Introducción a React 

Objetivos

- Que es React
- Que es Vite
- Conociendo JSX
- Crear nuestro primer proyecto con React y Vite
- Agregar CSS a un proyecto React
- Traspasar información a través de props
- Destructuring de props
- Renderizar fragmentos o componentes según condiciones
- Instalar y utilizar React Bootstrap

¿Qué es React?

React es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reactivas. Es una de las bibliotecas más populares para el desarrollo de aplicaciones web modernas y se basa en el concepto de componentes reutilizables.

Las principales características y conceptos clave de React incluyen:

- **Componentes:**
  React divide la interfaz de usuario en componentes pequeños y reutilizables. Estos componentes encapsulan su propia lógica y representan una parte específica de la interfaz.

JSX (JavaScript XML)
React utiliza JSX, una extensión de sintaxis que permite escribir código HTML similar dentro de JavaScript. Esto facilita la creación y renderizado de componentes en React.

Virtual DOM:
React utiliza un Virtual DOM (DOM virtual) para mejorar el rendimiento. El Virtual DOM es una representación en memoria del DOM real y permite a React realizar cambios de manera eficiente, minimizando las manipulaciones directas del DOM.

Reconciliación:
React realiza una comparación eficiente entre el Virtual DOM y el DOM real para identificar los cambios y actualizar solo las partes relevantes de la interfaz de usuario. Esto ayuda a mejorar el rendimiento y la velocidad de la aplicación.

React Hooks:
Introducidos en versiones más recientes de React, los Hooks permiten que los componentes funcionales de React tengan estado y utilicen características previamente reservadas para componentes basados en clases. Esto simplifica la creación y gestión de estados en componentes funcionales.

¿Qué es Vite?

Vite es una herramienta de construcción (build tool) y desarrollo (development tool) rápida y eficiente para aplicaciones web desarrolladas con tecnologías modernas como JavaScript, TypeScript, Vue.js y React. Fue creada para mejorar la experiencia de desarrollo y optimizar el tiempo de construcción en proyectos web.

Algunas características clave de Vite incluyen:

Desarrollo rápido:
Vite ofrece un servidor de desarrollo que utiliza el "módulo nativo" de JavaScript (ECMAScript Modules) para servir la aplicación en el navegador. Esto permite un inicio rápido y una recarga en caliente (hot module replacement) muy rápida. En lugar de empaquetar y compilar todo el código antes de ejecutar la aplicación, Vite solo compila los módulos a medida que se solicitan, lo que agiliza el tiempo de desarrollo.

Soporte para Vue.js y React:
Vite fue inicialmente creado para Vue.js, pero también es compatible con React y otras bibliotecas basadas en ECMAScript Modules.

Resolución de dependencias esencialmente "cero-config":
En la mayoría de los casos, no se necesita un archivo de configuración (como un archivo "vite.config.js") para comenzar con Vite. La herramienta puede resolver automáticamente las dependencias y ajustar la configuración en función del entorno y el tipo de proyecto.

Extensión JSX

JSX (JavaScript XML) es una extensión de sintaxis utilizada en React y otras bibliotecas similares para definir la estructura de las interfaces de usuario de manera más sencilla y declarativa. JSX permite escribir código HTML similar dentro de JavaScript, lo que facilita la creación de componentes en React.

En lugar de crear elementos DOM directamente utilizando JavaScript puro, JSX permite definir la estructura de los elementos DOM dentro de bloques de código JavaScript utilizando una sintaxis similar al HTML. Aquí hay un ejemplo de cómo se ve JSX:

JSX:

```
import React from 'react';

const MiComponente = () => {
  return (
    <div>
      <h1>Mi componente</h1>
      <p>Este es mi componente</p>
    </div>
  );
  );
}
```

En este ejemplo, MiComponente es un componente funcional en React que devuelve JSX. La estructura dentro del bloque de retorno se asemeja al código HTML, pero en realidad, es código JavaScript que será transformado en elementos DOM cuando se ejecute la aplicación.

