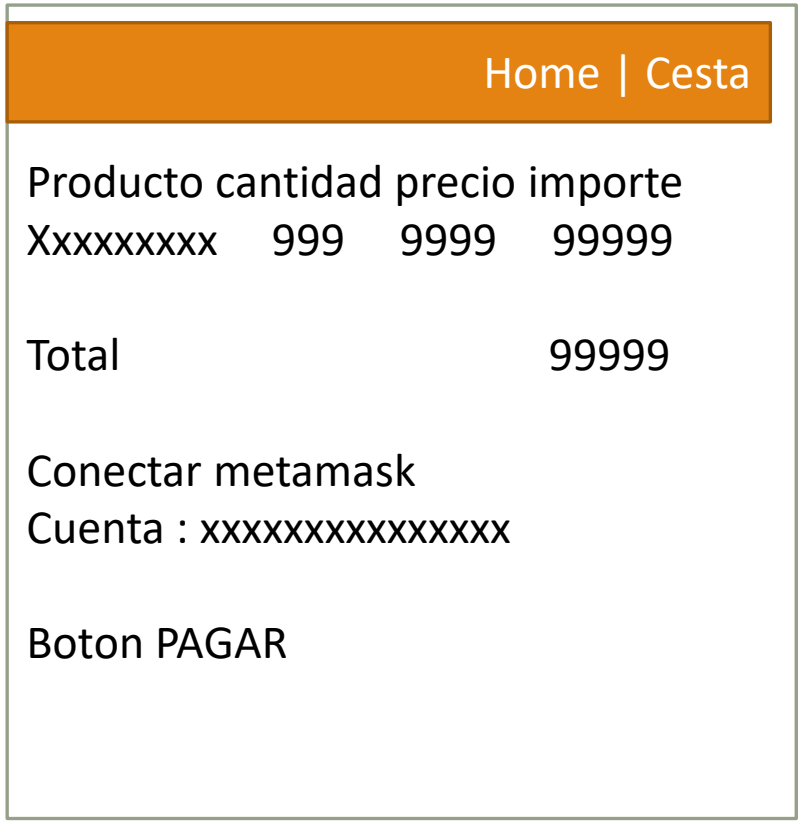
PROYECTO CESTA

DESARROLLO DE UNA CESTA CON PAGO EN ETH

CESTA

1. Se trata de hacer una cesta de la compra con los productos de la base de datos northwind.
2. Tendremos una home con la lista de productos
3. Tendremos una página que nos presente el producto
4. En la página del producto podremos poner la cantidad
5. Tendremos una página donde presentar la cesta
6. En la página de la cesta podremos cambiar la cantidad o ir a la página del producto
7. Tendremos una cabecera que en la podamos ir a la cesta o a la lista de los productos.
8. En la página de la cesta podremos pagar con ETH a través del metamask y de un provider local creado con Docker en nuestra máquina.

