Logotipo

El contenido generado por IA puede ser incorrecto.

Santiago Yosa González

**Ingeniero Sistemas de Información**



Logotipo

El contenido generado por IA puede ser incorrecto.}

**Contenido web:**

Características y distribución, formatos usuales, tipo de archivo, tamaño de las imágenes, fotografías y vínculos, diseño visual, factores del diseño visual, licencias para el uso y publicación de recursos, características, técnicas de configuración, métodos de administración de la base de datos del contenido web.

**Unidad I**

**Estructuración de Contenidos**

**Situación Caso de Uso**

Imaginen que trabajan en el equipo de desarrollo de **Netflix**. Tienen un catálogo de 15,000 películas.

* **Método Tradicional (HTML):** ¿Escribirían 15,000 veces la etiqueta <div class="movie-card">...</div>? Si el diseñador cambia el tamaño de la fuente del título, tendrían que editar 15,000 archivos. Eso es inviable.
* **Método React:** Ustedes no crean 15,000 tarjetas. Ustedes crean **UNA (1)** sola plantilla maestra (Componente) y tienen una **Base de Datos** (Lista de información). React se encarga de mezclar la plantilla con los datos para generar la interfaz.

Hoy aprenderemos a dejar de escribir contenido "quemado" (hardcoded) y empezaremos a crear **estructuras de datos** que alimentan **componentes visuales**.

**Objetivo:** Despertar la necesidad técnica de la reutilización.

**Contenido web: Características y Distribución**

**Naturaleza del Contenido (Componentes):** En React, el contenido no se distribuye en páginas HTML tradicionales, sino en **componentes**. Un componente es una pieza de la interfaz de usuario (UI) independiente y reutilizable (como un botón o todo un panel) que tiene su propia lógica y apariencia. Estos actúan como bloques (similares a Legos) que se combinan para formar la aplicación.

**Distribución (Single Page Application - SPA):** React se utiliza comúnmente para crear SPAs. Esto significa que hay un único archivo HTML (index.html) que actúa como contenedor raíz. El contenido interno se actualiza dinámicamente mediante JavaScript sin recargar la página completa.

**Empaquetado (Bundling):** Para la distribución en producción, el código pasa por un proceso de *bundling* (usando herramientas como Vite o Webpack). Este proceso minifica el código, lo hace "feo" (uglify) para proteger la lógica y realiza *tree shaking* (sacudir el árbol) para eliminar código no utilizado y optimizar la carga.

**Formatos usuales y Tipo de archivo**

• **JSX/TSX:** El formato estándar en React no es HTML puro, sino **JSX** (JavaScript XML), una extensión de sintaxis que permite escribir estructuras similares a HTML dentro de JavaScript. Si se usa TypeScript, la extensión del archivo será .tsx.

• **Archivos de Lógica:** Los componentes se definen usualmente en archivos .js o .jsx (o .ts/.tsx para TypeScript).

• **JSON:** Se utiliza ampliamente para la configuración (como package.json para dependencias) y para el intercambio de datos, como al recibir respuestas de una API.

**Imágenes, Fotografías y Vínculos**

• **Importación de Imágenes:** En React, las imágenes se tratan como módulos. Deben importarse en el archivo JavaScript (ej. import logo from './logo.png') para ser utilizadas en el atributo src de una etiqueta <img>:

La imagen debe estar dentro de la carpeta src (por ejemplo, en src/imagenes).

2. Importa la imagen al inicio del archivo.

3. Úsala en el atributo src entre llaves {}.

• **Accesibilidad:** Es imperativo incluir el atributo alt en las imágenes para describir el contenido.

• **Vínculos y Navegación:** Para la navegación interna en aplicaciones React, no se suelen usar enlaces <a> tradicionales que recargan la página. Se utilizan librerías como react-router-dom para manejar el enrutamiento y cambiar lo que se renderiza en pantalla según la URL.

