan realizado con nuestro Smart Contract y las mostrará por pantalla.



```
index.f27ec100.js:79
▼ Array(5) 1
  ▶ 0: {addressTo: '0x428a42972C5048E797b707F71c3082d', addressFrom: '0x428a42972C5048E797b707F71c3082d', timestamp: '2023-10-27T12:00:00', message: 'Transaction message', keyword: 'Transaction keyword', amount: 1000000000000000000}
  ▶ 1: {addressTo: '0xFA24CB28476be12c648206C1e89cfa', addressFrom: '0xFA24CB28476be12c648206C1e89cfa', timestamp: '2023-10-27T12:00:00', message: 'Transaction message', keyword: 'Transaction keyword', amount: 1000000000000000000}
  ▶ 2: {addressTo: '0xFA24CB28476be12c648206C1e89cfa', addressFrom: '0xFA24CB28476be12c648206C1e89cfa', timestamp: '2023-10-27T12:00:00', message: 'Transaction message', keyword: 'Transaction keyword', amount: 1000000000000000000}
  ▶ 3: {addressTo: '0x428a42972C5048E797b707F71c3082d', addressFrom: '0x428a42972C5048E797b707F71c3082d', timestamp: '2023-10-27T12:00:00', message: 'Transaction message', keyword: 'Transaction keyword', amount: 1000000000000000000}
  ▶ 4: {addressTo: '0x428a42972C5048E797b707F71c3082d', addressFrom: '0x428a42972C5048E797b707F71c3082d', timestamp: '2023-10-27T12:00:00', message: 'Transaction message', keyword: 'Transaction keyword', amount: 1000000000000000000}
  length: 5
```

En la anterior imagen podemos observar desde la consola del navegador, la array donde se almacenan las transacciones del usuario y su longitud.



En la anterior figura podemos observar como se han recogido las transacciones del usuario y se han mostrado en la página.

Transferencia de los datos al contexto.

Se realizará la transferencia de datos al contexto para que puedan estar disponibles en la aplicación a nivel global.

```
return (
  //Pasamos los objetos al contexto para poder utilizarlo
  <TransactionContext.Provider
    value={{
      connectWallet,
      currentAccount,
      formData,
      setFormData,
      handleChange,
      sendTransaction,
      transactionCount,
      transactions,
      isLoading,
    }}
  >
    {children}
  </TransactionContext.Provider>
);
```

En la anterior figura podemos observar como se pasan al contexto todas las variables y métodos definidos previamente.

3. Implementación de blockchain en aplicación web.

Formulario de envío de transacciones.

```
const Input = ({ placeholder, name, type, value, handleChange }) => (
  <input
    placeholder={placeholder}
    type={type}
    step="0.0001"
    value={value}
    onChange={(e) => handleChange(e, name)}
    className="my-2 w-full rounded-sm p-2 outline-none bg-transparent text-white border-none text-sm white-glassmorphism"
  />
);
```

En la anterior imagen podemos observar un input del formulario de envío de transacciones, realizo un asset previo para poder rehusarlo a la hora de realizar el formulario

```
<div className="p-5 sm:w-96 w-full flex flex-col justify-start items-center blue-glassmorphism">
  <Input
    placeholder="Address To"
    name="addressTo"
    type="text"
    handleChange={handleChange}
  />
  <Input
    placeholder="Amount (ETH)"
    name="amount"
    type="number"
    handleChange={handleChange}
  />
  <Input
    placeholder="Keyword (Gif)"
    name="keyword"
    type="text"
    handleChange={handleChange}
  />
  <Input
    placeholder="Enter Message"
    name="message"
    type="text"
    handleChange={handleChange}
  />
</div>
```

En la anterior imagen podemos observar el formulario completo para realizar una transacción.

```
const {
  connectWallet,
  currentAccount,
  formData,
  sendTransaction,
  handleChange,
  isLoading,
} = useContext(TransactionContext);
```

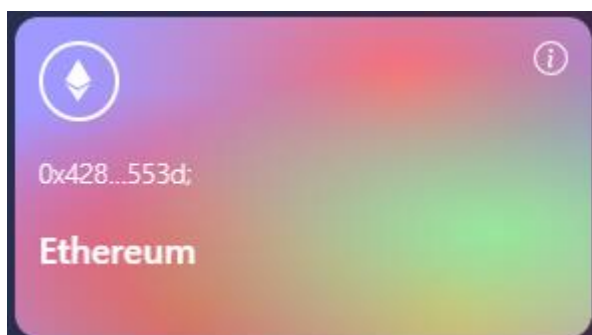
En la anterior imagen podemos observar como se recogen los datos del contexto previamente mostrados para tenerlos disponibles en esta clase del proyecto.

```
{!currentAccount && (
  <button
    type="button"
    onClick={connectWallet}
    className="flex flex-row justify-center items-center my-5 bg-[#06005c] p-3 rounded-full cursor-pointer hover:bg-[#2546bd]"
  >
    <p className="text-white text-base font-semibold">
      Conectar Wallet
    </p>
  </button>
)}
```

En la anterior imagen podemos observar como se realiza la validación de que si el usuario ha conectado su wallet y se quita el botón de conectar wallet.

```
/* Cogemos la cuenta del usuario en el caso de que esté conectado */
{shortDireccion(currentAccount)};
```

En la anterior imagen podemos ver como llamamos al método shortDireccion el cual reducirá el tamaño de dirección de la wallet del usuario para mostrarlo en la tarjeta de usuario.



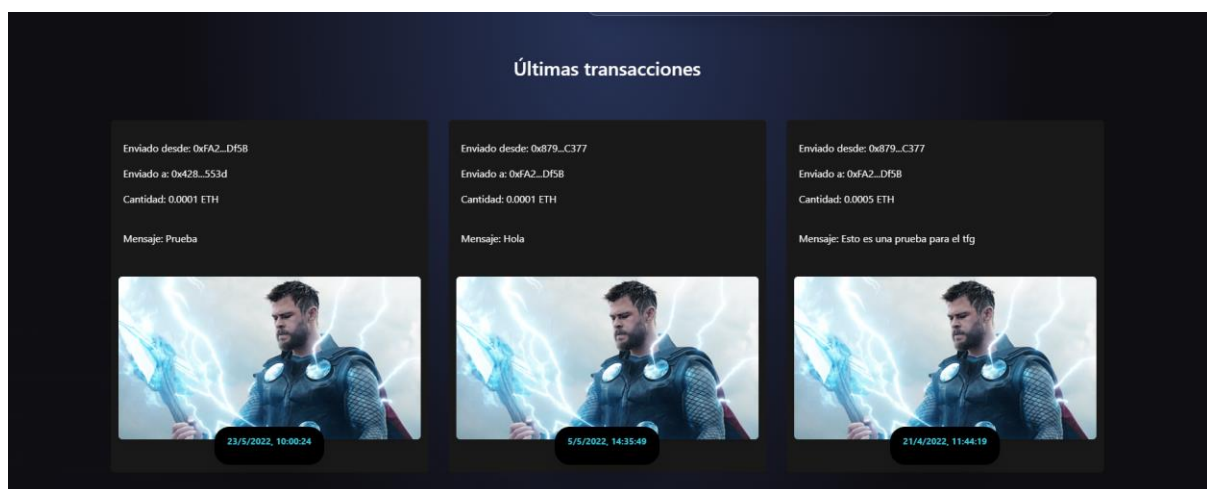
En la anterior imagen podemos observar cómo una vez el usuario haya conectado su cuenta, se recogerán su dirección de wallet y se mostrará en la tarjeta (la dirección se mostrará reducida, ya que las direcciones de las wallets de Ethereum contienen muchos caracteres).

Muestra por pantalla de las transacciones del usuario.

