

Guía de ejercicios 1 - Introducción a React



¡Hola! Te damos la bienvenida a esta nueva guía de ejercicios.

¿En qué consiste esta guía?

En la siguiente guía podrás trabajar los siguientes aprendizajes:

- Crear una aplicación en React con componentes JSX sin props, agregando estilos y JavaScript a partir de un requerimiento.
- Personalizar los componentes a través de un mecanismo llamado props.
- Reutilizar componentes usando props como fuente de datos.
- Importar componentes de terceros en una aplicación de React.

¡Vamos con todo!



Tabla de contenidos

Versión de React	3
Setup necesario	3
Profundizando en JSX	4
Actividad 1: conociendo JSX	4
Agregando expresiones de JavaScript con JSX	9
Agregando CSS en JSX con clases	11
Recomendaciones importantes:	16
Personalizando componentes con props	16
Actividad 2: Trabajando con props	16
Pasando múltiples props	18
Destructuring de las propiedades (props)	21
Props con valores por defecto	22
Render Condicional	24
Operador ternario	24
Utilizando el operador ternario en JSX	24
Entendamos el código	26
Instalando componentes en React	27
Actividad 3: agregando componentes de terceros en aplicaciones React	27
Actividad 4: Agregando un componente dentro de otro componente	31



¡Comencemos!

Versión de React

En React hay muchas versiones, pero en esta guía trabajaremos con la versión 18, por lo que es importante revisar tu instalación de acuerdo al setup indicado.

Setup necesario

Para esta guía necesitas:

1. Tener instalado y actualizado npm.
2. Instalar [Vite](#), lo cual podemos hacer al momento de crear nuestro primer proyecto.

Vite es una herramienta de construcción de aplicaciones web que ayudará a mejorar el rendimiento de proyectos basados en JavaScript, sobre todo aquellos que utilizan frameworks o librerías como React.



Imagen 01. Logo de Vite.

Fuente: [Vite](#).

Otra herramienta que nos sirve para crear proyectos en react es usando **create-react-app**, sin embargo, utilizaremos **Vite**, ya que este acelera el proceso de desarrollo haciendo que la aplicación sea más rápida y eficiente, además de brindar más flexibilidad y libertad para personalizar nuestros proyectos.



Si necesitas repasar cómo crear desde 0 una aplicación en React y los conceptos básicos de creación de proyecto, archivos y componentes, puedes visualizar el **Material complementario de React** disponible en la plataforma; este recurso usa **create-react-app** para la creación de proyectos, y servirá para conocer sus principales diferencias.

Profundizando en JSX

Para aprender más sobre componentes tenemos que profundizar en JSX.



"JSX es una extensión de la sintaxis de JavaScript. Recomendamos usarlo con React para describir cómo debería ser la interfaz de usuario. JSX puede recordarte a un lenguaje de plantillas, pero viene con todo el poder de JavaScript." (Presentando JSX –, s. f.)



Actividad 1: conociendo JSX

Para comprender cómo se comporta JSX, crearemos un nuevo proyecto llamado `trabajando-con-jsx`.

- **Paso 1:** Creamos nuestro primer proyecto usando Vite y desde la terminal ejecutamos el siguiente comando:

```
npm create vite
```

- **Paso 2:** Si no tenemos instalado Vite, nos aparecerá un mensaje indicando que necesitamos instalarlo, presionamos en la opción entregada (y) para proceder con su instalación:

```
desafio@desafio:~/Escritorio/React$ npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) █
```

Imagen 02. Instalación de Vite.
Fuente: Desafío Latam.

- **Paso 3:** Ahora nos solicitará un nombre para el proyecto, en nuestro caso lo llamaremos **trabajando-con-jsx** y presionamos Enter.

```
desafio@desafio:~/Escritorio/React$ npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
? Project name: > vite-project
```

Imagen 03. Nombre del proyecto.

Fuente: Desafío Latam.

- **Paso 4:** Luego de escribir el nombre del proyecto, Vite nos da la opción de seleccionar un Framework para comenzar a trabajar, donde usando las teclas direccionales bajaremos hasta seleccionar React:

```
desafio@desafio:~/Escritorio/React$ npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... trabajando-con-jsx
? Select a framework: > - Use arrow-keys. Return to submit.
  Vanilla
  Vue
  > React
  Preact
  Lit
  Svelte
  Others
```

Imagen 04. Seleccionar un framework.

Fuente: Desafío Latam.

- **Paso 5:** Seleccionado el framework, debemos seleccionar una “variante” para nuestro proyecto; para trabajar con la sintaxis de JSX seleccionamos JavaScript:

```
desafio@desafio:~/Escritorio/React$ npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... trabajando-con-jsx
✓ Select a framework: > React
? Select a variant: > - Use arrow-keys. Return to submit.
  TypeScript
  TypeScript + SWC
  > JavaScript
  JavaScript + SWC
```

Imagen 05. Seleccionar variante.

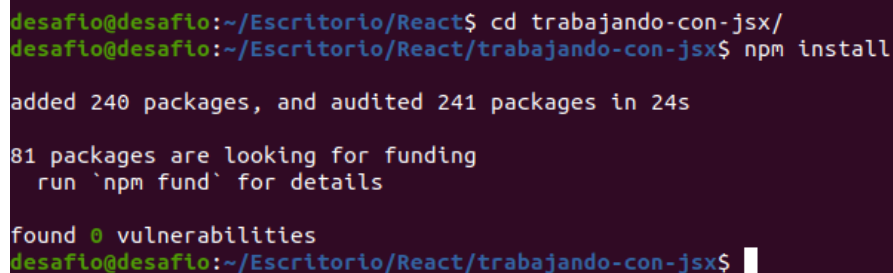
Fuente: Desafío Latam.