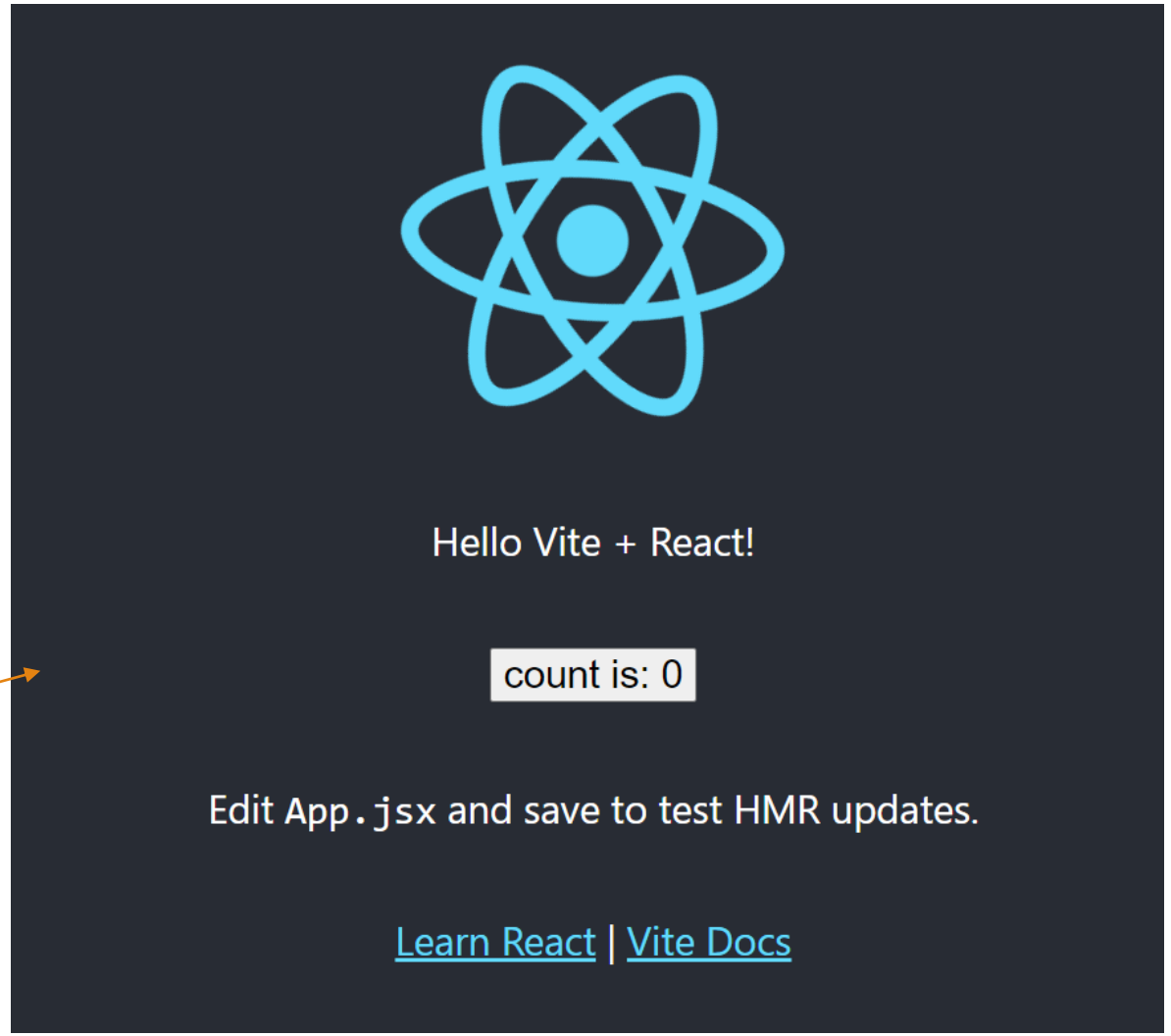
DISEÑO ESQUEMÁTICO



CREAMOS UN PROYECTO REACT
yarn create vite curso-cesta-12 --template React
Nos crea el directorio curso-cesta-12

PASAMOS AL DIRECTORIO, abrimos el code
cd curso-cesta-12
code .

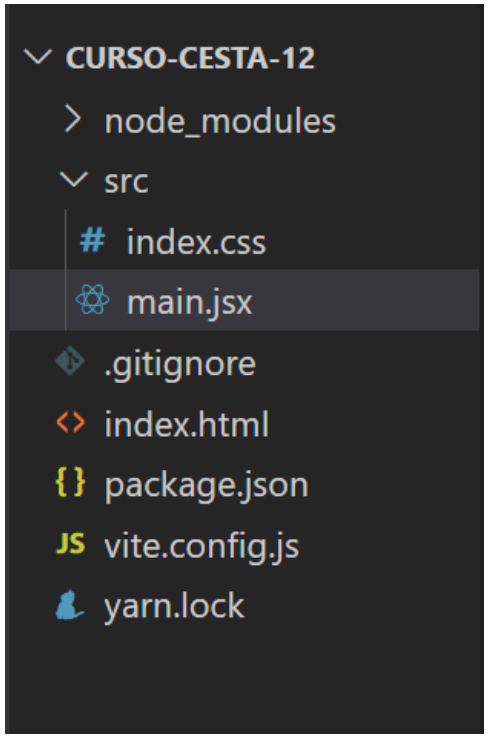
DENTRO DEL CODE
Menu-> Terminal -> Nuevo Terminal
Ejecturar yarn (nos instala las dependencias)
yarn dev
Ir al navegador y acceder a http://localhost:3000
Debe de aparecer la imagen de la derecha



Limpiamos el proyecto para quitar cosas que no necesitamos

```jsx
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'

ReactDOM.render(
  <React.StrictMode>
    <h1>Hola</h1>
  </React.StrictMode>,
  document.getElementById('root')
)
```

Main.jsx



Directorio

Metemos el Boostrap

https://getbootstrap.com/docs/5.0/getting-started/introduction/

Metemos en el index.html

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstra
    <title>Vite App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

ANTES DE BS

DESPUES DE BS
NOTESE EL CAMBIO DE LETRA

```
ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home></Home>} >
          <Route index element={<Productos />} />
          <Route path="*" element={<Productos />} />
          <Route path="productos" element={<Productos />} />
          <Route path="cesta" element={<Cesta />} />
          <Route path="productos/:id" element={<Producto />} />
        </Route>
      </Routes>
    </BrowserRouter >
  </React.StrictMode>
  ,
  document.getElementById('root')
)
```

Metemos el Router
https://reactrouter.com/docs/en/v6/getting-started/overview
yarn add react-router-dom@6 (desde el terminal del code)