Es importante tener en cuenta que JSX no es obligatorio para usar React, pero se ha convertido en la forma más común y recomendada de definir interfaces de usuario en aplicaciones React debido a su legibilidad y facilidad de uso.

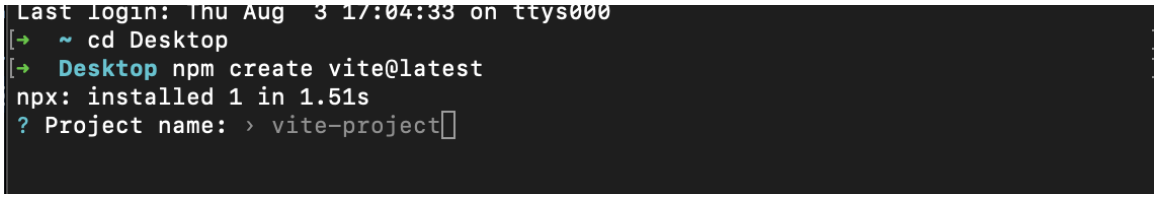
Creando nuestro primer proyecto con React y Vite

Para crear nuestro primer proyecto con React y Vite, debemos tener instalado Node.js y npm en nuestro equipo. Si no los tienes instalados, puedes descargarlos e instalarlos desde el sitio web oficial de Node.js.

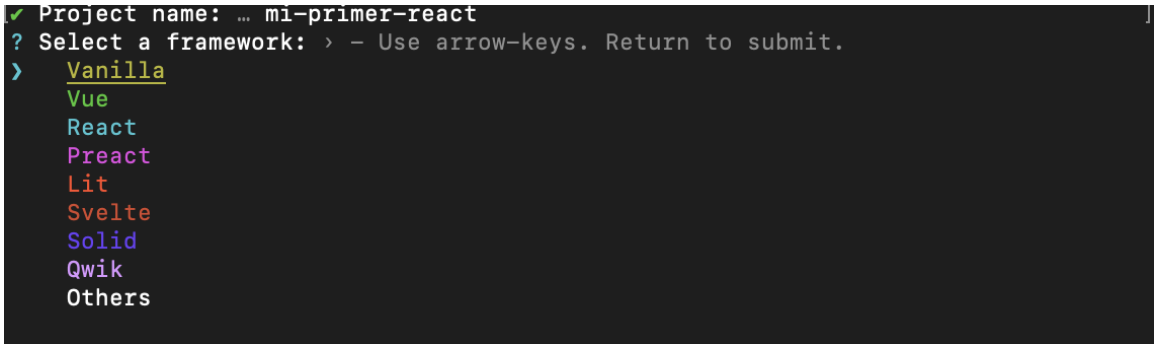
Una vez que tengamos Node.js y npm instalados, podemos crear nuestro primer proyecto con React y Vite utilizando el siguiente comando:

npm create vite@latest

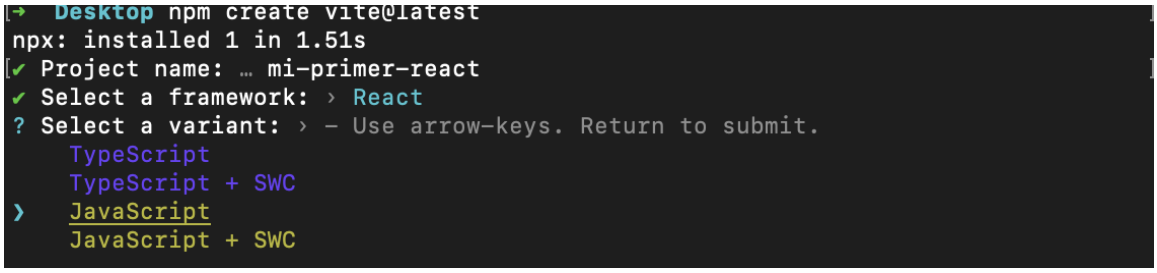
Luego el siguiente paso nos solicitara un nombre para nuestro proyecto.