Un usuario no conectado no podrá visualizar sus transacciones, en su lugar se le indicará que conecte su wallet:

Conecta tu cuenta para ver las últimas transacciones

Un usuario conectado podrá visualizar sus transacciones:



Para realizar este funcionamiento desarrollaremos lo siguiente:

```
// Cogemos los datos de la cuenta actual conectada si existe  
const { currentAccount, transactions } = useContext(TransactionContext);
```

Recogemos los datos necesarios del contexto, en este caso necesitaremos recoger los datos de la cuenta actual del usuario y su lista de transacciones.

```

const TransactionCard = ({
  addressTo,
  addressFrom,
  timestamp,
  message,
  keyword,
  amount,
  url,
}) => {
  // Cogemos la url del gif
  const gifUrl = useFetch({ keyword });
  return (
    <div className="bg-[#181918] m-4 flex flex-1 2xl:min-w-[450px] 2xl:max-w-[500px] sm:min-w-[270px] sm:max-w-[300px] flex-col p-3 rounded-md hover:shadow-2xl">
      <div className="flex flex-col items-center 2-full mt-3">
        <div className="w-full mb-6 p-2">
          /* Introducimos las direcciones, URL en caso de que se clique sobre la carta reenviara al control de las transacciones desde la página oficial*/
          <a
            href={`https://ropsten.etherscan.io/address/${addressFrom}`}
            target="_blank"
            rel="noopener noreferrer"
          >
            <p className="text-white text-base">
              Enviado desde: {shortDireccion(addressFrom)}
            </p>
          </a>
          <a
            href={`https://ropsten.etherscan.io/address/${addressTo}`}
            target="_blank"
            rel="noopener noreferrer"
          >
            <p className="text-white text-base">
              Enviado a: {shortDireccion(addressTo)}
            </p>
          </a>
          <p className="text-white text-base">Cantidad: {amount} ETH</p>
          /* Si hay un mensaje para la transaccion lo mostramos */
          {message && (
            <div>
              <br />
              <p className="text-white text-base">Mensaje: {message}</p>
            </div>
          )}
        </div>
        /* Introducimos el gif */
        <img
          src={gifUrl || url}
          alt="gif"
          className="w-full h-64 2x:h-96 rounded-md shadow-lg object-cover"
        />
        <div className="bg-black p-3 px-5 w-max rounded-3xl -mt-5 shadow-2xl">
          <p className="text-[#37c7da] font-bold">{timestamp}</p>
        </div>
      </div>
    </div>
  );
}

```

En la anterior imagen podemos observar el prefab de lo que sería mostrar una transacción, en este caso cada transacción llevará un Gif que se relacionará con la palabra clave de la transacción.

Introducción de sistema de Gifs en la aplicación.

Se ha introducido un sistema de Gifs, a la hora de realizar una transacción, el usuario que realice la transacción podrá introducir una palabra clave la cual se asociará con un Gif de la biblioteca de Giphy.

Para esto trabajaremos en la clase useFetch, donde llamamos a la API de los Gifs.

```
const API_KEY = import.meta.env.VITE_GIPHY_API;
```

En la anterior imagen podemos observar como se almacena la key para poder trabajar con esta biblioteca de Gifs.


```
const useFetch = ({ keyword }) => {
  const [gifUrl, setGifUrl] = useState("");
```

En la anterior imagen podemos observar como se almacena y se pone el gif.

```
const fetchGifs = async () => {
  try {
    // Cogemos la url que utilizaremos. Buscada en la documentacion de Giphy
    const response = await fetch(
      `https://api.giphy.com/v1/gifs/search?api_key=${API_KEY}&q=${keyword}
        .split("")
        .join("")}&limit=1`
    );
    const { datos } = await response.json();

    setGifUrl(datos[0]?.images?.downsized_medium?.url);
  } catch (error) {
    // En el caso de que el gif introducido sea erróneo introduciremos uno
    setGifUrl(
      "https://i.pinimg.com/originals/73/d3/a1/73d3a14d212314ab1f7268b71d639c15.gif"
    );
  }
}
```

En la anterior imagen podemos observar el método con el que recoge y trata los datos del Gif, en caso de que la biblioteca no esté disponible en el momento de realizar la transacción se introducirá un Gif aleatorio almacenado en la aplicación.

```
{transactions.reverse().map((transaccion, index) => (
  <TransactionCard key={index} {...transaccion} />
))}
```

En la anterior imagen podemos observar como una vez ya implementado el sistema de Gifs, se realiza un mapeo de las transacciones disponibles para mostrar y se utiliza el prefab de una transacción previamente mostrado. La realización del prefab permite trabajar de una manera más rápida y hace que el código sea más reutilizable.

4. Implementación de APIs en la aplicación.

Este proyecto trabaja con 3 APIs externas:

- Giphy, previamente explicada, se utiliza como biblioteca de Gifs y lo mostramos junto a las transacciones.
- CoinRanking, API de donde recogeremos datos sobre valores de criptomonedas y datos de estas.
- Bing News, API de la que recogeremos las últimas noticias del mundo de las criptomonedas.

Para la implementación de estas APIs se ha utilizado el portal de APIs RapidAPI.

4.1. Implementación de CoinRanking.

El primer paso para implementar una API a este proyecto ha sido darse de alta en RapidAPI y suscribirnos a la API que deseamos usar. Una vez ya cumplidos estos pasos podremos acceder a nuestras credenciales y empezar a trabajar con ellas. En este caso se ha realizado en la clase CryptoApi.js.

```
// Cogemos los datos de las criptomonedas
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
```

En la anterior imagen podemos observar la importación de Redux Toolkit(explicado en tecnologías) y recogemos dos métodos que nos permiten trabajar con la API.

```
const cryptoApiHeaders = {
  "X-RapidAPI-Host": "coinranking1.p.rapidapi.com",
  "X-RapidAPI-Key": "37cb977581mshb805568f51f3856p160085jsn58f2c7f88484",
};
```

En la anterior imagen podemos observar como se guardan los headers(datos de acceso a la Api a la que vamos a realizar peticiones y la URL del portal).

```
const baseUrl = "https://coinranking1.p.rapidapi.com";
```

En la anterior imagen podemos observar como almacenamos la variable del portal para trabajar con ella más adelante.

Crearemos el request que enviaremos como petición:

```
const createRequest = (url) => ({ url, headers: cryptoApiHeaders });
```

En la anterior imagen podemos observar como se crea el request junto a los datos previamente explicados.

Crearemos el método para hacer una petición a la API y su query correspondiente con los datos que queramos pedir:

```
// Metodo para coger las criptomonedas
export const cryptoApi = createApi({
  reducerPath: "cryptoApi",
  baseQuery: fetchBaseQuery({ baseUrl }),
  endpoints: (builder) => ({
    getCryptos: builder.query({
      query: (count) => createRequest(`/coins?limit=${count}`),
    }),
  }),
});
```

En la anterior imagen podemos observar cómo trabajamos con el método(createApi) importado previamente. Podemos observar como se realiza la query con la petición, en este caso se solicita a la API datos sobre las coins(criptomonedas) con un límite establecido ya que existen muchas criptomonedas y no queremos el acceso a todas ellas.

```
export const { useGetCryptosQuery } = cryptoApi;
```

En la anterior imagen podemos observar como se exporta la query para poder hacer una petición desde otra clase(la clase en la que trataremos los datos.)

4.1.1. Tratamiento de datos y muestra en la aplicación.

Estadísticas globales.

Realizaremos una petición para recoger los datos de las estadísticas globales de las criptomonedas.

```
import { useGetCryptosQuery } from "../services/cryptoApi";
```

En la anterior imagen podemos observar como importamos el método query previamente explicado.