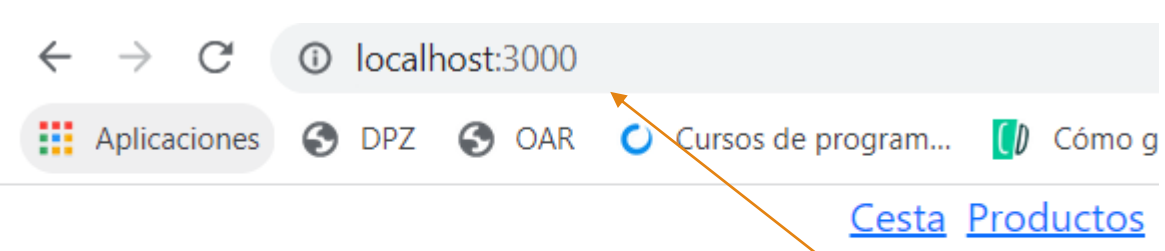
src > Home.jsx > Home

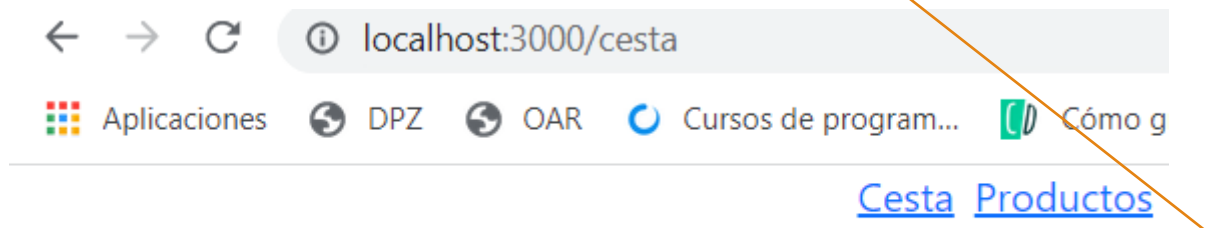
```jsx
import React from 'react'
import { Outlet , Link} from 'react-router-dom'

export default function Home() {
    return (
        <div className='container'>
            <div className='text-end '>
                <Link className='mx-2' to="/cesta">Cesta</Link>
                <Link to="/productos">Productos</Link>
            </div>
            <div>
                <Outlet />
            </div>
        </div>
    )
}

```

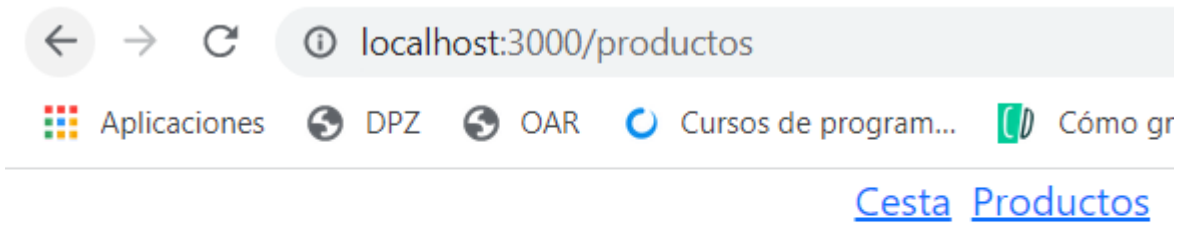
Home.jsx

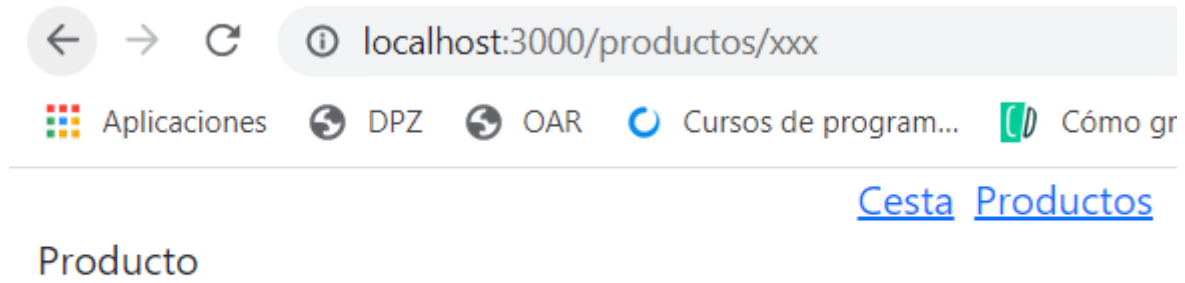
Esta cabecera siempre sale

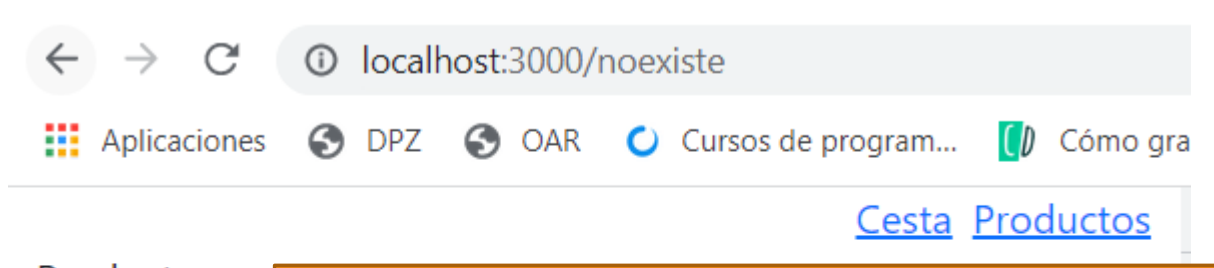
Ahí se mete el contenido de las rutas hijas











La ruta * va a productos
Por eso /noexiste -> productos

Si no vamos a ninguna sale /productos

Posibles rutas

Vamos a hacer la página de productos

Tenemos que acceder a la base de datos
Para hacer la **select * from Products**

Para ello empezamos un servidor nodejs con acceso a postgres

Creamos un web server nodejs

Creamos desde el terminal un directorio mkdir curso-nodejs-12
cd curso-nodejs-12
yarn init –y
yarn add express
code .