En el siguiente paso vamos a seleccionar el Framework con el que vamos a trabajar como el objetivo de esta unidad es React seleccionaremos ese.



A continuación debemos seleccionar una variante en este caso seleccionamos JavaScript.



Para finalizar usamos el comando **cd** para cambiarnos a la carpeta del proyecto y ejecutamos el siguiente comando:

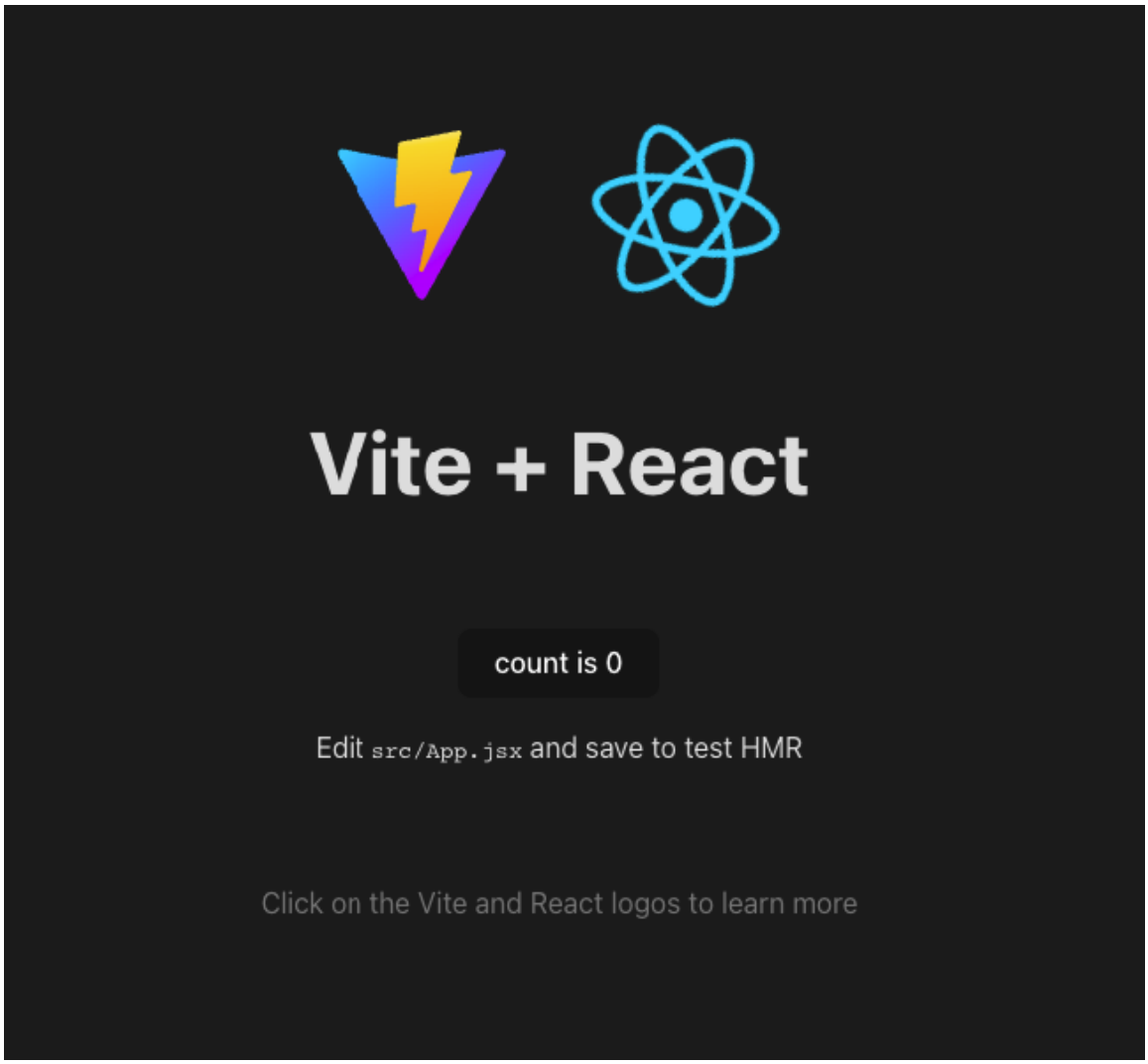
npm install

Este comando instalará las dependencias para que el proyecto funcione sin problemas.

