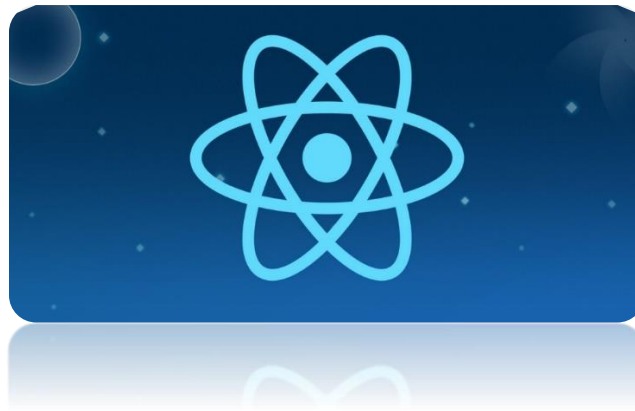


## React Basics



[React](#) es una biblioteca de JavaScript de código abierto diseñada para crear interfaces de usuario interactivas en aplicaciones web. Fue desarrollada por Facebook y se utiliza ampliamente en la industria para construir aplicaciones de una sola página (Single-Page Applications), aplicaciones móviles, interfaces de usuario y otros tipos de aplicaciones interactivas.

La principal característica de React es su enfoque en la construcción de componentes reutilizables. Un componente en React es una unidad autónoma y modular que encapsula el comportamiento y la interfaz de usuario de una parte específica de la aplicación. Estos componentes pueden combinarse para formar interfaces más complejas y, debido a su naturaleza reutilizable, permiten un desarrollo eficiente y escalable.

React utiliza un enfoque basado en el DOM virtual (Virtual DOM) para manejar de manera eficiente los cambios en la interfaz de usuario. En lugar de realizar cambios directamente en el DOM del navegador, React crea una representación virtual de los elementos y los compara con la versión actual del DOM. Luego, actualiza selectivamente solo los elementos que han cambiado, lo que resulta en un rendimiento optimizado y una experiencia de usuario más rápida.

Además, React promueve el uso de JavaScript junto con JSX, una extensión de sintaxis que permite escribir código HTML similar dentro de los archivos JavaScript. Esto facilita la construcción de componentes y la composición de interfaces de usuario, al tiempo que mantiene la separación entre la lógica y la presentación.

React también cuenta con una amplia comunidad y ecosistema de herramientas y bibliotecas complementarias que facilitan el desarrollo con esta tecnología. Algunas de las herramientas populares asociadas con React incluyen React Router para el enrutamiento, Redux para la administración del estado, Next.js para el renderizado del lado del servidor y muchos otros.

En resumen, React es una biblioteca de JavaScript que facilita la creación de interfaces de usuario interactivas y reutilizables en aplicaciones web y móviles, utilizando componentes modulares y un enfoque eficiente para el manejo de cambios en la interfaz de usuario.

## Iniciemos con el proceso

Vamos a abrir nuestra terminal y allí vamos a utilizar el siguiente comando: **npm create vite**

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y|
```

Colocamos **Y** y presionamos enter, a continuación, nos preguntara cual es el nombre del proyecto:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
? Project name: » vite-project|
```

Vamos a colocar **react\_basics** y presionamos enter:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
? Select a framework: » - Use arrow-keys. Return to submit.
>  Vanilla
   Vue
   React
   Preact
   Lit
   Svelte
   Others
```

Acá nos permite seleccionar el framework que vamos a utilizar en este caso seleccionamos **React** y presionamos enter:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
✓ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
>  TypeScript
   TypeScript + SWC
   JavaScript
   JavaScript + SWC
```

Ahora seleccionamos javascript y presionamos enter:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in D:\DATA\Documents\react_basics...

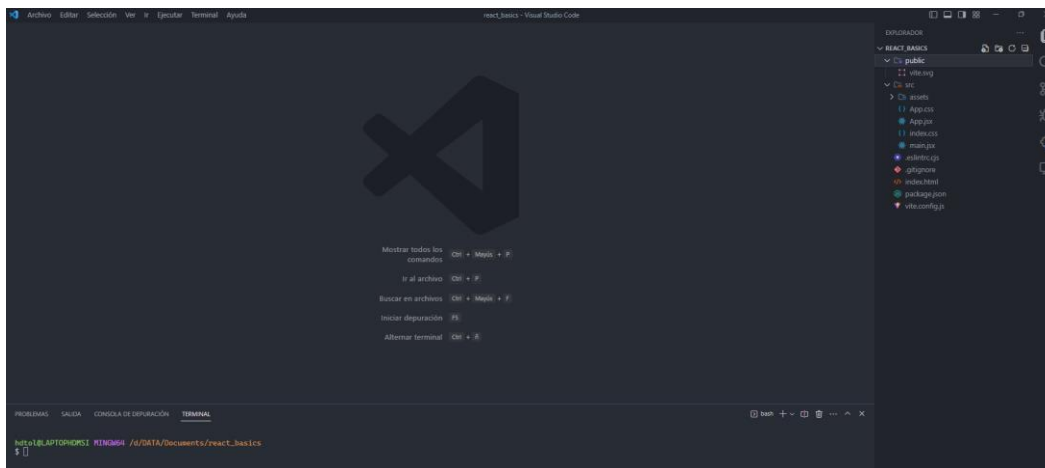
Done. Now run:

  cd react_basics
  npm install
  npm run dev

D:\DATA\Documents>|
```

Ahora ingresamos a nuestra carpeta recién creada **react\_basics** y allí vamos a abrir nuestro VS code con el comando **code** .

```
D:\DATA\Documents\react_basics>code .|
```



Ahora en nuestro terminal vamos a hacer la instalación de las dependencias a través del comando **npm install**

```
hdtol@LAPTOPHDMI MINGW64 /d/DATA/Documents/react_basics
$ npm install

added 239 packages, and audited 240 packages in 27s

80 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

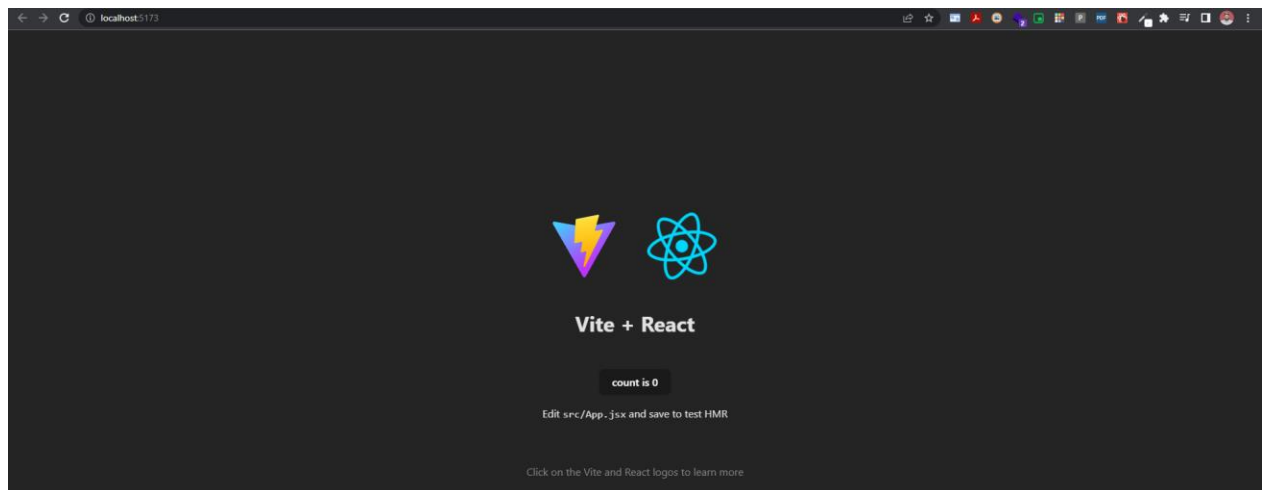
Ahora vamos a ejecutar el comando **npm run dev** para abrir nuestro servidor y poder ver el proyecto en la web:

```
hdtol@LAPTOPHDMI MINGW64 /d/DATA/Documents/react_basics
$ npm run dev

> react_basics@0.0.0 dev
> vite

VITE v4.3.8 ready in 482 ms
  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h to show help
```

De esta manera ya tendremos nuestro proyecto ejecutándose en el localhost:



Hablemos de [Vite](#) es una herramienta de desarrollo web rápida y liviana creada por Evan You, el creador de Vue.js. Está diseñada para mejorar la experiencia de desarrollo al permitir un tiempo de compilación instantáneo y un servidor de desarrollo extremadamente rápido.

