¡Les damos la bienvenida!

¿Comenzamos?

Esta clase va a ser

- grabada

Clase 04. REACT JS

Componentes I

Temario

03

JSX y transpiling

- ✓ Sugar syntax
- ✓ POLYFILLS y la retrocompatibilidad
- ✓ BUNDLING CON WEBPACK
- ✓ TRANSPILING
- ✓ JSX

04

Componentes I

- ✓ Componentes I: Introducción
- ✓ Patrones

05

Componentes II

- ✓ Componentes II: Introducción
- ✓ Efectos

CODERHOUSE

Objetivos de la clase

- **Comprender** qué son los componentes y qué problemas resuelven.
- **Conocer** los tipos de componentes.
- **Implementar** los componentes vistos.

CLASE N°03

Glosario

Sugar Syntax: refiere a la sintaxis agregada a un lenguaje de programación con el objetivo de hacer más fácil y eficiente su utilización. Favorece su escritura, lectura y comprensión.

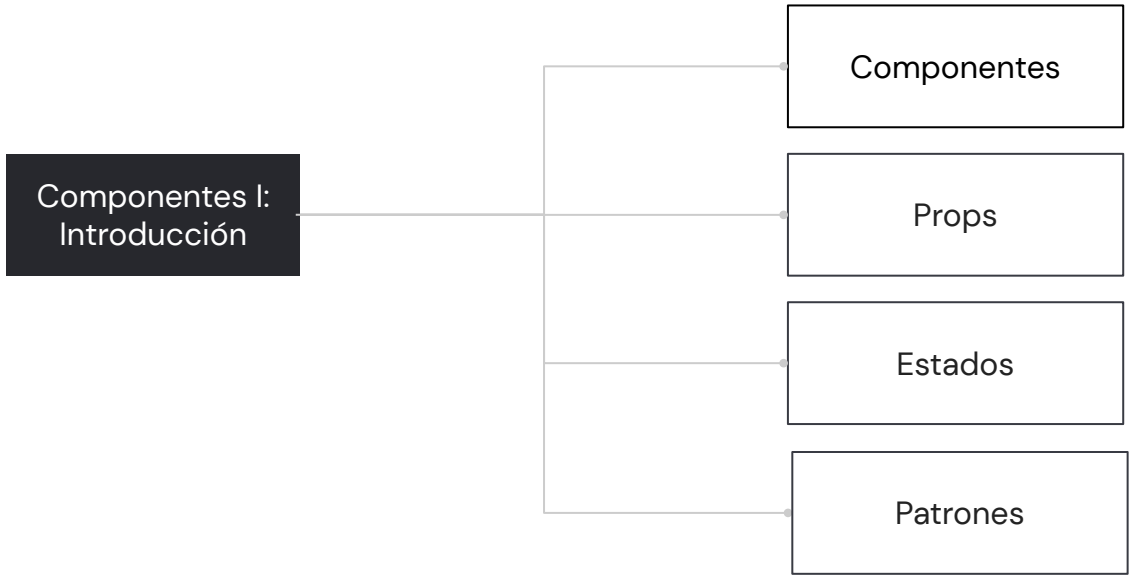
Webpack: es un module bundler o empaquetador de módulos.

Eject: es una acción permanente, que permite tener un control más específico del bundling, a costa de que de ahora en adelante tendremos que encargarnos de

Transpiling: es el proceso de convertir código escrito en un lenguaje, a su representación en otro lenguaje.

JSX: es una extensión de sintaxis de Javascript que se parece a HTML. Oficialmente, es una extensión que permite hacer llamadas a funciones y a construcción de objetos. No es ni una cadena de caracteres, ni HTML.