Crearemos un prefab donde se mostrarán los datos:

```
//Carta de servicios, para reutilizarlo
const ServicesCard = ({ color, tittle, icon, subtitle }) => (
  <div className="flex flex-row justify-start items-center white-glassmorphism p-3 m-2 hover:shadow-xl">
    <div
      className={`w-10 h-10 rounded-full flex justify-center items-center ${color}`}
    >
      <img src={icon} />
    </div>
    <div className="ml-5 flex flex-col flex-1">
      <h1 className="mt-2 text-white text-lg">{tittle}</h1>
      <p className="mt-2 text-white text-sm md:w-9/12">{subtitle}</p>
    </div>
  </div>
);
```

Realizaremos una petición a la query y almacenaremos los datos:

```
//Cogemos los datos de las criptomonedas
const { data, isFetching } = useGetCryptosQuery(5);
// Creamos una variables donde guardamos los
const globalStats = data?.data?.stats;
```

En la anterior imagen podemos observar como se realiza la petición(con un límite de 5 criptomonedas) y como almacenamos los datos de la misma. Podemos observar a la hora de recoger los datos que recogeremos las estadísticas de las criptomonedas:

```
const globalStats = data?.data?.stats;
```

En la anterior imagen podemos observar como accedemos a las “stats” ya que son los datos que mostraremos en la página.

```

/* Cogemos los valores de las criptomonedas reales y las mostramos en la página, utilizaremos millify para hacer los valores mas leibles*/
<ServicesCardIcons
  color="bg-[#8000FF]"
  title="Total Cryptocurrencies"
  icon=<GiAbstract030 fontSize={21} className="text-white" />
  subtitle={globalStats.total}
/>
<ServicesCardIcons
  color="bg-[#83FF33]"
  title="Total Exchanges"
  icon=<Gi3DGlasses fontSize={21} className="text-white" />
  subtitle={millify(globalStats.totalExchanges)}
/>
<ServicesCardIcons
  color="bg-[#FF7733]"
  title="Total Market Cap"
  icon=<GiAmplitude fontSize={21} className="text-white" />
  subtitle={millify(globalStats.totalMarketCap)}
/>
<ServicesCardIcons
  color="bg-[#3334FF]"
  title="Volumen total 24h"
  icon=<GiArmorDowngrade fontSize={21} className="text-white" />
  subtitle={millify(globalStats.total24hVolume)}
/>
<ServicesCardIcons
  color="bg-[#C70039]"
  title="Total Mercados"
  icon=<GiArrowings fontSize={21} className="text-white" />
  subtitle={millify(globalStats.totalMarkets)}
/>

```

En la anterior imagen podemos observar como realizamos la impresión de datos en la página y como accedemos a ellos. Accederemos a los siguientes datos que hemos almacenando realizando la query a la API:

- globalstats.total, mostrará el número total de criptomonedas que hay disponibles.
- globalstats.totalExchanges, mostrará el número total de Exchanges disponibles.
- Globalstats.totalMarketCup, mostrará el market cup total de todas las criptomonedas.
- Globalstats.total24hVolume, mostrará el volumen total de movimientos realizados en las últimas 24 horas.
- Globalstats.totalMarkets, mostrará el número total de mercados.

Estadísticas de Cryptomonedas



Total Cryptocurrencies

14061



Total Exchanges

181



Total Market Cap

1.3T



Volumen total 24h

86.6B



Total Mercados

27.5K

En la anterior imagen podemos observar como quedarían mostrados los datos.

Top de criptomonedas.

Con la misma petición realizada previamente ahora recogeremos otros datos diferentes para realizar un listado donde se muestren las cinco criptomonedas que más valor tengan.

Utilizaremos un prefab mostrado previamente para mostrar los datos que recogemos.

```
// //Cogemos la lista de las criptomonedas para el top de las criptomonedas
const { lista: cryptoList, isFetching2 } = useGetCryptosQuery(count);
const cryptos = data?.data?.coins;
```

En la anterior imagen podemos observar como esta vez hemos accedido a los datos de las “coins” las criptomonedas.

```
{cryptos?.map((currency) => (
  <ServicesCard
    color={currency.color}
    tittle={` ${currency.rank}. ${currency.name}`}
    icon={currency.iconUrl}
    subtitle={` Precio: ${millify(currency.price)}
              Market Cap: ${millify(
                currency.marketCap
              )}
              Daily Change: ${millify(
                currency.change
              )}%`}
    key={currency.uuid}
  />
)}}}
```

En la anterior imagen podemos observar como utilizo el prefab para mostrar el top de las criptomonedas y los datos que mapeamos:

- Currency.color, recogemos el color base de la criptomoneda.
- Currency.rank, recogemos el número en el cual se encuentra de ranking la criptomonedas.
- Currency.name, recogemos el nombre de la criptomoneda.
- Currency.price, precio actual de una criptomoneda.
- Currency.marketCap, recogemos el market cap de las criptomoneda.
- Currency.change, recogemos el cambio de la criptomoneda.
- Currency.uuid, recogemos el id de la criptomoneda, necesario para poder identificar cada una.

Top 5 Criptomonedas



1. Bitcoin

Precio: 29.1K Market Cap: 554B Daily Change: -2.6%



2. Ethereum

Precio: 1.8K Market Cap: 221.2B Daily Change: -7.2%



3. Tether USD

Precio: 1 Market Cap: 73.6B Daily Change: 0.2%



4. USDC

Precio: 1 Market Cap: 53.7B Daily Change: -0%



5. Binance Coin

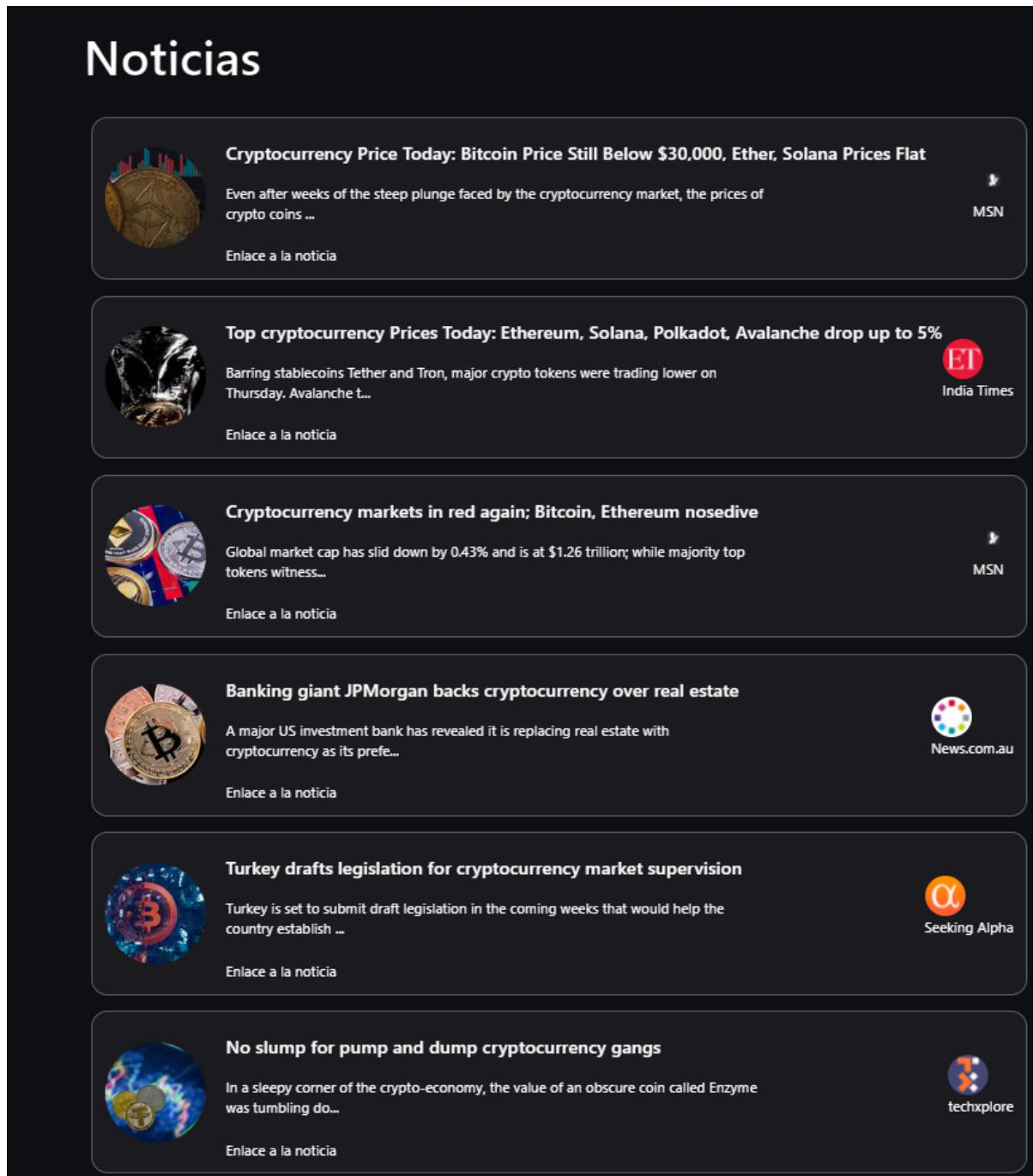
Precio: 312 Market Cap: 45.8B Daily Change: -6.2%

En la anterior imagen podemos observar el top de criptomonedas, con su logo correspondiente y sus datos actuales.

