



Agencia de Habilidades
para el Futuro

<Talento
Tech />

React JS

Clase 02 | JSX y Componentes

¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

Índice

Primeros Pasos

- Repaso HTML, CSS y JavaScript.
- Instalación con Vite y Node.js.
- Estructura básica del proyecto React.
- Introducción a JSX y creación de componentes funcionales.

JSX y Componentes

- Repaso de JSX y componentes funcionales.
- Creación de componentes más complejos.
- Bienvenida a TechLab
- Situación Inicial en TechLab

Layout en React

- Creación de la estructura básica de la aplicación.
- Desarrollo de los primeros componentes reutilizables (Header, Footer, Nav, Main, Gallery).
- Visualización de los componentes en el navegador.
- TalentoLab

Objetivos de la Clase

- 1** Comprender la importancia y las ventajas del uso de JSX.
- 2** Construir componentes más complejos que utilicen estructuras dinámicas.
- 3** Aprender a utilizar **props** para pasar datos entre componentes.
- 4** Practicar la reutilización y personalización de componentes.

JSX

Componentes funcionales

JSX

En la clase anterior vimos cómo JSX nos permite escribir código que combina HTML y JavaScript para describir la interfaz de usuario. JSX es un atajo elegante y más intuitivo que el uso directo de [React.createElement](#).

Por ejemplo:

```
const Bienvenida = () => <h1>¡Hola, Mundo!</h1>;
```



Componentes Funcionales

Además, vimos que un **componente funcional** es una función de JavaScript que devuelve un bloque de JSX. Los componentes son las piezas fundamentales de cualquier aplicación React. Hoy daremos un paso más allá y exploraremos cómo construir componentes más complejos.

Simplicidad

Permiten crear interfaces de usuario de manera declarativa y eficiente.

Reutilización

Facilitan la creación de código modular y reutilizable en toda la aplicación.

Ejemplo:

```
export default function Bienvenida() {  
  return <h1>¡Hola, Mundo!</h1>;  
}
```

ⓘ Este componente simple demuestra cómo JSX permite integrar HTML directamente en nuestro código JavaScript.

Patrón contenedor

En React, muchas veces diferenciamos dos tipos de componentes:

Componentes presentacionales (o “dumb”):

- Se enfocan en cómo se ve la interfaz.
- Renderizan la UI basándose en props.
- No contienen lógica de negocio, solo muestran datos.
- Ejemplo: un `UserCard` que recibe un usuario como prop y lo muestra con nombre, foto y descripción.

Componentes contenedores (o “smart”):

- Se encargan de la **lógica** y de manejar el **estado**.
- Obtienen datos (desde una API, un hook, un store, etc.) y definen cómo pasarlos a los presentacionales.
- Deciden qué mostrar o qué acciones ejecutar.
- Ejemplo: `UserListContainer` que hace la petición a la API, gestiona loading y error, y le pasa los usuarios a un componente `UserList`.

👉 Este patrón ayuda a **separar responsabilidades**:

- Los **contenedores** manejan la lógica y el “qué hacer”.
- Los **presentacionales** muestran los datos y el “cómo se ve”.

Ejemplo de esqueleto de un contenedor

```
export const ComponenteContainer = () => {
  // 1. Acá declararíamos un "estado" para
  guardar datos

  // 2. Acá iría la lógica para obtener los
  datos cuando el componente se monta // 
  (por ejemplo, una llamada a una API).

  // 3. Una vez obtenidos los datos, se los
  pasariámos al componente presentacional //
  En este punto, asumiremos que ya tenemos
  los datos cargados

  return ( );
}
```

En este ejemplo podemos observar cómo funciona el **patrón contenedor** en React:

- El componente **container** es el que concentra la **lógica y las peticiones de datos** (por ejemplo, llamadas a una API, validaciones o transformaciones de la información).
- Todo ese procesamiento ocurre **antes del return**, donde se define qué se va a renderizar.
- Una vez que los datos están listos, el contenedor cumple el rol de **“pasamanos”**: simplemente le transfiere la información a otro componente encargado de mostrarla.
- Ese otro componente es un **presentacional**, que no se preocupa por cómo se obtuvieron los datos, sino que se limita a recibirlos mediante **props** y renderizarlos en la interfaz

ⓘ **Gracias a él podemos dividir responsabilidades entre componentes:**

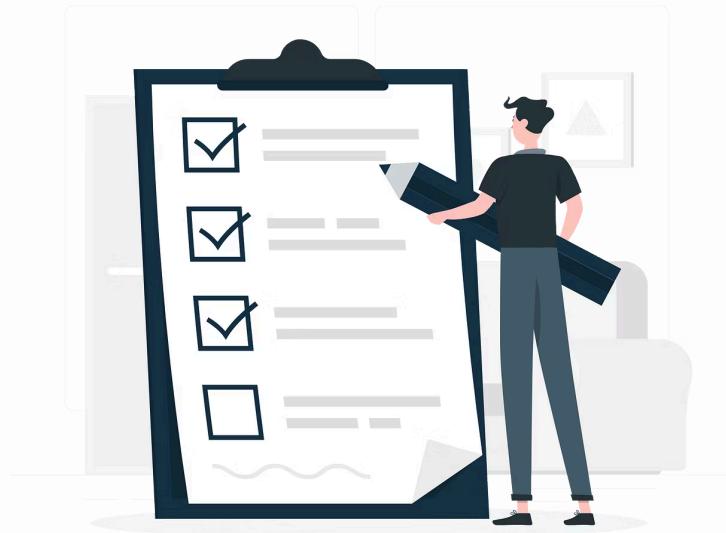
- El **contenedor** se ocupa de la lógica y de traer los datos.
- Otro componente puede encargarse del **mapeo** y mostrar un “cargando” mientras espera.
- Y finalmente, un componente **Tarjeta** se dedica solo a mostrar los datos ya listos.

Así cada componente cumple su función y el código se vuelve más ordenado y fácil de mantener.

Creación de componentes

Un componente puede incluir lógica y múltiples elementos para formar interfaces más sofisticadas. Por ejemplo, un componente que lista usuarios:

```
function ListaUsuarios() {
  const usuarios = ['Ana', 'Luis', 'Pedro',
'María'];
  return (
    <ul>
      {usuarios.map(usuario => (
        <li key={usuario}>{usuario}</li>
      ))}
    </ul>
  );
}
```



Análisis del Componente ListaUsuarios

1 Array de Datos

Definimos un array de usuarios dentro del componente.

2 Método Map

Utilizamos map para iterar sobre el array de usuarios.

3 Renderizado Dinámico

Generamos elementos dinámicamente para cada usuario.

Props

Introducción a Props

Los props son propiedades que se pasan a los componentes como argumentos.

Función

Permiten la comunicación y personalización de componentes.

Flexibilidad

Hacen que los componentes sean más reutilizables y adaptables.



Ejemplo de Componente

```
export function Boton({ texto, color = "green" }) {  
  //No se recomienda usar los estilos de esta forma. Es solo un  
  ejemplo  
  const estilo = {  
    backgroundColor: color,  
    color:"white",  
    padding:"10px",  
  };  
  return <button style={estilo}>{texto}</button>;  
}
```



Uso del Componente

```
import { Boton } from "./components/Boton";
import "./App.css";

function App() {
  return (
    <div>
      <Boton texto="Aceptar" color="green" />
      <Boton texto="Cancelar" color="red" />
    </div>
  );
}
export default App;
```



Ventajas de Usar Props



Reutilización

Permite crear componentes genéricos adaptables a diferentes contextos.



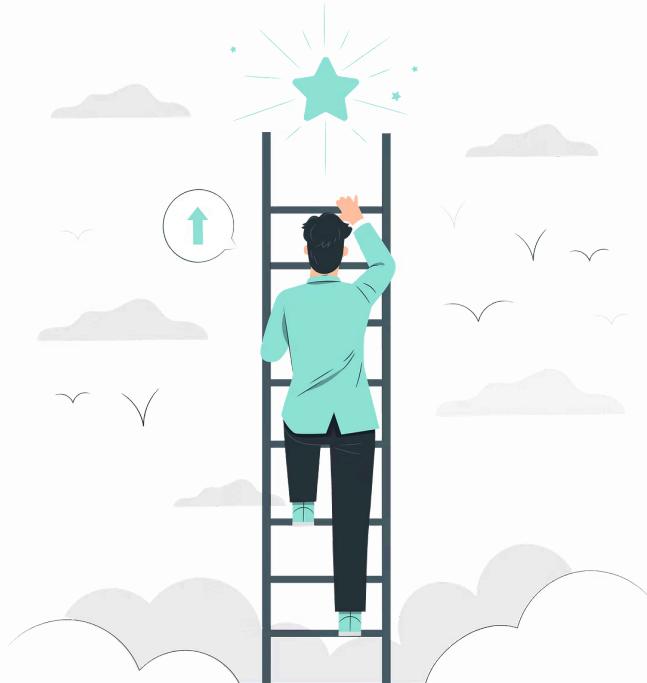
Modularidad

Facilita la creación de interfaces complejas a partir de piezas simples.



Mantenibilidad

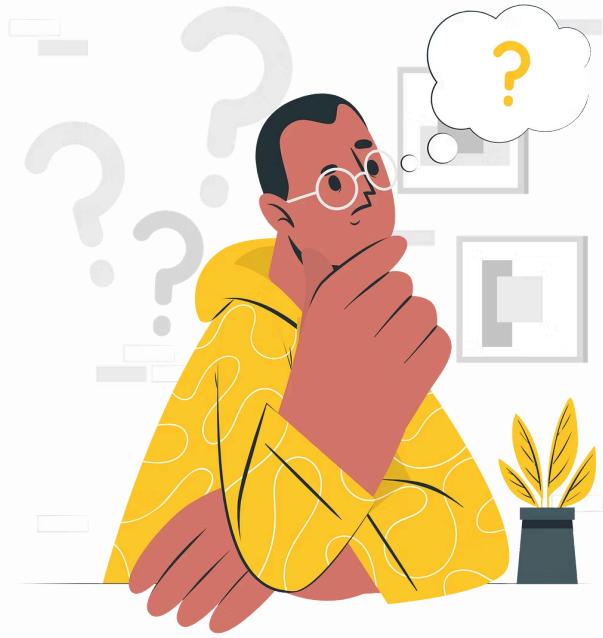
Mejora la organización del código y simplifica las actualizaciones.



Reflexión Final

En esta clase, exploramos el potencial de JSX y los componentes funcionales, comprendiendo cómo React combina lo mejor de HTML y JavaScript para construir interfaces dinámicas y reutilizables. Aprendimos que JSX simplifica el desarrollo al permitir una integración fluida entre estructura y lógica, haciendo que el código sea más claro y mantenible.

También introdujimos las **props**, una herramienta fundamental para personalizar y reutilizar componentes en diferentes contextos. Esto nos permitió crear interfaces más flexibles y adaptables, sentando las bases para el desarrollo de aplicaciones escalables y eficientes.



Preguntas para reflexionar

- ¿Qué ventajas observas al usar props en componentes React?
- ¿Cómo podrías reutilizar un componente en diferentes partes de una aplicación?
- ¿Cuál es la diferencia entre un componente estático y uno dinámico que utiliza props?

Recursos Adicionales



Documentación React

Consulta la documentación oficial de React para profundizar en los conceptos.



Guía de Props

Explora la guía sobre props en React para dominar su uso.



Ejercicios Prácticos

¡Talento Lab te está esperando!



- ⓘ Imaginá que recibís una invitación para participar en el proceso de selección de **Talento Lab** una startup ubicada en Buenos Aires.

¿Cuál sería el reto?

Completar una pasantía de aprendizaje que pondrá a prueba todas tus habilidades y aprendizaje. A partir de este momento un equipo de expertos te guiarán en este emocionante viaje.

Acerca de TalentoLab.



En **TechLab**, convertimos ideas en herramientas digitales innovadoras y confiables, ofreciendo servicios de desarrollo con un enfoque en la calidad y eficiencia. Nuestro compromiso es desarrollar soluciones que optimicen procesos y potencien negocios, combinando creatividad, tecnología de punta y un compromiso absoluto con la satisfacción de nuestros clientes. Tu éxito es nuestra prioridad, y nuestras soluciones están diseñadas para marcar la diferencia.

Equipo Tech Lab:





Silvia

Product Owner



Luis

Diseñador UX/UI



Matias

Desarrollador



Sabrina

Desarrolladora



Situación inicial en TechLab.

¡Únete al equipo TechLab! ✨

La familia de **TechLab** está buscando un nuevo integrante para colaborar en el desarrollo de un emocionante producto innovador. Buscamos personas con un perfil **proactivo, autodidacta e innovador**, apasionadas por el aprendizaje continuo y con ganas de afrontar grandes desafíos.



Hemos preparado una serie de **ejercicios prácticos** diseñados para evaluar tus habilidades técnicas. Estos retos nos permitirán conocerte mejor y comprobar cómo aplicas tus conocimientos en situaciones reales.

Ejercicio Práctico

Obligatorio

1) Crea un componente que reciba un array de productos como prop y los muestre en una lista ordenada.

Ejemplo de array:

```
const productos = ['Manzanas', 'Peras',  
'Naranjas'];
```

2) Crea un componente Tarjeta que reciba props para mostrar un título, una descripción y un botón personalizado.

Ejemplo de uso:

```
<Tarjeta titulo="Oferta especial"  
descripcion="20% de descuento en todos
```

```
los productos"  
botonTexto="Ver más" />
```

3) Usa el componente Boton que creamos para estilizar diferentes acciones en una página.



¡Nuevo cuestionario en Campus!

- i No olvides que los cuestionarios son de carácter obligatorio para poder avanzar con la cursada.