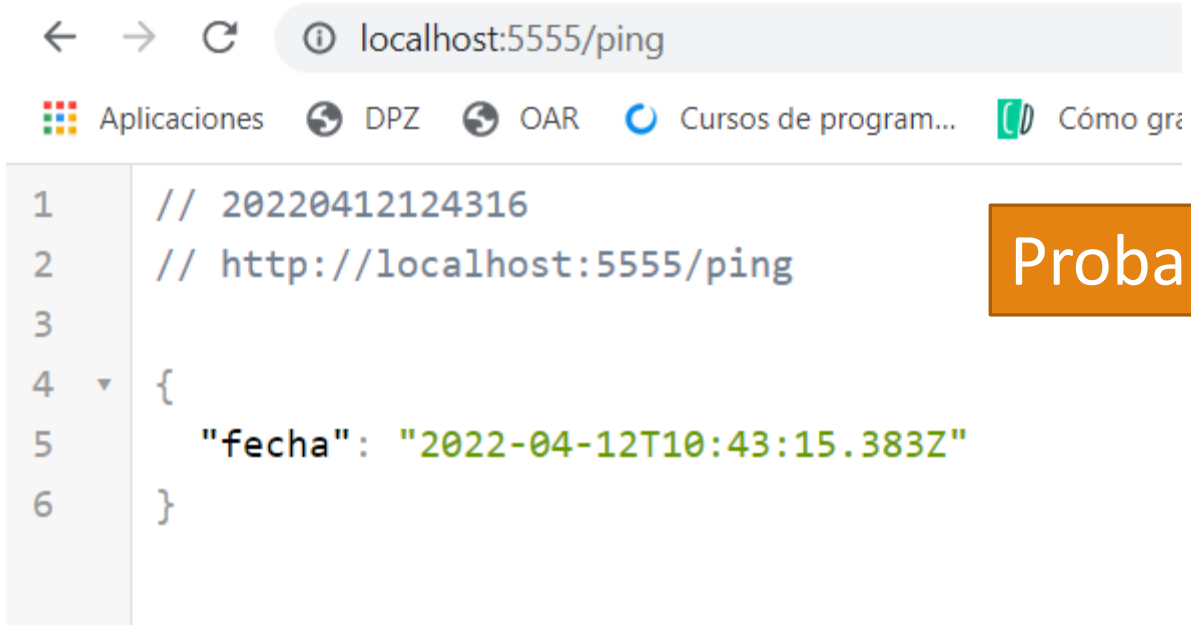
Dentro del code
Creamos un fichero llamado app.js (no tiene nada)
Abrimos una terminal en vscode con la opción de menú Terminal -> nuevo terminal
Ejecutamos en el termina: npx nodemon app.js

Escribimos en el fichero

```
const express = require("express")

const app = express();

app.get("/ping", async (req, res) => {
    res.send({ fecha: new Date().toISOString() })
})

app.listen(5555, () => {
    console.log("listening")
})
```

En la terminal no hay errores

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
listening
```



Probamos el servidor web desde el navegador

Uso del curl desde el command line

```
C:\Users\jviejo\curso-nodejs-12>curl localhost:5555/ping
{"fecha":"2022-04-12T10:47:21.486Z"}
```

```
const mysql = require("mysql8")

var pool = mysql.createPool({
    connectionLimit: 10,
    host: 'localhost',
    user: 'root',
    password: 'my-secret-pw',
    database: 'northwind'
});
function q(sql, parameters) {
    return new Promise((resolver, reject) => {
        pool.query(sql, parameters, function (error, results,
fields) {
            if (error) reject(error);

            return resolver([results, fields]);
        });
    });
}
module.exports = {
    q
}
```

Fichero db.js

Modulo para poder acceder a la base de datos

Hemos usado mysql

Incorporamos el módulo

```
const db = require('./db')
const app = express();


app.get("/products", async (req, res) => {
    const [results, fields] = await db.q("select * from Products")
    res.send(results)
})
```

Ruta para obtener productos

Uso de async / await

Habría que controlar errores con try/catch

Incorporamos el módulo

```
const db = require('./db')
const app = express();


app.get("/products", async (req, res) => {
    const [results, fields] = await db.q("select * from Products")
    res.send(results)
})
```

Ruta para obtener productos

Uso de async / await

Habría que controlar errores con try/catch

