
Decoinpay: Una aplicación web para ayudar a las empresas a aceptar pagos en criptomonedas



Proyecto para premio Don Bosco
Edición 35

Autor

Diego Blas Gastón

Tutor

Gorka Sanz Lopategui

Área: Innovación tecnológica

Categoría: Tecnologías de la Información y comunicación

Centro docente: Salesianos zaragoza, España

Resumen

Decoinpay, una aplicación web, que permite a las empresas aceptar pagos con criptomonedas en sus establecimientos, ya sean físicos u online. El proyecto consta de dos componentes, el servidor y el cliente. El servidor integra una API para poder acceder a la base de datos privada de Decoinpay y así poder ver todos los datos del usuario. El lado del cliente está diseñado para que sea fácil e intuitivo, permitiendo que incluso un usuario nuevo en las criptomonedas pueda realizar una transacción o ver todos los pagos que ha realizado o recibido con solo hacer clic en un botón. El estar basado en blockchain y tener un sistema de usuarios gestionado a través de una cartera virtual es lo que le hace a Decoinpay no ser una web de la Web 2.0 sino de la Web 3.0.

La página web es accesible a través del siguiente enlace:

[*https://decoinpay.com*](https://decoinpay.com)

El código de Decoinpay se encuentra en el siguiente repositorio de GitHub:

[*https://github.com/DiegoBlasg/Decoinpay*](https://github.com/DiegoBlasg/Decoinpay)

Palabras clave

Blockchain, criptomoneda, Metamask, web 3.0, React, nodejs, No-SQL, API, API REST, Docker, Nginx, Cartera virtual.

Índice

1 Introducción	4
1.1 Motivación	4
1.2 Objetivos	4
1.3 Organización de trabajo	5
2 Tecnologías usadas	6
2.1 React	6
2.2 Express	6
2.3 Mongoose	6
2.4 MongoDB	6
2.5 Node	7
2.6 Nginx	7
2.7 Docker	7
2.8 Bibliotecas de JavaScript	7
2.9 Bootstrap	8
2.10 Git	8
2.11 DigitalOcean	8
2.12 Let's Encrypt	8
3 Backend	9
3.1 Estructura de archivos	9
3.2 Carpeta src	9
3.2.1 database.js	10
3.2.2 app.js	10
3.2.3 index.js	10
3.2.4 API REST y seguridad	10
3.2.5 Carpeta Models	11
3.2.6 Carpeta Routes	11
3.2.7 Carpeta Controllers	11
3.3 Carpeta TerminalApp	11
4 Funcionamiento del frontend	12
4.1 Precios	12
4.1.1 Funcionamiento	13
4.2 Cuenta	13



4.2.1 Funcionamiento	14
4.3 API	14
4.4 Contratos	15
4.4.1 Funcionamiento	16
4.5 Gestión de usuarios	18
4.6 Estructura de archivos	19
5 API publica	20
6 Despliegue de la aplicación.....	21
6.1 Docker	21
6.2 Nginx	21
6.3 DigitalOcean.....	22
6.4 GitHub	22
7 Puntos de mejora	23
7.1 API de metamask	23
7.2 Futuras actualizaciones	24

Capítulo

1

En este capítulo se explica la motivación para desarrollar Decoinpay y presentarlo al concurso, los objetivos del proyecto y la organización de trabajo que se ha seguido.

Introducción

1.1 Motivación

Desde que me adentré en el mundo de las criptomonedas fui investigando todo lo que había detrás de ellas y descubrí un mundo con un gran potencial para la tecnología, la blockchain, un sistema capaz de autogestionarse sin la necesidad de ningún intermediario, todo a través de internet siendo además prácticamente imposible de hackear hoy en día.

Este sistema blockchain, cuanta más gente lo conozca y entienda más posibles avances tecnológicos se podrán conseguir, de ahí nace la idea de Decoinpay, una web 3.0 que ofrece un sistema a las empresas para poder manejar la gestión de pagos con criptomonedas y así adentrarse en este mundo ya no solo por las criptomonedas si no por toda la tecnología blockchain.

1.2 Objetivos

Hay varios objetivos a cumplir en este proyecto, el principal es que las empresas, sin importar si son grandes o pequeñas puedan aceptar pagos con criptomonedas de forma gratuita, rápida, segura y de forma descentralizada, aparte de tener una buena gestión de dichos pagos, de la forma más sencilla posible ya tengan una tienda online o física.

Aparte del objetivo principal existen otros objetivos casi igual de importantes, como por ejemplo que cada vez más personas se adentre en el mundo de las criptomonedas hasta hacer de ellas algo normal y poder obtener de ellas todos los beneficios como la sencillez para usarlo, la seguridad, la descentralización, los bajos costes, la rapidez y confidencialidad de las transacciones, evitar la falsificación etc., también, otro objetivo sería ayudar a las empresas a aprovechar todo el potencial que tiene el mundo de las criptomonedas ya que están en continuo crecimiento.

Un objetivo también muy importante de esta plataforma es ser lo menos intermediario y lo más descentralizado posible, lo menos intermediario posible para que las empresas ya sean grandes o pequeñas puedan aceptar pagos sin comisiones y que sea lo más sencillo posible sin que haya una entidad poniendo obstáculos de por medio, y lo más descentralizado posible para que la relación cliente negocio solo sea entre el cliente y el negocio y así aumentar la confianza, ya que el manejo de los fondos no recae en la plataforma si no que los usuarios mantienen en todo momento el total control de sus activos.

1.3 Organización de trabajo

Desde la fecha de inicio, entre noviembre y diciembre, el proyecto se dividió en 3 fases: la fase de aprendizaje, la de desarrollo y la fase final del proyecto.

En la primera fase, la de aprendizaje, consistía en aprender lo básico de las diferentes tecnologías que se utilizarían posteriormente en el proyecto, aprendiendo lo básico de react, de node, de express y de mongodb, esta fase tenía una duración prevista de hasta mitad de diciembre la cual se cumplió sin problema.

Durante la segunda mitad de diciembre fue cuando empezó la fase 2 del proyecto la cual consistía en desarrollar la aplicación con actualizaciones estables cada semana, con un plazo previsto que acabase en febrero el cual se cumplió incluso unos días antes gracias a las vacaciones de navidad lo cual permitió dedicarle muchas más horas al proyecto.

