



FULL STACK

**Comenzamos en unos
minutos**

ACADEMY
by NUMEN

Componentes

¿Qué problemas resuelven los componentes?

⚠️ Cuando navegamos por una página tradicional, en realidad navegamos sobre varias páginas que son recargadas cada vez que hacemos click sobre las mismas. Por ejemplo, The Restaurant.

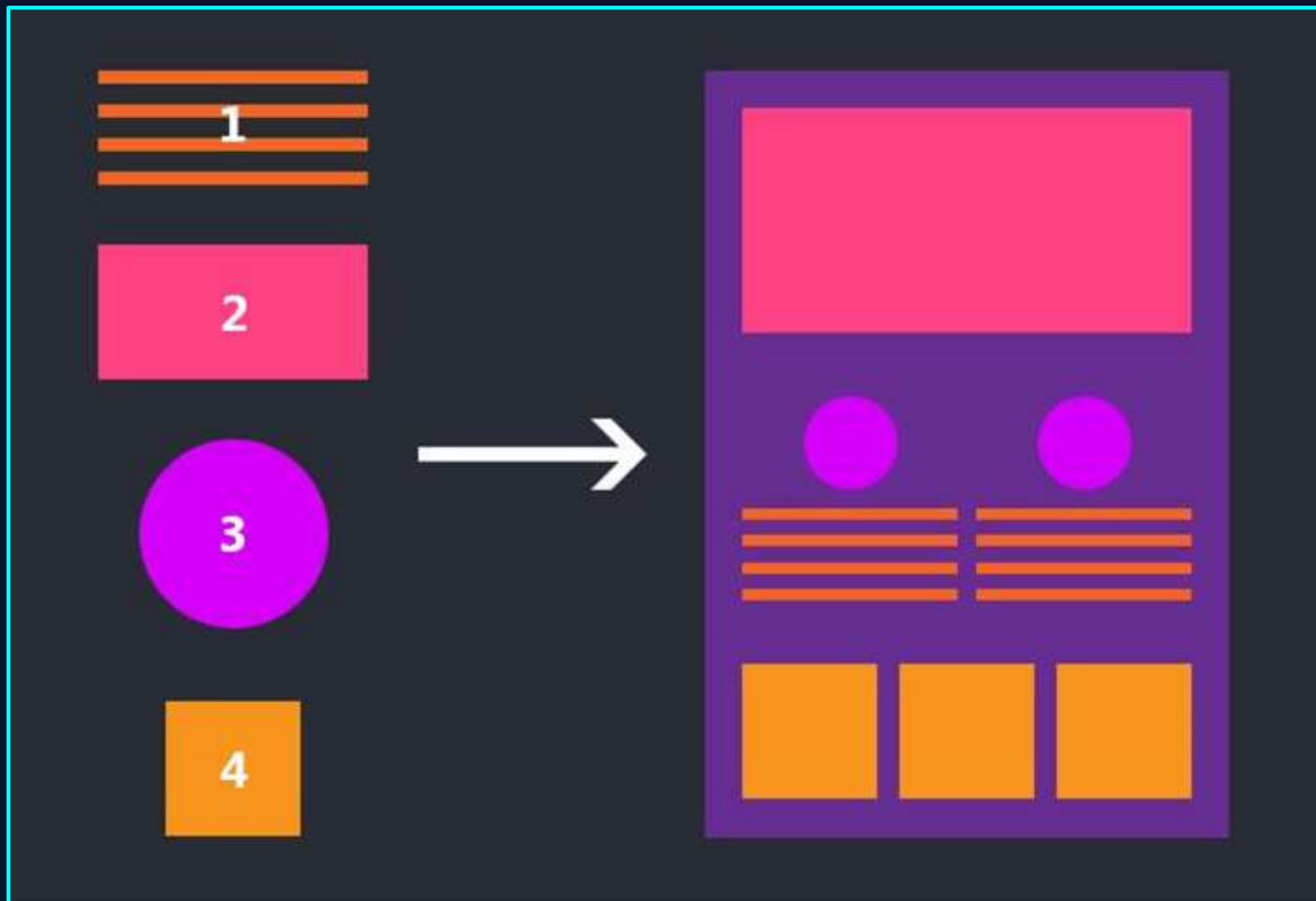
⚠️ No obstante, esto implica redundar código y quitar rendimiento y velocidad a nuestra aplicación.

⚠️ Además, al desarrollar aplicaciones muy grandes, el código se complejiza demasiado y se dificulta la lectura y escalabilidad.