**Diseño Visual y Factores del Diseño**

• **Clases CSS:** En lugar del atributo class de HTML, en React se utiliza **className** para asignar estilos, ya que "class" es una palabra reservada en JavaScript.

• **Estilos en Línea:** Se pueden aplicar estilos directamente a los elementos usando un objeto de JavaScript en el atributo style. Las propiedades CSS que tienen guiones (como background-color) se escriben en formato **camelCase** (ej. backgroundColor).

• **CSS Modules e Importación:** Es común crear archivos CSS separados por componente (ej. Boton.css) e importarlos en el archivo del componente para aplicar reglas de estilo específicas.

• **Renderizado Condicional:** El diseño visual puede cambiar dinámicamente. Se pueden asignar clases o estilos basándose en el estado (por ejemplo, si un botón es de "clic" o de "reiniciar" cambia su color) usando operadores ternarios.

**Características Técnicas**

• **Virtual DOM:** React utiliza un DOM Virtual para mejorar el rendimiento. Cuando hay cambios, React actualiza el DOM Virtual, lo compara con el DOM real y solo actualiza las partes que han cambiado, evitando renderizar toda la página de nuevo.

• **Hooks:** Permiten a los componentes funcionales tener estado y ciclo de vida. Los más comunes son:

    ◦ useState: Para manejar variables reactivas (estado).

    ◦ useEffect: Para manejar efectos secundarios como llamadas a APIs o sincronización con sistemas externos.

    ◦ useRef: Para referencias mutables que no causan re-renderizado.

• **Props (Propiedades):** Mecanismo para pasar datos de un componente padre a un componente hijo. Son de lectura única para el hijo.

**Técnicas de Configuración**

• **Inicialización:** Se utilizan herramientas de línea de comandos (CLI) para configurar el entorno. Un ejemplo moderno es **Vite** (usando npm create vite@latest) o anteriormente create-react-app. Esto estructura las carpetas, instala dependencias y configura el servidor de desarrollo.

• **Estructura de Carpetas:** No hay una regla fija, pero se sugiere organizar por "Componentes", "Hooks", "Modelos" y "Servicios" dentro de la carpeta src.

• **Dependencias:** Se gestionan a través de archivos package.json y se instalan usando comandos como npm install.

**Métodos de Administración de la Base de Datos (Manejo de Datos)**

React es una librería de Frontend (interfaz), por lo que no administra bases de datos directamente. Sin embargo, administra la **data** del contenido web de las siguientes formas:

• **Consumo de APIs:** React obtiene datos de bases de datos externas mediante peticiones HTTP (fetch o Axios) a servicios o APIs REST. Estos datos se guardan en el estado local del componente para ser mostrados.

• **Estado Global (Context):** Para evitar pasar datos manualmente por muchos niveles de componentes (*prop drilling*), se utiliza **Context API** para proveer datos a todo el árbol de componentes (similar a una base de datos global en memoria para la sesión actual).

• **Persistencia Local:** Se puede utilizar localStorage para guardar información en el navegador del usuario (como tokens de sesión) para que los datos persistan al recargar la página.

**Gestión de Recursos Multimedia en Vite**

Para optimizar la carga (performance), debemos saber dónde poner las imágenes.

1. **Carpeta /public:** Los archivos aquí se sirven tal cual. Se usan para imágenes estáticas que no cambian de nombre (ej: logo.png, favicon.ico). Se accede con string directo: <img src="/logo.png" />.
2. **Carpeta /src/assets:** Las imágenes aquí pasan por el proceso de compilación (bundling). Vite puede optimizarlas, renombrarlas para caché y convertirlas a base64 si son pequeñas. Se deben **importar** en el JS.

**Regla de Oro (Formatos):**

* **JPG/JPEG:** Fotografías con muchos colores.
* **PNG:** Imágenes con transparencia o gráficos planos simples.
* **SVG:** Iconos y logotipos vectoriales (escalan sin pixelarse).
* **WebP:** El estándar moderno. Menor peso, misma calidad. Úsenlo siempre que puedan.