Por último usamos nuestro editor preferido y abrimos el proyecto recién creado. Una vez abierto el proyecto en la terminal ejecutamos el siguiente comando para desplegar el proyecto:

npm run dev

Esto nos devolverá una URL la cual pegaremos en nuestro navegador y veremos la siguiente imagen a continuación:



Agregando JavaScript a nuestro proyecto React

Para agregar JavaScript a nuestro proyecto React, debemos crear un archivo JSX o usar uno existente y agregar el código JavaScript dentro de la funcion que retorna el html del componente pero arriba de la clausula return.

Ejemplo:

```
import React from 'react';

const MiComponente = () => {
  console.log('Hola desde el componente');
  return (
    <div>
      <h1>Mi componente</h1>
      <p>Este es mi componente</p>
    </div>
  );
  );
}
```

Ejemplo usando variables:

```
import React from 'react';

const MiComponente = () => {
  const fullName = 'Juan Perez'
  return (
    <div>
      <h1>Mi componente</h1>
      <p>Hola mi nombre es {fullName}</p>
    </div>
  );
  );
}
```

Como podemos apreciar podemos pasar variables de JavaScript directamente al código HTML que está en la función esto nos dice que hay una infinidad de cosas que podemos hacer con JavaScript y React.

Agregando CSS a nuestro proyecto React

En React existen varias formas de agregar estilos a tu proyecto pero vamos a partir por lo esencial en HTML usábamos la palabra class para asignar una clase de esta forma podíamos llamarla desde un archivo CSS para posteriormente asignar los estilos de nuestra preferencia en React esto no es muy distinto la única diferencia está en que no usamos la palabra class si no la palabra className para asignar un nombre a una clase.

Ejemplo:

```
import React from 'react';

const MiComponente = () => {
  return (
    <div className="nombre de la clase">
      <h1>Mi componente</h1>
      <p>Hola mi nombre es {fullName}</p>
    </div>
  );
  );
}
```

Ahora que ya sabemos como asignar clases a nuestros elementos, vamos a ver algunas formas de usar CSS en nuestro proyecto

CSS Tradicional:
Puedes utilizar hojas de estilo CSS tradicionales para estilizar tu proyecto React. Puedes crear archivos CSS independientes y luego importarlos en tus componentes React.

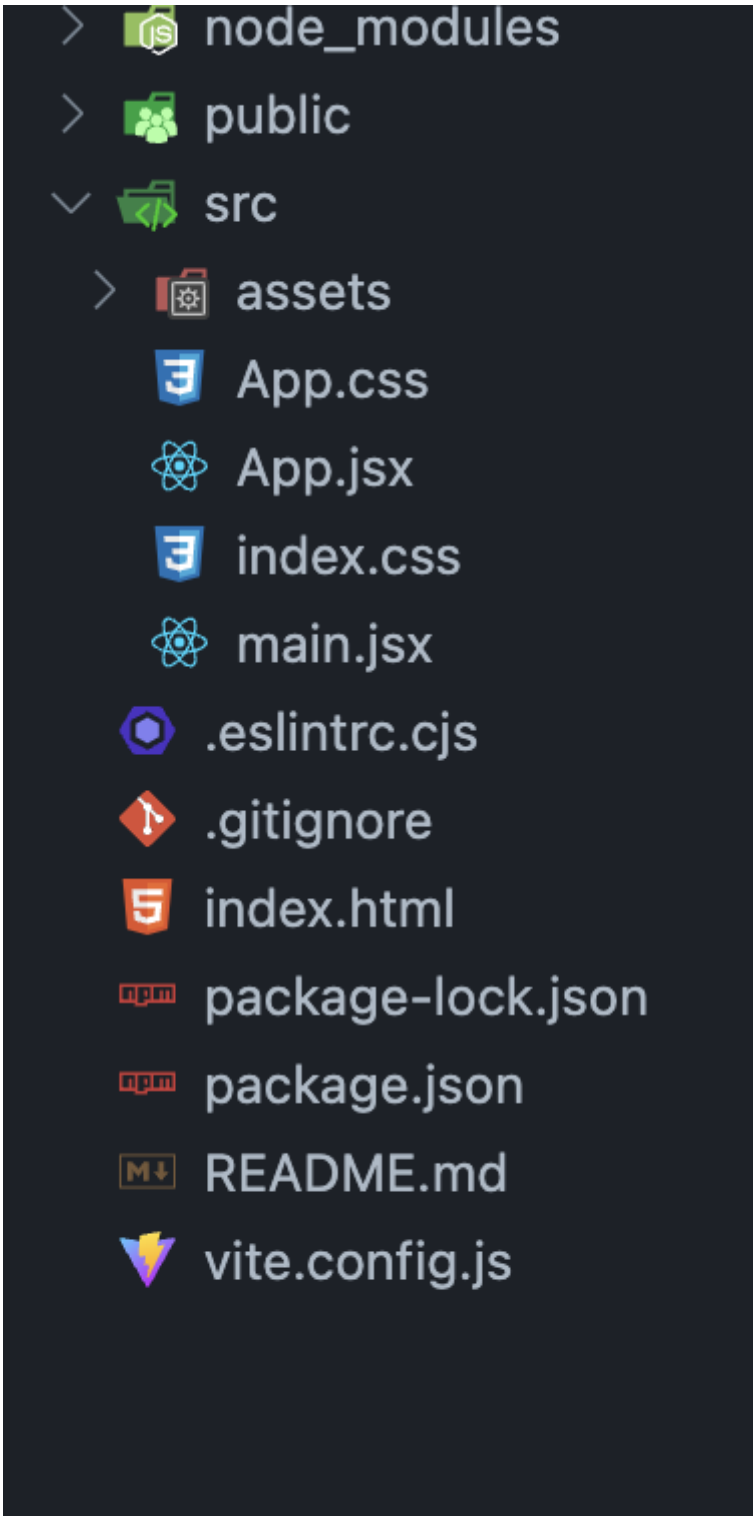
Por ejemplo:

```
import React from 'react';
import './styles.css'; // Importa el archivo de estilos

const MiComponente = () => {
  return (
    <div className="container">
      <h1 className="title>Mi componente</h1>
      <p>Mi primer componente con estilos</p>
    </div>
  );
  );
}
```

Archivo css:

```
/* styles.css */
.container {
  background-color: #f0f0f0;
  padding: 20px;
}

.title {
  font-size: 24px;
  color: #333;
}
```

Así como podemos crear hojas de estilo personalizadas para cada componente existe un archivo global en el cual podemos dar estilo a nuestro proyecto este archivo se llama index.css y se encuentra en la carpeta src.



¿Qué son los props?

Los "props" (abreviatura de "properties" o propiedades en español) son un mecanismo fundamental para pasar datos de un componente padre a un componente hijo. Los props permiten que los componentes interactúen y se comuniquen entre sí al pasar información desde el componente principal (padre) al componente secundario (hijo).

Estas propiedades se pasan como argumentos al componente y se pueden acceder desde su interior. Los props son inmutables, lo que significa que un componente hijo no puede modificar directamente los props que recibe de su componente padre.

Ejemplo de cómo se usan los props en React:

```
Componente hijo
function Saludo(props) {
  return <p>Hola, {props.nombre}!</p>;
}
Componente padre que utiliza el componente hijo
function App() {
  return (
    <div>
    <Saludo nombre="Juan" />
    <Saludo nombre="María" />
    </div>
  );
}

export default App;
```

Los props se pueden utilizar para transmitir cualquier tipo de dato, ya sean cadenas de texto, números, objetos o incluso funciones. Esto permite la composición y reutilización de componentes en una aplicación React, lo que es esencial para construir interfaces de usuario dinámicas y flexibles.