4.2. Implementación de Bing News.

Se implementa el proyecto la API Bing News para mostrar las últimas noticias del mundo del blockchain en la aplicación.

Utilizaremos el portal de RapidAPI para trabajar con esta API y de esta manera suscribirnos a ella, con esto obtendremos los headers para acceder a ella(key, url, etc).



En la anterior imagen podemos observar el apartado de noticias ya mostradas en la aplicación web.

```
//Cogemos los datos de las noticias
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
```

En la anterior imagen podemos observar la importación de Redux Toolkit(explicado en tecnologías) y recogemos dos métodos que nos permiten trabajar con la API.

```
const cryptoNewsHeaders = {
  "X-BingApis-SDK": "true",
  "X-RapidAPI-Host": "bing-news-search1.p.rapidapi.com",
  "X-RapidAPI-Key": "37cb977581mshb805568f51f3856p160085jsn58f2c7f88484",
};
```

En la anterior imagen podemos observar como se guardan los headers(datos de acceso a la Api a la que vamos a realizar peticiones y la URL del portal).

```
const baseUrl = "https://coinranking1.p.rapidapi.com";
```

En la anterior imagen podemos observar como almacenamos la variable del portal para trabajar con ella más adelante.

Crearemos el request que enviaremos como petición:

```
const createRequest = (url) => ({ url, headers: cryptoNewsHeaders });
```

En la anterior imagen podemos observar como se crea el request junto a los datos previamente explicados.

Crearemos el método para hacer una petición a la API y su query correspondiente con los datos que queramos pedir:

```
export const cryptoNewsApi = createApi({
  reducerPath: "cryptoNewsApi",
  baseQuery: fetchBaseQuery({ baseUrl }),
  endpoints: (builder) => ({
    getCryptoNews: builder.query({
      query: ({ newsCategory, count }) =>
        createRequest(
          `/news/search?q=${newsCategory}&safeSearch=Off&textFormat=Raw&freshness=Day&count=${count}`
        ),
    }),
  }),
});
```

En la anterior imagen podemos observar cómo trabajamos con el método(createApi) importado previamente. Podemos observar cómo se realiza la query con la petición, en este caso se solicita a la API, las noticias con unos parámetros en concreto:

- Categoría de las noticias que queremos obtener, en este caso noticias sobre el mundo de blockchain.
- Búsqueda segura.
- El formato del texto.
- El tiempo con el que se mostrarán nuevas noticias, en este caso se mostrarán noticias que tengan un menos de 24 horas desde su salida.
- Contador de las noticias que queremos mostrar en la aplicación web.

```
export const { useGetCryptoNewsQuery } = cryptoNewsApi;
```

En la anterior imagen podemos observar cómo se exporta la query para poder hacer una petición desde otra clase(la clase en la que trataremos los datos.)

4.2.1. Tratamiento de datos y muestra en la aplicación.

Realizaremos una petición para recoger las noticias, primero deberemos de importar la query previamente explicada para poder acceder a ella.

```
import { useGetCryptoNewsQuery } from "../services/cryptoNewsApi";
```

En la anterior imagen podemos observar como importamos la query para trabajar con ella.

Realizaremos un prefab donde dictaremos el formato de las noticias y sentaremos una base para mostrarlas todas de la misma manera:

```
//Carta de servicios, para reutilizarlo
const ServicesCard = ({
  color,
  tittle,
  icon,
  subtitle,
  proveedor,
  nombreProveedor,
  enlace,
}) => (
  <div className="flex flex-row justify-start items-center white-glassmorphism p-3 m-2 hover:shadow-xl">
    <div
      className={`w-30 h-20 rounded-full flex justify-center items-center ${color}`}
    >
      <img src={icon} className="w-30 h-30 rounded-full" />
    </div>
    <div className="ml-5 flex flex-col flex-1">
      <h1 className="mt-2 text-white text-lg">{tittle}</h1>
      <p className="mt-2 text-white text-sm md:w-9/12">{subtitle}</p>
      <a
        className="mt-2 text-white text-sm md:w-9/12"
        target="_blank"
        href={enlace}
      >
        Enlace a la noticia
      </a>
    </div>
    <div className="flex-end">
      <img src={proveedor} className="w-10 h-10 rounded-full flex-end" />
      <p className="text-white flex-end">{nombreProveedor}</p>
    </div>
  </div>
);
```

Introduciremos una imagen “demo” en caso de que a la hora de recoger la noticia esta no tenga adjuntada ninguna:

```
const demoImage =
  "https://www.bing.com/th?id=OVFT.mpzuVZnv8dwIMRfQGPbOPC&pid=News";
```

Recogeremos los datos realizando previamente la petición:

```
//Cogemos los datos de las noticias
const { data: cryptoNews } = useGetCryptoNewsQuery({
  newsCategory: "Cryptocurrency",
  count: simplified ? 6 : 12,
});
```

En la anterior imagen podemos observar como realizamos la petición, introduciendo los campos necesarios que hemos dictado a la hora de crear la petición.

```
//Si no tiene datos devolvemos un cargando
if (!cryptoNews?.value) return <Loader />;
```

En la imagen anterior podemos observar que si el valor que nos devuelve las noticias es nulo(puede ser porque falle la API o no haya conexión a internet) se introducirá una animación de cargando hasta que la petición se realice de una manera afirmativa.

```
{cryptoNews?.value.map((news, i) => (
  <ServicesCard
    color="#FFFFFF"
    tittle={` ${news.name}`}
    icon={news?.image?.thumbnail?.contentUrl || demoImage}
    subtitle={
      news.description.length > 100
      ? ` ${news.description.substring(0, 100)}...`
      : news.description
    }
  )
  proveedor={
    news.provider[0]?.image?.thumbnail?.contentUrl || demoImage
  }
  nombreProveedor={news.provider[0]?.name}
  enlace={news.url}
  key={i}
)}
)}
))}
```

En la anterior imagen podemos observar como se realiza el mapeo de las noticias una vez recogidas, utilizaremos el prefab previamente mostrado para su implementación en la página y los datos que mostraremos sobre las noticias son:

- News.name, nombre de la noticia.
- News.image, imagen de la noticia o en su defecto se pondrá la imagen demo previamente mostrada.
- News.description, este será el grueso de la noticia, su contenido será reducido ya que en nuestra página no queremos mostrar la noticia entera.
- News.provider, proveedor de la noticia el cual indicaremos.
- News.image.provider, imagen del proveedor de la noticia la cual indicaremos.
- News.url, dirección a la noticia, en caso de que el usuario decida seguir leyendo la noticia solo deberá de entrar al enlace.