¿Qué solución ofrece React?

✓ La solución que ofrece React es trabajar por medio de componentes. Estos son *piezas reutilizables de código* que tienen una funcionalidad propia y luego son ensamblados para conformar una sola aplicación con varias funcionalidades.







OPERATING MODELS

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)



DATA & ANALYTICS

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)



BUSINESS STRATEGY

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)

Componente
Caja con
Imagen



OPERATING MODELS

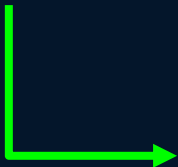
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)



Componente
Caja con
Título,
Descripción y
Botón



DATA & ANALYTICS

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)



BUSINESS STRATEGY

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Image from [Freepik](#)

[LEARN MORE](#)



Alex Grinfield

programming guru



Ann Richmond

creative leader



Jeffrey Brown

manager



Componente
Caja con
Imagen,
Título,
Descripción
y Links a
Redes
Sociales



Alex Grinfield
programming guru



Ann Richmond
creative leader



Jeffrey Brown
manager



Some facts about us

Sample text. Click to select the text box. Click again or double click to start editing the text. Image from [Freepik](#)

LEARN MORE



PROJECTS

100



CLIENTS

40



OUR TEAM

20



AWARDS

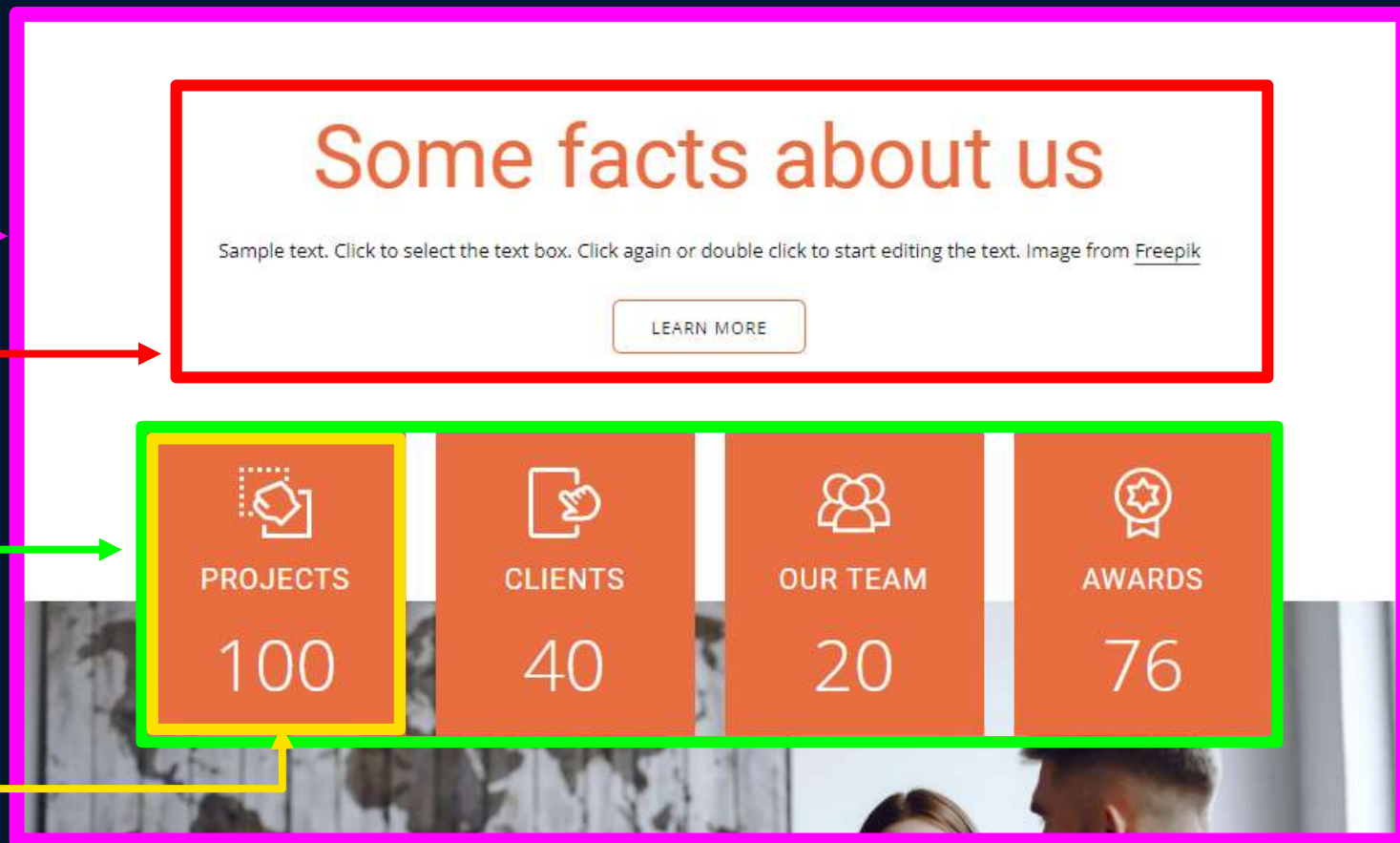
76

Componente
Contenedor
de Sección

Componente
Encabezado
de Sección

Componente
Contenedor
de Tarjetas

Componente
Tarjeta con
Ícono, Título
y Número



JOIN OUR NEWSLETTER

Contact us

3045 10 Sunrize Avenue, 123-456-7890

Mon – Fri: 9:00 am – 8:00 pm,

Sat – Sun: 9:00 am – 10 pm

contacts@esbnyc.com

Follow us



©2021 Privacy policy

JOIN US

Item 1

Enter your Name

Enter a valid email address

Enter your message

SUBMIT

Componente
Contenedor
de Sección

Componente
Contacto

Componente
Redes
Sociales

Componente
Formulario

JOIN OUR NEWSLETTER

Contact us

3045 10 Sunrize Avenue, 123-456-7890

Mon – Fri: 9:00 am – 8:00 pm,

Sat – Sun: 9:00 am – 10 pm

contacts@esbnyc.com

Follow us



©2021 Privacy policy

JOIN US

Item 1

Enter your Name

Enter a valid email address

Enter your message

SUBMIT

¿Cómo se construye un componente?