**Simulación de Base de Datos (JSON)**

Como no tenemos SQL aún, nuestra "base de datos" será un **Arreglo de Objetos** en JavaScript. Esto es lo que eventualmente recibirán de un servidor, así que es vital que recordemos como es la estructura JSON.

// database.js (Simulación)

export const productos = [

{

id: 1,

nombre: "Laptop Gamer",

precio: 1500,

imagen: "https://via.placeholder.com/150"

},

{

id: 2,

nombre: "Mouse Óptico",

precio: 25,

imagen: "https://via.placeholder.com/150"

}

];

**El concepto de PROPS (Propiedades)**

Las **Props** son la forma en que el componente "Padre" le pasa información al "Hijo".

* Son de **lectura única** (el hijo no puede modificar sus props).
* Se ven como atributos HTML, pero son variables JS.

**En el Componente Padre (El Emisor)**

Aquí es donde "enviamos" la información. Se parece mucho a escribir atributos HTML normales.

**En el Componente Hijo (El Receptor)**

Aquí es donde la función "recibe" el paquete. React empaqueta todos los atributos del padre en un solo objeto llamado props.

Imagen de la pantalla de un celular de un mensaje en letras blancas

El contenido generado por IA puede ser incorrecto.

**Forma A: Tradicional (Sin Destructuring)**

* *Menos usada hoy en día, pero buena para entender qué es el objeto props.*

// La función recibe SIEMPRE un objeto como primer argumento

function TarjetaProducto(props) {

// props = { precio: 1500 }

return <h3>Precio: ${props.precio}</h3>;

}

**Forma B: Moderna (Con Destructuring - La que usaremos)**

* *Extraemos directamente las propiedades que necesitamos.*

// Desestructuración (Extraer)

// ↓

function TarjetaProducto({ precio }) {

// Ya puedo usar la variable 'precio' directamente

return <h3>Precio: ${precio}</h3>;

}

**Preguntas Frecuentes (FAQs)**

**¿Por qué no puedo poner la imagen directamente en el HTML img src="foto.jpg"?**

Porque en React, si la imagen está en src/assets, la ruta cambia cuando se compila la app. Debes importarla (import foto from './assets/foto.jpg') para que Vite sepa dónde encontrarla y optimizarla.

**¿Qué pasa si tengo 100 productos, tengo que escribir <Tarjeta /> 100 veces?**

No. Usamos una función de JavaScript llamada .map(). Esta función recorre tu arreglo de datos y devuelve un componente <Tarjeta /> por cada elemento automáticamente. Es la técnica de **Renderizado de Listas**.

**Demo**

Crear una galería de "Destinos Turísticos" usando un array de datos y un componente reutilizable.

**Paso 1: Crear la estructura de datos (src/data/destinos.js)**

Separamos los datos de la vista. Buenas prácticas de organización.

export const dataDestinos = [

{

id: 1,

ciudad: "Tokio",

pais: "Japón",

img: "https://images.unsplash.com/photo-1540959733332-eab4deabeeaf?w=400",

precio: 1200

},

{

id: 2,

ciudad: "París",

pais: "Francia",

img: "https://images.unsplash.com/photo-1431274172761-fca41d930114?w=400",

precio: 950

}

];

**Paso 2: Crear el componente UI (src/components/DestinoCard.jsx)**

Creamos el molde. Usamos CSS en línea o clases simples para el diseño visual. Observa cómo recibimos props.