A diferencia de otras herramientas de construcción como webpack o Parcel, que se basan en la creación de un único paquete final antes de ejecutar la aplicación, Vite adopta un enfoque de desarrollo basado en la modularidad y la carga bajo demanda.

En lugar de agrupar todos los archivos de la aplicación en un solo archivo en tiempo de compilación, Vite utiliza la capacidad de los navegadores modernos para importar módulos directamente desde el sistema de archivos local durante el desarrollo. Esto significa que cada módulo individual se compila y se sirve por separado, lo que resulta en un tiempo de compilación y recarga instantáneos cuando se realizan cambios en el código fuente.

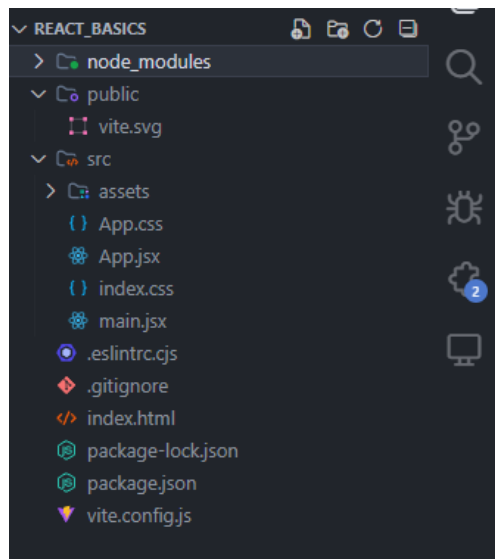
Vite también tiene una integración nativa con los marcos de JavaScript más populares, como Vue.js, React y Preact, lo que permite una configuración sencilla y un flujo de desarrollo optimizado para estas tecnologías.

Además de su velocidad de desarrollo, Vite ofrece otras características útiles, como la recarga en caliente (hot module replacement) que actualiza solo los componentes modificados en lugar de recargar toda la página, un entorno de desarrollo con soporte para TypeScript y CSS preprocesados, y la capacidad de generar un paquete optimizado y minificado para la producción.

En resumen, Vite es una herramienta de desarrollo web rápida y eficiente que mejora la experiencia de desarrollo al ofrecer un tiempo de compilación instantáneo, un servidor de desarrollo veloz y una carga bajo demanda de módulos. Es especialmente popular en combinación con frameworks como Vue.js, React y Preact.

## Hablemos de las carpetas de nuestro proyecto

La estructura de carpetas de un proyecto de React utilizando Vite sigue una convención comúnmente utilizada en el ecosistema de React. A continuación, se describe cada una de las carpetas principales y para qué se utilizan:



1. **node\_modules**: Esta carpeta se crea automáticamente al instalar las dependencias del proyecto utilizando herramientas como npm (Node Package Manager) o Yarn. Aquí se almacenan todas las bibliotecas y paquetes de terceros que son necesarios para el funcionamiento del proyecto.

2. **public**: En esta carpeta se almacenan los archivos estáticos que se servirán directamente al navegador, como imágenes, archivos CSS, fuentes, etc. Estos archivos se copian directamente en la raíz del directorio de salida al construir el proyecto.

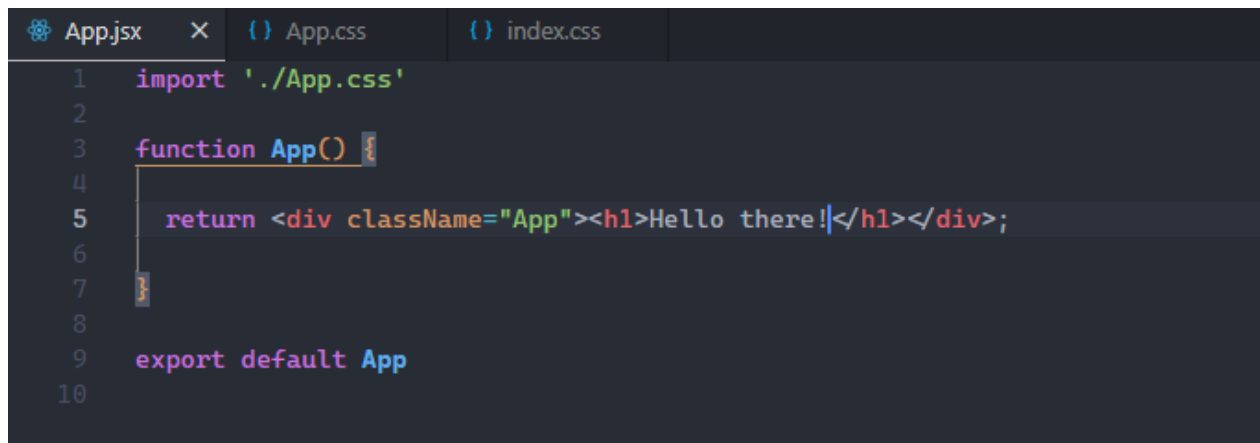
3. **src**: Esta carpeta es donde se ubica el código fuente de la aplicación. A continuación, se describen las subcarpetas y archivos más comunes dentro de la carpeta **src**:

- **index.html**: Es el archivo HTML principal de la aplicación. Aquí se incluirán los puntos de montaje de la aplicación React y se vincularán los archivos CSS o scripts necesarios.
- **main.js o index.js**: Es el punto de entrada principal de la aplicación. Aquí se configura y se inicia la aplicación React.
- **components**: Esta carpeta se utiliza para almacenar los componentes de React reutilizables. Cada componente generalmente se coloca en su propio archivo.
- **pages**: Aquí se pueden almacenar los componentes específicos de cada página de la aplicación. Por lo general, cada archivo en esta carpeta representa una página o una ruta de la aplicación.
- **assets**: En esta carpeta se pueden almacenar archivos estáticos adicionales, como imágenes, iconos o archivos de datos.

- **styles:** Aquí se pueden colocar los archivos CSS o preprocesados (como Sass o Less) utilizados en la aplicación.
  - **utils:** En esta carpeta se pueden almacenar funciones de utilidad, helpers o módulos que se utilizan en varias partes del proyecto.
  - **services:** Esta carpeta se utiliza para almacenar módulos de servicios o API que interactúan con servicios externos, como solicitudes HTTP o almacenamiento local.
4. **dist:** Esta carpeta se crea cuando se realiza una compilación o construcción del proyecto. Contiene los archivos estáticos generados y optimizados listos para ser desplegados en producción.
  5. **public/index.html:** Este archivo es una plantilla HTML que se utiliza durante el desarrollo y se utiliza para servir la aplicación React. Durante la construcción, este archivo se reemplaza por el archivo **index.html** ubicado en la carpeta **public**.

Estas son las carpetas y archivos principales que se encuentran comúnmente en un proyecto de React utilizando Vite. Sin embargo, ten en cuenta que la estructura puede variar dependiendo de las preferencias y necesidades del proyecto, así como de las convenciones establecidas por el equipo de desarrollo.