- **Paso 6:** Con estos pasos creamos nuestro proyecto en React. Desde la misma terminal accedemos a la carpeta raíz del proyecto usando el siguiente comando:

```
cd trabajando-con-jsx
```

- **Paso 7:** Ya ubicados dentro de la carpeta, debemos instalar las dependencias necesarias para ejecutar el proyecto, esto lo realizaremos usando este comando:

```
npm install
```



```
desafio@desafio:~/Escritorio/React$ cd trabajando-con-jsx/  
desafio@desafio:~/Escritorio/React/trabajando-con-jsx$ npm install  
  
added 240 packages, and audited 241 packages in 24s  
  
81 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
desafio@desafio:~/Escritorio/React/trabajando-con-jsx$
```

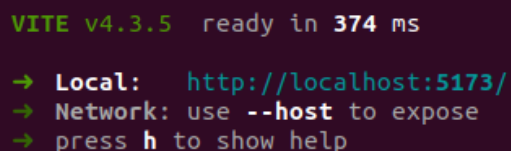
Imagen 06. Instalación de dependencias.

Fuente: Desafío Latam.

- **Paso 8:** Antes de ejecutar `npm run dev` para levantar el servidor, abrimos nuestro editor de código escribiendo en la terminal:

```
code .
```

- **Paso 9:** Solo nos queda levantar el servidor de nuestro proyecto y acceder a la url que nos muestra la terminal:



```
VITE v4.3.5 ready in 374 ms  
  
→ Local:   http://localhost:5173/  
→ Network: use --host to expose  
→ press h to show help
```

Imagen 07. `npm run dev`.

Fuente: Desafío Latam.

- **Paso 10:** Entramos a la url que muestra la terminal, y veremos los logos de Vite y React, esto nos indica que el proyecto está ejecutándose correctamente:



Imagen 08. Proyecto Vite + React.
Fuente: Desafío Latam.

- **Paso 11:** Luego, en nuestro editor de código, crearemos el componente `Producto`. Para esta ocasión y siguiendo las recomendaciones de los desarrolladores de React, crearemos una carpeta llamada `components` dentro de nuestra carpeta `src`, y a su vez, en ella ubicamos el componente `Producto` mencionado anteriormente.

```
// src/components/Producto.jsx
const Producto = () => {
  return <h1> Producto </h1>;
};

export default Producto;
```

- **Paso 12:** Configuramos el componente `App.jsx` y borramos todo el código dentro de las etiquetas `<>` para poder agregar nuestro código e importar el componente. Observaremos que la importación ahora se realiza primero accediendo a la carpeta `components` y dentro de ella llamamos al componente `Producto`.



Nota: Esta configuración y estructura la utilizaremos a lo largo del curso para poder tener ordenado nuestro árbol de directorios.

```
// src/App.jsx
import Producto from './components/Producto';

function App() {
  return (
    <div>
      <Producto />
    </div>
  );
}

export default App;
```

- **Paso 13:** Verificamos en el navegador que el componente producto se esté mostrando correctamente.

Producto

Imagen 09. Mostrando componente Producto
Fuente: Desafío Latam



Hasta este punto ya tenemos la base para iniciar la incorporación de lógica de JavaScript en nuestro componente Producto. Recuerda repetir los pasos mostrados anteriormente, una vez consolidado los conocimientos sigue avanzando.

Agregando expresiones de JavaScript con JSX

Recordemos que JSX significa JavaScript XML y es una extensión de Javascript que nos permite agregar nodos de HTML de forma sencilla. En React usualmente lo vemos dentro del `return()` de nuestros componentes, donde para incorporar JavaScript debemos hacerlo antes del `return`.

Veamos esto utilizando como base el mismo proyecto de `trabajando-con-jsx`

- **Paso 1:** Partamos agregando un JavaScript sencillo a un componente `console.log()`

```
// src/components/Producto.jsx
const Producto = () => {
  console.log("Creando un producto");
  return <p> Producto </p>;
};

export default Producto;
```

Comprobemos el resultado del `console.log()` en nuestro navegador, abriendo el inspector de elementos; esto nos debería mostrar en la consola el mensaje "Creando un producto".

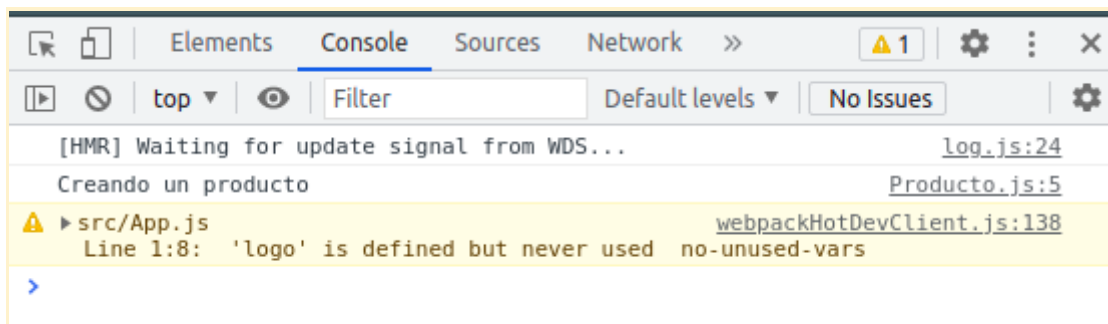


Imagen 10. Imprimiendo información en consola
Fuente: Desafío Latam

También podemos agregar JavaScript dentro del `return` o de los nodos creados utilizando llaves `{expresionDeJavascript}`. Veamos esto con un ejemplo

- **Paso 2:** Agregando expresiones de JavaScript en nuestro JSX.

```
// components/Producto.jsx

const Producto = () => {
  const productName = "Producto 1"
  return <h1> {productName} </h1>;
};

export default Producto;
```

Dentro del código vemos que se define una constante `productName` y se le asigna un valor. Luego, dentro del `return` se incorpora una etiqueta de párrafo que a través de las llaves `{}` le pasamos nuestra constante para mostrar el valor de "Producto 1".