🔥 React nos da la posibilidad de hacer **componentes de clase** y **componentes funcionales**. A su vez, los componentes funcionales pueden ser declarados utilizando la sintaxis de ES6 y de versiones anteriores.

```
import { Component } from 'react';

export default class componente extends Component {
  render() {
    return (
      <>
        <p>Componente de Clase</p>
      </>
    );
  }
}
```

```
const componente = () => {
  return (
    <div>Componente Funcional</div>
  )
};

export default componente;
```


¿Herencia o Composición?

✗ POO frecuentemente utiliza la *herencia* para pasar métodos y propiedades de los padres a los hijos. Esto implica crear clases de objetos que funcionan como prototipo para crear otras instancias diferentes de dichos objetos, que heredan sus métodos y propiedades del objeto padre. En otras palabras, se parte de lo general, y luego se aborda lo particular.

✓□ Por otra parte, la programación funcional se vale de la *composición*, que implica el camino inverso: desde lo específico a lo general. Consiste en *abstraer* la lógica en pequeñas funciones que se enfocan en cumplir una tarea particular. Luego estas *componen* funciones más grandes, que a su vez *componen* otras, y así luego obtienes una aplicación compuesta de múltiples funciones.

¿Cómo ensamblar los componentes?

🔥 Para poder ensamblar los componentes es necesario usar la sintaxis de ES6 *import* y *export*. En primer lugar, hay que exportar el componente hijo para ponerlo a disponibilidad del resto de la aplicación.

```
export default Component
```

🔥 En segundo lugar, hay que importar el componente hijo dentro del componente padre. Finalmente, hay que utilizarlo dentro de JSX.

```
import Component from './ruta'
```

Reutilización de Componentes

🔥 No tengas miedo de dividir los componentes en otros más pequeños.

🔥 Extraer componentes puede parecer un trabajo pesado al principio, pero tener una paleta de componentes reutilizables vale la pena en aplicaciones más grandes. Una buena regla en general es que si una parte de tu interfaz de usuario se usa **varias veces** (Button, Panel, Avatar), o es lo suficientemente **compleja** por sí misma (App, FeedStory, Comment), es buen candidato para extraerse en un componente independiente.