MAPA DE CONCEPTOS

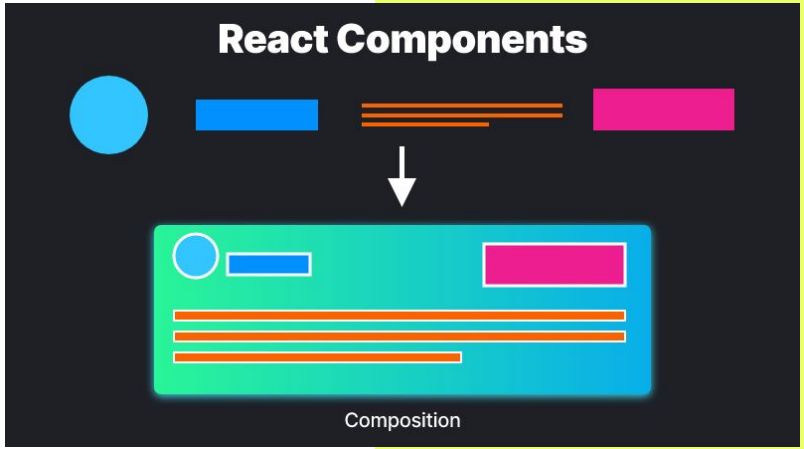


Componentes I:
Introducción

¿Qué es un componente?

Es un conjunto de elementos que cumplen una función específica en la interfaz de usuario.

Se utilizan para construir la jerarquía de todos esos elementos. Cada componente puede contener otros componentes como hijos.



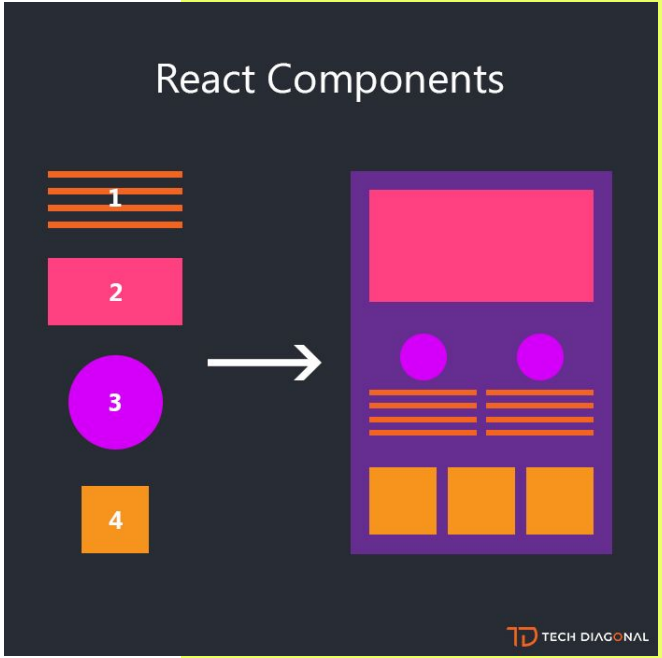
CODERHOUSE

Diseño modular

Los componentes permiten separar la interfaz de usuario en piezas **independientes, reutilizables** y pensar en cada pieza de **forma aislada**.

Al desarrollar crearemos componentes para resolver pequeños problemas, que son fáciles de visualizar y comprender.

Luego, unos componentes se apoyarán en otros para solucionar problemas mayores y al final la aplicación será un conjunto de componentes trabajando entre sí.



CODERHOUSE

Ventajas del enfoque

- ✓ Favorece la separación de responsabilidades: cada componente debe tener una única tarea.
- ✓ La aplicación es más fácil de entender.
- ✓ Mejora el rendimiento de la aplicación.
- ✓ Se simplifica la tarea de hacer pruebas unitarias.

Componentes I

CODERHOUSE

Funcionamiento

Podemos pensar a nuestra aplicación de React como una máquina, por ejemplo, un automóvil.

Así como un automóvil está compuesto por muchas piezas que trabajan en conjunto para generar un desplazamiento, una aplicación de React está compuesta por muchos componentes que trabajan juntos para crear una experiencia de usuario coherente y funcional.

CODERHOUSE

Componentes basados en funciones

Hoy en día todos los componentes de React se crean a partir de funciones, lo que hace que sea mucho más fácil de escribir el código y entenderlo.