Veamos en nuestro navegador si el comportamiento es correcto.

Producto 1

Imagen 11. Expresiones de JavaScript en componentes
Fuente: Desafío Latam

- **Paso 3:** Además de pasar valores como se mostró anteriormente, también podemos realizar operaciones matemáticas dentro de las llaves `{}`, Veamos esto en el siguiente código.

```
// src/components/Producto.jsx
import React from 'react'

const Producto = () => {
  const productName = "Producto 1"
  return (
    <>
      <h1> {productName} </h1>
      <p>Operaciones matemática Suma: {2 + 2}</p>
      <p>Operaciones matemática Resta: {2 - 2}</p>
      <p>Operaciones matemática Multiplicación: {2 * 2}</p>
      <p>Operaciones matemática División: {2 / 2}</p>
    </>
  )
}
```

```
    <p>Operaciones matemática Módulo: { 2 % 2 }</p>
  </>
);
};
export default Producto;
```

- **Paso 4:** Comprobemos el estado en el navegador de las operaciones matemáticas de ejemplo que agregamos.

Producto 1

Operaciones matemática Suma: 4

Operaciones matemática Resta: 0

Operaciones matemática Multiplicación: 4

Operaciones matemática División: 1

Operaciones matemática Módulo: 0

Imagen 12. Operaciones matemáticas
Fuente: Desafío Latam

Como pudimos observar en los pasos anteriores con JSX, podemos utilizar expresiones matemáticas de nuestros componentes, de la misma forma podemos usar cualquier otro tipo de expresión de JavaScript, por ejemplo, transformando minúsculas a mayúsculas o concatenando.

Agregando CSS en JSX con clases

Adicionalmente, en JSX podemos agregar CSS, lo que podemos hacer a través de clases o de la etiqueta style.

Veamos esto usando como base la misma aplicación `trabajando-con-jsx`.

- **Paso 5:** Agreguemos CSS a través de clases con el atributo `className`

```
// src/components/Producto.jsx
const Producto = () => {
  const productName = "Producto 1"
  return (
    <div>
      <h1 className="producto"> {productName} </h1>
    </div>
  )
}
```

```
    <p>Operaciones matemática Suma: {2 + 2}</p>
    <p>Operaciones matemática Resta: {2 - 2}</p>
    <p>Operaciones matemática Multiplicación: {2 * 2}</p>
    <p>Operaciones matemática División: {2 / 2}</p>
    <p>Operaciones matemática Módulo: { 2 % 2 }</p>
  </div>
);
};

export default Producto;
```



Recordemos que para pasarle clases a nuestras etiquetas HTML, el atributo está definido como `class`, pero bajo ese nombre es una palabra reservada de JavaScript, por lo que si dejamos el atributo como `class` nos levantará un error. Veamos en la consola del navegador el error:

```
✖ Warning: Invalid DOM property `class`. Did you mean `className`? index.js:1
  at h1
  at div
  at Producto
  at div
  at App
```

Imagen 13. Error propiedad class

Fuente: Desafío Latam

Para solucionar este inconveniente, React implementó `className` para asignarle clases a nuestras etiquetas HTML definidas en JSX.

- **Paso 6:** Ahora que nuestro producto tiene una clase vía `className`, agreguemos CSS dentro del archivo `index.css`.



Nota: Cuando creamos un proyecto usando Vite, este nos crea dos archivos CSS (`App.css` - `index.css`).

La diferencia entre estos archivos radica en su alcance y propósito, el archivo `index.css` se utiliza para agregar estilos globales a la aplicación, mientras que `App.css` se utiliza para agregar estilos específicos para el componente `App`.

En este caso, nuestros estilos los agregaremos al final dentro del archivo de estilos **index.css**.

```
/* src/index.css */  
  
.producto{  
  font-size: 32px;  
  color: red;  
}
```

- **Paso 7:** Comprobemos los cambios en el navegador:

Producto 1

Operaciones matemática Suma: 4

Operaciones matemática Resta: 0

Operaciones matemática Multiplicación: 4

Operaciones matemática División: 1

Operaciones matemática Módulo: 0

Imagen 14. Mostrando estilos modificados
Fuente: Desafío Latam



¡Excelente! Ya conoces una de las formas de agregar estilos a nuestros elementos HTML en React. Ahora veremos otra de las formas, pero antes de aprenderlo, recordemos algunos aspectos técnicos fundamentales.

Comencemos con el concepto de objetos literales en JavaScript, para ello, revisemos el siguiente código de ejemplo extraído del sitio mozilla developer.

```
var nombreObjeto = {  
  miembro1Nombre: miembro1Valor,  
  miembro2Nombre: miembro2Valor,  
  miembro3Nombre: miembro3Valor  
}
```



Según, MDN Web Docs *"Un objeto como este se conoce como un objeto literal – literalmente hemos escrito el contenido del objeto tal como lo fuimos creando.* (Conceptos básicos de los objetos JavaScript - Aprende sobre desarrollo web | MDN, 2021).