Componente dentro de componente

🔥 Veamos un ejemplo de cómo insertar un componente dentro de otro.

```
const Tarjeta = () => {
  return (
    <figure style={{
      border: "solid 2px black",
      width: "200px",
      height: "275px",
      textAlign: "center"
    }}>
      <figcaption>
        <h1>Villa Langostura</h1>
        <p>Un hermoso lugar para
          pasar las vacaciones
        </p>
      </figcaption>
      {/* Aqui va el boton */}
    </figure>
  )
};

export default Tarjeta;
```



```
const Boton = () => {
  return <button style={{
    color: "white",
    backgroundColor: "darkgreen",
    padding: "10px 20px",
    borderRadius: "5px",
    fontSize: "1.5rem"
  }}>Clickeame</button>;
};

export default Boton;
```

Props

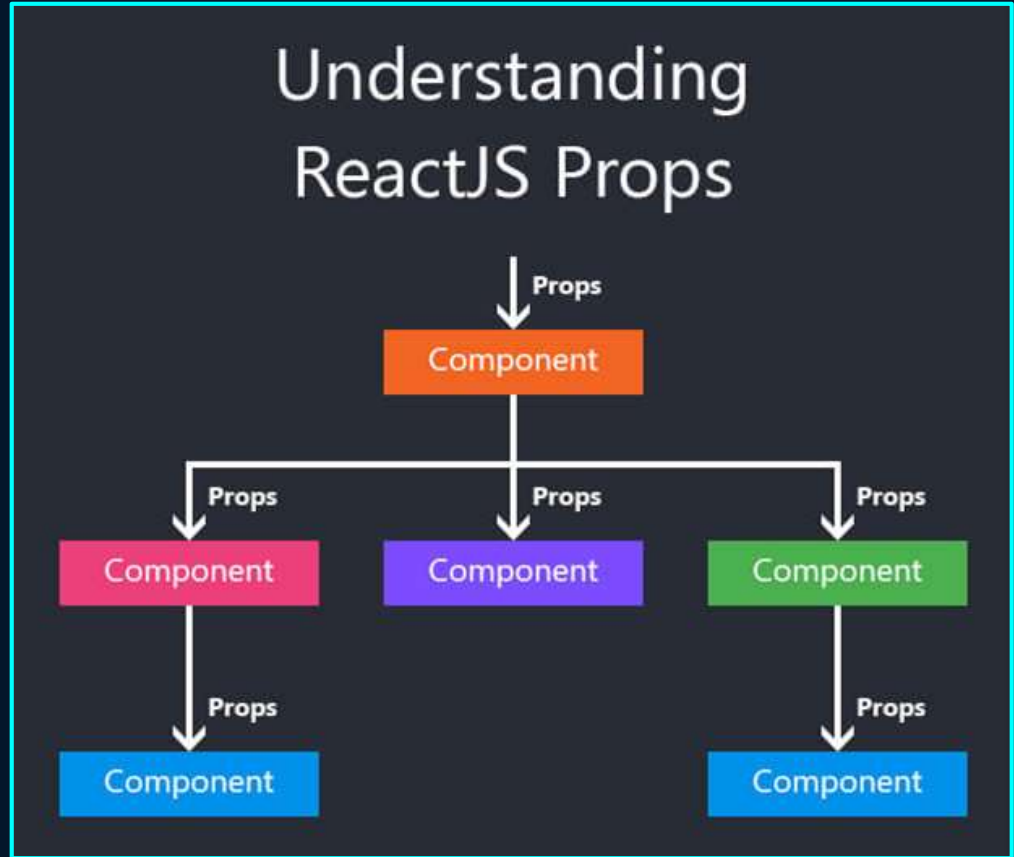
Componentes y Props

💡 Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada. Pero no solo esto...

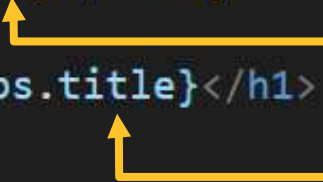
🔥 Parte del poder que tienen los componentes reside en las **props**. Conceptualmente, los componentes son como las funciones de JavaScript. En este sentido, aceptan entradas arbitrarias (“props”) y devuelven elementos que describen lo que debe aparecer en la pantalla.

☒ Las *props* son propiedades que permiten implementar el flujo de datos unidireccional.

¡ Por flujo de datos **unidireccional** nos referimos al hecho de que el valor de las props de los componentes hijos es almacenado en los componentes padres. Así, los datos descenden desde arriba hacia abajo, en *una sola dirección*.





```
const Title = (props) => {  
  return (  
    <h1>{props.title}</h1>  
  )  
}  
  
export default Title;
```



! Definimos las props como argumento y luego añadimos una propiedad a este objeto.