Vamos a modificar **App.jsx** de la siguiente manera:

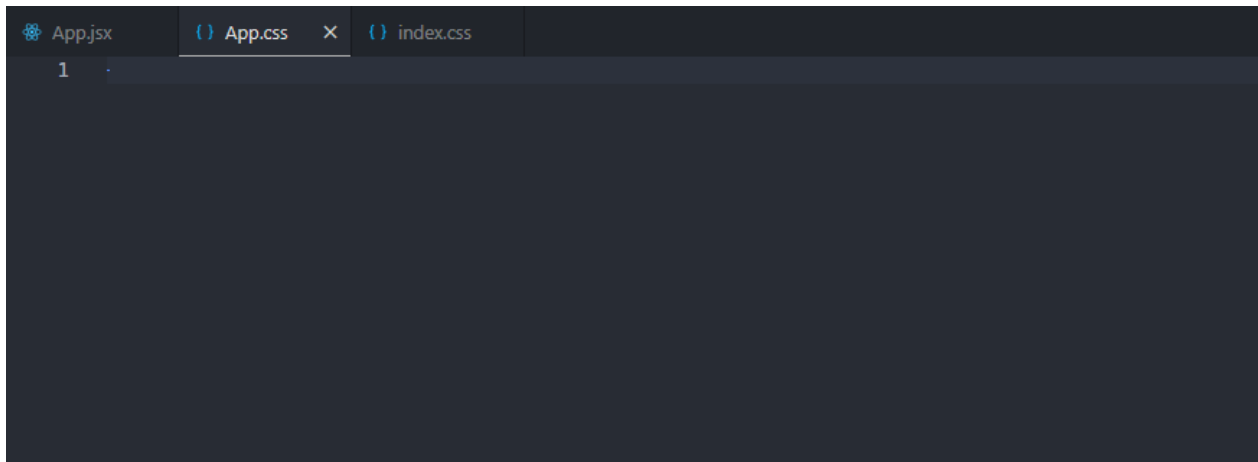


```
1  import './App.css'
2
3  function App() {
4
5      return <div className="App"><h1>Hello there!</h1></div>;
6
7
8
9  export default App
10
```

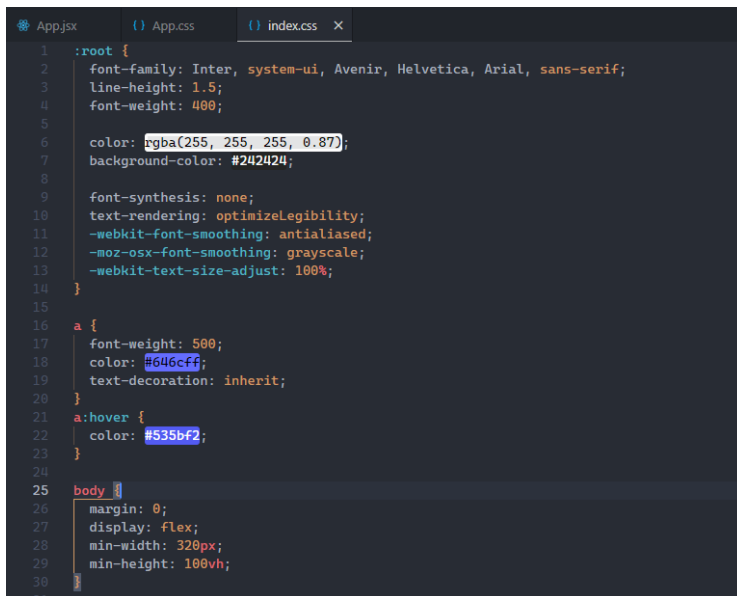
1. **import './App.css':** Esta línea de código importa un archivo CSS llamado **App.css**. Este archivo contiene estilos CSS específicos para el componente **App**.
2. **function App() { ... }:** Define el componente **App** como una función de JavaScript. En React, los componentes pueden ser funciones o clases. En este caso, se está utilizando una función.
3. **return <div className="App"></div>;** Esta línea de código es el cuerpo del componente **App**. Utiliza la sintaxis de JSX, que es una extensión de JavaScript que permite escribir código HTML-like dentro de JavaScript. En este caso, está devolviendo un elemento **div** con la clase CSS "App".
4. **export default App:** Esta línea exporta el componente **App** como el componente principal de este archivo. Esto significa que otros archivos que importen este archivo podrán utilizar este componente como **App**.

En resumen, el código define un componente funcional **App** que devuelve un elemento **div** con la clase CSS "App". Este componente puede ser utilizado en otros lugares de la aplicación como el componente principal. La importación del archivo **App.css** indica que este componente tiene estilos CSS específicos que se aplicarán a él.

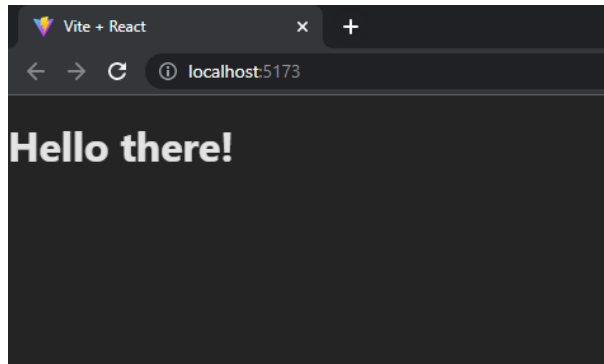
Y modificamos también **App.css** dejándolo sin estilos por el momento.



Recordemos que también tenemos **index.css** que nos permite modificar al igual que **App.css** nuestros estilos visuales

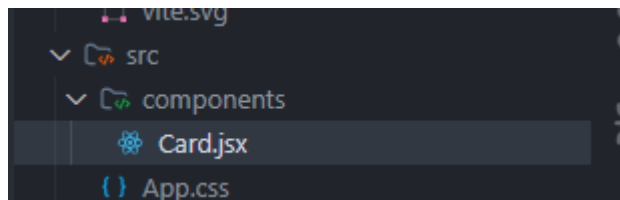


Por defecto ya vienen algunos cargados así que vamos a dejarlos por el momento, y el estilo visual nos va a quedar así:

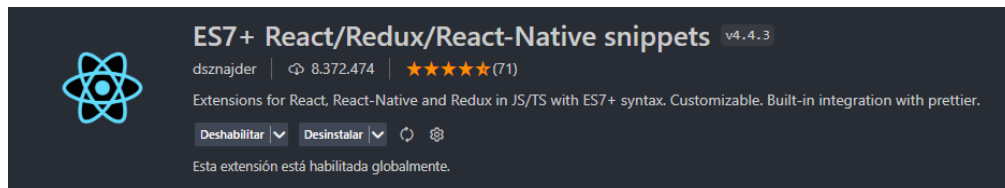


Vamos a crear nuestro primer componente, un componente es una pieza de código que renderiza una parte de la interfaz. Los componentes pueden ser parametrizados, reutilizados y pueden contener su propio estado. En React los componentes se crean usando funciones o clases.

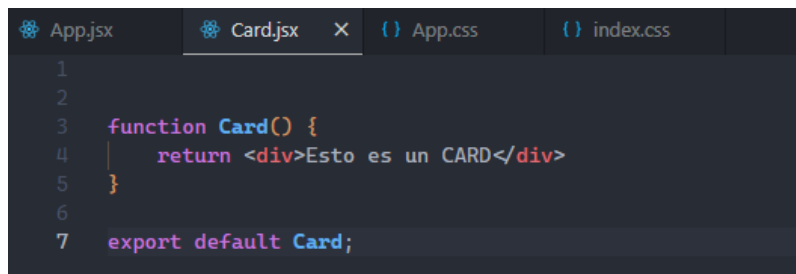
Vamos a ubicarnos en la carpeta **src** y creamos la carpeta **components** y dentro creamos un archivo llamado **Card.jsx**



Ahora para trabajar de una mejor manera se sugiere tener instalada la extensión



Esta nos permitirá agilizar la creación de código al momento de crear componentes y demás si escribimos en nuestro **Card.jsx** la abreviación **rfce** nos creará rápidamente el código, mientras tanto vamos a hacerlo manual para explicar cómo funciona el componente



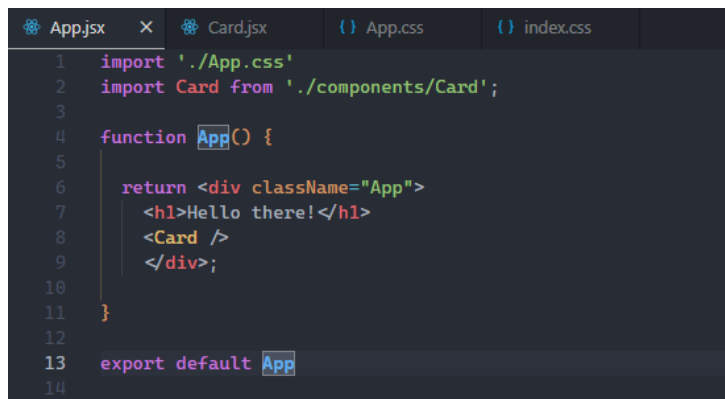
1. **function Card() { ... }:** Define el componente **Card** como una función de JavaScript. En React, los componentes pueden ser funciones o clases. En este caso, se está utilizando una función.