Por último, la fase final, en la que quedaba todo lo relacionado con el concurso como hacer la memoria, preparar la presentación del proyecto, hacer el video y el cartel.

Capítulo 2

En este capítulo se muestran las tecnologías usadas, con una pequeña descripción de que es lo que hacen, y su uso en el proyecto.

Tecnologías usadas

2.1. React

Es una biblioteca de JavaScript de código abierto creada por Facebook para desarrollar interfaces de una manera más ordenada, pudiendo con una de sus muchas funcionalidades crear componentes reutilizables, con esta tecnología se ha desarrollado el frontend de Decoinpay.

2.2. Express

Express.js, es un marco de aplicación web de backend para Node.js, está diseñado para crear aplicaciones web y API, esta tecnología junto a mongoose son con las que se ha desarrollado la API REST de Decoinpay.

2.3. Mongoose

Mongoose es una biblioteca de programación orientada a objetos de JavaScript que crea una conexión entre MongoDB y el marco de la aplicación web Express.

2.4. MongoDB

Es un sistema de base de datos NoSQL, orientado a documentos y de código abierto, en lugar de guardar datos en tablas, MongoDB guarda estructuras de datos

JSON, similar a BSON, este sistema se ha utilizado como base de datos de Decoinpay.

2.5. Node

Node.js es un entorno controlado por eventos diseñado para crear aplicaciones escalables, permitiendo establecer y gestionar múltiples conexiones al mismo tiempo. Toda la aplicación de Decoinpay funciona entorno a node como su punto de unión gracias a su gestor de paquetes, npm.

2.6. Nginx

Nginx es un servidor web/proxy inverso ligero de alto rendimiento, En Decoinpay se ha utilizado para poder usar toda la página desde la misma ip y desde el mismo puerto y poder junto a Let's Encrypt, explicado en el capítulo 2.12, obtener el certificado SSL y así tener la página usando el protocolo https que le aporta mayor seguridad.

2.7. Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos, tecnología que ha sido útil para que la web funcione igual tanto en el ordenador local como en la máquina virtual en la nube y que junto con GitHub Actions obtenga una facilidad mayor a la hora de desplegar la aplicación como se explica en el capítulo 6.3.

2.8. Bibliotecas de JavaScript

Una biblioteca de JavaScript es una biblioteca de código pre-escrito en JavaScript que permite un desarrollo más fácil de aplicaciones basadas en JavaScript, las bibliotecas utilizadas en este proyecto son las siguientes: dotenv, crypto-js, ethers, chart.js, axios, commander, cors y filesystem. Estas bibliotecas se explican con detalle en el capítulo 3.

2.9. Bootstrap

Bootstrap es un framework de maquetación CSS o un conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web, Bootstrap se ha utilizado para el diseño y la maquetación de Decoinpay.

2.10. Git

Git es un software de control de versiones, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. En Decoinpay se ha utilizado para poder tener un seguimiento de las actualizaciones del código, y prevenir algún posible error o pérdida de datos.

2.11. DigitalOcean

DigitalOcean es un proveedor estadounidense de servidores virtuales privados. La compañía alquila instalaciones de centros de cómputo existentes, en Decoinpay se ha utilizado para alojar la página web en una máquina virtual de esta empresa.

2.12. Let's Encrypt

Let's Encrypt es una autoridad de certificación que proporciona certificados X.509 gratuitos para el cifrado de seguridad de nivel de transporte, en Decoinpay se ha utilizado junto con nginx para que la página pueda estar activa a través del protocolo https y así otorgarle una mayor seguridad.

Capítulo 3

En este capítulo se muestra todo el funcionamiento del backend, que es una API REST y el cómo y por qué se ha creado este sistema de conexión con la base de datos.

Backend

El backend, realizado con express y mongoose explicados anteriormente en los capítulos 2.2 y 2.3, es la parte de la aplicación que se encarga de gestionar las peticiones recibidas del frontend y manipularlos para realizar diferentes funciones en la base de datos, en cuanto a las librerías usadas las cuales se explican más adelante se encuentran: commander y file-system para la aplicación de terminal, crypto-js para la encriptación de datos y cors para el intercambio de datos de origen cruzado.

3.1 Estructura de archivos

El backend del proyecto se estructura en 2 carpetas fundamentales, la primera carpeta, TerminalApp en la que se ha creado una aplicación de terminal para el visionado y guardado de copias de seguridad de los datos de la base de datos de Decoinpay todo a través de la propia terminal, y en la segunda carpeta, src, es en donde se encuentra todo el código de la API REST.

3.2. Carpeta src

Dentro de esta carpeta se encuentra el código fuente de la API REST de Decoinpay la cual se compone de 3 modulos separados en carpetas: Controllers, Models y Routes y tres archivos fundamentales para la configuración del backend, app.js, database.js index.js.

3.2.1 database.js

Este archivo se utiliza para poder conectarse a la base de datos, en el se define la ruta donde se encuentra y toda la configuración necesaria para poder realizar la conexión.

3.2.2 app.js

En este archivo es donde se configura el backend: que rutas están disponibles, en que puerto se encuentra, y que midelwares se usan.

Las rutas especificadas en este fichero son links a los ficheros de la carpeta Routes, esta refactorización se hace para una mayor organización del código.

3.2.3 index.js

Este archivo es el que se ejecuta por defecto al llamar al backend, el cual hace de nexo entre los ficheros app.js y database.js.

3.2.4 API REST y seguridad

Una API REST es una aplicación web en el lado del backend en la que se crean una serie de rutas configuradas con una serie de métodos los cuales hacen ciertas funcionalidades como la conexión a la base de datos, esta API puede ser consumida mediante el protocolo http por peticiones get, post, delete, put...

En cuanto a la seguridad del backend de Decoinpay, la API se ha configurado para que sea necesario que cada petición, ya sea post, get o de cualquier tipo contenga una cabecera con la wallet (identificador de la cartera virtual) que proporciona metamask, encriptada con una contraseña propia de Decoinpay, esa wallet se descripta en el backend y se utiliza para devolver solo los datos que le pertenezcan al usuario con wallet, así el backend se asegura que solo el frontend de Decoinpay ha realizado la consulta y que el usuario que pide los datos es realmente quien dice ser.