Desestructuración props

Antes de ir a la desestructuración de props debemos saber que significa este nuevo concepto. La desestructuración es una característica de JavaScript que permite extraer múltiples propiedades de un objeto o elementos de un array en una sola declaración. Esto es especialmente útil cuando se trabaja con objetos o arreglos que tienen muchas propiedades o elementos, ya que evita la necesidad de acceder a cada propiedad o elemento por separado.

En JavaScript, la desestructuración se utiliza mediante sintaxis de patrones que coinciden con la estructura de los objetos o arreglos que deseas desestructurar. Hay dos tipos principales de desestructuración:

Ej desestructuración de objetos:
La desestructuración de objetos permite extraer propiedades específicas de un objeto y asignarlas a variables con el mismo nombre. Aquí tienes un ejemplo:

Desestructuración de objetos:

```
const persona = { nombre: 'Juan', edad: 30, ciudad: 'Madrid' };
Desestructuración de objeto
const { nombre, edad } = persona;

salida "Juan"
console.log(nombre);
salida 30
console.log(edad);
```

Desestructuración de arreglos:
La desestructuración de arreglos permite extraer elementos de un arreglo y asignarlos a variables en el orden en que aparecen en el arreglo. Aquí tienes un ejemplo:

Desestructuración de arreglos:

```
const numeros = [1, 2, 3, 4, 5];
Desestructuración de arreglo
const [primerNumero, segundoNumero] = numeros;

salida 1
console.log(primerNumero);
salida 2
console.log(segundoNumero);
```

Desestructuración también puede tener parámetros por defecto:
La desestructuración también se puede combinar con valores por defecto y lo que la hace aún más poderosa y flexible.

Ej:

```
const persona = { nombre: 'María', ciudad: 'Barcelona' };
Desestructuración de objeto con valor por defecto:
const { nombre, edad = 25 } = persona;
```

```
salida "Maria"
console.log(nombre);
salida 25
console.log(edad);
```

La desestructuración es una herramienta útil para escribir código más limpio y legible, y es ampliamente utilizada en JavaScript moderno, especialmente al trabajar con React, para extraer propiedades de objetos que se pasan como props a componentes funcionales.

Ahora que ya sabemos que es desestructuración veamos como se aplican a los props en react mediante los siguientes ejemplos:

Desestructuración el objeto props:

```
const Saludo = (props) => {
  const { nombre, edad } = props;
  return (
    <div>
    <p>Hola {nombre}>/p>
    <p>Tu edad es {edad}>/p>
    </div>
  );
};
```

Ejemplo de Desestructuración en la recepción de argumentos:

```
const MiComponente = ({ nombre, edad, ciudad }) => {
  return (
    <div>
    <p>Hola {nombre}>/p>
    <p>Tu edad es {edad}>/p>
    <p>Ciudad: {ciudad}>/p>
    </div>
  );
};
```

Renderizado condicional

El render condicional en React se refiere a la práctica de renderizar componentes o elementos JSX basados en ciertas condiciones. Permite que tu interfaz de usuario se adapte y muestre diferentes elementos en función del estado o los datos disponibles en tu aplicación.

Hay varias formas de lograr el render condicional en React una de ellas es:

Operador Ternario:
El operador ternario es una forma concisa de realizar un render condicional y es una de las opciones mas utilizadas.

Ejemplo:

```
const MiComponente = ({ isLoggedIn }) => {
  return (
    <div>
      {isLoggedIn ? <p>Bienvenido>/p> : <p>Inicia sesión.>/p>}
    </div>
    );
  };
```

Usando React Bootstrap

React Bootstrap es una biblioteca de componentes que combina el poder de React, una biblioteca de JavaScript para construir interfaces de usuario, con los estilos y componentes de Bootstrap, un popular marco de diseño front-end. React Bootstrap proporciona una forma fácil y conveniente de crear interfaces de usuario atractivas y responsivas utilizando los estilos y componentes predefinidos de Bootstrap, todo integrado de manera fluida con la arquitectura de componentes de React.