2. **return <div>Esto es un CARD</div>**: Esta línea de código es el cuerpo del componente **Card**. Utiliza la sintaxis de JSX, que es una extensión de JavaScript que permite escribir código HTML-like dentro de JavaScript. En este caso, está devolviendo un elemento **div** que contiene el texto "Esto es un CARD".
3. **export default Card;**: Esta línea exporta el componente **Card** como el componente principal de este archivo. Esto significa que otros archivos que importen este archivo podrán utilizar este componente como **Card**.

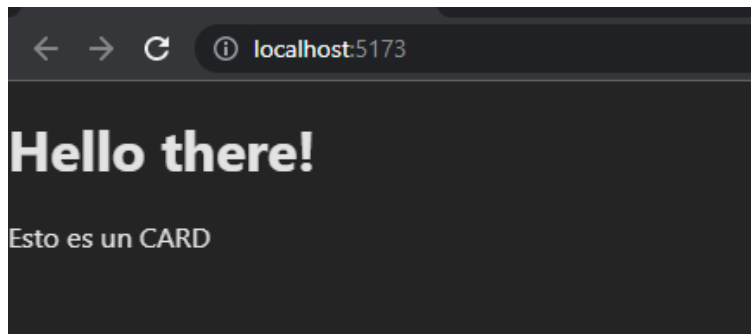
En resumen, el código define un componente funcional **Card** que devuelve un elemento **div** con el texto "Esto es un CARD". Este componente puede ser utilizado en otros lugares de la aplicación para representar un "card" o tarjeta. Al exportar el componente, se permite que otros archivos puedan importarlo y utilizarlo en sus propias aplicaciones React.

Para implementarlo lo hacemos de la siguiente manera dentro de nuestro archivo **App.jsx**



```
1 import './App.css'
2 import Card from './components/Card';
3
4 function App() {
5
6   return <div className="App">
7     <h1>Hello there!</h1>
8     <Card />
9   </div>;
10
11 }
12
13 export default App
14
```

1. Importamos el componente **import Card from './components/Card'**;
2. Agregamos **<Card />**

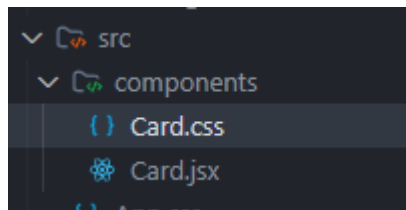


Y de esta manera hemos realizado nuestro componente de una manera muy sencilla.

Ahora vamos a modificar nuestro componente de una manera más bonita:

```
App.jsx | Card.jsx | App.css | index.css
1
2  function Card() {
3    return (
4      <div className="Card">
5        <h2>Titulo de la Tarjeta</h2>
6        <p>Descripcion de la tarjeta</p>
7      </div>
8    )
9  }
10
11
12  export default Card;
```

Procedemos a crear nuestro **Card.css**:



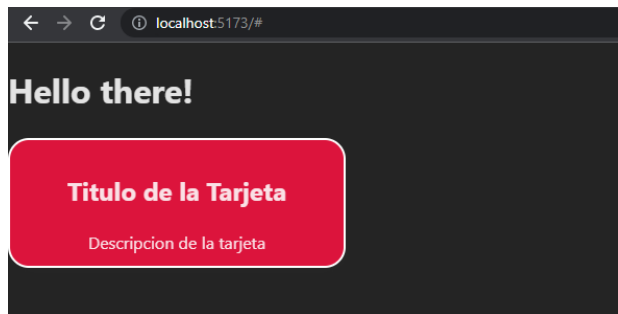
Y lo dejamos de la siguiente manera:

```
App.jsx | Card.jsx | Card.css | App.css | index.css
1  .Card {
2    width: 300px;
3    height: 100px;
4    border-radius: 20px;
5    box-shadow: 2px 2px 5px #3333;
6    background-color: crimson;
7  }
```

Y hacemos la importación de nuestro css en **Card.jsx**

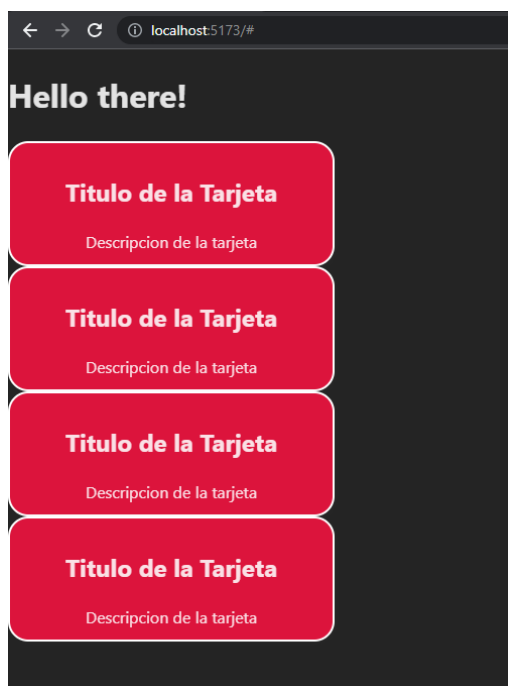
```
App.jsx | Card.jsx | Card.css | App.css | index.css
1  .Card {
2    width: 300px;
3    height: 100px;
4    border-radius: 20px;
5    box-shadow: 2px 2px 5px #3333;
6    background-color: crimson;
7    padding: 10px;
8    text-align: center;
9    border: 2px solid white;
10   gap: 10;
11 }
12
13 h2{
14   align-items: center;
15   justify-content: center;
16   display: flex;
17 }
18
19 p{
20   align-items: center;
21   justify-content: center;
22   display: flex;
23 }
```

Y nos quedaría de la siguiente manera:



Y si necesitamos mas tarjetas de este tipo simplemente agregamos la línea `<Card />`

```
App.jsx  Card.jsx  Card.css  App.css
1  import './App.css'
2  import Card from './components/Card';
3
4  function App() {
5
6    return <div className="App">
7      <h1>Hello there!</h1>
8      <Card />
9      <Card />
10     <Card />
11     <Card />
12   </div>;
13
14
15
16  export default App
17
```



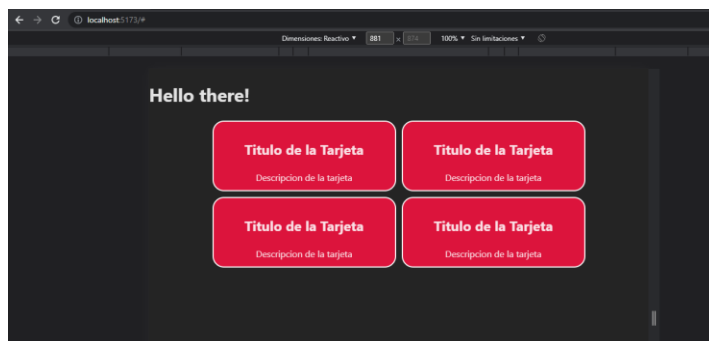
Modificamos un poco para que no se nos queden pegadas, agregamos un div con clase container y agregamos dentro las cards

```
App.jsx x Card.jsx {} Card.css {} App.css
1 import './App.css'
2 import Card from './components/Card';
3
4 function App() {
5
6   return <div className="App">
7     <h1>Hello there!</h1>
8     <div className='container'>
9       <Card />
10      <Card />
11      <Card />
12      <Card />
13    </div>
14  </div>;
15
16
17
18
19 export default App
20
```

Agregamos la clase .container en Card.css

```
App.jsx x Card.jsx {} Card.css {} App.css
1 .Card {
2   width: 300px;
3   height: 100px;
4   border-radius: 20px;
5   box-shadow: 2px 2px 5px #33333;
6   background-color: crimson;
7   padding: 10px 50px 20px;
8   text-align: center;
9   border: 2px solid white;
10 }
11
12 h2{
13   align-items: center;
14   justify-content: center;
15   display: flex;
16 }
17
18 p{
19   position: relative;
20   margin: 10px;
21   font-size: 14px;
22   line-height: 1;
23 }
24
25 .container{
26   display: flex;
27   gap: 10px;
28   flex-wrap: wrap;
29   justify-content: center;
30   align-items: center;
31 }
```

Y nos quedaría de la siguiente manera:



Ahora hablemos de las propiedades, las **props** son las propiedades de un componente. Son datos que se pasan de un componente a otro. Por ejemplo, si tienes un componente Button que muestra un botón, puedes pasarle una prop text para que el botón muestre ese texto:

```
function Button(props) {  
  return <button>{props.text}</button>  
}
```

Podríamos entender que el componente Button es un botón genérico, y que la prop text es el texto que se muestra en el botón. Así estamos creando un componente reutilizable.