```
app.get("/products", async (req, res) => {
    try {
        const [results, fields] = await db.q("select * from Products“,[])
        res.send(results)
    } catch (error) {
        res.send({ error: error.message })
    }
})
```

Uso de try/catch para controlar los errores

Hacemos el componente Productos

Instalamos yarn add react-query
Creamos un objeto queryClient y envolvemos el BrowserRouter con el elemento QueryClientProvider

```
import { QueryClient, QueryClientProvider } from 'react-query'

const queryClient = new QueryClient()

  <QueryClientProvider client={queryClient}>
      <BrowserRouter>
        ….
   </BrowserRouter >
 </QueryClientProvider>
```

```jsx
import React from 'react'
import { useQuery } from 'react-query'

export default function Productos() {
  const {data,  isLoading} = useQuery("products", () => {
    return fetch("http://localhost:5555/products")
      .then(res => res.json())
  })
  if (isLoading) {
    return <div>Cargando...</div>
  }
  return (
    <div>{JSON.stringify(data)}</div>
  )
}
```

El hook useQuery nos permite acceder al web server

El hook devuelve data e isLoading

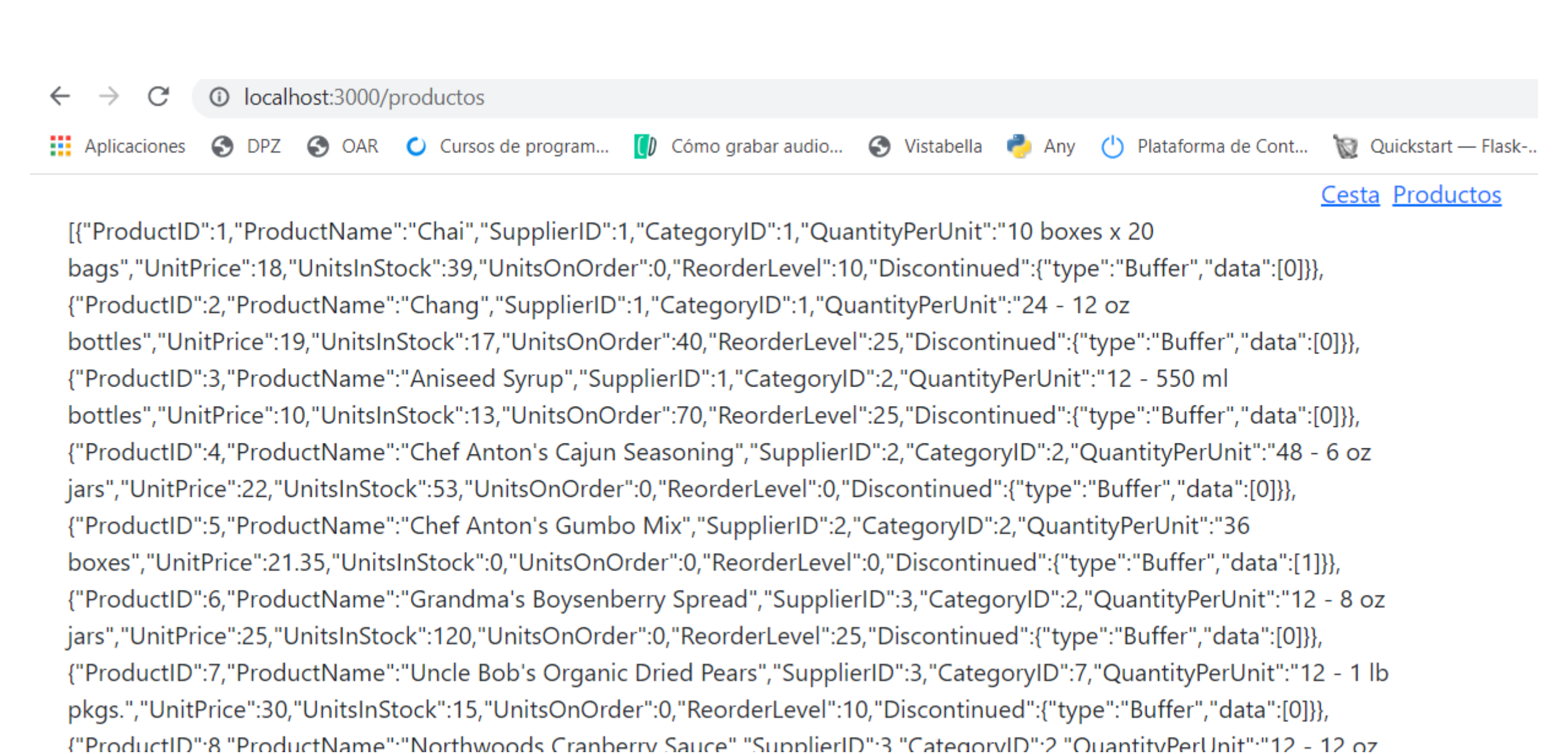
Presentación en bruto

Al acceder tenemos un error de CORS, que solucionamos instalando CORS en el web server

Access to fetch at 'http://localhost:5555/products' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

Instalamos yarn add cors (en la aplicación del web server