Nota: aquí puedes ir a la página para verlo mejor: [Conceptos básicos de los objetos JavaScript - Aprende sobre desarrollo web | MDN](#)

- **Paso 8:** Agreguemos ahora estilos en línea con el atributo style dentro del primer párrafo.

```
// src/components/Producto.jsx
const Producto = () => {
  const productName = "Producto 1"
  return (
    <div>
      <h1 className="producto"> {productName} </h1>
      <p style={{fontSize:'12px', color: 'blue'}}>Operaciones
matemática Suma: {2 + 2}</p>
      <p>Operaciones matemática Resta: {2 - 2}</p>
      <p>Operaciones matemática Multiplicación: {2 * 2}</p>
      <p>Operaciones matemática División: {2 / 2}</p>
      <p>Operaciones matemática Módulo: { 2 % 2 }</p>
    </div>
  );
};

export default Producto;
```

Aquí vemos que el estilo asignado a las operaciones de tipo Suma reciben estilo a través de un objeto literal.

- **Paso 9:** Otra forma de implementar los estilos en JavaScript con el atributo style, es pasando los valores que queremos incorporar, pero almacenándolos en una constante o variable, para posteriormente pasarlos a nuestra etiqueta HTML con la sintaxis de expresiones. Veamos un ejemplo a continuación:

```
// src/components/Producto.jsx
const Producto = () => {
  const productName = "Producto 1"
```

```
const myStyle = {fontSize:'12px', color: 'brown'}
return (
  <div>
    <h1 className="producto"> {productName} </h1>
    <p style={{fontSize:'12px', color: 'blue'}}>
      Operaciones matemática Suma: {2 + 2}
    </p>
    <p style={ myStyle }>
      Operaciones matemática Resta: {2 - 2}
    </p>
    <p>Operaciones matemática Multiplicación: {2 * 2}</p>
    <p>Operaciones matemática División: {2 / 2}</p>
    <p>Operaciones matemática Módulo: { 2 % 2 }</p>
  </div>
);
};

export default Producto;
```

- **Paso 15:** Observemos que los estilos incorporados hayan sido aplicados correctamente en el navegador.

Producto 1

Operaciones matemática Suma: 4

Operaciones matemática Resta: 0

Operaciones matemática Multiplicación: 4

Operaciones matemática División: 1

Operaciones matemática Módulo: 0

Imagen 14. Estilos CSS múltiples
Fuente: Desafío Latam



Recuerda: Para ejecutar expresiones de JavaScript dentro de JSX debemos envolver en llaves las expresiones {expresión de javascript}.



Nota: En plataforma podrás acceder al proyecto explicado en este material de estudio con el nombre de **Repositorio Ejercicio - Trabajando con JSX**.



Recomendaciones importantes:

1. Antes de continuar, intenta anotar todos los pasos para ir recordando lo aprendido.
2. Con estos apuntes, comienza un proyecto nuevo desde cero, e intenta realizar los pasos anteriores sin mirar la actividad. Si te faltó algún paso, no dudes en volver a consultar la guía.



Puedes profundizar en JSX revisando la documentación oficial de React. [Presentando JSX – React](#)

Personalizando componentes con props

A continuación aprenderemos a crear componentes de React que sean más flexibles, por ejemplo, creando productos con distintos nombres o con distintos precios, lo que podemos lograr a través de **props**.

Los **props** (abreviación de propiedades) son valores que puede recibir un componente, y nos sirven para personalizarlos y utilizarlos para distintas situaciones. Una vez que aprendamos a crearlos, podremos desarrollar productos con características de la siguiente forma:

```
<Producto title="Mi primer producto"/>
```

Aquí, **title** es simplemente un nombre que nosotros especificamos, pudo haber sido título, precio o cualquier otro.



Actividad 2: Trabajando con props

Para comprender el uso de props trabajaremos sobre el proyecto `trabajando-con-jsx`. Una vez dentro del proyecto seguiremos los siguientes pasos:

- **Paso 1:** Modificaremos el componente `Producto` agregando props, de este modo y como se muestra en el siguiente ejemplo de código, le estamos diciendo a React que el componente recibirá algunas propiedades.
Para concentrarnos en los props, borraremos el código agregado anteriormente en el componente de producto.

```
// src/components/Producto.jsx  
const Producto = (props) => {
```



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



Nota: Esto es muy similar a cuando una función recibe argumentos y los utilizamos dentro de ella. Los props son los argumentos que reciben un componente, los que a su vez se envían a un objeto que por convención llamaremos props.

Una vez configurado nuestro componente `Producto`, debemos ir a nuestro componente padre `App.jsx` e indicar las propiedades que serán enviadas al componente `Producto`.

- *Paso 2:* Pasamos la propiedad `title` desde el `App.jsx` a nuestro componente `Producto`. Para ello copiaremos 3 veces el componente `Producto` y le modificaremos el valor de la propiedad `title`:

```
/* App.jsx */  
import Producto from "../Producto";  
  
function App() {  
  return (  
    <>  
      <Producto title="Producto1" />  
      <Producto title="Producto2" />  
      <Producto title="Producto3" />  
    </>  
  );  
}  
  
export default App;
```

Veamos el resultado de esta configuración en el navegador web a continuación:

Producto 1

Producto 2

Producto 3

Imagen 16. Propiedades en componentes
Fuente: Desafío Latam

Pasando múltiples props

Como observamos en el ejemplo anterior, podemos pasar información del componente padre `App.jsx` a nuestro componente hijo `Producto` utilizando los props, para lo cual definimos únicamente la propiedad nombre.