Antes aquellos componentes que se encargaban del funcionamiento de la app, se creaban a partir de la clase Components para poder implementar las características de los componentes de React. Lo que hacía más complejo el modelo mental.

CODERHOUSE

Características principales de los componentes

- ✓ Pueden recibir propiedades (props).
- ✓ Tienen la capacidad de hacer render de un único elemento. Aunque este elemento puede tener muchos elementos dentro.
- ✓ Pueden tener estados.

Ejemplo en vivo

Cursos
Vamos al código

Propiedades/Props

Props en React

Las props en React se utilizan para pasar datos de un componente padre a un componente hijo, manteniendo el flujo unidireccional de los datos.

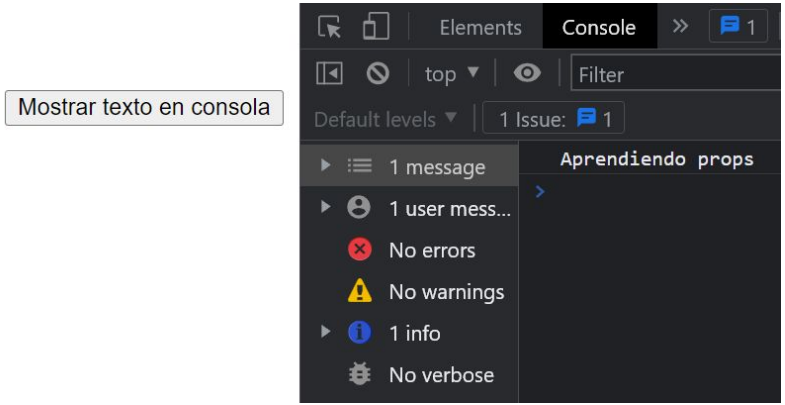
Las props son objetos que contienen datos específicos que un componente necesita para renderizarse correctamente.

Esto permite la reutilización de componentes en diferentes partes de la aplicación con diferentes datos.

Props en React

```jsx
const Button = ({ callback }) => {
  return <button onClick={callback}>Mostrar texto en consola</button>
}

const ParentComponent = ({ text }) => {

  const handleClick = () => {
    console.log(text)
  }

  return <Button callback={handleClick}/>
}


const App = () => {
  return (
    <div className="App">
      <ParentComponent text="Aprendiendo props"/>
    </div>
  );
}
```

Si alguna prop es una función (callback), el componente hijo puede llamarla para provocar efectos secundarios en el componente padre.



CODERHOUSE

Render de un único elemento

Los componentes deben retornar un solo elemento o componente, pero este elemento o componente sí puede tener más de un elemento o componente hijo.

```
const ParentComponent = () => {
  return (
    <div>
      <ChildComponent/>
      <ChildComponent />
    </div>
  )
}
```

En caso de no necesitar que el elemento padre forme parte del árbol del DOM, se puede utilizar React.Fragment, o lo que es lo mismo utilizar un tag sin tipo <></>

```
const ParentComponent = () => {
  return (
    <>
      <ChildComponent/>
      <ChildComponent />
    </>
  )
}
```

Ejemplo en vivo

Cursos
Vamos al código

ACTIVIDAD EN CLASE

Ta Te Ti

¡Para pensar!

- ¿Qué acciones del usuario impactan en la UI?
- ¿Qué variables necesito para representar la lógica?
- ¿Qué componentes necesito?

github

CODERHOUSE

Break

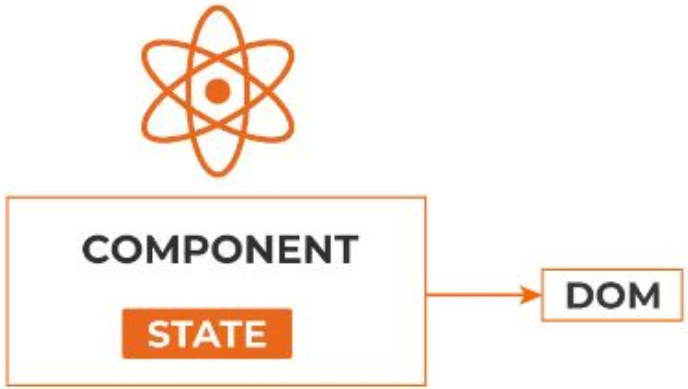
¡10 minutos y volvemos!