```
const express = require("express")
const db = require('./db')
const app = express();
const cors = require('cors');
app.use(cors());
```


localhost:3000/productos
Aplicaciones
DPZ
OAR
Cursos de program...
Cómo grabar audio...
Vistabella
Any
Plataforma de Cont...
Quickstart — Flask-..
Cesta
Productos
[{"ProductID":1,"ProductName":"Chai","SupplierID":1,"CategoryID":1,"QuantityPerUnit":"10 boxes x 20 bags","UnitPrice":18,"UnitsInStock":39,"UnitsOnOrder":0,"ReorderLevel":10,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":2,"ProductName":"Chang","SupplierID":1,"CategoryID":1,"QuantityPerUnit":"24 - 12 oz bottles","UnitPrice":19,"UnitsInStock":17,"UnitsOnOrder":40,"ReorderLevel":25,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":3,"ProductName":"Aniseed Syrup","SupplierID":1,"CategoryID":2,"QuantityPerUnit":"12 - 550 ml bottles","UnitPrice":10,"UnitsInStock":13,"UnitsOnOrder":70,"ReorderLevel":25,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":4,"ProductName":"Chef Anton's Cajun Seasoning","SupplierID":2,"CategoryID":2,"QuantityPerUnit":"48 - 6 oz jars","UnitPrice":22,"UnitsInStock":53,"UnitsOnOrder":0,"ReorderLevel":0,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":5,"ProductName":"Chef Anton's Gumbo Mix","SupplierID":2,"CategoryID":2,"QuantityPerUnit":"36 boxes","UnitPrice":21.35,"UnitsInStock":0,"UnitsOnOrder":0,"ReorderLevel":0,"Discontinued":{"type":"Buffer","data":[1]}},
{"ProductID":6,"ProductName":"Grandma's Boysenberry Spread","SupplierID":3,"CategoryID":2,"QuantityPerUnit":"12 - 8 oz jars","UnitPrice":25,"UnitsInStock":120,"UnitsOnOrder":0,"ReorderLevel":25,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":7,"ProductName":"Uncle Bob's Organic Dried Pears","SupplierID":3,"CategoryID":7,"QuantityPerUnit":"12 - 1 lb pkgs.","UnitPrice":30,"UnitsInStock":15,"UnitsOnOrder":0,"ReorderLevel":10,"Discontinued":{"type":"Buffer","data":[0]}},
{"ProductID":8,"ProductName":"Northwoods Cranberry Sauce","SupplierID":3,"CategoryID":2,"QuantityPerUnit":"12 - 12 oz

```
if (isLoading) {
    return <div>Cargando...</div>
  }
  return (
    <div>
      <table className="table">
        <thead>
          <tr>
            <th>Nombre</th>
          </tr>
        </thead>
        <tbody>
          {data.map(product => (
            <tr key={product.id}>
              <td>
                <Link to={`/productos/${product.ProductID}`}> {product.ProductName}</Link>
              </td>
            </tr>
          ))}
        </tbody>
      </table>