Ahora, supongamos que al ser un producto queremos pasarle un precio, **¿cómo lo harías?** Veamos un ejemplo a continuación:

- **Paso 3:** Agregamos la propiedad precio a nuestro componente `Producto`.

```
// src/components/Producto.jsx
const Producto = (props) => {
  return (
    <>
      <h1>{props.title}</h1>
      <p>Precio: { props.price }</p>
    </>
  );
};

export default Producto;
```

- **Paso 4:** Modificamos nuestro componente padre `App.jsx` e incorporamos la propiedad que se desea pasar a `Producto`.

```
/* App.jsx */
import Producto from "../Producto";

function App() {
```

```
return (  
  <>  
    <Producto title="Producto1" price="400" />  
    <Producto title="Producto2" price="700" />  
    <Producto title="Producto3" price="1000" />  
  </>  
);  
}  
  
export default App;
```

Observemos en el navegador cómo se muestra esta configuración de múltiples props en un componente:

Producto 1

Precio: 400

Producto 2

Precio: 700

Producto 3

Precio: 1000

Imagen 17. Múltiples props
Fuente: Desafío Latam

Con este ejercicio, evidenciamos cómo generar y pasar información dinámica de un componente padre a un componente hijo con los props.

Ahora, es momento de analizar una situación, los productos pueden tener más de una propiedad. En lugar de hacer `{ props.nombre_propiedad }`, podemos aprovechar el destructuring de Javascript.



Desestructuración se define como “...una expresión de JavaScript que permite desempacar valores de arreglos o propiedades de objetos en distintas variables”. (La Desestructuración - JavaScript | MDN, 2021)

Observemos un ejemplo traído de Mozilla Developer al respecto de la desestructuración:

```
// Prueba esto en la consola del navegador.  
  
const user = {  
  id: 42,  
  is_verified: true  
};  
  
const {id, is_verified} = user; // Aquí hacemos el destructuring  
  
console.log(id); // 42  
console.log(is_verified); // true
```

Destructuring de las propiedades (props)

- **Paso 5:** Apliquemos destructuring a nuestras props.

```
// src/components/Producto.jsx
const Producto = ({title, price}) => {
  return (
    <>
      <h1 class="producto">{title}</h1>
      <p>Precio: { price }</p>
    </>
  );
};

export default Producto;
```

Observemos cómo se ve nuestra aplicación y que los cambios hechos funcionen correctamente.

Producto 1

Precio: 400

Imagen 18. Múltiples props
Fuente: Desafío Latam

Ya comprobamos que utilizando destructuring en las props, la aplicación sigue funcionando sin problemas. Veamos a continuación cómo asignar valores por defecto a nuestras propiedades.

¡Sigamos! 😊

Props con valores por defecto

Podemos asignar valores a un **prop** en caso de que no se especifique al momento de utilizar el componente, para aprender a pasar valores seguiremos utilizando el proyecto `trabajando-con-jsx`.

- **Paso 6:** Para pasar valores por defecto, agregaremos un valor inicial a nuestros props desestructurados, lo que también es posible hacerlo sobre el objeto sin desestructurar):

```
// src/components/Producto.jsx
const Producto = ({title = 'Título por defecto', price = '0'}) => {
  return (
    <>
      <h1 class="producto">{title}</h1>
      <p>Precio: { price }</p>
    </>
  );
};

export default Producto;
```

- **Paso 7:** Ahora en nuestro componente `App.jsx` replicaremos el componente `Producto` y uno de ellos estará sin la asignación de valores mediante las props. Veamos el siguiente código:

```
/* App.jsx */
import Producto from './components/Producto';

function App() {
  return (
    <>
      <Producto />
      <Producto title="Producto 1" price="400" />
      <Producto title="Producto 2" price="700" />
      <Producto title="Producto 3" price="1000" />
    </>
  );
}

export default App;
```

Observemos en nuestro navegador si los valores definidos por defecto están correctamente mostrados.

Titulo por defecto

Precio: 0

Producto 1

Precio: 400

Producto 2

Precio: 700

Producto 3

Precio: 1000

Imagen 19. Props por defecto

Fuente: Desafío Latam

Excelente, hasta este punto hemos logrado comunicar y pasar propiedades desde nuestro componente Padre al Hijo, también asignamos valores por defecto en caso de que los datos no sean pasados.



Te invitamos a replicar los pasos anteriores sin recurrir a la guía. Puedes modificar la temática del ejercicio e ir probando cosas nuevas, considera que mientras más proyectos y ejercicios realizas, más rápido será tu aprendizaje.

¡Continuemos! 😊

Render Condicional

En React, cuando hablamos de render condicional nos referimos a mostrar información si se cumple cierta condición. Para lograr esto ocuparemos operadores ternarios.

Operador ternario



Según Mozilla Developers, “El operador condicional (ternario) es el único operador en JavaScript que tiene tres operandos. Este operador se usa con frecuencia como atajo para la instrucción if” (Presentando JSX –, s. f.)

```
condición ? expr1 : expr2
```



Aquí puedes revisar la fuente: [Operador condicional \(ternario\) - JavaScript | MDN](#))

Utilizando el operador ternario en JSX

A continuación modificaremos el proyecto `trabajando-con-jsx` para que muestre información o no dependiendo de una variable que declaremos.

Veamos los pasos de esta implementación:

- **Paso 1:** En nuestro `App.jsx` agregaremos una nueva propiedad que la denominaremos `stock`, la cual la asignaremos a cada producto, además de nombres respectivos para asociarlos a una fruta. Veamos cómo queda el código del componente.

```
//App.jsx
import Producto from './components/Producto';

function App() {
  return (
    <div>
      <div>
        <Producto title="Frutilla" price="400" stock="0" />
        <Producto title="Maracuyá" price="700" stock="10" />
        <Producto title="Durazno" price="1000" stock="20" />
      </div>
    </div>
  );
}
```



```
    </div>
  );
}

export default App;
```

- **Paso 2:** En nuestro archivo `index.css` agregaremos un nuevo estilo para marcar como tachado aquellos productos cuyo stock sea 0:

```
/* index.css */
.sinStock {
  text-decoration: line-through;
}
```

- **Paso 3:** En nuestro componente `Producto.jsx`, debemos agregar la nueva propiedad `stock` a la desestructuración de props.

```
// src/components/Producto.jsx
const Producto = ({ title = 'Título por defecto', price = '1000', stock
}) => {
```

A continuación, utilizaremos el operador ternario de JavaScript para modificar los estilos de cada producto a partir de una condición. Si nuestro stock es menor o igual a 0 vamos a tachar el producto, en caso contrario se mantendrá sin tachaduras.

- **Paso 4:** Para lograr lo anterior, debemos agregar estilos haciendo uso de `className`. Dicha clase se puede agregar condicionalmente utilizando el operador ternario. Veamos cómo queda el componente `Producto.jsx`.

```
// src/components/Producto.jsx
const Producto = ({ title = 'Título por defecto', price = '1000', stock
}) => {
  return (
    <>
      <p className={stock <= 0 ? 'sinStock' : null}> // Aquí
        {title} - Precio: {price} <span>stock: {stock}</span>
      </p>
    </>
  );
};
export default Producto;
```

Entendamos el código

- Nuestro componente `Producto.jsx` está recibiendo una nueva **props** llamada `stock`.
- Dentro de nuestro `return`, estamos devolviendo un párrafo que muestra el título del producto, el precio y dentro de una etiqueta `` la propiedad `stock`.
- Le estamos asignando estilos con `className` al párrafo de manera condicional, esto se lee entonces de la siguiente manera. Si el `stock` de nuestro producto es **menor o igual** a 0 **entonces** asigna la clase `sinStock`, en caso contrario no modifiques los estilos con la instrucción `null`.
- Esta clase `sinStock` fue la que definimos en nuestro archivo `index.css`

Observemos esto en el navegador web para comprobar su funcionamiento.

Frutilla - Precio: 400 stock: 0

Maracuyá - Precio: 700 stock: 10

Durazno - Precio: 1000 stock: 20

Imagen 20. Operador Ternario condición true.
Fuente: Desafío Latam



¡Felicitaciones! Logramos modificar y asignar estilos de manera condicional en un componente de React haciendo uso de CSS y el operador ternario que nos provee JavaScript.



Nota: Podrás acceder al repositorio del proyecto explicado en este material de estudio, en la plataforma con el nombre de **Repositorio Ejercicio - Trabajando con props**.



Tips: Te invitamos a repetir los pasos mostrados anteriormente, para que de este modo puedas consolidar los aprendizajes y temas planteados en este material de estudio. Recuerda que la práctica hace al maestro y es la mejor manera de comprender y aplicar los conocimientos.

Instalando componentes en React

En esta ocasión aprenderás a incorporar componentes de terceros en React. Para ello, ocuparemos la librería [react-bootstrap](#) versión 5,02 para utilizar los elementos y funcionalidades que ofrece este Framework de CSS.

A lo largo de este material de estudio, haremos la importación de algunos componentes y realizaremos las configuraciones necesarias para que puedan ser mostrados en el navegador.



Actividad 3: agregando componentes de terceros en aplicaciones React

- **Paso 1:** Creamos un nuevo proyecto usando `npm create vite`. Le asignamos el nombre **reutilizando-componentes** y seguimos los pasos que realizamos al comienzo de la guía (Seleccionar React y luego JavaScript).
- **Paso 2:** Desde nuestra terminal accedemos al directorio del proyecto, lo abrimos con el editor de código y levantamos el servidor con `npm run dev`
- **Paso 3:** Modificamos el componente `App.jsx` mostrando en un `h1` el texto "Reutilizando componentes".

```
/* App.jsx */
import './App.css';

function App() {
  return (
    <>
      <h1> Reutilizando componentes </h1>
    </>
  );
}

export default App;
```

- **Paso 4:** Instalamos el paquete `react-bootstrap` especificando la versión de Bootstrap que utilizaremos.

```
npm install react-bootstrap@next bootstrap@5.0.2
```

Para comprobar que instalamos la versión correcta, podemos revisarla en nuestro `package-lock.json`.

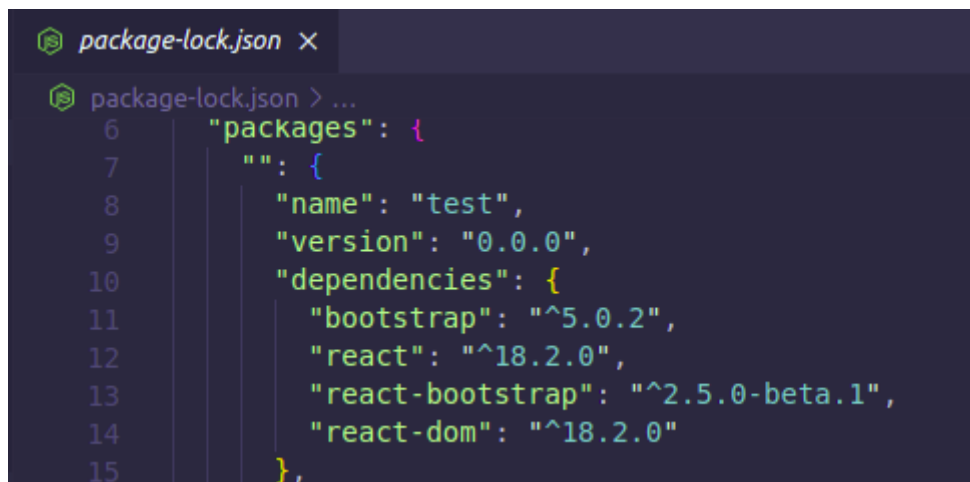


Imagen 21. Comprobar versión de Bootstrap.
Fuente: Desafío Latam.

- **Paso 5:** Importamos bootstrap en nuestro componente padre `App.jsx` bajo el siguiente código.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

- **Paso 6:** Agregamos el componente Button de bootstrap a la aplicación.

```
/* App.jsx */

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css'; // Importamos CSS
import Button from 'react-bootstrap/Button'; // Importamos el componente

function App() {
  return (
    <>
      <h1> Reutilizando componentes </h1>
      <Button> Botón de Bootstrap </Button> { /* Utilizamos el
componente */ }
    </>
  );
}
export default App;
```

Cada componente hay que importarlo individualmente antes de utilizarlo, esto en función de no tener que cargar código innecesario en nuestro proyecto.

Observemos cómo se muestra esto en el navegador.

Reutilizando componentes

Botón de Bootstrap

Imagen 22. Componente Button
Fuente: Desafío Latam



Importante: No confundir la etiqueta `<button>` con el componente `Button` en React, la convención es escribir estos componentes con mayúsculas.



A continuación te compartimos la [documentación oficial](#) de `react-bootstrap` para que puedas observar todos los componentes disponibles.

- **Paso 7:** También podemos integrar otros componentes como las Cards, para eso tenemos que copiar el ejemplo de la documentación y adicionalmente importar el componente, al integrar todo en `App.jsx` quedaría:

```
/* App.jsx */

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

function App() {
  return (
    <>
      <Card style={{ width: "18rem" }}>
        <Card.Img variant="top" src="http://placekitten.com/g/200/300"
      />
      <Card.Body>
        <Card.Title>Card Title</Card.Title>
        <Card.Text>
          Some quick example text to build on the card title and make
```

```
up the bulk of the card's content.  
    </Card.Text>  
    <Button variant="primary">Go somewhere</Button>  
  </Card.Body>  
</Card>  
</>  
);  
}  
  
export default App;
```

Observemos los cambios ahora en el navegador:

Reutilizando componentes

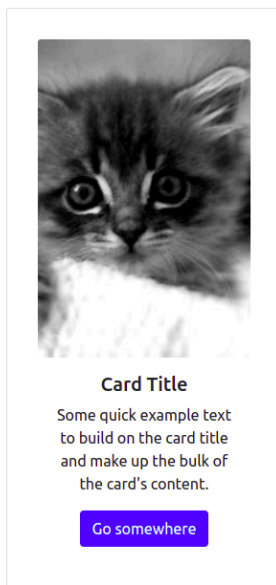


Imagen 23. Componente Card
Fuente: Desafío Latam



Nota: En plataforma podrás acceder al proyecto explicado en este material de estudio con el nombre de **Repositorio Ejercicio - Agregando componentes de terceros en aplicaciones React.**



Te invitamos a replicar los pasos anteriores sin recurrir a la guía, donde también puedes modificar la temática del ejercicio e ir probando cosas nuevas, considera que mientras más proyectos y ejercicios realizas, más rápido será tu aprendizaje.



Actividad 4: Agregando un componente dentro de otro componente

- **Paso 1:** Seguiremos trabajando en el proyecto **reutilizando-componentes**, esta vez convertiremos nuestra Card en un componente llamado `MyCard`, que integraremos dentro de un componente llamado `Boton`, que contendrá un Button de Bootstrap.
- **Paso 2:** Creamos el componente `MyCard.jsx` y eliminamos la etiqueta `<CardText>` y `<Button>` que se encontraba dentro de `<Card>`

```
/* src/components/MyCard.jsx */

import Card from 'react-bootstrap/Card';

const MyCard = () => {
  return (
    <>
      <Card style={{ width: "18rem" }}>
        <Card.Img variant="top" src="http://placekitten.com/g/200/300"
      />
        <Card.Body>
          <Card.Title>Card Title</Card.Title>
        </Card.Body>
      </Card>
    </>
  );
};

export default MyCard;
```

- **Paso 3:** Realizamos las importaciones necesarias dentro de `App.jsx`

```
/* App.jsx */

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css'; //
import MyCard from './components/MyCard';

function App() {
  return (
    <>
```

```
    <h1> Reutilizando componentes </h1>
    <MyCard /> { /* Utilizamos el componente */ }
  </>
);
}

export default App;
```

Vemos el resultado en el navegador:

Reutilizando componentes

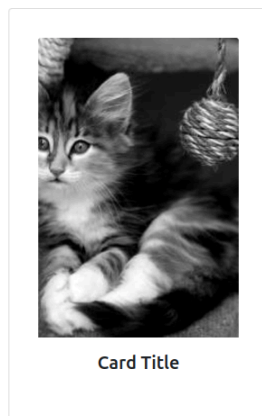


Imagen 24. Componente MyCard
Fuente: Desafío Latam

- **Paso 4:** Ahora, crearemos el componente llamado `Boton`, el cual importamos dentro de nuestro otro componente `MyCard`.

```
/* src/components/Boton.jsx */

import Button from 'react-bootstrap/Button';

const Boton = () => {
  return (
    <>
      <Button variant="primary">Botón dentro de un componente</Button>
    </>
  );
};

export default Boton;
```


- **Paso 5:** Importamos el componente `Boton` dentro del componente `MyCard`, y lo utilizaremos abajo del título de la tarjeta.

```
/* src/components/MyCard.jsx */

import Card from 'react-bootstrap/Card';
import Boton from './Boton';

const MyCard = () => {
  return (
    <>
      <Card style={{ width: "18rem" }}>
        <Card.Img variant="top" src="http://placekitten.com/g/200/300"
      />
        <Card.Body>
          <Card.Title>Card Title</Card.Title>
          <Boton />
        </Card.Body>
      </Card>
    </>
  );
};

export default MyCard;
```

Vemos el resultado en el navegador:

Reutilizando componentes

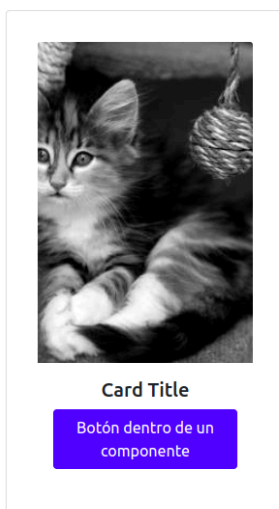


Imagen 25. Componente Boton dentro de otro componente
Fuente: Desafío Latam.

- **Paso 6:** Ahora usaremos props para pasar información a ambos componentes desde de `App.jsx`.

Las props que usaremos serán la imagen de la tarjeta, su título, el color del botón, y el texto del botón, para esto, iremos al componente `MyCard`, y declaramos las propiedades que recibirá desde `App`.

```
/* src/components/MyCard.jsx */

import Card from 'react-bootstrap/Card';
import Boton from './Boton';

const MyCard = ({image, title, colorButton, textButton}) => {
  return (
    <>
      <Card style={{ width: "18rem" }}>
        <Card.Img variant="top" src={image} />
        <Card.Body>
          <Card.Title>{title}</Card.Title>
          <Boton colorButton={colorButton} textButton={textButton} />
        </Card.Body>
      </Card>
    </>
  );
};

export default MyCard;
```

Si analizamos el código, vemos que declaramos 4 props que recibirá el component, y dentro de la tarjeta usamos las props **image** y **title**. Si observamos el componente `Boton`, este envía las props que recibe a su propio componente.

- **Paso 7:** Veamos cómo utilizamos las props dentro del componente `Boton.jsx`. Recibimos las props `colorButton` y `textButton`, las cuales vienen desde `MyCard`, que a su vez nacen en `App.jsx`.

```
/* src/components/Boton.jsx */

import Button from 'react-bootstrap/Button';

const Boton = ({colorButton, textButton}) => {
  return (
    <>
```

```
        <Button {colorButton}>{textButton}</Button>
      </>
    );
  };

export default MyCard;
```

- **Paso 8:** Realizado esto, modificaremos nuestro archivo App en donde le pasaremos la ruta de la imagen que utilizaremos, el título de la tarjeta, el color y también el texto del botón.

```
/* App.jsx */

import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css'; //
import MyCard from './components/MyCard';

function App() {
  return (
    <>
      <h1> Reutilizando componentes </h1>
      <MyCard
        image="http://placekitten.com/g/200/300"
        title="Gatito"
        colorButton="success"
        textButton="Adoptar"
      /> { /* Definimos las props que se pasaran a MyCard */ }
    </>
  );
}

export default App;
```

Si vemos el navegador veremos el siguiente resultado:

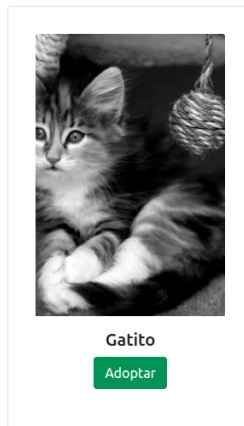


Imagen 26. Props desde App
Fuente: Desafío Latam.

Si ahora quisiéramos agregar otra tarjeta, pero con botones distintos, podemos copiar el componente MyCard en App y cambiar los valores.

```
<MyCard
  image="http://placekitten.com/g/200/300"
  title="Gatito"
  colorButton="success"
  textButton="Adoptar"
/>

<MyCard
  image="http://placekitten.com/g/200/300"
  title="Gatito"
  colorButton="warning"
  textButton="Alimentar"
/>
```

Veamos el resultado en el navegador:



Imagen 27. Modificar boton
Fuente: Desafío Latam.



Nota: En plataforma podrás acceder al ejercicio explicado en este material de estudio con el nombre de **Repositorio Ejercicio - Reutilizando componentes**.



¡Felicitaciones! Has logrado incorporar Bootstrap y sus componentes en una aplicación de React. Es muy buena práctica repetir los pasos acá explicados generando tus propias aplicaciones, ya que te ayudará a consolidar los aprendizajes a través de la práctica.

A lo largo de este curso, utilizaremos y crearemos diversos componentes, en ellos también podemos gestionar cambios de estado, navegación, entre otros. Esto lo veremos en próximas unidades.