Debe considerarse además que al usar cualquier expresión JavaScript dentro de JSX debe envolverlos con {}, en este caso el objeto props, de otra forma JSX lo considerará como texto plano.

Para usarlo, indicamos el nombre del componente y le pasamos las props que queremos:

```
<Button text="Haz clic aquí" />  
<Button text="Seguir a @hdtoledo" />
```

Las props son una forma de parametrizar nuestros componentes igual que hacemos con las funciones. Podemos pasarle cualquier tipo de dato a un componente, incluso otros componentes.

Ahora vamos a agregar props a nuestro componente card, vamos a modificar de la siguiente manera **Card.jsx**

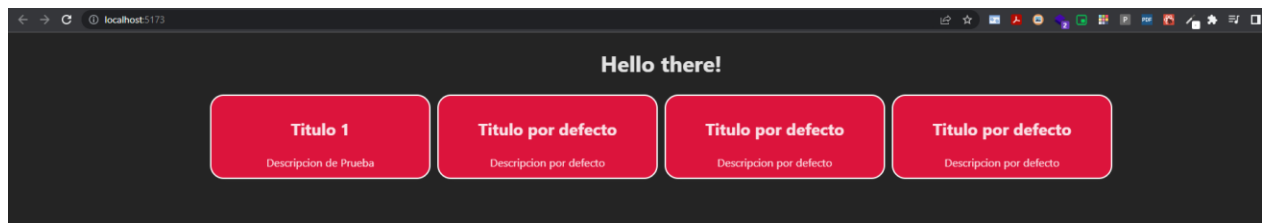
```
1  
2  
3 import "./Card.css"  
4  
5 function Card({title = "Titulo por defecto", description = "Descripcion por defecto"}) {  
6   return (  
7     <div className="Card">  
8       <h2>{title}</h2>  
9       <p>{description}</p>  
10    </div>  
11  )  
12 }  
13  
14 export default Card;
```

Dentro de la function Card estamos pasando unos atributos que son title y description que vienen con unos textos por defecto, y estamos accediendo a las propiedades de estos a través del **h2** y **p** utilizando **{title}** y **{description}**

Y en **App.jsx** podemos modificar de las siguientes maneras:

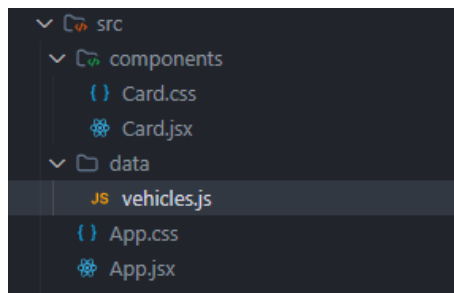
```
App.jsx x Card.jsx
1 import './App.css'
2 import Card from './components/Card';
3
4 function App() {
5
6   return <div className="App">
7     <h1>Hello there!</h1>
8     <div className="container">
9       <Card title="Titulo 1" description="Descripcion de Prueba"/>
10      <Card />
11      <Card />
12      <Card />
13    </div>
14  </div>;
15
16
17 }
18
19 export default App
20
```

Aplicamos title y description sobre nuestra card y nos quedaría de la siguiente manera



De esta manera podemos acceder a los datos dentro de un objeto y pasar la información.

Vamos a simular que tenemos una data de vehículos y que necesitamos pasar mas de 100 datos sobre nuestras tarjetas, vamos a crear la carpeta **data** y el archivo **vehicles.js**



Ahora agregamos nuestra data en el archivo, lo pueden descargar de [acá](#),

```
App.jsx x vehicles.js x Card.jsx
1 const vehicles = [
2   {
3     name: "Coche",
4     description: "Un vehiculo de transporte terrestre con capacidad para varias personas.",
5     image: "https://cdn.computerhoy.com/sites/navi.axelspringer.es/public/media/image/2022/06/coche-hidrogeno-2739247.jpg",
6   },
7   {
8     name: "Bicicleta",
9     description: "Un vehiculo de transporte de una o dos personas que se impulsa mediante pedales.",
10    image: "https://cdn.brujulabike.com/media/29040/conversions/talla-de-bicicleta-1240.jpg",
11  },
12  {
13    name: "Motocicleta",
14    description: "Un vehiculo de transporte de dos ruedas y motor de combustión interna.",
15    image: "https://www.pruebaderinta.com/wp-content/uploads/2016/08/tipos-chasis-moto-a.jpg",
16  },
17  {
18    name: "Camión",
19    description: "Un vehiculo de transporte de carga pesada utilizado para transportar mercancías.",
20    image: "https://cdn.computerhoy.com/sites/navi.axelspringer.es/public/media/image/2021/12/iveco-camion-autonomo-2569959.jpg",
21  },
22  {
23    name: "Avión",
24    description: "Un vehiculo de transporte aéreo que permite volar a través de las altitudes.",
25    image: "https://cdn.computerhoy.com/sites/navi.axelspringer.es/public/media/image/2021/12/avion-united-airlanes-2551587.jpg",
26  },
27 ]
```

Vamos a modificar nuestro **App.jsx**

```
import './App.css'
import Card from './components/Card';
import vehicles from './data/vehicles';

function App() {
  const vehicleList = vehicles.map((v) => {
    return <Card title={v.name} description={v.description}/>;
  });

  return <div className="App">
    <h1>Hello there!</h1>
    <div className='container'>
      {vehicleList}
    </div>
  </div>;
}

export default App
```

1. Importaciones:

- **import './App.css'**: Importa un archivo CSS llamado **App.css**. Esto permite aplicar estilos específicos al componente **App**.
- **import Card from './components/Card'**: Importa el componente **Card** desde un archivo ubicado en la carpeta **components** del proyecto. Esto permite utilizar el componente **Card** en el componente **App**.
- **import vehicles from './data/vehicles'**: Importa un array llamado **vehicles** desde un archivo ubicado en la carpeta **data** del proyecto. Estos datos de vehículos serán utilizados en la lógica del componente **App**.

2. Definición del componente **App**:

- **function App() {...}**: Define el componente **App** como una función. Este es un componente funcional en React.

3. Lógica del componente **App**:

- **const vehicleList = vehicles.map((v) => {...})**: Crea una variable **vehicleList** que contiene el resultado de mapear cada elemento del array **vehicles**. Para cada vehículo (**v**), se devuelve un componente **Card** con las propiedades **title** y **description** que se obtienen de cada vehículo.

4. Renderizado del componente **App**:

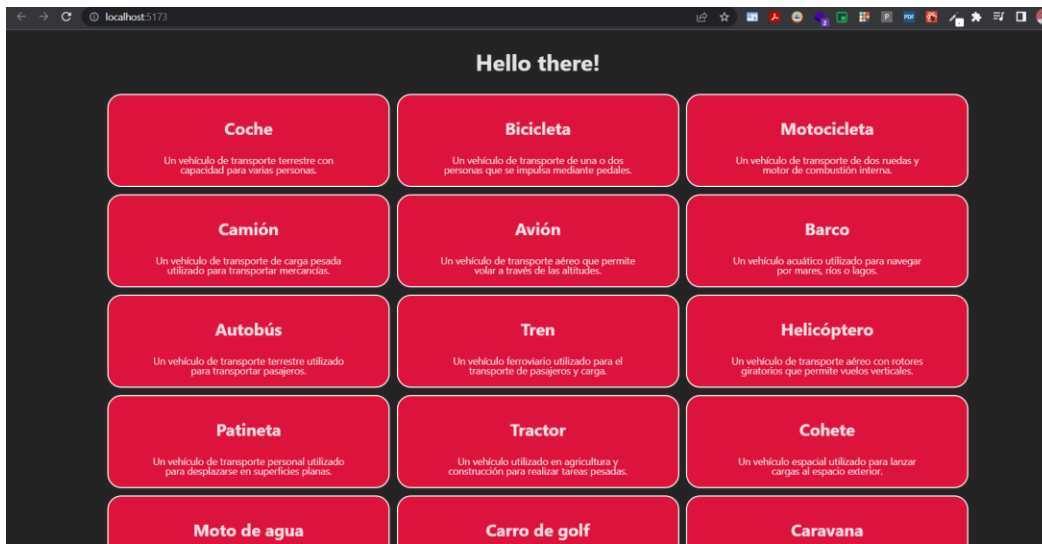
- **return <div className="App">...</div>**: Renderiza el contenido del componente **App**. El contenido está envuelto en un elemento **div** con la clase CSS "App".
- **<h1>Hello there!</h1>**: Renderiza un encabezado **h1** con el texto "Hello there!". Este texto se muestra en la interfaz de usuario.
- **<div className='container'>...</div>**: Renderiza un elemento **div** con la clase CSS "container". Este elemento envuelve la lista de tarjetas de vehículos.

- **{vehicleList}**: Renderiza la variable **vehicleList**, que contiene la lista de tarjetas de vehículos generadas mediante el mapeo del array **vehicles**.

##### 5. Exportación del componente **App**:

- **export default App**: Exporta el componente **App** para que pueda ser utilizado en otros componentes o archivos del proyecto.

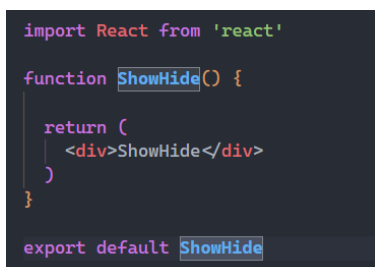
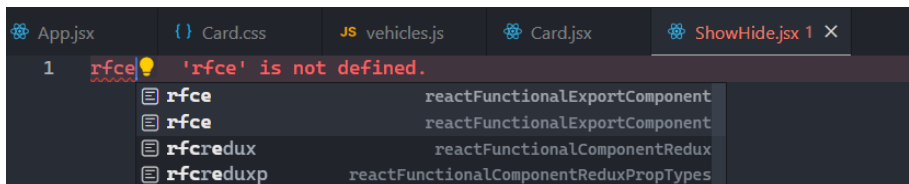
En resumen, este código importa un componente **Card**, un array de datos de vehículos y un archivo CSS. Luego, utiliza estos recursos para renderizar una lista de tarjetas de vehículos en el componente **App**. Cada tarjeta muestra el nombre y la descripción de un vehículo.



De esta manera podemos traer la información a través de nuestra data que se encuentra almacenada en un array dentro de **vehicles.js**

Aplica las imágenes que vienen dentro del array, creando un parámetro mas en la **function Card** y agregando el **css**.

Ahora vamos a crear un componente que se llame **ShowHide.jsx** para practicar otros conceptos, en este utilizamos la abreviación de **rfce**





Hablemos de que es un **hook**, los Hooks son una API de React que nos permite tener estado, y otras características de React, en los componentes creados con una function.

Los hooks son funciones predefinidas por React que puedes utilizar dentro de tus componentes funcionales para agregar y manejar estado, efectos, contexto y más. Al utilizar hooks, puedes escribir componentes funcionales más poderosos y reutilizables sin necesidad de convertirlos en componentes de clase.

Algunos de los hooks más comunes en React son:

1. `useState`: Permite agregar y utilizar estado interno en componentes funcionales. Puedes declarar una variable de estado y una función para actualizarla.
2. `useEffect`: Permite realizar efectos secundarios en componentes funcionales, como la llamada a una API, suscripciones a eventos, cambios en el DOM, entre otros. Se ejecuta después de que el componente se renderiza.
3. `useContext`: Permite acceder al contexto de React en componentes funcionales. Puedes acceder a los valores proporcionados por el contexto y sus actualizaciones.
4. `useRef`: Permite crear una referencia mutable que persiste durante la vida útil del componente. Puedes utilizarlo para acceder a elementos del DOM, almacenar valores mutables, etc.
5. `useMemo`: Permite memorizar el resultado de una función costosa para evitar cálculos innecesarios. Solo se actualiza cuando las dependencias cambian.
6. `useCallback`: Permite memorizar una función para evitar la recreación en cada renderizado del componente. Útil cuando necesitas pasar una función a componentes hijos.

Estos son solo algunos ejemplos de los hooks disponibles en React. Cada hook tiene un propósito específico y te permite añadir diferentes funcionalidades a tus componentes funcionales. Puedes crear tus propios hooks personalizados para reutilizar lógica en diferentes componentes.

Es importante tener en cuenta que los hooks deben seguir ciertas reglas de uso, como llamarse solo en el nivel superior de un componente funcional, no llamarlos dentro de bucles, condiciones o funciones anidadas. Esto garantiza que React pueda rastrear correctamente el estado y los efectos de los hooks.

Vamos a modificar de la siguiente manera nuestro **ShowHide.jsx**

```
import React from 'react'
import { useState } from 'react'

function ShowHide() {
  const [Show, setShow] = useState(true);

  return (
    <div>
      {Show ? <h2>Escondeme !</h2> : ""}
    </div>
  )
}

export default ShowHide
```

### 1. Importaciones:

- **import React from 'react'**: Importa el módulo **React** necesario para definir componentes en React.
- **{ useState } from 'react'**: Importa el hook **useState** desde el módulo **react**. El hook **useState** permite agregar y utilizar estado interno en componentes funcionales.

### 2. Definición del componente **ShowHide**:

- **function ShowHide() {...}**: Define el componente **ShowHide** como una función. Este es un componente funcional en React.

### 3. Estado interno utilizando **useState**:

- **const [Show, setShow] = useState(true)**: Utiliza el hook **useState** para declarar una variable de estado llamada **Show** y una función para actualizarla llamada **setShow**. La variable de estado **Show** se inicializa con el valor **true**, lo que significa que inicialmente el componente muestra el mensaje "Escondeme !".

### 4. Renderizado del componente **ShowHide**:

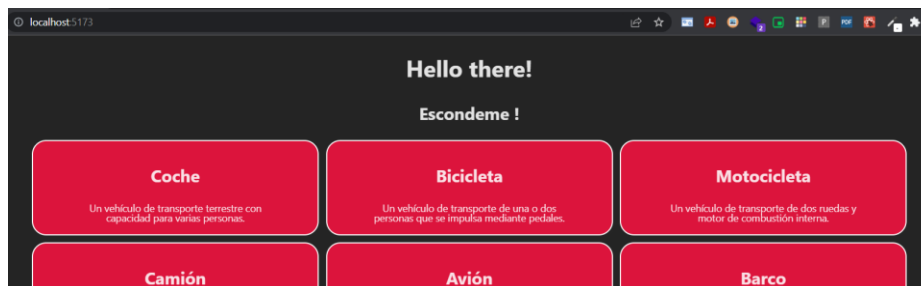
- **<div>...</div>**: Renderiza un elemento **div** que contiene el contenido del componente.
- **{Show ? <h2>Escondeme !</h2> : ""}**: Utiliza una expresión ternaria para renderizar un elemento **h2** con el texto "Escondeme !" si **Show** es **true**, de lo contrario, muestra una cadena vacía. En otras palabras, si **Show** es **true**, se mostrará el mensaje "Escondeme !", de lo contrario, no se mostrará nada.