    </div>
  )
```

Si se está cargando

Procesa todos los productos

Cambio de ruta

```jsx
import React from 'react'
import { useParams } from 'react-router-dom'
import { useQuery } from 'react-query'
export default function Producto() {
  const params = useParams()
  const { data, isLoading } = useQuery("products", () => {
    return fetch(`http://localhost:5555/products/${params.id}`)
      .then(res => res.json())
  })
  if (isLoading) {
    return <div>Cargando...</div>
  }
```

Ruta del servidor

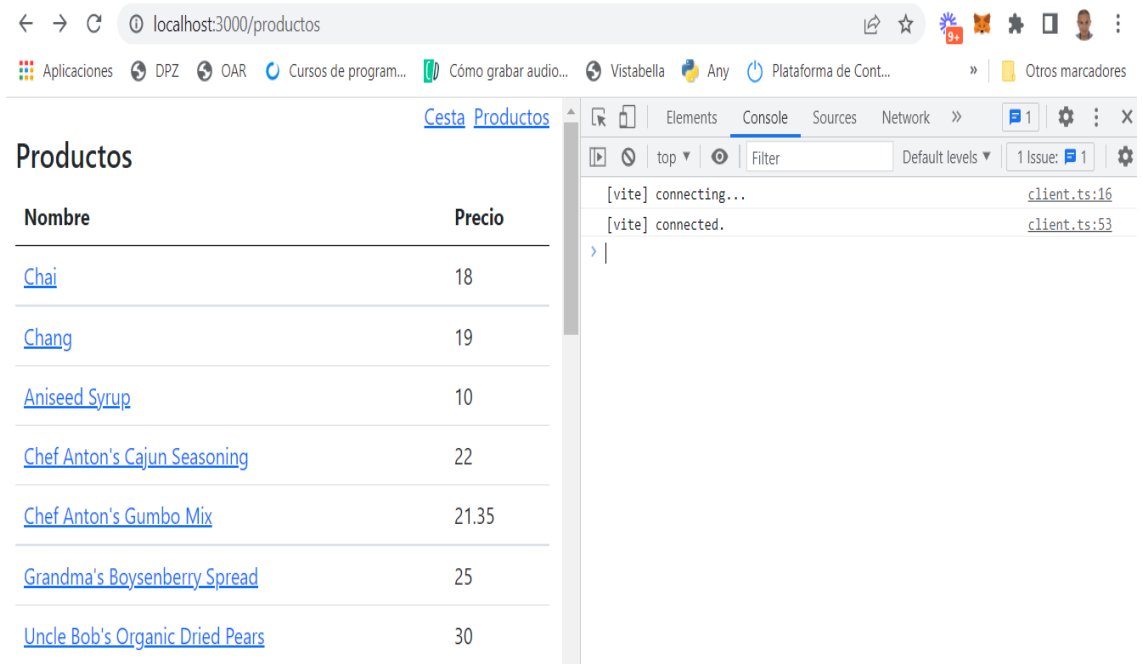
```jsx
app.get("/products/:id", async (req, res) => {
    try {
        const [results, fields] = await db.q("select * from Products where
ProductID=?", [req.params.id])
        res.send(results)
    } catch (error) {
        res.send({ error: error.message })
    }
})
```

En el servidor node

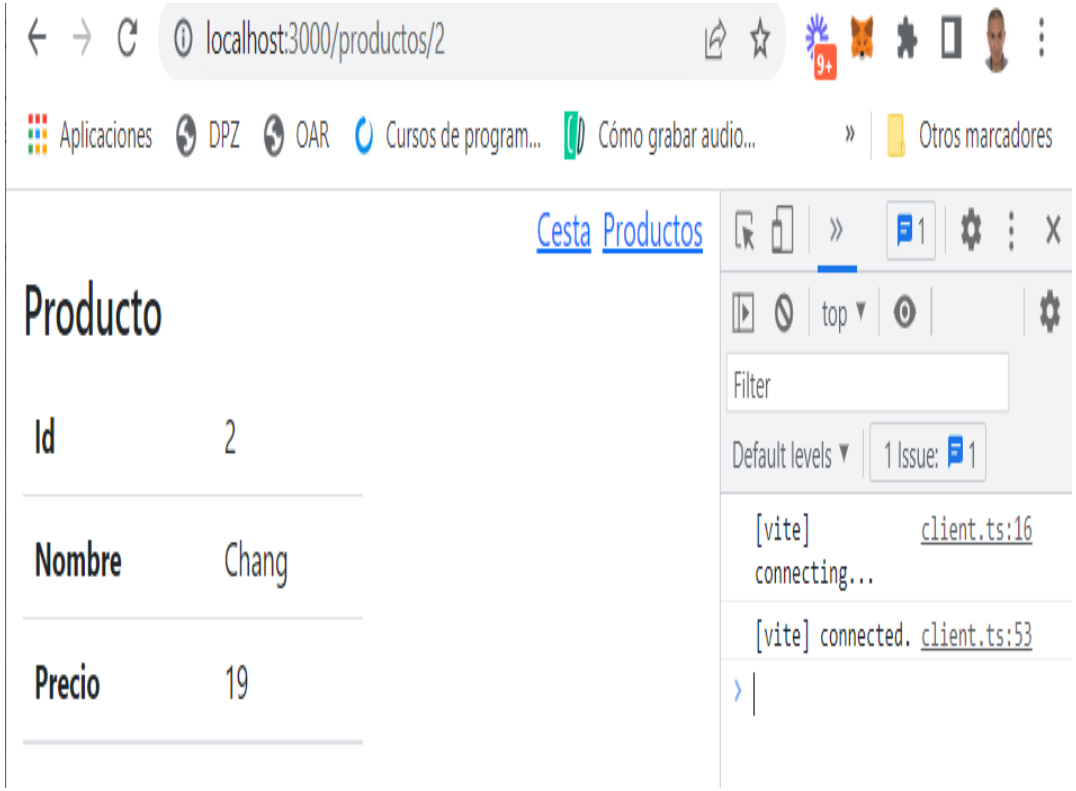
Render de producto

```
return (
    <div>
      <h3>Producto</h3>
      <table className="table w-50">
        <thead>
          <tr>
            <th>Id</th>
            <td>{data[0].ProductID}</td>
          </tr>
          <tr>
            <th>Nombre</th>
            <td>{data[0].ProductName}</td>
          </tr>
          <tr>
            <th>Precio</th>
            <td>{data[0].UnitPrice}</td>
          </tr>
        </thead>
      </table>
</div>
```

Productos queda así



Producto



Añadimos un formulario para pedir la cantidad

Cuando pinchemos aceptar, debemos añadir al estado global, el producto y la cantidad

Tenemos que añadir un contexto global que tenga el estado