! Luego importamos el componente hijo en el componente padre, y le damos el valor que deseemos a sus props.



```
import Title from './components/Title'  
  
function App() {  
  return (  
    <>  
      <Title title="Título String"></Title>  
    </>  
  )  
}  
  
export default App
```

¿Qué datos podemos pasar como props?

```
function App() {  
  return (  
    <Propiedades  
      cadena="Hola, soy una cadena"  
      numero={33}  
      booleano={true ? "Verdadero" : "Falso"}  
      arreglo={[1,2,3,4]}  
      objeto={{  
        nombre: "Academia",  
        apellido: "Numen",  
      }}  
      funcion={() => 3 * 4}  
      elementoJSX=<p>Hola, soy un párrafo</p>  
      componenteReact=<Contact />  
    />  
  );  
}  
  
export default App;
```

```
const Propiedades = props => {  
  return (  
    <>  
      <ul>  
        <li>{props.cadena}</li>  
        <li>{props.numero}</li>  
        <li>{props.booleano}</li>  
        <li>{props.arreglo}</li>  
        <li>{props.objeto.nombre}</li>  
        <li>{props.funcion()}</li>  
        <li>{props.elementoJSX}</li>  
        <li>{props.componenteReact}</li>  
      </ul>  
    </>  
  );  
}  
  
export default Propiedades;
```

Mapeando una prop (parte 1)

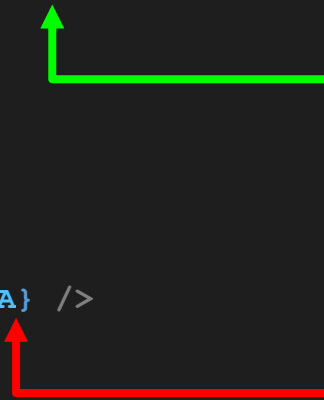
```
import Lista from "../components/Lista";

const LISTA = [
  {id: 1, titulo: 'Título uno'},
  {id: 2, titulo: 'Título dos'},
  {id: 3, titulo: 'Título tres'},
]

function App() {

  return (
    <Lista lista={LISTA} />
  )
}

export default App;
```

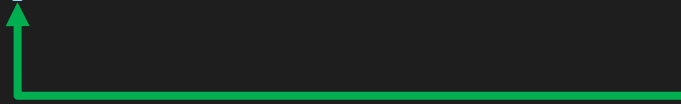
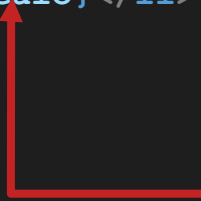
A green arrow originates from the 'LISTA' variable in the array definition and points to the 'lista={LISTA}' prop in the JSX element. A red arrow originates from the 'lista={LISTA}' prop and points to the 'LISTA' variable in the array definition, illustrating the mapping of the prop to the data source.

Primero, declaro los valores que voy a asignarle a las props del componente a reutilizar.

Luego los asigno a la prop de mi componente.

Mapeando una prop (parte 2)

```
const Lista = props => {  
  return (  
    <ul>  
      {props.lista.map(item =>  
        <li key={item.id}>{item.titulo}</li>  
      )}  
    </ul>  
  );  
};  
  
export default Lista;
```

A green arrow originates from the `props` parameter in the function signature `props =>` and points to `props.lista` inside the `map` function call.A red arrow originates from the `props` parameter in the function signature `props =>` and points to `props.lista` inside the `map` function call.

En primer lugar, creamos nuestro componente reutilizable y utilizamos *props* para que el contenido renderizado sea dinámico. Para ello, pasamos *props* como parámetro del componente funcional.

Luego, declaramos con *dot notation* una propiedad de dicho objeto, y accedemos a aquellas sub-propiedades que vayamos a utilizar.

Props.children

¡ Esta es una propiedad especial de los componentes hechos con React que permiten incluir diferentes cosas dentro de los tags de apertura y clausura al llamar y utilizar un component.

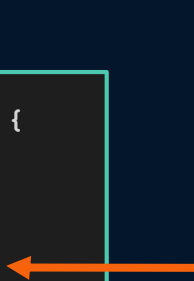
```
const string = "Contenido dinámico"

function App() {
  return (
    <>
      <Section> {string} </Section>
      <Section> Contenido Estático </Section>
    </>
  )
}

export default App;
```

```
const Section = props => {
  return (
    <div>
      {props.children}
    </div>
  )
};

export default Section;
```



En este ejemplo, le pasamos como *children* contenido dinámico declarado en Vanilla JS y contenido estático inmutable.

ACADEMY
by NUMEN