3.2.5 Carpeta Models.

Para crear la base de datos primero se tiene que definir los modelos o esquemas de datos, para ello en la carpeta llamada Models se encuentran los 3 esquemas que componen la base de datos de Decoinpay separados en 3 archivos: Contracts.js, Users.js y Transactions.js, en ellos se define que propiedades tiene el esquema y de que tipo son, String, Boolean, int...

3.2.6 Carpeta Routes

En este módulo se definen las rutas que serán utilizadas por el frontend a través del protocolo http para la interacción con los datos de la base de datos, para ello, cada ruta tiene diferentes métodos, get, post, delete o put, y cada método se le asigna una función que se encuentra en la carpeta Controllers, así entonces, si el frontend hace por ejemplo una petición http get a una ruta definida en los ficheros de esta carpeta, el backend ejecutara una función, que responderá enviando al frontend un archivo json. Esta carpeta se compone de 3 archivos: contracts.js, users.js y transactions.js uno por cada esquema.

3.2.7 Carpeta controllers

Por último, en el módulo Controllers, con un archivo por cada esquema: contracts.controllers.js, users.controllers.js y transactions.controllers.js, se encuentran todas las funciones que se ejecutan cuando se entran a las diferentes rutas, ya sean de buscar, agregar, borrar, o actualizar datos.

3.3 Carpeta TerminalApp

En esta carpeta se encuentra una aplicación de terminal llamada decoinvew creada para poder ver los datos de la base de datos y poder guardar esos datos en archivos .json y así tener copias de seguridad, todo desde la terminal, para ello se ha utilizado la librería de javascript commander, en un archivo commands.js se han especificado los diferentes comandos que se pueden ejecutar, y los diferentes parámetros que se le pueden pasar a dicho comando, y en el módulo Controllers, se han especificado los diferentes métodos que se ejecutan al ejecutar cada comando.

Capítulo 4

Funcionamiento del frontend

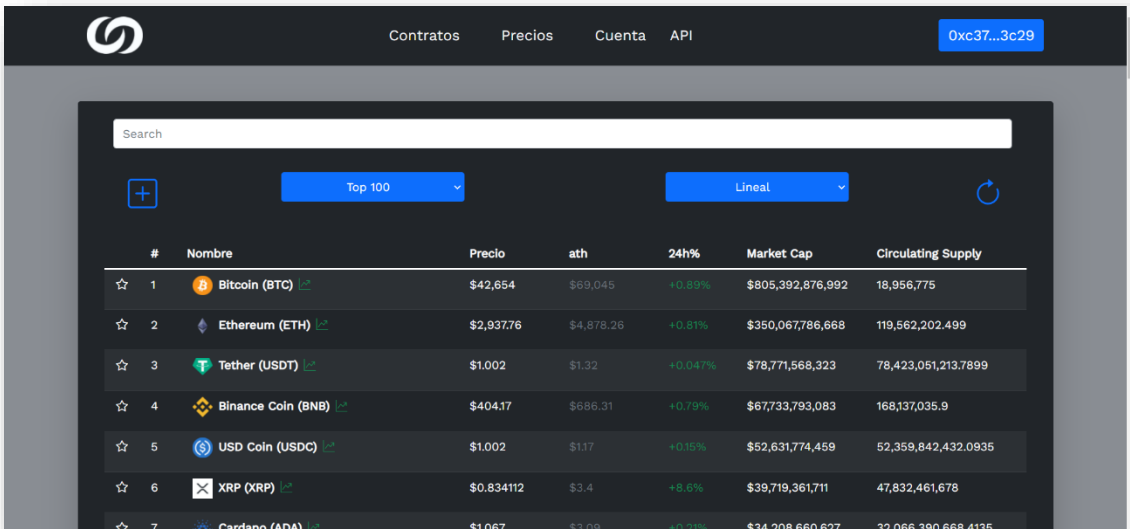
La plataforma de Decoinpay se divide en 4 apartados principales en el que el usuario puede navegar: Contratos, Precios, Cuenta y API.

4.1 Precios

Este apartado es el primero que se muestra al abrir la página, en él se pueden observar los precios del top 100 de criptomonedas ordenadas de mayor a menor Market Cap.

Se puede elegir entre dos modos de vista: lineal, en el cual las criptomonedas aparecen en una tabla, o cuadrícula, en la que las criptomonedas aparecen en forma de cartas, estas criptomonedas se pueden añadir a favoritos para posteriormente poderlas ver en el apartado de favoritas, en el que también se pueden añadir nuevas criptomonedas que no aparezcan en el top 100 añadiendo solo el contrato de dicha moneda.

Este apartado también tiene un sistema para poder filtrar las criptomonedas con un buscador, buscando por el nombre o abreviatura de la moneda.



The screenshot shows the 'Precios' section of the Decoinpay interface. It features a search bar at the top, a 'Top 100' filter, and a 'Lineal' view selector. Below these is a table listing the top 10 cryptocurrencies by market cap. Each row includes a star icon for favorites, a rank, the coin's name and symbol, its price, all-time high (ath), 24-hour percentage change, market capitalization, and circulating supply.

#	Nombre	Precio	ath	24h%	Market Cap	Circulating Supply
1	Bitcoin (BTC)	\$42,654	\$69,045	+0.83%	\$805,392,876,992	18,956,775
2	Ethereum (ETH)	\$2,937.76	\$4,878.26	+0.81%	\$350,067,786,668	119,562,202.499
3	Tether (USDT)	\$1.002	\$1.32	+0.047%	\$78,771,568,323	78,423,051,213.7899
4	Binance Coin (BNB)	\$404.17	\$686.31	+0.79%	\$67,733,793,083	168,137,035.9
5	USD Coin (USDC)	\$1.002	\$1.17	+0.19%	\$52,631,774,459	52,359,842,432.0935
6	XRP (XRP)	\$0.834112	\$3.4	+8.6%	\$39,719,361,711	47,832,461,678
7	Cardano (ADA)	\$1.067	\$3.09	+0.27%	\$34,208,660,627	32,066,390,668.4135

4.1.1 Funcionamiento

Para poder mostrar los precios de las criptomonedas se utiliza la API gratuita de CoinGecko la cual nos devuelve un array de objetos con todos los datos de las 100 primeras criptomonedas. Para mostrárselas al usuario, con un map se recorre todo el array y se muestra por pantalla.