Estados y Hooks

Estados

El concepto más importante hoy en día para entender el funcionamiento de React es el de estados.

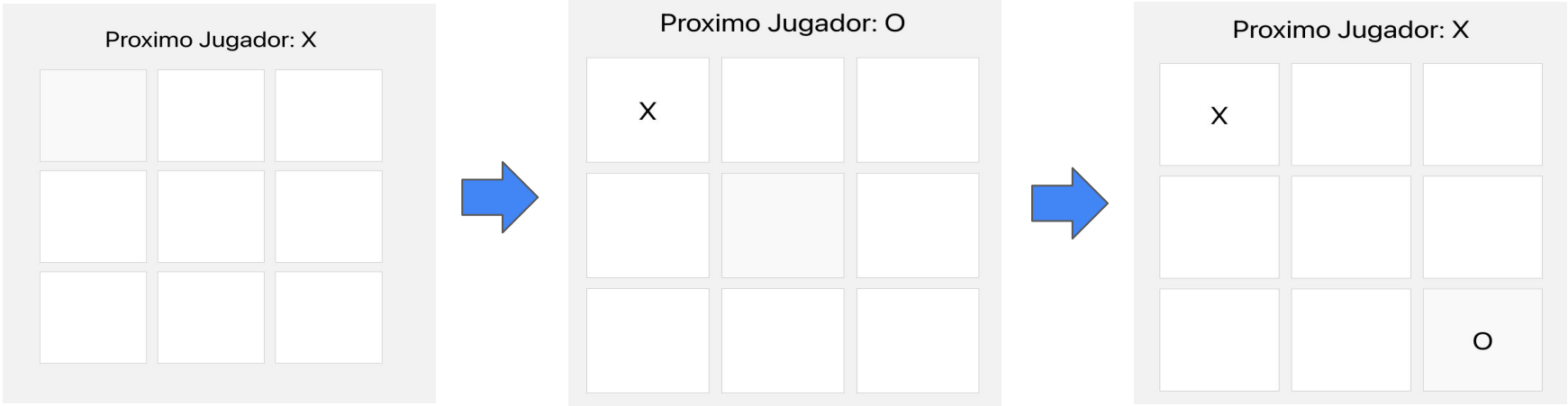
Los estados de un componente en React se utilizan para representar la información que puede cambiar durante la vida útil del componente y afectar su representación en la interfaz de usuario.



Estados

¿Qué acciones del usuario impactan en la UI?

¿Qué variables necesito para representar la lógica?

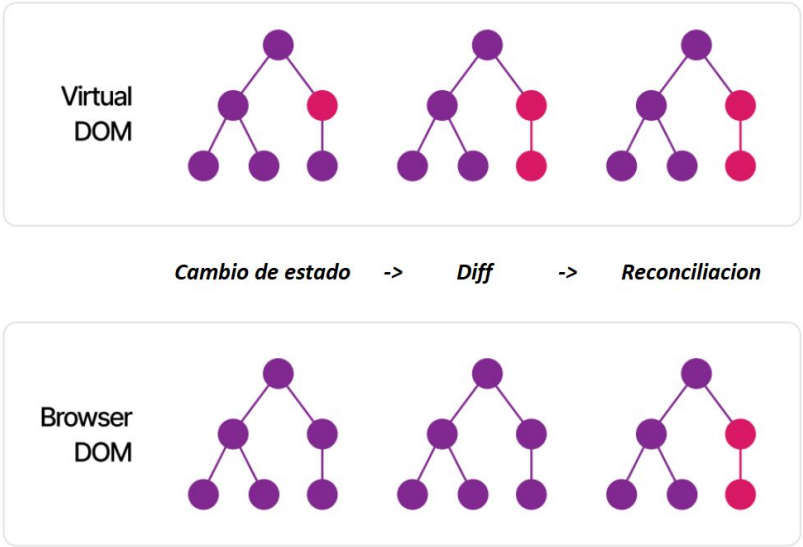


Estados

Todo cambio de estado va inicializar el proceso de render a partir del nodo donde se produjo el cambio de estado y así se generará el nuevo Virtual DOM que luego será reconciliado.