// Importamos React (opcional en versiones nuevas, pero buena práctica mencionar)

import './DestinoCard.css'; // Asumimos un CSS básico para dar tamaño

export function DestinoCard({ ciudad, pais, imagen, precio }) {

return (

<div className="card-turismo">

<img src={imagen} alt={ciudad} className="card-img" />

<div className="card-body">

<h2>{ciudad}</h2>

<p className="subtitulo">{país}</p>

<span className="precio">US$ {precio}</span>

<button>Reservar</button>

</div>

</div>

);

}

**Paso 3: Unificar en el Padre (src/App.jsx)**

Aquí ocurre la magia. Importamos los datos y el componente, y usamos .map() para generar la lista.

import { dataDestinos } from './data/destinos';

import { DestinoCard } from './components/DestinoCard';

import './App.css';

function App() {

return (

<div className="contenedor-principal">

<h1>Nuestros Destinos</h1>

<div className="grilla-destinos">

{/\* Iteramos sobre la "base de datos" \*/}

{dataDestinos.map((destino) => (

<DestinoCard

key={destino.id} // VITAL: Identificador único para React

ciudad={destino.ciudad}

pais={destino.pais}

imagen={destino.img}

precio={destino.precio}

/>

))}

</div>

</div>

);

}

export default App;18/02/2026

**Demo 2**

Primero, definimos la información. En un archivo separado, por simplicidad didáctica, creamos nuestro arreglo.

// src/data/teamData.js

export const teamMembers = [

{

id: 1,

nombre: "Santiago Yosa",

rol: "Senior Frontend Dev",

bio: "Experto en React y amante del café.",

foto: "https://i.pravatar.cc/150?img=11" // Avatar aleatorio

},

{

id: 2,

nombre: "Valeria Ruiz",

rol: "UX/UI Designer",

bio: "Diseñando experiencias centradas en el usuario.",

foto: "https://i.pravatar.cc/150?img=5"

},

{

id: 3,

nombre: "Carlos Díaz",

rol: "Backend Engineer",

bio: "Optimizando bases de datos y APIs.",

foto: "https://i.pravatar.cc/150?img=3"

}

];

**Paso 2: El Componente (El Molde)**

Creamos la tarjeta. Aquí es donde ocurre la **magia de las Props**.

Observen cómo en lugar de escribir "Santiago", escribo {nombre}. Esas llaves le dicen a React: "Aquí va una variable, no texto plano".

src/components/TeamCard.jsx

Desestructuramos las props que necesitamos recibir

export function TeamCard({ nombre, rol, bio, foto }) {

return (

<div className="card">

{/\* 2. Usamos las props en el atributo src y alt \*/}

<img src={foto} alt={`Foto de ${nombre}`} className="card-img" />

<div className="card-info">

{/\* 3. Inyectamos el texto dinámico \*/}

<h2 className="card-title">{nombre}</h2>

<h4 className="card-subtitle">{rol}</h4>

<p className="card-bio">{bio}</p>

</div>

<button onClick={() => alert(`Contactando a ${nombre}...`)}>

Contactar

</button>

</div>

);

}

**Paso 3: La Implementación (El Montaje) -** Aquí unimos los datos con el molde.

*Recordemos* el .map(): Por cada miembro que encuentres en la lista, quiero que 'dibujes' una <TeamCard /> y le entregues sus datos correspondientes

return (

<div className="container">

<h1 className="title">Conoce al Equipo</h1>

<div className="grid-cards">

{/\* Recorremos el arreglo \*/}

{teamMembers.map((miembro) => (

<TeamCard

key={miembro.id} // OBLIGATORIO: ID único para React

nombre={miembro.nombre} // Pasamos dato del objeto -> a la prop

rol={miembro.rol}

bio={miembro.bio}

foto={miembro.foto}

/>

))}

</div>

</div>

);

**Demo 3 Con CSS** El Archivo de Datos (datos.js)

// src/datos.js

export const equipo = [

{

id: 1,

nombre: "Ana García",

rol: "Desarrolladora Frontend",

foto: "https://i.pravatar.cc/150?u=a042581f4e29026024d"

},

{

id: 2,

nombre: "Carlos López",

rol: "Arquitecto de Software",

foto: "https://i.pravatar.cc/150?u=a042581f4e29026704d"

},

{

id: 3,

nombre: "Elena Torres",

rol: "Diseñadora UX/UI",

foto: "https://i.pravatar.cc/150?u=a042581f4e29026704e"

}

];

**El Componente Reutilizable (TarjetaPerfil.jsx)**

// src/TarjetaPerfil.jsx

// 1. Recibimos las propiedades (props) entre llaves {}

export function TarjetaPerfil({ nombre, rol, foto }) {

// Estilos simples para la tarjeta

const estiloCard = {

border: "1px solid #ccc",

borderRadius: "10px",

padding: "20px",

textAlign: "center",

width: "200px",

boxShadow: "2px 2px 10px rgba(0,0,0,0.1)"

};

return (

<div style={estiloCard}>

{/\* 2. Usamos la prop 'foto' en el src \*/}

<img

src={foto}

alt={`Foto de ${nombre}`}

style={{ borderRadius: "50%", width: "100px" }}

/>

{/\* 3. Inyectamos las props de texto \*/}

<h3 style={{ margin: "10px 0", color: "#333" }}>{nombre}</h3>

<p style={{ color: "#666", fontSize: "0.9rem" }}>{rol}</p>

<button style={{ cursor: "pointer", marginTop: "10px" }}>

Contactar

</button>

</div>

);

}

**La Aplicación Principal**

Donde ocurre la magia. Importamos los datos y el componente, y usamos .map() para generar la lista.

// src/App.jsx

import { equipo } from "./datos"; // Importamos los datos

import { TarjetaPerfil } from "./TarjetaPerfil"; // Importamos el componente

function App() {

return (

<div style={{ padding: "40px", fontFamily: "sans-serif" }}>

<h1>Nuestro Equipo de Trabajo</h1>

{/\* Contenedor Flex para poner las tarjetas una al lado de la otra \*/}

<div style={{ display: "flex", gap: "20px", flexWrap: "wrap" }}>

{/\* Mapeamos el arreglo 'equipo' \*/}

{equipo.map((miembro) => (

<TarjetaPerfil

key={miembro.id} // OBLIGATORIO: Identificador único

nombre={miembro.nombre} // Pasamos datos del objeto a la prop

rol={miembro.rol}

foto={miembro.foto}

/>

))}

</div>

</div>

);

}

export default App;

**Troubleshooting (Solución de Errores Comunes)**

**Error:** "Warning: Each child in a list should have a unique 'key' prop."

* **Causa:** Al usar .map(), React necesita saber qué elemento es cuál para optimizar el renderizado.
* **Solución:** Siempre agrega key={item.id} (o un valor único) en el componente dentro del map.

**Error:** Las imágenes no cargan (aparece el ícono de imagen rota).

* **Causa:** Probablemente usaste una ruta relativa (./img/foto.jpg) en un array de datos, pero la imagen no está siendo importada ni está en public.
* **Solución:** Para empezar, usen URLs absolutas (https://...) o muevan las imágenes a la carpeta public y usen /nombre.jpg.

**Error:** Props son undefined.

* **Causa:** Llamaste a la prop title en el padre, pero en el hijo intentaste leer { titulo }.
* **Solución:** Los nombres deben coincidir exactamente.

**Ejercicio Independiente Aplicado (Extraclase)**

Construcción de un "Blog de Noticias Tech".

1. Crea una base de datos local (data.js) con al menos 4 noticias. Cada noticia debe tener: id, titulo, resumen, fecha, autor y categoria.
2. Diseña un componente NewsItem.
3. Usa **CSS Grid** en el contenedor padre para que se vea responsive (columnas).
4. **Reto:** Agrega un pie de página al componente que diga "Leer más..." (aunque el link no funcione aún).

**Criterios de Logro:**

* El código utiliza .map() correctamente.
* No existe código HTML repetido (hardcoded) para las noticias.
* Uso correcto de la prop key.
* Las imágenes se visualizan correctamente.

Logotipo

El contenido generado por IA puede ser incorrecto.