Clase 5 — Introducción a React

Diego Muñoz

16 de septiembre de 2025

Introducción

- React es una biblioteca para construir interfaces de usuario.
- Nos permite dividir la UI en **componentes** reutilizables.
- Usa **JS moderno + JSX** y un modelo de estado propio.

Node.js y npm/yarn

- **Node.js**: entorno para ejecutar JavaScript fuera del navegador.
- Necesario para instalar dependencias y correr herramientas modernas.
- **npm**: gestor de paquetes oficial de Node.
- **yarn**: alternativa más rápida y con mejoras en UX.

```
# instalar dependencias con npm
npm install

# instalar alternativa yarn
npm install --global yarn

# instalar dependencias con yarn
yarn
```

Vite

- Herramienta moderna para crear proyectos frontend.
- Muy rápida comparada con `create-react-app`.
- Soporta React, Vue, Svelte, etc.

```
# crear un proyecto react con vite
yarn create vite nombre-proyecto
```

React crudo

```
import React from "react";
import { createRoot } from "react-dom/client";

function App() {
  return React.createElement("h1", null, "Hola React");
}

const root = createRoot(document.getElementById("root"));
root.render(React.createElement(App));
```

React con JSX

```
import React from "react";
import { createRoot } from "react-dom/client";

function App() {
  return <h1>Hola React con JSX</h1>;
}

const root = createRoot(document.getElementById("root"));
root.render(<App />);
```

- JSX → sintaxis similar a HTML, pero en JavaScript.
- El transpiler lo transforma a `React.createElement`.

Componentes (definición)

```jsx
function Card({ title, body }) {
  return (
    <div className="card">
      <h3>{title}</h3>
      <p>{body}</p>
    </div>
  );
}
```

Componentes (uso)

```
function App() {
  return (
    <div>
      <Card title="Intro a JS" body="Variables y funciones" />
      <Card title="React" body="Estado y componentes" />
    </div>
  );
}
```

Hooks: useState (concepto)

- React maneja **estado** con hooks.
- `useState` devuelve un valor + función para actualizarlo.

```
import { useState } from "react";
```

Hooks: useState (ejemplo)

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Valor: {count}</p>
      <button onClick={() => setCount(count + 1)}>Sumar</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```

Eventos y listas (estado inicial)

```
function App() {
  const [tasks, setTasks] = useState([
    { id: 1, text: "Leer", done: false }
  ]);
```

Eventos y listas (función toggle)

```
function toggle(id) {
  setTasks(tasks.map(
    t => t.id === id ? { ...t, done: !t.done } : t
  ));
}
```

```
  return (
    <div>
      {tasks.map(t => (
        <div
          key={t.id}
          onClick={() => toggle(t.id)}
          style={{ textDecoration: t.done ? "line-through" : "none" }}
        >
          {t.text}
        </div>
      ))}
    </div>
  );
}
```

Buenas prácticas iniciales

1. Componentes pequeños y reutilizables.
2. Estado local con `useState`.
3. Props inmutables: no modificar lo que llega al componente.
4. Keys únicas al renderizar listas.
5. Mantener JSX legible y limpio.

Proyecto de ejemplo

Referir a ejemplo en GitHub.

Resumen

- Node.js y yarn/npm: base del ecosistema.
- Vite: crear y correr proyectos rápidamente.
- JSX simplifica la creación de elementos.
- Componentes = funciones puras + props.
- `useState`: primer hook para manejar estado.
- Listas y eventos → ya estamos listos para proyectos más grandes.