Los estados son un concepto de React, no de JavaScript, por lo que para implementarlo vamos a necesitar de los Hooks de React.

Pero... **¿qué son los Hooks?**

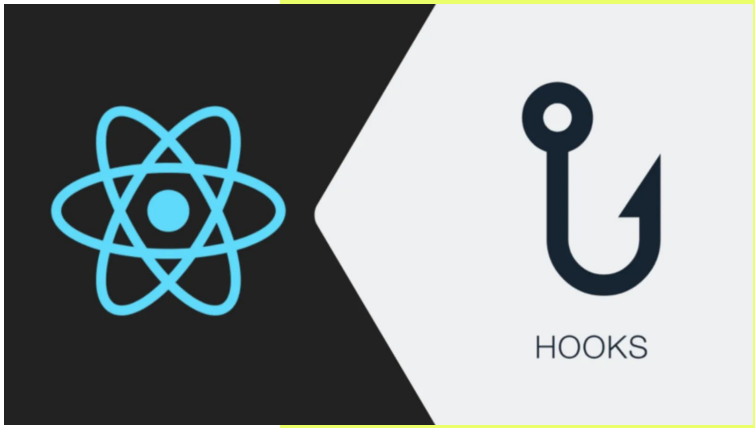


Hooks

Los hooks fueron introducidos en React 16.8 y permiten utilizar las características de React en los componentes funcionales, en lugar de tener que usar la clase Component.

Los hooks son funciones que se pueden usar dentro de los componentes funcionales para agregar funcionalidad adicional de React.

Son varias las funcionalidades, pero una de ellas son los estados, con el hook useState.



Características de los Hooks

- Comienzan con el prefijo “use” para que React pueda identificarlos y realizar validaciones.
- Solamente pueden utilizarse dentro de componentes funcionales.
- Deben ejecutarse siempre, es decir, en cada renderizado y el orden de ejecución debe ser siempre el mismo. No puede estar dentro de if, for, etc.
- Deben ejecutarse siempre en el cuerpo de la función del componente, por lo que normalmente se los ejecuta al inicio.

Hooks

Spoiler 😯

Cualquier función que se declare con el prefijo “use”, React la tratará como un hook realizando las validaciones correspondientes y llamará a otros hooks dentro de esta función sin ser esta un componente, ya que si este hook padre cumple las validaciones y, dentro de él, los hooks cumplen los requisitos correspondientes, pasarán todas las validaciones y podrá ejecutarse sin problemas.

A estos hooks declarados por el desarrollador para encapsular una lógica de componente específica se les llama **custom hooks**.

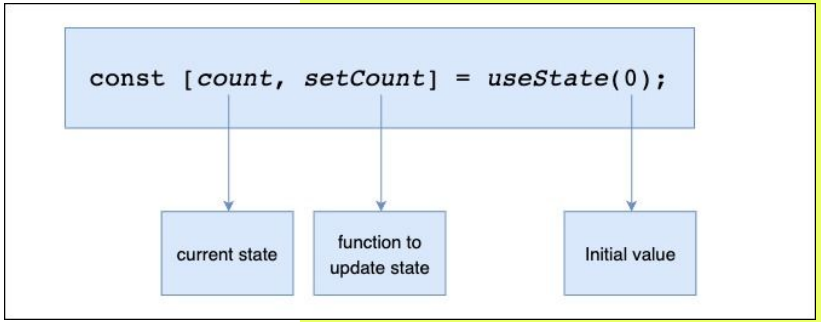
useState

useState es un hook de React que permite a los componentes funcionales tener estado.