Para instalar react bootstrap en nuestro proyecto ejecutamos el siguiente comando en nuestra terminal

npm install react-bootstrap bootstrap
Este comando instalará las dependencias necesarias para comenzar a usar bootstrap en nuestro proyecto

Para comenzar a usar react bootstrap en nuestro proyecto debemos importar los componentes que necesitamos en nuestro archivo App.js,por ejemplo:

```
import Button from 'react-bootstrap/Button';
import 'bootstrap/dist/css/bootstrap.min.css';
```

Para usar los componentes de react bootstrap en nuestro proyecto simplemente debemos usarlos como cualquier otro componente de react, por ejemplo:

```
import Button from 'react-bootstrap/Button';
import 'bootstrap/dist/css/bootstrap.min.css';

const App = () => {
  return (
    <div>
    <Button variant="primary">Primary</Button>
    </div>
  );
};
```

Reutilizando componentes

Una de las ventajas de usar componentes es que podemos reutilizarlos en cualquier parte de nuestra aplicación, por ejemplo, si tenemos un componente que muestra una card, podemos reutilizar ese componente en cualquier parte de nuestra aplicación donde necesitemos mostrar cards.

Para reutilizar un componente simplemente debemos crear una carpeta llamada components y comenzar a crear archivos con nuestros componentes reutilizables para posteriormente importarlos en cualquier parte de nuestra aplicación donde los necesitemos.

A continuación veamos un ejemplo reutilizando una card en distintas vistas

Card.js

```
import React from 'react';
import Card from 'react-bootstrap/Card';

const Card = () => {
  return (
    <Card style={{ width: '18rem' }}>
    <Card.Img variant="top" src="holder.js/100px180" />
    <Card.Body>
    <Card.Title>Card Title</Card.Title>
    <Card.Text>
    Some quick example text
    of the card's content.
    </Card.Text>
    <Button variant="primary">Go somewhere</Button>
    </Card.Body>
    </Card>
  );
};

export default Card;
```

Home.js

```
import React from 'react';
import Card from './components/Card';

const Home = () => {
  return (
    <div>
    <Card />
    </div>
  );
};

export default Home;
```

Profile.js

```
import React from 'react';
import Card from './components/Card';

const Profile = () => {
  return (
    <div>
    <Card />
    </div>
  );
};

export default Profile;
```

Pasando props a componentes reutilizables

Una de las ventajas de usar componentes es que podemos pasarles props para que estos componentes se adapten a nuestras necesidades, por ejemplo, si tenemos un componente que muestra una card, podemos pasarle props para que este componente muestre la información que nosotros queramos. A continuación veamos un ejemplo pasando props a un componente reutilizable.

Card.js

```
import React from 'react';
import Card from 'react-bootstrap/Card';

const Card = (props) => {
  return (
    <Card style={{ width: '18rem' }}>
    <Card.Img variant="top" src="holder.js/100px180" />
    <Card.Body>
    <Card.Title>{props.title}</Card.Title>
    <Card.Text>
    {props.text}
    </Card.Text>
    <Button variant="primary">Go somewhere</Button>
    </Card.Body>
    </Card>
```

```
  );
};
export default Card;
```

Home.js

```
import React from 'react';
import Card from './components/Card';

const Home = () => {
  return (
    <div>
    <Card title="Home" text="Home text" />
    </div>
  );
};

export default Home;
```

Profile.js

```
import React from 'react';
import Card from './components/Card';

const Profile = () => {
  return (
    <div>
    <Card title="Profile" text="Profile text" />
    </div>
  );
};

export default Profile;
```

Para ver la lista completa de componentes que nos ofrece react bootstrap podemos visitar la documentación oficial de react bootstrap en el siguiente enlace:

https://react-bootstrap.github.io/docs/components/accordion