```
… FALTAN IMPORT
import { createContext } from 'react'
export const Context = createContext(null)
const queryClient = new QueryClient()
function App() {
  const [estado, setEstado] = React.useState({
    cesta: []
  })
  return <React.StrictMode>
    <Context.Provider value={[estado, setEstado]}>
      <QueryClientProvider client={queryClient}>
        <BrowserRouter>
          <Routes>

          </Routes>
        </BrowserRouter>
      </QueryClientProvider>
    </Context.Provider>
  </React.StrictMode>
}
```

Creamos context

Creamos el estado en el componente root

Inicializo el Context

```
import { useForm } from 'react-hook-form'
```

Modulo para formularios

Buscamos si el producto está en la cesta, para inicializar la cantidad

```
  const cantidad = estado.cesta.find(i => i.producto.ProductID == params.id)?.cantidad
  const { register, handleSubmit, watch, formState: { errors } } = useForm(
    { mode: 'onChange', defaultValues: { cantidad: cantidad } }
  );
```

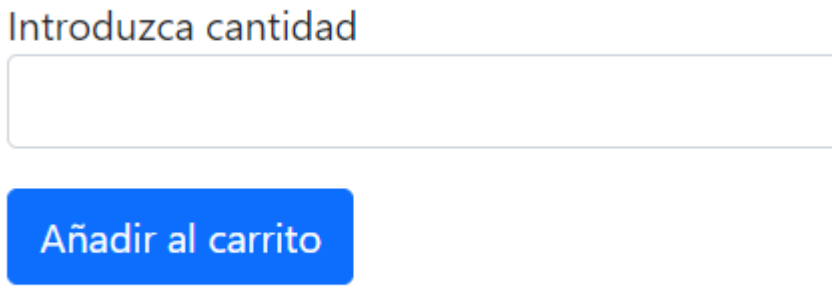
El hook useForm se usa par gestionar el formulario.
Tiene un register y un handleSubmit

El onSubmit se ejecuta cuando pinchemos o demos Enter

```
<form onSubmit={handleSubmit(onSubmit)}>
        <div className="form-group">
          <label>Introduzca cantidad</label>
          <input {...register('cantidad')} type="number" className="form-control" />
        </div>
        <button type="submit" className="btn btn-primary mt-3">Añadir al carrito</button>
</form>
```

…register('cantidad') añade un campo al formulario

Este es el formulario



Se obtiene el estado global con useContext(Context)

```
 const [estado, setEstado] = useContext(Context)

function onSubmit(datos) {
    if (datos.cantidad == 0)
      return;

    setEstado({
      ...estado, cesta:
        [...estado.cesta.filter(i => i.producto.ProductID != data[0].ProductID),
      {
         producto: data[0],
         total: datos.cantidad * data[0].UnitPrice,
         cantidad: datos.cantidad }]
      })
  }
```

En datos están los datos del formulario. Datos.cantidad tiene lo que sea tecleado en cantidad

setEstado toma el estado actual y cambia la cesta, en la cesta mete todos los elementos distintos del producto que tenemos en pantalla y luego mete el registro en pantalla

```
const [estado, setEstado] = useContext(Context)
// operación de acumulación de el total final de la compra
const total = estado.cesta.reduce((acumulador, item) => acumulador
+ item.total, 0)

<table className="table">
        <thead>
          <tr>
            <th>Nombre</th>
            <th>Precio</th>
            <th>Cantidad</th>
            <th>Total</th>
          </tr>
        </thead>

  </table>
 <h3>Total compra {total}</h3>
```

```
<tbody>
            {estado.cesta.map(i => (
              <tr key={i.producto.ProductID}>
                <td>
                  <Link
to={`/productos/${i.producto.ProductID}`}>{i.pr
oducto.ProductName}</Link>
                </td>
                <td>
                  {i.producto.UnitPrice}
                </td>
                <td>
                  {i.cantidad}
                </td>
                <td>
                  {i.total}
                </td>
              </tr>
            ))}
          </tbody>
```

La cesta está en el estado y lo presentamos con el componente Cesta.jsx

Presentar la cuenta activa ETH

```
useEffect(() => {
    ethereum && ethereum.request({ method: 'eth_requestAccounts'
}).then(i => {
      setCuenta(i[0])

      ethereum.on('accountsChanged', (i) => {
        setCuenta(i[0])
      })
    });
  }, [])
```

El useEffect se ejecuta una sola vez, cuando el componente se activa ya que tiene [] en las dependencias

Cuando cambio la cuenta del metamask también cambia la cuenta por el Ethereum.on

```
async function pagar() {
  setTxOk(false)
  setTxRechazo(false)
  const transactionParameters = {
    to: '0x280f1db3d104dad8705c0696fb81bd8e78141caf', // Cuenta del comercio.
    from: ethereum.selectedAddress, // la cuenta activa del metamask
    value: ethers.utils.parseEther(total.toString()).toHexString()
  };
  try {
    const txHash = await ethereum.request({
      method: 'eth_sendTransaction',
      params: [transactionParameters],
    });
    setTxOk(txHash) // tx ok

  } catch (error) {
    setTxRechazo(error) // cancelada por la razón que sea

  } finally {
    // final
  }
```