La ejecución de la función Hook useState retorna un array con dos elementos:

- El primer elemento es el valor actual del estado.
- El segundo es una función que se utiliza para actualizar el estado, al llamar esta función con un nuevo valor pasado como argumento.

Es **importante** destacar que esta función no actualiza el valor del estado de manera síncrona, sino que React lo actualiza cuando inicia el próximo ciclo de renderizado.



CODERHOUSE

useState

Para inicializar el estado, se debe proporcionar un valor inicial como argumento a useState.

Como esta función retorna un array de dos elementos podemos desestructurarlo para darle un nombre más apropiado a nuestro estado y a su función de actualización.

El nombre debe ser representativo del contenido y la función debe ser el mismo nombre con el prefijo “set”.

```
const App = () => {
  const [buttonText, setButtonText] = useState('Haz clic aquí')

  return (
    <div>
      <button
        onClick={() => setButtonText('¡Gracias por hacer clic!')}
      >
        {buttonText}
      </button>
    </div>
  )
}
```

CODERHOUSE

Ejemplo en vivo

Ta Te Ti
Vamos al código

Patrones

Componentes de presentación

CODERHOUSE

Componentes de presentación

Son aquellos que simplemente se limitan a mostrar datos y tienen poca o nula lógica asociada a manipulación del estado (por eso, también son llamados stateless components).

Características:

- ✓ Orientados al aspecto visual.
- ✓ No tienen dependencia con fuentes de datos (por ejemplo: Llamadas a APIs, Redux).
- ✓ Normalmente no tienen estado.

Componentes de presentación

Los componentes de presentación **usualmente no tienen estado**.

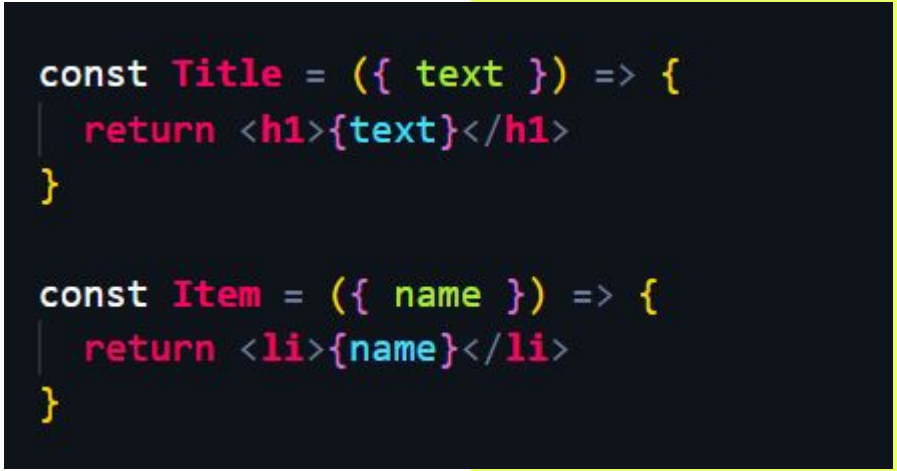
En caso de tenerlos serán para trabajar características visuales del componente pero no para almacenar datos de la aplicación, sino que reciben datos de aplicación por props.

```
const Titulo = ({nombre} = props) => (
    <h1>{ nombre }</h1>
);

const Item = (props) => (
<li><a href='#'>{ props.valor }</a></li>
);

const Input = (props) => (
    <input type='text' placeholder={ props.placeholder} />
);
```

CODERHOUSE

Componentes de presentación

La **ventaja** más evidente de estos componentes es la posibilidad de reutilizarlos siempre que queramos, sin tener que recurrir a escribir el mismo código una y otra vez.



CODERHOUSE

Componentes contenedores

CODERHOUSE

Componentes contenedores

Tienen como propósito encapsular a otros componentes y proporcionarles las propiedades que necesitan.