4.3. Almacenamiento de los datos de las APIs.

Las APIs precisan de una clase de “store” para guardar los datos que se utilizan. Por lo cual crearemos una clase Store donde realizaremos estas tareas.

```
import { configureStore, getDefaultMiddleware } from "@reduxjs/toolkit";
```

En la anterior imagen podemos observar cómo importamos dos métodos de redux toolkit, el cual nos facilita métodos a la hora de trabajar con APIs.

```
import { cryptoApi } from "../services/cryptoApi";  
import { cryptoNewsApi } from "../services/cryptoNewsApi";
```

En la anterior imagen podemos observar como se importan las APIs previamente explicadas.

```
export default configureStore({  
  reducer: {  
    [cryptoApi.reducerPath]: cryptoApi.reducer,  
    [cryptoNewsApi.reducerPath]: cryptoNewsApi.reducer,  
  },  
  middleware: (getDefaultMiddleware) =>  
    getDefaultMiddleware().concat(cryptoApi.middleware),  
});
```

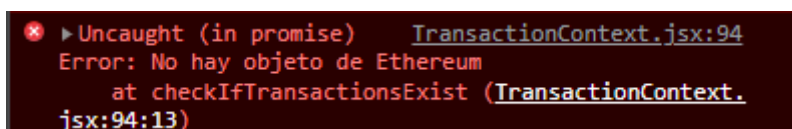
En la imagen anterior podemos observar cómo trabaja el store con las APIs y también como se realiza la optimización de la misma mediante un reducer(ya que las APIs no tienen por qué ser óptimas para el rendimiento y de esta manera podemos asegurarnos de que es más óptimo y menos pesado a la hora de cargarlo en la aplicación web).

5. Testeo de la aplicación.

En el siguiente apartado podremos observar los métodos de testeo y el testeo del mismo para comprobar todos los funcionamientos del proyecto.

5.1. Testeo de la conexión con MetaMask.

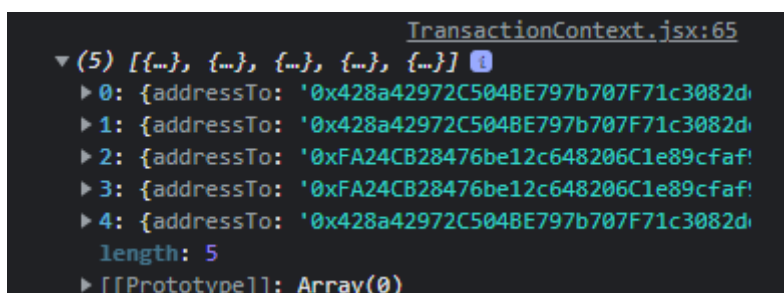
Este testeo consiste en dos partes, primero verificar si la aplicación no detecta ningún usuario conectado, segundo verificar que una vez el usuario conectado la aplicación recoge los datos de este.



```

✖ ▶ Uncaught (in promise) TransactionContext.jsx:94
Error: No hay objeto de Ethereum
    at checkIfTransactionsExist (TransactionContext.
jsx:94:13)
  
```

En la anterior imagen podemos observar como se muestra que el usuario no está conectado.



```

TransactionContext.jsx:65
▼ (5) [{...}, {...}, {...}, {...}, {...}] ⓘ
  ▶ 0: {addressTo: '0x428a42972C5048E797b707F71c3082d'
  ▶ 1: {addressTo: '0x428a42972C5048E797b707F71c3082d'
  ▶ 2: {addressTo: '0xFA24CB28476be12c648206C1e89cfaf'
  ▶ 3: {addressTo: '0xFA24CB28476be12c648206C1e89cfaf'
  ▶ 4: {addressTo: '0x428a42972C5048E797b707F71c3082d'
    length: 5
  ▶ [[Prototype]]: Array(0)
  
```

En la anterior imagen podemos observar como una vez conectado el usuario podemos mostrar las transacciones y datos de este. Esto verifica que el usuario está conectado y podemos recoger sus datos.

5.2. Testeo de transacción.

En este testeo se realizará una transacción y se comprobará que todo funcione de una manera correcta. Se seguirán los siguientes pasos:

- Rellenar el formulario para realizar una transacción.



0xFA24CB28476be12c648206C1e89cfaf9f115Df5B

0,0001

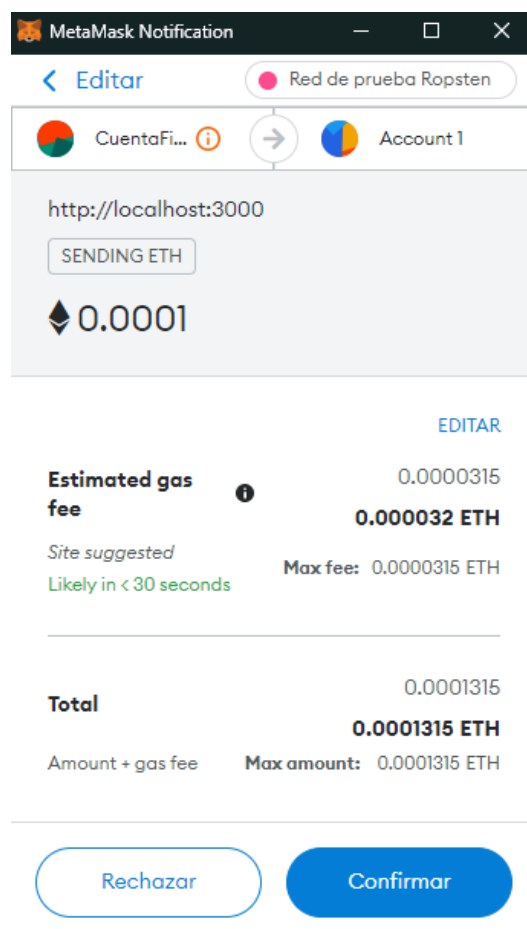
Testeo

Testeo para documentacion

Enviar

En la anterior imagen podemos observar como se rellena el formulario para realizar una transacción.

- Pulsaremos el botón de enviar y se nos abrirá una ventana en MetaMask para confirmar la transacción.



MetaMask Notification

< Editar Red de prueba Ropsten

CuentaFi... Account 1

http://localhost:3000

SENDING ETH

0.0001

EDITAR

Estimated gas fee 0.0000315
0.000032 ETH

Site suggested
Likely in < 30 seconds

Max fee: 0.0000315 ETH

Total 0.0001315
0.0001315 ETH

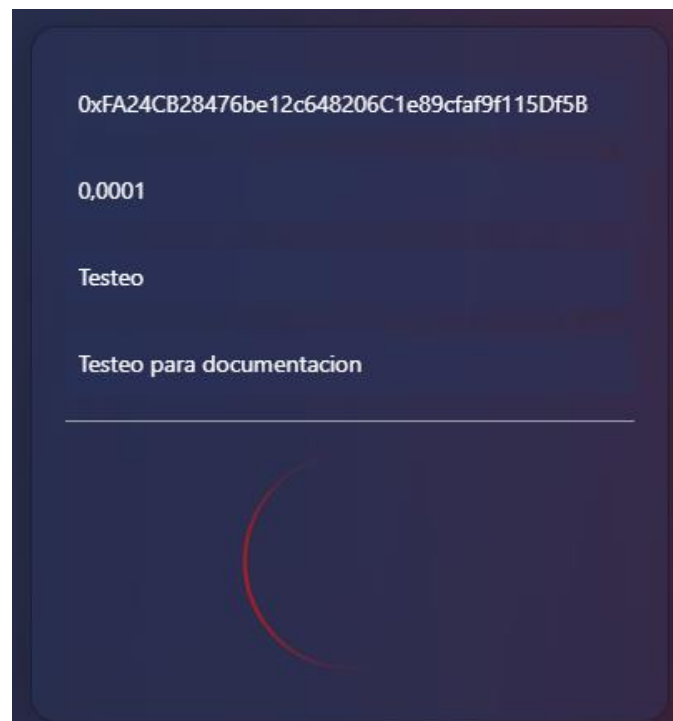
Amount + gas fee Max amount: 0.0001315 ETH

Rechazar Confirmar

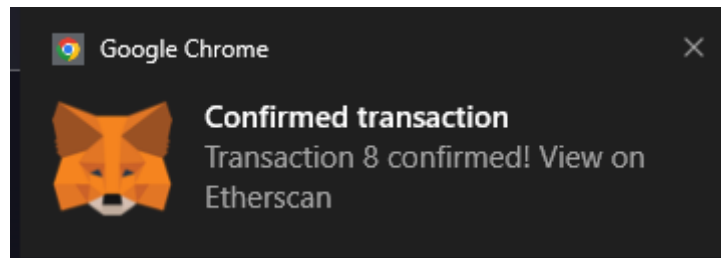
En la anterior imagen podemos observar los datos tales como las comisiones al realizar la transacción y que el usuario deberá de confirmar para realizarla.



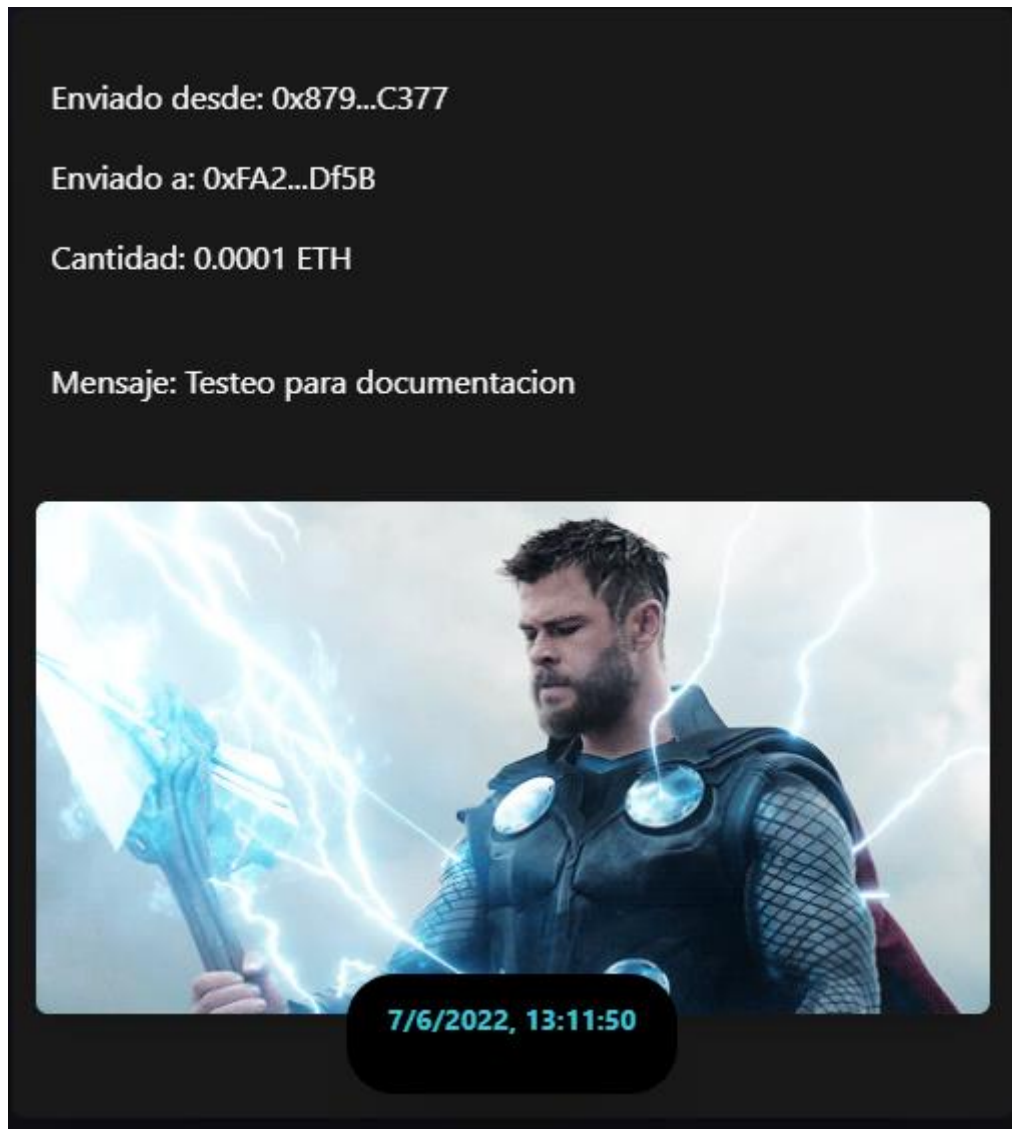
En la anterior imagen podemos observar como se indica que se utilizará el smart contract diseñado para la aplicación y que ya hemos explicado.



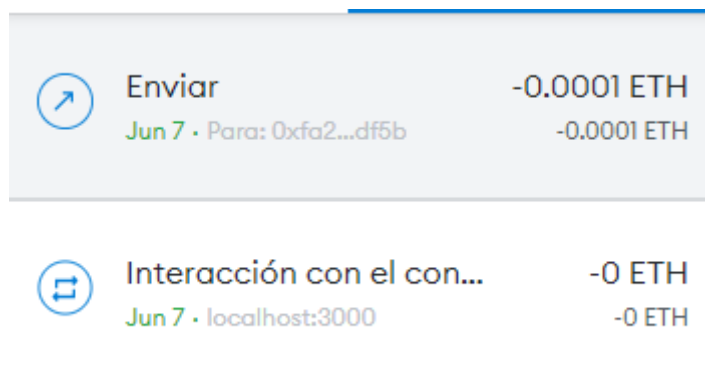
En la anterior imagen podemos observar la animación de “cargando” hasta que se realice la transacción.



En la anterior imagen podemos observar la notificación de transacción confirmada.



En la anterior imagen podemos observar como la transacción ya se ha registrado en el usuario y en el blockchain y podemos mostrarla.

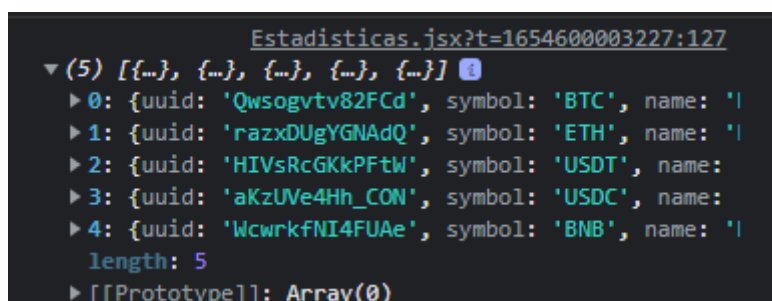


En la anterior imagen podemos observar desde MetaMask la transacción, la cantidad enviada y la interacción con el contrato que no supone ningún coste.

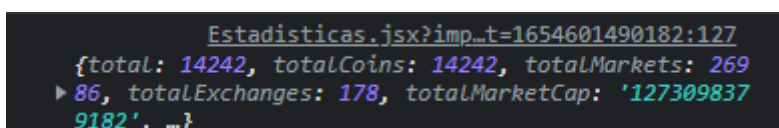
5.3. Testeo de APIs.

5.3.1. Testeo de API CoinRanking.

El testeo de la API se realizará observando los datos que nos devuelve mostrándonos por la consola de la aplicación web, esta nos indicará si la conexión ha sufrido algún error o nos devolverá los datos.



En la anterior imagen podemos observar los datos que nos devuelve la API sobre las criptomonedas que se utilizarán en el top de criptomonedas.



En la anterior imagen podemos observar la muestra de datos de las estadísticas de criptomonedas de la petición que realizamos a la API.

5.3.2. Testeo de Bing News.

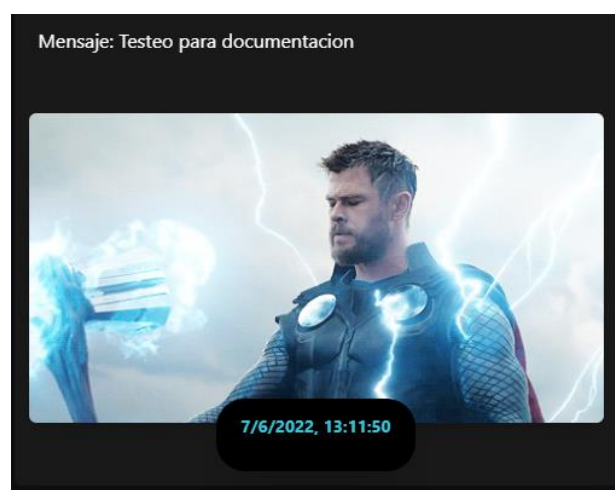
El testeo de la API se realizará observando los datos que nos devuelve mostrándolos por la consola de la aplicación web, esta nos indicará si la conexión ha sufrido algún error o nos devolverá los datos.

```
News.jsx?t=1654601603842:114
{ _type: 'News', readLink: 'https://api.cognitive.mi
crosoft.com/api/v7/news/search?q=Cryptocurrency', q
ueryContext: {...}, totalEstimatedMatches: 44, sort:
Array(2), ...}
  ▶ queryContext: { _type: 'QueryContext', originalQue
    readLink: "https://api.cognitive.microsoft.com/ap
  ▶ sort: (2) [{...}, {...}]
    totalEstimatedMatches: 44
  ▼ value: Array(6)
    ▶ 0: { _type: 'NewsArticle', name: 'Top cryptocurre
    ▶ 1: { _type: 'NewsArticle', name: 'Why Cryptocurre
    ▶ 2: { _type: 'NewsArticle', name: 'Cryptocurrency
    ▶ 3: { _type: 'NewsArticle', name: 'Cryptocurrency
    ▶ 4: { _type: 'NewsArticle', name: 'Amid crypto tur
    ▶ 5: { _type: 'NewsArticle', name: 'Cryptocurrency
      length: 6
    ▶ [[Prototype]]: Array(0)
    _type: "News"
  ▶ [[Prototype]]: Object
```

En la anterior imagen podemos observar la recogida de datos de las noticias. Podemos observar la query con la cual se realiza esta petición a la API.

5.4. Testeo de API Giphy.

El testeo de la API se realizará a la hora de realizar una transacción y visualizar que el GIF que se ha introducido en la transacción es correcto. En el caso de que la keyword no coincida se introducirá uno de manera predeterminada.



En la anterior imagen podemos observar el GIF introducido al realizar una transacción.

TECNOLOGÍA

En este apartado mostraré y detallare las tecnologías que se han utilizado a lo largo de todo el proyecto para conseguir el resultado final. Se dividirán en 5 grupos: tecnologías para el desarrollo de la aplicación web, tecnologías para el desarrollo del entorno blockchain, lista de todas las dependencias utilizadas en el proyecto, el trabajo con APIs externas y el procesamiento de datos que llevan detrás y por último las tecnologías con las que se ha trabajado a lo largo del proyecto que no tienen por qué estar en el resultado final pero sin ellas no hubiese sido posible la realización de este.

Se mostrarán las herramientas utilizadas para el desarrollo del proyecto.

Tecnologías utilizadas para el desarrollo de la aplicación:

- **React.** Biblioteca de JavaScript para crear interfaces de usuario. En este proyecto se ha utilizado para crear todas las interfaces de usuario.
- **CSS.** Lenguaje de estilo de diseño gráfico. Utilizado para dar diseño a la aplicación.
- **JavaScript.** Lenguaje de programación utilizado para el desarrollo del proyecto.
- **Giphy.** Biblioteca de Gifs implementada, cada vez que un usuario realice una transacción al introducir la KeyWord está buscará un gif de esta biblioteca y lo asociará a la transacción. De esta manera al ver las transacciones no veremos tan solo números y datos sino que también un Gif.

Tecnologías utilizadas para el desarrollo del entorno blockchain:

-
- **Solidity.** Lenguaje de programación orientado a objetos para escribir contratos inteligentes. Utilizado en este proyecto para desarrollar el Smart Contract sobre el que se realiza las transacciones.
- **JavaScript.**

Lista de dependencias NodeJs utilizadas:

-
- **Ant Desing:**
 - o **Ant Design Icons.** Biblioteca de React y lenguaje de diseño para construir interfaces, en este proyecto se ha utilizado para conseguir algunos iconos.
- **ReduxJs:**
 - o **ReduxJs toolkit.** Librería de JavaScript para el manejo de estado de las aplicaciones. En este proyecto utilizado el toolkit que proporciona soporte para la biblioteca immer.js que cambia el código de manera inmutable. Utilizado para los scripts de las APIs.
- **TailwindCss:**

- **Tailwind Css forms.** Framework utilizado para crear diseños a medida. Utilizado a lo largo del proyecto para diseño y pequeños detalles.
- **Antd.** Kit de interfaz de usuario.
- **Postcss.** Herramienta de JavaScript que automatiza las funciones del código CSS.
- **Autoprefixer.** Plugin de PostCSS que permite que el código CSS se adapte a todos los navegadores/circunstancias.
- **Axios.** Cliente HTTP basado en promesas para NodeJS y el navegador.
- **Ethers.** Librería que permite interactuar con Ethereum y su ecosistema. Utilizado en el proyecto para esto mismo.
- **Framer motion.** Librería de animaciones de React. Utilizada en el proyecto para el Loader de la página.
- **HTML React parser.** Utilizado para convertir HTML String a elementos de React.
- **Millify.** Utilizado para convertir números grandes e ilegibles en números que se puedan leer.
- **Moment.** Librería de JavaScript que permite trabajar de una manera cómoda con fechas. Utilizada para recoger la fecha correcta a la hora de realizar una transacción.
- **React:**
 - **React dom,** librería que proporciona métodos específicos del DOM.
 - **React icons,** librería de iconos .
 - **React redux.**

Tecnologías utilizadas para el desarrollo y testeo del proyecto:

-
- **NodeJs.** Utilizada para descargar las dependencias necesarias para el proyecto.
- **Npm.** Utilizado como gestor de paquetes NodeJs.
- **Vite.** Herramienta frontend utilizada para el desarrollo de la aplicación y como servidor local para el testeo de esta.
- **Hardhat.** Entorno de desarrollo de Ethereum, utilizado para compilar el Smart contract y subirlo a la red blockchain.

Herramientas utilizadas para el desarrollo del proyecto.

- **Visual Studio Code.** Utilizado como editor de código, utilizado para trabajar con la terminal que contiene el mismo. Gestionar el control de versiones.
- **Google Chrome/Edge.** Visualizar a tiempo real el avance del proyecto, además de visualizar datos por la consola del mismo.

APIs implementadas:

- **CoinRanking.** API que nos ofrece una gran cantidad de datos sobre criptomonedas, entorno blockchain. En este proyecto se ha utilizado para recoger los últimos datos y valores sobre criptomonedas y ofrecerlas en la aplicación como información.
- **Bing News.** API que nos ofrece un gran catálogo de noticias y novedad al momento. En este proyecto se ha implementado para ofrecer al usuario las últimas novedades enseñando las noticias de una manera breve y ofreciendo el enlace en caso de que al usuario le interese más.
- **RapidAPI.** Portal de APIs que se ha utilizado para el previo testeo de estas APIs. Portal que ofrece una gran cantidad de APIs y cómo poder implementarlas en un proyecto.

METODOLOGÍA

Metodología usada en el proyecto.

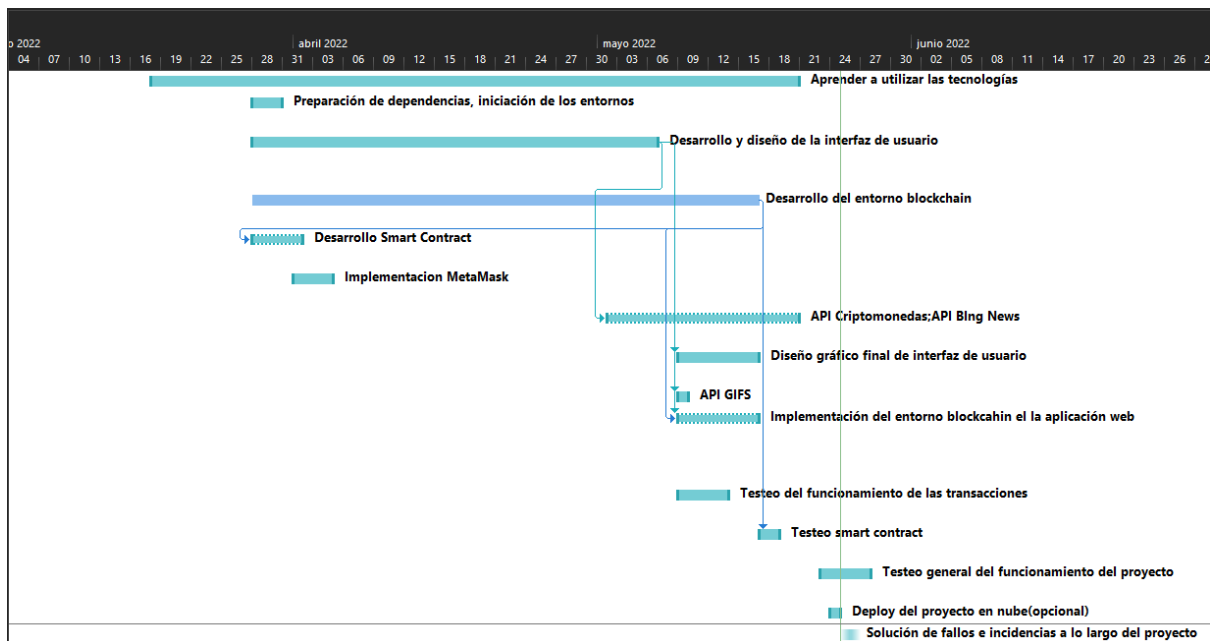
La metodología utilizada para la realización de este proyecto ha sido una metodología de trabajo personal. Desarrollando cada parte del proyecto hasta que fuese terminada y una vez terminada esa tarea empezar la siguiente. En el caso de tareas más extensas como la interfaz de la aplicación web o el desarrollo del entorno blockchain.

Diagrama de Gantt.

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Aprendizaje	46 días	vie 18/03/22	vie 20/05/22		Aprender a utilizar las tecnologías
Preparación inicial del proyecto	3 días	lun 28/03/22	mié 30/03/22		Preparación de dependencias, iniciación de los entornos
Desarrollo y diseño de la interfaz de usuario	30 días	lun 28/03/22	vie 06/05/22		Desarrollo y diseño de la interfaz de usuario
Desarrollo del entorno blockchain	36 días	lun 28/03/22	lun 16/05/22		Desarrollo del entorno blockchain
Desarrollo de Smart Contract	5 días	lun 28/03/22	vie 01/04/22	4	Desarrollo Smart Contract
Implementación MetaMask	2 días	vie 01/04/22	lun 04/04/22		Implementacion MetaMask
Implementación de APIs	15 días	lun 02/05/22	vie 20/05/22	3	API Criptomonedas;API Blng News
Diseño de la interfaz gráfica del usuario	6 días	lun 09/05/22	lun 16/05/22	3	Diseño gráfico final de interfaz de usuario
API GIFS	1 día	lun 09/05/22	lun 09/05/22	3	API GIFS
Desarrollo de implementación de aplicación con blockchain	6 días	lun 09/05/22	lun 16/05/22	3;4	Implementación del entorno blockcahin el la aplicación web
Testeo de transacciones	5 días	lun 09/05/22	vie 13/05/22	4	Testeo del funcionamiento de las transacciones
Testeo Smart Contract	2 días	mar 17/05/22	mié 18/05/22	4	Testeo smart contract
Testeo general del	5 días	lun	vie		Testeo general del

proyecto		23/05/22	27/05/22		funcionamiento del proyecto
Deploy en nube	1 día	mar 24/05/22	mar 24/05/22		Deploy del proyecto en nube(opcional)
Solución de errores	2 días				Solución de fallos e incidencias a lo largo del proyecto

En la siguiente figura podemos observar el diagrama de Gantt de una manera gráfica:



Para la comprobación y constatación de las tareas y tiempos realizados en el proyecto se puede constatar con las entregas realizadas en Microsoft Planner, GIT, diagramas de caso de uso y el diagrama RFTP.

TRABAJOS FUTUROS

Tras la finalización de este proyecto no sé si seguiré desarrollando este proyecto para una futura comercialización de este o puesta en escena a nivel mayor.

Los conocimientos adquiridos mientras lo he realizado es una cosa que siempre llevaré conmigo.

CONCLUSIONES

Realizar este proyecto considero que ha sido un gran desafío y una manera de poner a prueba los conocimientos estudiados en Desarrollo de Aplicaciones Multiplataforma.

El proyecto ha sido extenso y con muchas horas de dedicación, ya no solo en cuanto a desarrollarlo sino en cuanto a aprender sobre estas nuevas tecnologías, entenderlas y aprender a trabajar con ellas. Cada paso del proyecto ha sido un desafío.

Realizar una aplicación web me ha permitido desarrollar una buena habilidad trabajando con ellas, ya que se sale un poco de la manera de trabajar que he tenido previamente a este proyecto.

Trabajar con nuevas tecnologías, como el ecosistema blockchain y desarrollar en él, me ha brindado la oportunidad de aprender muchos conocimientos y tecnologías. Descubrir un mundo muy moderno y tecnológico que me apasiona y que estoy seguro de que de seguiré trabajando en él.

Cuando decidí realizar este proyecto sabía que me estaba poniendo un desafío complejo al trabajar con tecnologías nuevas. Leer mucha documentación sobre los desarrolladores de las tecnologías ha sido la manera más eficaz de entender cómo funcionan las mismas.

Incorporar en el proyecto el trabajo con APIs externas, considero que ha sido una buena práctica y descubrir un mundo que no conocía y el gran catálogo de APIs y la cantidad de oportunidades que te pueden ofrecer al trabajar con ellas me ha llevado a tenerlas en cuenta a la hora de realizar cualquier proyecto.

La conclusión profesional que tengo tras terminar este proyecto es que en cuanto al trabajo o proyectos personales hay que buscar algo siempre que te apasione y que tengas ganas de trabajar con ello sea más duro o más fácil. Aprender nuevas tecnologías y estar siempre en lo último, renovarse.

REFERENCIAS

- Documentación React.* (2022). React. <https://es.reactjs.org/docs/getting-started.html>
- Documentación Giphy.* (2022). Giphy. <https://developers.giphy.com/docs/sdk#web>
- Documentación Solidity.* (2022). Solidity. <https://solidity-es.readthedocs.io/es/latest/>
- Documentación AntDesign.* (2022). AntDesign. <https://ant.design/docs/react/introduce>
- Documentación Redux.* (2022). Redux. <https://es.redux.js.org/>
- Documentación TailwindCSS.* (2022). TailwindCSS. <https://v2.tailwindcss.com/docs>
- Documentación PostCss.* (2022). PostCss. <https://github.com/postcss/postcss/tree/main/docs>
- Documentación Autoprefixer.* (2022). Autoprefixer. <https://openbase.com/js/autoprefixer>
- Documentación Ethers.* (2022). Ethers: <https://docs.ethers.io/v5/>
- Documentación Millify.* (2022). Millify: <https://openbase.com/js/millify/documentation>
- Documentación Moment.* (2022) Moment: <https://momentjs.com/docs/>
- Documentación NodeJs.* (2022). NodeJs: <https://nodejs.org/es/docs/>
- Documentación Npm.* (2022). Npm: <https://docs.npmjs.com/packages-and-modules>
- Documentación Vite.* (2022). Vite: <https://vitejs.dev/guide/why.html>
- Documentación HardHat.* (2022). HardHat: <https://hardhat.org/getting-started/>
- Documentación RapidAPI.* (2022). RapidAPI: <https://docs.rapidapi.com/>
- Documentación API Bing News Search.* (2022). API Bing News Search: <https://www.microsoft.com/en-us/bing/apis/bing-news-search-api>
- Documentación Coinranking.* (2022). Coinranking: <https://developers.coinranking.com/api/documentation>