Para poder añadir a favoritas, se hace uso de la base de datos en la que, con algoritmos específicos, al pulsar en la estrella envía una petición al servidor para añadir esa moneda a un array de favoritas que posee cada usuario e igual para eliminarla, y en el momento que se seleccione la opción de ver favoritos simplemente se recorre el array de favoritas que tiene el usuario y se muestran por pantalla.

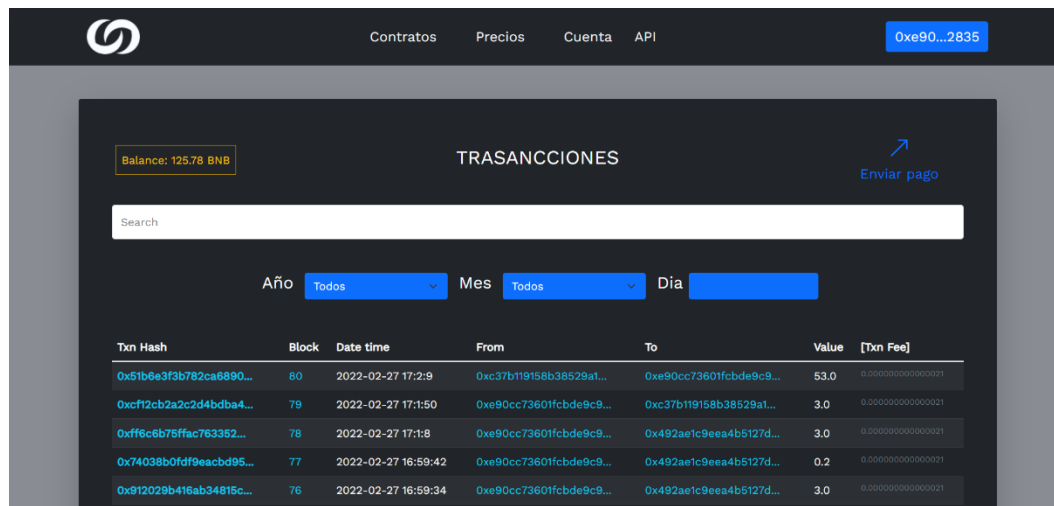
Para poder añadir nuevas criptomonedas se utiliza la API de Pancakeswap, la cual es una API gratuita a la que le puedes pedir información de cualquier criptomoneda o token que se encuentre dentro de la blockchain de BSC, solo pasándole su contrato (identificador del token). Todos los contratos que añade el usuario se guardan en la base de datos en un array llamando `added_tokens`, y cuando se quiere visionar las monedas, se recorre ese array haciendo la petición a la API de Pancakeswap por cada contrato.



Para poder realizar el filtro al buscar, cuando el usuario escribe algo en el buscador, se vuelve a ejecutar la llamada a la API de Pancakeswap o de CoinGecko pero esta vez solo se muestran las monedas que contengan en su nombre el texto que ha escrito el usuario.

4.2 Cuenta

En este apartado se pueden ver la cantidad de BNB que posee la cuenta, realizar transacciones pichando en el botón de enviar pago, ver todas las transacciones realizadas o recibidas pudiéndolas buscar por hash, bloque, cartera desde donde se envía o la cartera que la ha recibido, aparte, también tiene un filtro por fecha.



4.2.1 Funcionamiento

Para poder ver las transacciones, cada usuario en la base de datos tiene un array de transacciones, así pues, cada vez que se carga la página se recorre ese array y se muestran todas las transacciones que ha realizado el usuario, además de todas las que ha recibido, siempre y cuando la transacción no se haya realizado en un contrato. Si en la barra de search no hay nada se mostraran todas las transacciones, pero si sí que hay texto, se mostraran solo las transacciones en la que, en uno de sus campos, coincida con el texto escrito.

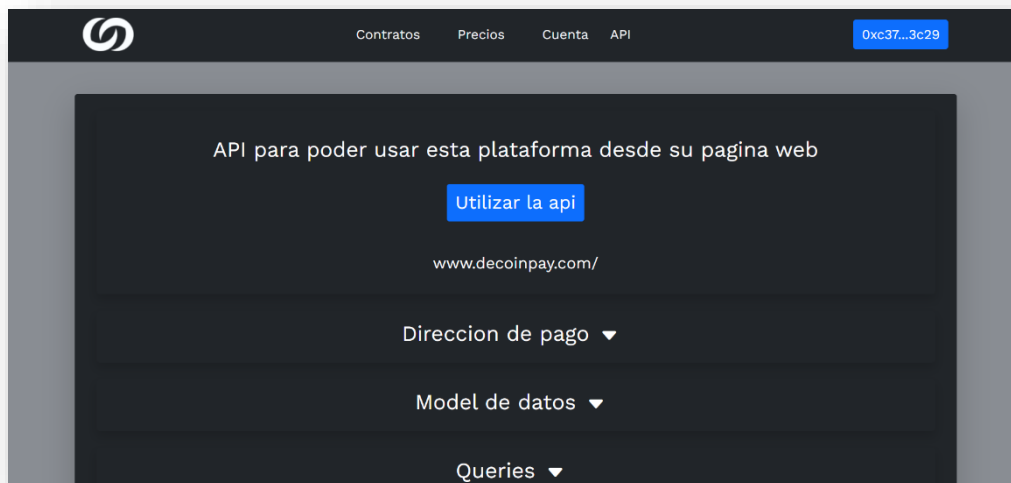
Para el visionado de los BNBs que posee la cuenta se utiliza la API de metamask con el objeto window.ethereum y su método getBalance(), que devuelve la cantidad de BNBs que posee la cuenta.

Para enviar un pago, pulsando en el botón de enviar pago se abre un modal en el que hay que insertar la cartera a la que va dirigida la transacción y el importe que se desea pasar, una vez ingresados los datos y dado a enviar, se ejecuta una función creada usando la librería de ethers-js para poder conectarse a la blockchain, se abre la extensión de metamask para que se acepte la transacción y una vez realizada guarda los datos en el array de transacciones para poder posteriormente mostrársela al usuario.