Se encargan de modificar el estado de la aplicación para que el usuario vea el cambio en los datos (por eso, también son llamados state components).

```
const Container = () => {
  const [country, setCountry] = useState('Argentina')

  const handleOnConfirm = (countryName) => {
    setCountry(countryName)
  }

  return (
    <div>
      <Form onConfirm={handleOnConfirm}/>
      <Title text={country}/>
    </div>
  )
}
```

Componentes contenedores

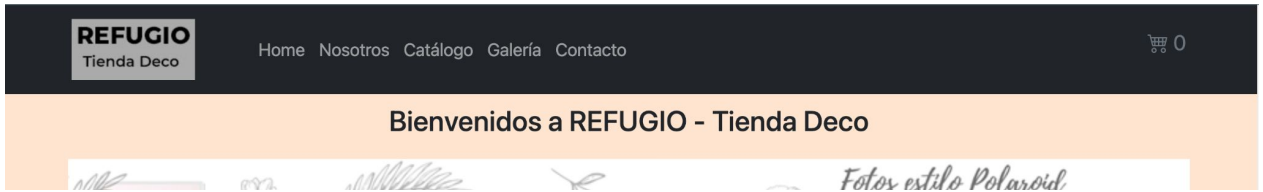
Características:

- ✓ Orientados al funcionamiento de la aplicación.
- ✓ Contienen componentes de presentación y/u otros contenedores.
- ✓ Se comunican con las fuentes de datos. (API, Flux, etc)
- ✓ Tienen estado para representar el cambio en los datos.

Primera pre-entrega de tu Proyecto final

CODERHOUSE

Primera pre-entrega



Ejemplo de proyecto final:

Refugio Tienda Deco:

https://refugio-tiendadeco-marceloluismoreno.netlify.app/

Primera pre- entrega

E-commerce

Formato: Link al último commit de tu repositorio en Github. Debe tener el nombre "PreEntrega1+Apellido"

Consigna.

- ✓ Crea una carpeta dentro de src llamada components que contenga la implementación del componente NavBar dentro del archivo NavBar.js. Su funcionalidad es la de renderizar una barra de menú (Navbar).

Objetivos.

- ✓ Crear el menú e-commerce de tu proyecto.

Primera pre-entrega

Se debe entregar.

- ✓ Brand (título/nombre de la tienda)
- ✓ Un listado de categorías clickeables
- ✓ Incorpora alguna librería de estilos con bootstrap/materialize u otro de tu preferencia (opcional).

Deberás corroborar que tu proyecto cuente con:

Componente Navbar.Js

Primera pre- entrega

Crea tu landing

Formato: Link al último commit de tu repositorio en Github. Debe tener el nombre "PreEntrega1+Apellido"

Consigna.

- ✓ Crea un componente CartWidget con un ícono y una notificación mostrando un número hardcodeado (fijo). Este servirá luego para indicar la cantidad de elementos que tenemos en el carrito, pero por ahora, mostrará un número hardcodeado (colocado en el código). Ubica este componente (CartWidget) dentro de Navbar.. Agrega algunos estilos con bootstrap/materialize u otro.
- ✓ Crea un componente contenedor ItemListContainer.js con una prop greeting, y muestra el mensaje dentro del contenedor con el styling integrado.

Objetivos.

- ✓ Crear la landing de tu proyecto.

Primera pre-entrega

Se debe entregar.

- ✓ Crea un componente CartWidget.js que haga rendering de un ícono Cart, e inclúyelo dentro de NavBar.js para que esté visible en todo momento.
- ✓ Crea un componente ItemListContainer. Impórtalo dentro de App.js, y abajo de NavBar.js.

Deberás corroborar que tu proyecto cuente con:

Componente Navbar.Js

¿Preguntas?

Resumen
de la clase hoy

- ✓ Introducción y tipos de componentes
- ✓ Children
- ✓ Props

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación