02 - Tecnicatura Superior en Ciencias de Datos e Inteligencia Artificial

Componentes, Props e Integración de Tailwind CSS en React con Vite

1. ¿Qué es un Componente en React?

Un **componente** en React es una **función de JavaScript** que devuelve **JSX**, una extensión de la sintaxis que permite escribir HTML directamente dentro de JavaScript.

Los componentes permiten **dividir la interfaz en piezas independientes y reutilizables**, lo que hace que el código sea más modular y mantenible.

Ejemplo de un componente funcional:

```
function Saludo() {
  return <h1>Hola, mundo</h1>;
}
```

También se puede escribir como una **función flecha**:

```
const Saludo = () => {
  return <h1>Hola, mundo</h1>;
};
```

¿Dónde se usan?

Los componentes se utilizan para representar cualquier parte de la UI: botones, formularios, tarjetas, encabezados, etc.

2. ¿Qué son las Props (Propiedades)?

Las **props** (abreviación de *properties*) son **valores que se pasan desde un componente padre a un componente hijo**, y permiten que los componentes sean **dinámicos y reutilizables**.

Ejemplo:

```
function Saludo(props) {
  return <h1>Hola, {props.nombre}</h1>;
}

// Uso:
<Saludo nombre="Lucía" />
```

También se puede usar **desestructuración** para mayor claridad:

```
const Saludo = ({ nombre }) => {
  return <h1>Hola, {nombre}</h1>;
};

// Uso:
<Saludo nombre="Lucía" />
```

¿Para qué sirven las props?

- Personalizar la salida del componente
- Reutilizar un mismo componente con distintos datos
- Comunicar componentes (padre → hijo)

3. ¿Cómo integrar Tailwind CSS en un proyecto de React con Vite?

Tailwind CSS es un framework de utilidades que permite aplicar estilos directamente en las clases de los elementos, sin escribir archivos CSS personalizados.

Ventajas:

- Rapidez para prototipar
- Consistencia visual
- Totalmente personalizable
- Ideal para trabajar con React y componentes reutilizables

Pasos para integrarlo en un proyecto creado con Vite + React:

1. Crear el proyecto con Vite:

```
npm create vite@latest
```

Elegí:

- Framework: `React`
- Variante: `JavaScript + SWC`

Ingresá a la carpeta del proyecto:

```
cd mi-app
```

2. Instalar Tailwind y sus dependencias:

```
npm install tailwindcss @tailwindcss/vite
```

3. Configurar el archivo de configuración de Vite:

Editar el archivo `vite.config.js`, importando tailwindcss y añadiendo el pulgin al arreglo:

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react-swc";
import tailwindcss from "@tailwindcss/vite";

export default defineConfig({
    plugins: [react(), tailwindcss()],
});
```

4. Importar Tailwind en tu archivo CSS:

En `src/App.css` :

```
@import "tailwindcss";
```

Ejemplo usando Tailwind en un componente:

```
jsx
CopyEdit
const Boton = () => {
  return (
    <button className="bg-blue-500 text-white px-4 py-2 rounded hover:b
g-blue-600">
      Enviar
    </button>
  );
};
```

4. Cómo usar componentes de Tailblocks en un proyecto React

¿Qué es Tailblocks?

Tailblocks.cc es una biblioteca gratuita de **componentes prediseñados con Tailwind CSS** listos para copiar y pegar. Incluye secciones como encabezados, testimonios, formularios, galerías y más.

Estos bloques están diseñados en **HTML** con clases de **Tailwind**, por lo que son una excelente base visual para construir interfaces rápidas y responsivas.

¿Puedo usarlos directamente en React?

No exactamente. Como los bloques de Tailblocks están en HTML puro, necesitamos **adaptarlos** para que funcionen correctamente dentro de un archivo `.jsx` en React. JSX es más estricto que HTML tradicional.

Pasos para importar y adaptar un componente de Tailblocks a React

1. Copiar el bloque desde Tailblocks.cc

Elegí un componente, por ejemplo un formulario de contacto, y copiá el código HTML.

2. Pegar el código en un componente React

Creá un nuevo archivo de componente en tu carpeta `src/components/`, por ejemplo:

```
// src/components/FormularioContacto.jsx
const FormularioContacto = () => {
  return (
    <section className="text-gray-600 body-font relative">
      {/* contenido aquí */}
    </section>
  );
};

export default FormularioContacto;
```

✨ 3. Modificar el código HTML para que sea JSX válido

Las principales modificaciones que debés hacer:

HTML (original)	JSX (React)
`class`	`className`
`for` (en etiquetas `<label>`)	`htmlFor`
Etiquetas vacías como `<input />`	**Dejarlas cerradas correctamente**
Atributos booleanos (`checked`, `disabled`)	Escribirlos como `checked={true}`

Ejemplo de conversión:

HTML original:

```
html
CopyEdit
```

```
<label for="email" class="text-sm text-gray-600">Email</label>
<input type="email" id="email" name="email" class="border rounded p-2"
/>
```

JSX modificado:

```
<label htmlFor="email" className="text-sm text-gray-600">Email</label>
<input type="email" id="email" name="email" className="border rounded
p-2" />
```

4. Importar y usar el componente

En tu archivo `App.jsx` o donde lo necesites:

```
import FormularioContacto from './components/FormularioContacto';

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

Tip extra:

Si copiás componentes más complejos que incluyen imágenes, íconos o SVGs, asegurate de:

- Cerrar correctamente todas las etiquetas (`<img />` , `<br />` , etc.)
- Si hay enlaces a fuentes externas (como íconos SVG externos), verificá que estén bien cargados o reemplazalos con componentes propios

Recursos:

- Tailblocks
- Playground online para ver el JSX
- Documentación oficial de JSX