4.3 API

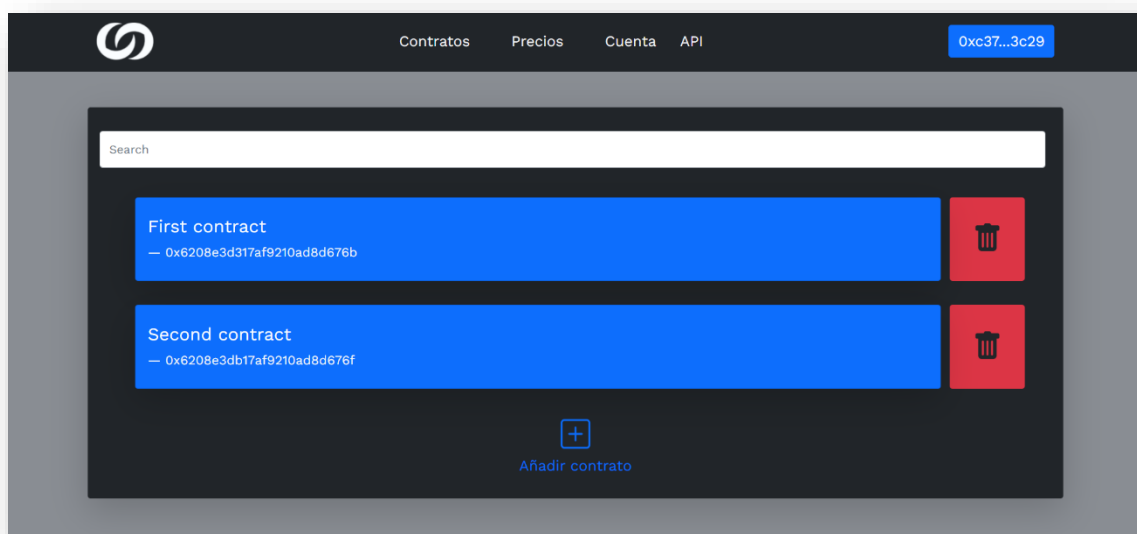
Este apartado muestra la información de uso de la API publica de Decoinpay la cual se explica en el capítulo 5, para que un usuario propietario de una página web pueda aceptar pagos usando dicha API.

En esta página se puede ver la información sobre: cuáles son las claves de acceso de cada contrato para poder usar la API, claves de acceso las cuales se tienen que poner en la cabecera de cada petición, cosa que se explica en el desplegable de Queries, dirección a la que se tendrá que redirigir el usuario para que pueda pagar, cual es el modelo o esquema de datos de la base de datos que guarda la información de las transacciones y todas las consultas que se pueden hacer a la API de Decoinpay y cómo debe estar estructurada.



4.4 Contratos

Este apartado es el principal objetivo de este proyecto, en él se puede gestionar todos los contratos o pasarelas de pago, un contrato sirve para poder recibir transacciones y poder tener un control y un seguimiento de ellas, en este apartado se puede: crear, modificar y eliminar contratos, entrar a ver todas las transacciones que el contrato ha recibido, generar QR para que el contrato reciba pagos y añadir usuarios permitidos para que ellos también puedan gestionar el contrato.



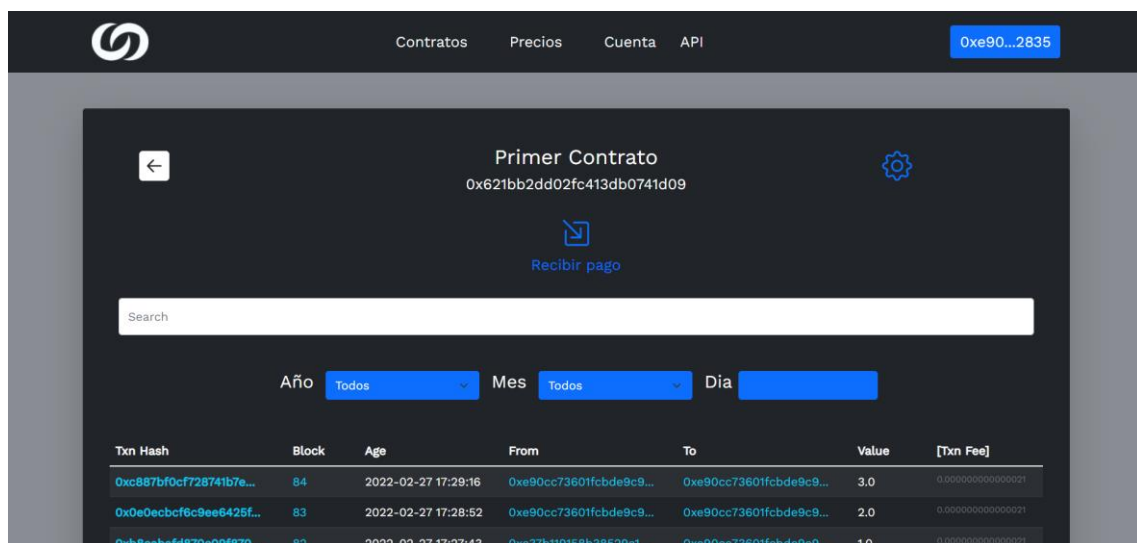
4.4.1 Funcionamiento

Al cargar la página se hace una petición a la API de Decoinpay en la que se le pide que devuelva todos los contratos en los que la propiedad `wallet_id`, del esquema de contratos coincida con la wallet del usuario conectado con metamask, además de todos los contratos a los que se tiene acceso de otras cuentas, los cuales están en la propiedad `array contracts_with_acces`, y el frontend se encarga de mostrar los datos que devuelven dichas peticiones.

Para crear un contrato, pinchando en el botón de añadir contrato e insertando el nombre que se le quiere dar, se envía una petición post a la API con los datos insertados y la cabecera con la wallet encriptada para que cree un nuevo objeto contrato con la wallet del usuario y el nombre insertado.

Al pinchar en un contrato, accedes a la página info, en la que se pueden ver las transacciones que ha recibido el contrato, y la información básica como el id y el nombre.

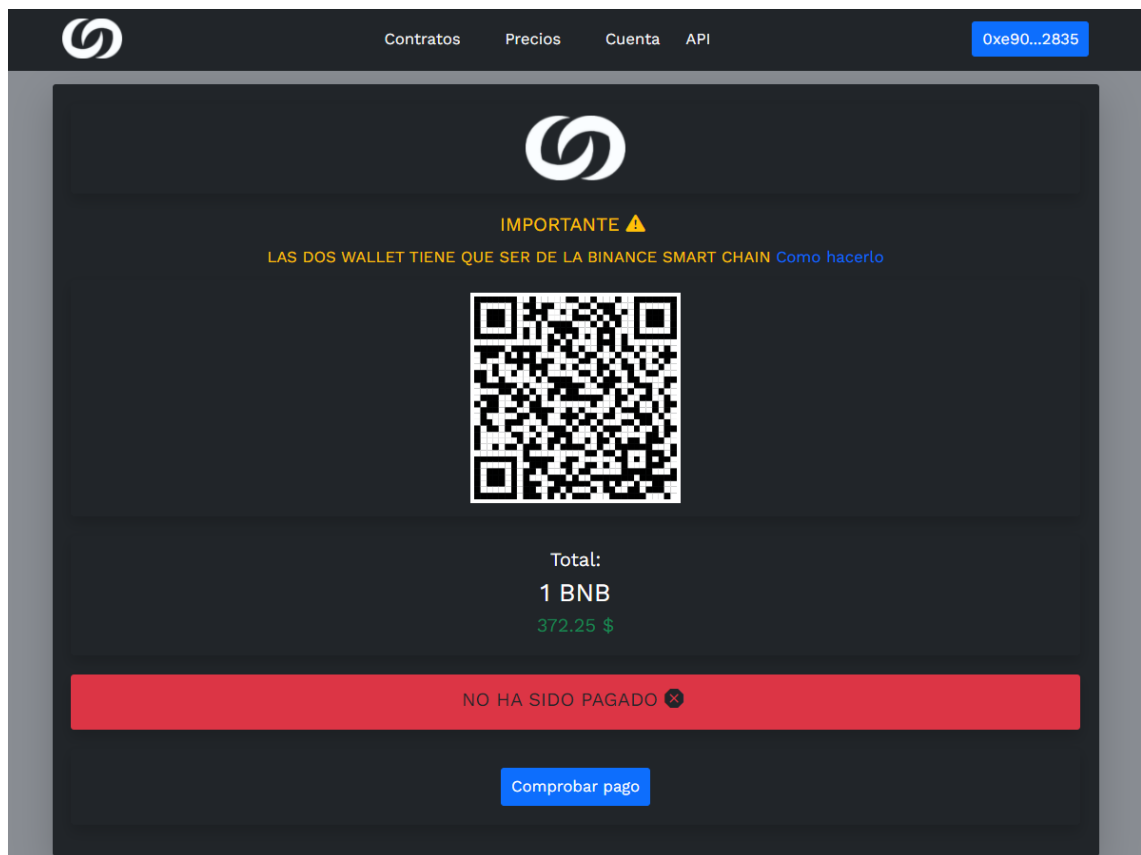
En la base de datos cada contrato al igual que el usuario tiene un array de transacciones, así pues, igual que en el apartado cuenta en esta página se recorre ese array transacciones para mostrar todas las transacciones que se ha recibido, pudiendo también filtrarlas en la barra de búsqueda por los mismos campos que en el apartado de cuenta, por otra parte en esta página se puede crear un código QR para recibir una transacción cuyo funcionamiento se describe a continuación, también se puede entrar a los ajustes del contrato pinchando en el engranaje.



Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0xc887bf0cf728741b7e...	84	2022-02-27 17:29:16	0xe90cc73601fcbde9c9...	0xe90cc73601fcbde9c9...	3.0	0.0000000000000021
0x0e0ecbcfc6c9ee6425f...	83	2022-02-27 17:28:52	0xe90cc73601fcbde9c9...	0xe90cc73601fcbde9c9...	2.0	0.0000000000000021
0xb8cabafd870c09f870...	82	2022-02-27 17:27:43	0xc37b119158b38529a1...	0xe90cc73601fcbde9c9...	1.0	0.0000000000000021

Para que un contrato pueda recibir una transacción se realizan los siguientes pasos de la siguiente forma:

- Primero el dueño del contrato o un usuario permitido pincha en recibir transacción e ingresa el importe que desea recibir, esto crea en la base de datos un objeto transacción con el importe introducido de la transacción y automáticamente se añade la propiedad isPaid por defecto en false para saber si dicha transacción se ha realizado o no y el id del contrato para saber a donde tiene que ir dirigida esa transacción.
- Después de haber creado el objeto transacción, con el id de dicha transacción que se obtiene del método post al crear la transacción se redirige a la página receivepayment/id-de-la-transaccion en la que se muestra la información de si ha sido pagado y el valor del importe en BNBs y en dólares, aparte de un código QR para que el cliente que tenga que realizar la transacción lo pueda leer con el móvil.
- Por último, el cliente lee el QR y le redirige a la página dopayment/id-de-la-transacción para que pueda pagar, en el momento que paga, se comprueba que todo ha ido bien y no ha habido nada extraño, la propiedad de isPaid pasa a true y la transacción se guarda en el array de transacciones tanto del usuario como en la del contrato y en la página de receivepayment se mostraría en verde indicando que la transacción ha sido pagada.



En la pestaña de ajustes del contrato se puede modificar su nombre, modificándolo a través de una petición put a la API y ver, añadir, eliminar, modificar y buscar usuarios permitidos, los cuales podrán administrar ese mismo contrato desde sus cuentas, para ello su funcionamiento es el siguiente, al pinchar en añadir wallet se tiene que insertar la wallet del usuario que se quiere añadir y un alias, una vez añadidos los datos y pulsado en añadir, el frontend enviara una petición put a la API para modificar el objeto usuario del usuario al que pertenece la wallet que se ha insertado, añadiendo a la propiedad array `contracts_with_acces` el id del contrato, array el cual se recorrerá al iniciar la página, sacando toda la información del contrato al que le pertenece la id y mostrándolo por pantalla.

4.5 Gestión de usuarios

La gestión de usuarios se realiza no ingresando usuario y contraseña sino con carteras virtuales a través de la extensión de metamask, este nuevo sistema de login con carteras virtuales es perfecto para una mayor facilidad de gestión de usuarios y una seguridad mayor en la página al estar las carteras encriptadas con el sistema de clave publica clave privada.

React ofrece una forma de crear variables y funciones globales para poder usarlas en cualquier componente, con `useContext`, este Hook se ha utilizado en Decoinpay para gestionar toda la información del usuario, en el `useContext` se ha creado una variable objeto inicialmente en null.

Cuando se carga la página comprueba que metamask este conectada y desbloqueada, si no lo está, el usuario tendrá que darle al botón de conectar, pero si sí está conectada y desbloqueada, se ejecuta una función para sacar la wallet, si esta wallet está en la base de datos, carga en la variable global de `useContext` el objeto usuario cuya propiedad `wallet_id` del esquema de datos de usuario coincida con la wallet conectada de metamask, sino coincide con ningún usuario, la función crea un nuevo usuario en la base de datos y carga en la variable global los datos del usuario creado.

Entonces en todo momento si la variable es null significa que el usuario no se ha conectado y no puede acceder a determinadas funcionalidades, pero si la variable es un objeto es que existe un usuario logeado y ese objeto será toda la información del

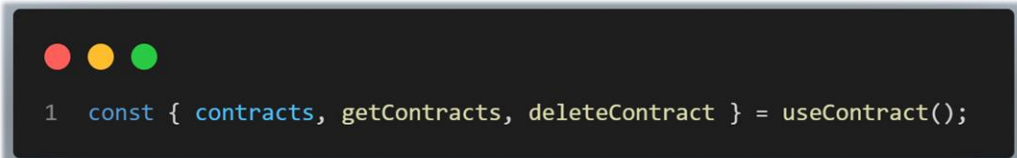
usuario, como las transacciones, los contratos que posee, los contratos a los que tiene acceso...

En este componente también se han creado las funciones para encriptar y desencriptar los datos que posteriormente se insertaran en las cabeceras de las peticiones a la API.

4.6 Estructura de archivos

Los archivos en el frontend se han dividido en carpetas por apartados, funcionando cada una como módulos o miniaplicaciones, teniendo las carpetas: Account para el apartado de cuenta, Api para el apartado información de la api, Contracts para el apartado de contratos, Payments para todo lo que tenga que ver con realizar transacciones, Prices para la página principal o la página precios, y Header que es el menú de Decoinpay, aparte de la carpeta hooks donde se encuentras las funciones globales que utiliza la página como las que tienen que ver con gestión de usuarios. Cada carpeta si son necesarias se compone de tres carpetas:

- Hooks, en esta carpeta es donde se alojan los customshooks, en los que se definen todas las funciones que utiliza el apartado. Posteriormente en cada componente se llama al hook específico con las funciones o estados (variables) que se desee, como en el siguiente ejemplo:



```
1  const { contracts, getContracts, deleteContract } = useContract();
```

haciendo así que el código de los componentes sea más limpio y las funciones se puedan reutilizar.

- Modals, en la que están todos los modales o ventanas emergentes que utiliza dicho apartado.
- Layout, en el que están todos los componentes que utiliza dicho apartado, aquí estaría lo que sería la maquetación o la parte visual de la página.

Capítulo 5

En este capítulo se explica cómo funciona la API pública y cómo las demás páginas web pueden usarla con el objetivo de aceptar pagos con criptomonedas usando la plataforma de Decoinpay.

API pública

Esta API fue creada para que los usuarios que tengan una página web puedan aceptar pagos con criptomonedas a través de la plataforma de Decoinpay desde su propia página. Cuando un usuario crea un contrato se le generan automáticamente sus claves API, que son una contraseña de 50 caracteres generada aleatoriamente y el ID del contrato, esto le servirá al usuario para poder hacer peticiones a la API. La API funciona de la siguiente forma:

- Estando en la situación un usuario que tiene una página que utiliza la API de Decoinpay, tiene un cliente que va a realizar un pago con criptomonedas, el cliente le da a pagar, entonces la página web tendría que hacer una petición POST a la ruta que se especifica en decoinpay.com/informationapi con una cabecera con sus claves API, con el importe de la transacción, la página a la que quiere que redirija cuando el cliente pague y la página que quiere que redirija si el cliente cancela la transacción.
- Una vez creada la transacción se tiene que redirigir al cliente a <https://www.decoinpay.com/pay/id-de-la-transaccion> con el ID de la transacción que devuelve al hacer el POST, una vez redirigido al cliente a la página de Decoinpay, Decoinpay se encarga de saber si el cliente ha pagado o no, si el cliente paga y todo ha ido bien Decoinpay cambia la transacción al estado de pagada y redirige al cliente a la ruta especificada, y si algo ha salido mal, lo redirige a la ruta de cancelación especificada con la transacción en el estado de no pagada.
- Y por último la página web en cuestión tendría que verificar que la transacción ha sido pagada y seguir con las funcionalidades de la página.

Capítulo 6

Despliegue de la aplicación

6.1 Docker

Al momento de subir la aplicación desde el equipo local de desarrollo hasta el host web donde se aloja el proyecto y por lo tanto cambiar de sistema operativo era necesario un sistema con el que no diese errores y además facilitase el despliegue de la aplicación, ese sistema fue Docker.

Gracias a Docker la aplicación se puede dividir en diferentes contenedores, uno para el frontend, otro para el backend, otro para la base de datos, otros 2 para el servidor de nginx y otro para Let's Encrypt. Para poder construir cada contenedor se utiliza el archivo llamado DockerFile que se encuentra dentro de la carpeta de la parte de la aplicación que se va a dockerizar ya sea el frontend el backend o nginx.

Se utiliza la herramienta de Docker-compose que permite orquestar todos los contenedores y así poder enlazar unos con otros. Esta herramienta se configura con el archivo docker-compose.yml, se especifica de cada contenedor los puertos, el nombre, las variables de entorno y los enlaces con otros contenedores. Gracias a este sistema a la hora de desplegar la aplicación en una nueva máquina funciona sin problemas y solo hay que escribir dos comandos.