### 5. Exportación del componente **ShowHide**:

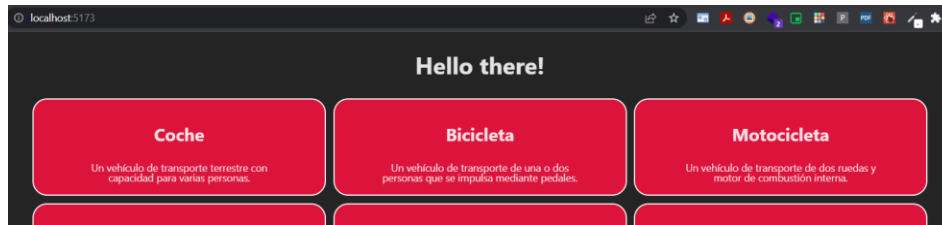
- **export default ShowHide**: Exporta el componente **ShowHide** para que pueda ser utilizado en otros componentes o archivos del proyecto.

En resumen, el componente **ShowHide** utiliza el hook **useState** para agregar y gestionar un estado interno llamado **Show**. Dependiendo del valor de **Show**, se muestra o no un mensaje "Escondeme !". Al cambiar el estado utilizando **setShow**, el componente se volverá a renderizar y mostrará o ocultará el mensaje según corresponda.

De esta manera se nos va a mostrar el texto en nuestra vista:



Si cambiamos el estado de true a False se ocultará.



Ahora hagámoslo de otra manera sin el operador ternario, cambiemos por el operador && y eliminamos la opción de falso

```
import React from 'react'
import { useState } from 'react'

function ShowHide() {
  const [Show, setShow] = useState(true);

  return (
    <div>
      {Show && <h2>Escondeme !</h2>}}
    </div>
  )
}

export default ShowHide
```

Si lo aplicamos de esta manera también funcionara, Show tiene dos propiedades, true o false, que son las que estamos modificando y por ende el valor siempre se vera reflejado, vamos a modificar ahora un poco y agregaremos un button:

```
import React from 'react'
import { useState } from 'react'

function ShowHide() {
  const [Show, setShow] = useState(true);

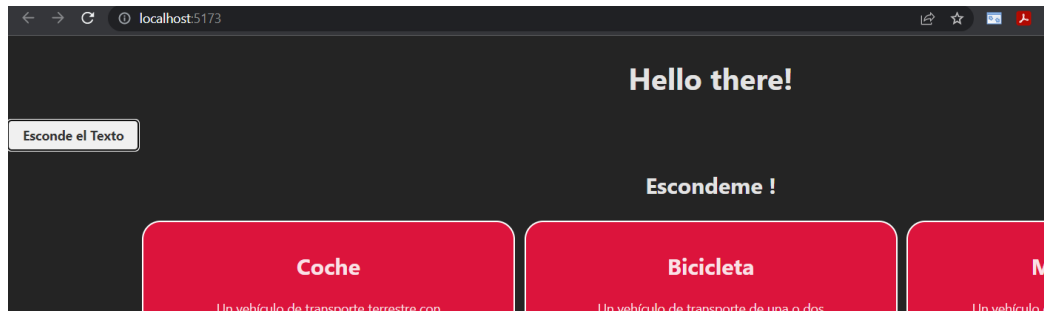
  const handleClick = (event) => {
    setShow(!Show);
  };

  return (
    <div>
      {Show && <h2>Escondeme !</h2>}}
      <button onClick={handleClick}>Esconde el Texto</button>
    </div>
  )
}

export default ShowHide
```

- En la primera línea, se define la función ShowHide.
- La línea **const [Show, setShow] = useState(true);** declara una variable de estado llamada **Show** y una función **setShow** para modificar su valor. El valor inicial de **Show** se establece en **true**, lo que significa que el texto se mostrará inicialmente.

- A continuación, se define una función llamada **handleClick** que se ejecuta cuando se hace clic en el botón. Esta función utiliza **setShow** para cambiar el valor de **Show** al valor opuesto, es decir, si **Show** es **true**, se cambiará a **false**, y viceversa.
- Dentro del bloque **return**, se devuelve el contenido JSX que será renderizado por React. En este caso, se muestra un **div** que contiene un elemento **h2** con el texto "Escondeme !" si **Show** es **true**. Además, hay un botón con el texto "Esconde el Texto" que al hacer clic ejecutará la función **handleClick**.
- Dependiendo del valor de **Show**, el texto y el botón se mostrarán o se ocultarán en la interfaz de usuario.



De esta manera si presionamos el botón se esconderá el texto y vice-versa ahora modificamos que el texto que dice en el botón cuando este oculto diga mostrar y si es el contrario que diga ocultar, vamos a utilizar un ternario para ello:

```
import React from 'react'
import { useState } from 'react'

function ShowHide() {
  const [Show, setShow] = useState(true);

  const handleClick = (event) => {
    setShow(!Show);
  };

  return (
    <div>
      <button onClick={handleClick}>{Show ? "Ocultar" : "Mostrar"} Texto</button>
      {Show && <h2>Escondeme !</h2>}
    </div>
  )
}

export default ShowHide
```

A través del ternario le estamos asignando las palabras Ocultar y Mostrar y de esta manera cambiarán si presionamos el botón, hasta acá hemos visto Estados, eventos de clic, renderizado condicional, ahora vamos a ocultar ShowHide y vamos a ver el tema de rutas.

En React, las rutas se utilizan para definir la navegación y el enrutamiento en una aplicación de una sola página (SPA, por sus siglas en inglés). Las rutas permiten que diferentes componentes se muestren en función de la URL actual.

React utiliza una biblioteca llamada [React Router](#) para manejar las rutas en una aplicación. React Router proporciona un conjunto de componentes que se utilizan para definir las rutas y gestionar la navegación entre ellas.

Algunos de los componentes clave que se utilizan en React Router son:

- **<BrowserRouter>**: Este componente se coloca alrededor de la aplicación y define el enrutador principal. Utiliza la API de Historial del navegador para manejar las rutas.
- **<Route>**: Se utiliza para definir una ruta específica en la aplicación. Se le asigna una **path** (ruta) y un componente que se debe renderizar cuando la URL coincide con la ruta especificada.
- **<Switch>**: Este componente se utiliza para agrupar las rutas y asegurarse de que solo se renderice el primer componente que coincide con la URL actual. Esto es útil para evitar que se muestren múltiples componentes cuando se tiene una URL que coincide con varias rutas.
- **<Link>**: Se utiliza para crear enlaces a otras rutas en la aplicación. Permite la navegación entre las diferentes páginas o componentes sin recargar la página completa.

Estos son solo algunos de los componentes principales que se utilizan en React Router. Hay otros componentes y funcionalidades adicionales disponibles para manejar casos más complejos, como parámetros en las rutas, rutas anidadas, redirecciones, entre otros.

En resumen, las rutas en React permiten definir la navegación en una aplicación de una sola página y proporcionan una forma de mostrar componentes específicos según la URL actual. Esto se logra utilizando la biblioteca React Router y sus componentes relacionados.

Ahora vamos a nuestra terminal y escribiremos el siguiente comando

```
hdtol@LAPTOPHDMSI MINGW64 /d/DATA/Documents/react_basics
$ npm install react-router-dom
```

Una vez instalado nos vamos a nuestro **main.jsx** y vamos a agregar algunas líneas:

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

import { BrowserRouter, RouterProvider } from 'react-router-dom'

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />
  }
])

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
)
```

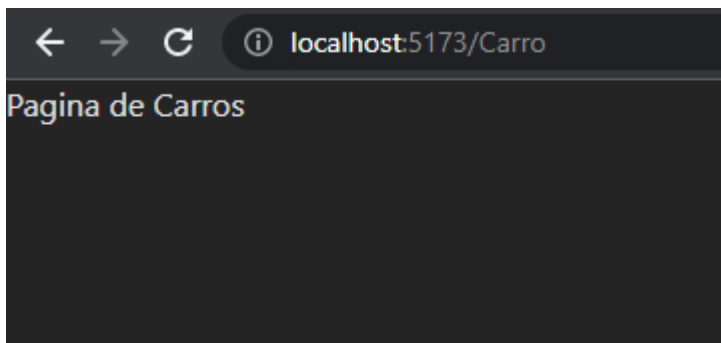
- **createBrowserRouter** y **RouterProvider** son componentes y funciones proporcionados por **react-router-dom**.
- **createBrowserRouter** se utiliza para crear un enrutador de navegación. Recibe un arreglo de objetos de configuración de ruta. En este caso, hay una única ruta definida con la propiedad **path** establecida en "/" (la ruta raíz) y la propiedad **element** establecida en el componente **<App />**.
- **RouterProvider** es un componente que proporciona el enrutador creado (**router**) a la aplicación para que pueda manejar la navegación.
- Luego, se utiliza **ReactDOM.createRoot** para renderizar la aplicación en el elemento con el id 'root' del documento HTML.
- La aplicación se renderiza en un elemento **<React.StrictMode>**, que es un contenedor especial proporcionado por React para detectar y resaltar posibles problemas en el código durante el desarrollo.
- Dentro del elemento **<React.StrictMode>**, se coloca el componente **RouterProvider** y se le pasa el enrutador (**router**) como prop a través de la propiedad **router**.

En resumen, este código crea un enrutador de navegación utilizando **createBrowserRouter** y define una única ruta para la ruta raíz ("/") que se renderizará utilizando el componente **<App />**. Luego, se utiliza **ReactDOM.createRoot** para renderizar la aplicación con el enrutador proporcionado por **RouterProvider**.

Ahora vamos a crear una ruta de prueba en el mismo archivo, solo modificamos lo siguiente:

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />
  },
  {
    path: "Carro",
    element: <div>Pagina de Carros</div>
  }
])
```

Si vamos a nuestro navegador y escribimos /Carro y presionamos enter nos mostrara lo siguiente:



De esta manera comprobamos que el enrutamiento fue creado y esta funcional.

Ahora vamos a crear nuestras rutas para los vehículos, hacemos la siguiente modificación:

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

import { createBrowserRouter, RouterProvider } from 'react-router-dom'
import vehicles from './data/vehicles.js'

const routes = [
  {
    path: "/",
    element: <App />
  },
];

vehicles.forEach((vehicle) => {
  routes.push({
    path: vehicle.name,
    element: <div>{vehicle.name}</div>
  });
});

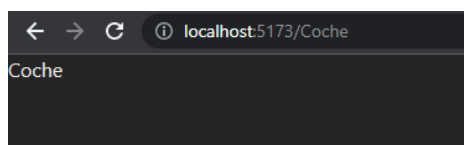
const router = createBrowserRouter(routes);

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>,
)
```

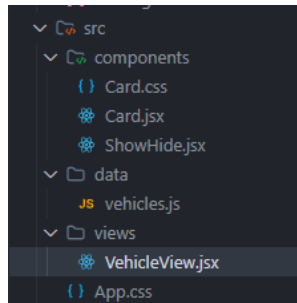
- **import vehicles from './data/vehicles.js'** importa los datos de vehículos desde el archivo **vehicles.js**.
- Se declara un arreglo llamado **routes** que inicialmente contiene una única ruta para la ruta raíz ("/") con el componente **<App />** asociado.
- A continuación, se utiliza el método **forEach** en el arreglo de **vehicles** para iterar sobre cada elemento del arreglo.
- Dentro del bucle **forEach**, se agrega una nueva ruta al arreglo **routes** para cada vehículo. La ruta se define utilizando el nombre del vehículo (**vehicle.name**) como la ruta y se asocia a un componente que muestra el nombre del vehículo en un elemento **<div>**.
- Finalmente, se utiliza **createBrowserRouter** para crear un enrutador de navegación utilizando el arreglo **routes** que se ha construido.

En resumen, importamos los datos de vehículos, creamos un arreglo de rutas inicializado con una ruta raíz y el componente principal **<App />**, y luego agregamos rutas adicionales basadas en los datos de los vehículos. El enrutador de navegación se crea utilizando las rutas definidas.

Podemos hacer la comprobación escribiendo la ruta en nuestro navegador con el nombre de cada uno de los vehículos:



Ahora vamos a crear una vista para que se nos muestre de una mejor manera nuestras rutas, así que vamos a crear en **src** la carpeta **views** y la página **VehicleView.jsx**



Creamos el rfec con la abreviación y hacemos lo siguiente:

```
import React from 'react'

function VehicleView({vehicle}) {
  return (
    <div>
      <h1>{vehicle.name}</h1>
      <h2>{vehicle.description}</h2>
      <img src={vehicle.image} alt={vehicle.name + " imagen"} />
    </div>
  )
}

export default VehicleView
```

- La función **VehicleView** recibe un objeto **vehicle** como argumento desestructurado de las props.
- En el retorno de la función, se devuelve un fragmento de JSX que contiene varios elementos HTML.
- **<h1>{vehicle.name}</h1>** renderiza el nombre del vehículo dentro de un encabezado de nivel 1 (**<h1>**).
- **<h2>{vehicle.description}</h2>** renderiza la descripción del vehículo dentro de un encabezado de nivel 2 (**<h2>**).
- **<img src={vehicle.image} alt={vehicle.name + " imagen"} />** renderiza una imagen del vehículo. La URL de la imagen se toma de la propiedad **image** del objeto **vehicle**. El atributo **alt** proporciona un texto alternativo para la imagen que combina el nombre del vehículo con la palabra "imagen".
- Todo el contenido se envuelve en un elemento **<div>** para agruparlos.
- Por último, se exporta el componente **VehicleView** como el valor predeterminado para que pueda ser utilizado en otros componentes.

Ahora nos vamos a modificar **main.jsx** en el element y le vamos a pasar nuestra vista de vehículos.

```
vehicles.forEach((vehicle) => {
  routes.push({
    path: vehicle.name,
    element: <VehicleView vehicle={vehicle} />
  });
});
```



De esta manera si comprobamos nuevamente las rutas ya obtendremos una vista funcional con los detalles de nuestros vehículos:



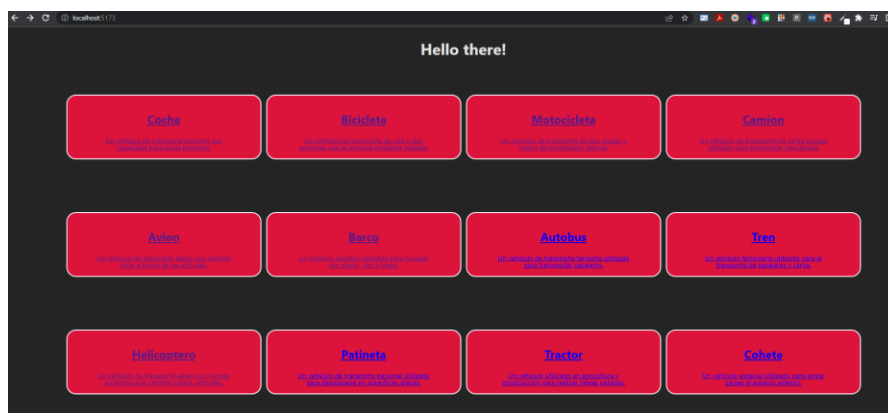
Ahora nos vamos a **Card.jsx** y vamos a crear un link para cada uno de nuestras tarjetas y que cuando le demos clic nos lleve a nuestra vista, para ello importamos el componente **link** de **react-router-dom** y hacemos lo siguiente:

```
import './Card.css'
import { Link } from 'react-router-dom';

function Card({title = "Titulo por defecto", description = "Descripcion por defecto"}) {
  return (
    <div className="Card">
      <Link to={title}>
        <h2>{title}</h2>
        <p>{description}</p>
      </Link>
    </div>
  )
}

export default Card;
```

De esta manera agregamos link y enlazamos toda la tarjeta para que cuando le demos clic nos lleve a la ruta correcta, una buena práctica debería aplicarse correctamente sobre la ruta y llevarnos a la ruta que es y no como en el ejemplo que estamos utilizando el titulo para ir a la ruta.



De esta manera aprendimos a manipular varias propiedades de React, agregamos interactividad, aplicamos estilos y jugamos con varios detalles para dar inicio a React.