6.2 Nginx

Estando el frontend en el puerto 3000 y el backend en el puerto 4000 se necesitaba una manera para que se pudiese acceder a ambos desde el puerto 80 o desde el 443 que equivalen a los protocolos http y https, para ello se utiliza nginx. En la configuración de nginx se decidió que si en la url empieza por Decoinpay.com/api apunta al puerto 4000 y por tanto al backend y cualquier otra url que no empezara por /api apuntara al puerto 3000 y por tanto al frontend. Esto se ha conseguido gracias a la configuración de Nginx en el que se ha configurado un reverse proxy y se ha enlazado con el contenedor de Let's encrypt anteriormente comentado en el capítulo 2.12 para poder tener la web segura en https

6.3 DigitalOcean

En este proveedor de servidores es donde se encuentra alojado todos los contenedores que componen la página web de Decoinpay, DigitalOcean ofrece una máquina virtual en la nube a la que a través de unas configuraciones se puede acceder desde un ordenador local a través del protocolo SSH.

Para poder actualizar el código cada vez que hay un cambio se hace uso de GitHub Actions cuyo funcionamiento se explica en el siguiente apartado, en la máquina virtual de DigitalOcean solo se ha creado y configurado un nuevo usuario y se ha instalado las configuraciones de los runners de GitHub Actions.



6.4 GitHub

Para poder subir los cambios desde el equipo local al host se utiliza la herramienta de GitHub y por tanto la de git. La metodología de trabajo con git, consistía en, desde el sistema local ir haciendo cambios y realizando commits a una rama llamada dev, una vez la aplicación era estable y se podía subir una nueva versión de la aplicación, se hacía un merge a la rama main (todos los datos de la rama dev pasan a la rama main) como una nueva release con un tag en específico.

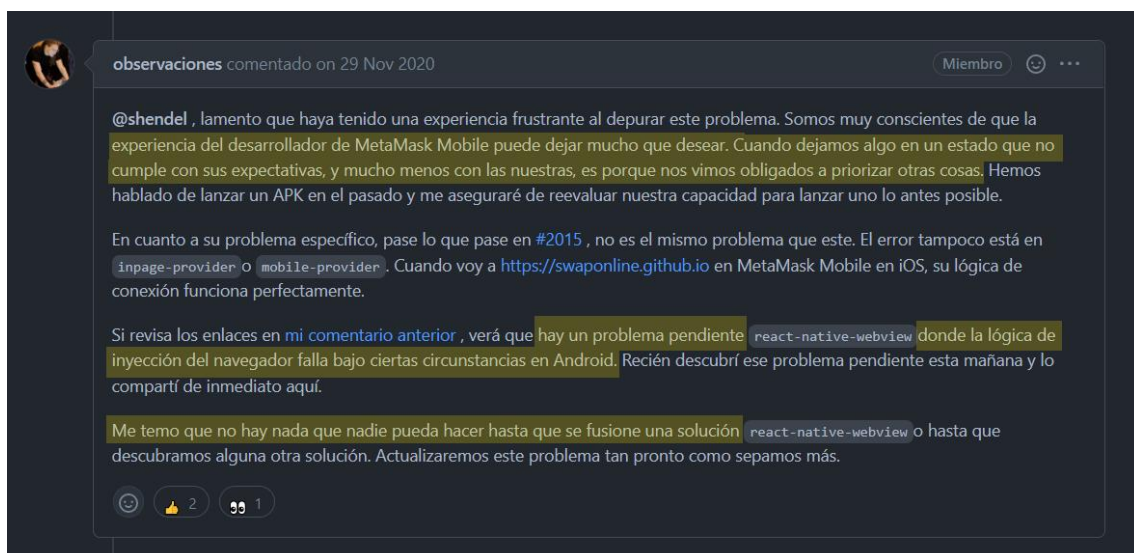
GitHub también ha servido para facilitar el despliegue de la aplicación ya que con su herramienta de GitHub Actions se consigue que cada vez que el repositorio se actualiza al hacerle un push en la rama principal, este automáticamente ejecuta una serie de comandos para que el Docker-compose que está corriendo en la máquina virtual de DigitalOcean se reinicie con todas las nuevas actualizaciones, todo ello con un archivo de configuración tanto en el repositorio como en el host web.

Capítulo 7

Puntos de mejora

7.1 API de metamask

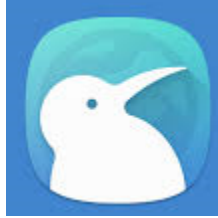
Durante el desarrollo de Decoinpay surgió un problema con la API de metamask al utilizar la aplicación desde un dispositivo móvil, como la aplicación no funciona sin metamask, el objetivo inicial del proyecto era poder usar la página web desde la propia aplicación de metamask, Decoinpay funciona bien excepto en dos ocasiones bastante importantes para el buen funcionamiento de la página, la API de Metamask tiene un error al momento de cargar la información si se accede a la página a través de un QR, o si se refresca la página después de un método post, por algunos problemas al cargar las cookies de metamask, en la siguiente imagen se ve como uno de los desarrolladoras confirma que existe y son conscientes del error pero tienen otras prioridades.



<https://github.com/MetaMask/metamask-mobile/issues/1975>

Como la pagina depende de la API de metamask y en los navegadores de móvil no se pueden instalar extensiones, la solución que se ha encontrado ha sido **Kiwi browser**, un navegador capaz de instalar extensiones en móvil y por tanto poder usar metamask, a día de hoy es posible entrar en la página de Decoinpay desde cualquier

navegador, pero para poder utilizar todas sus funcionalidades solo se puede desde Kiwi browser, en un futuro cuando la API de metamask se actualice o hay alguna forma mejor, el objetivo es utilizar la página desde la propia aplicación de metamask para mayor comodidad



Kiwi browser

7.2 Futuras actualizaciones

Una de las actualizaciones planificadas para el futuro sería realizar un swap de criptomonedas, para que los usuarios desde la página de Decoinpay puedan intercambiar unas criptomonedas por otras, para poder tener una mayor diversificación o simplemente para pasar los BNBs a una moneda estable como el BUSD y así facilitar al usuario el no tener que cambiar por diferentes páginas web, y que puedan tener todas las necesidades en una única página, también con ese objetivo sería una buena actualización la de visionado de graficas as profesional.

La segunda actualización prevista después del swap sería que la pagina tuviese su propia criptomoneda, por dos motivos: el primero sería para que los pagos se puedan realizar con la propia criptomoneda de la página, y el segundo para lanzar una ICO (Initial Coin Offering) y así poder tener o una fuente de ingreso o una forma de conseguir capital para el mantenimiento y el avance de la página.