

# Desarrollo de Apps Web con React

# Instructor:



Nicolás Gallardo

[nicolasgallardo91@gmail.com](mailto:nicolasgallardo91@gmail.com)

<https://www.linkedin.com/in/nicolas-gallardo>

<https://github.com/ngallardo91>

## **Desarrollo de Apps Web con React**

### **Parte 1 – Introducción – 4 Sesiones**

Martes 21/10, jueves 23, martes 28, jueves 30

### **Parte 2 – Trabajando con bibliotecas y creando componentes con React – 11 Sesiones**

Martes 4/11, martes 11, jueves 13, martes 18, jueves 20 ... y así hasta finalizar el jueves 11 de diciembre

**Siempre en el horario de 11:00 a 13:00 hs**

## Desarrollo de Apps Web con React

### Parte 1 – Introducción – 4 Sesiones

**HTML y CSS:** repasaremos los conceptos básicos.

**Javascript / Typescript:** para mejor calidad del código. Ejemplos, agregando tipado estático a JavaScript.

**ViteJS:** desarrollo rápido y eficiente de aplicaciones web. Instalación, ejemplos, ejercicios.

**React (versión 19):** Instalación. Iniciar un proyecto nuevo. Agregar código a un proyecto existente. Configurar el editor. Instalar herramientas de desarrollo.

## Desarrollo de Apps Web con React

### Fundamentos de HTML

- Estructura básica de un documento HTML
- Etiquetas más utilizadas (div, header, footer, section, article)
- Formularios y validaciones básicas
- Semántica y buenas prácticas
- Introducción al DOM

# ¿Qué es HTML?

HTML (HyperText Markup Language) define la estructura del contenido web.

- Es un lenguaje de programación.
- Estructura texto, imágenes, enlaces, formularios, etc.
- Forma parte del trío base: **HTML + CSS + JavaScript.**

# Página WEB Estructura Básica

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi página</title>
  </head>
  <body>
    <!-- Contenido visible -->
  </body>
</html>
```

## Partes principales:

- <head> → metadatos, título, SEO
- <body> → contenido visible

## Etiquetas comunes

- `<header>`, `<main>`, `<section>`, `<article>`, `<aside>`, `<footer>`
- `<div>` y `<span>` → agrupaciones sin significado semántico
- `<p>`, `<h1>`–`<h6>`, `<ul>`/`<ol>`/`<li>`, `<strong>`, `<em>`, `<small>`



## Semántica

Usar etiquetas con significado mejora:

- **Accesibilidad**
- **SEO**
- **Mantenimiento del código**

```
<header>Cabecera</header>  
<main>Contenido</main>  
<footer>Pie de página</footer>
```

## Formularios

```
<form>
  <fieldset>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" required>
  </fieldset>
  <button type="submit">Enviar</button>
</form>
```

Atributos útiles:

required, pattern, minlength, placeholder, etc.

## Multimedia

Etiquetas principales:

- `<img>` → imágenes
- `<audio>` → sonido
- `<video>` → video
- `<iframe>` → contenido externo

Atributos útiles: `controls`, `autoplay`, `loading="lazy"`

## Buenas prácticas

- ✓ Código indentado
  - ✓ Etiquetas cerradas
  - ✓ Atributos en minúsculas
  - ✓ Semántica correcta
- 
- ✗ No usar etiquetas obsoletas como  
<center> o <font>

## Desarrollo de Apps Web con React

### Fundamentos de CSS

- Selectores, clases e IDs
- Modelo de caja (Box Model)
- Flexbox y Grid
- Variables y funciones en CSS moderno

## Introducción a CSS (Cascading Style Sheets)

**CSS** controla el aspecto visual de una página web.

Permite definir colores, fuentes, márgenes, alineaciones y más.

- ◆ HTML = estructura
- ◆ CSS = diseño
- ◆ JavaScript = interactividad

## Cómo aplicar CSS

Inline(en linea)

```
<p style="color:red;">Texto rojo</p>
```

Interno (en el mismo archivo)

```
<style>
  p { color: red; }
</style>
```

Externo (archivo .css)

```
<link rel="stylesheet" href="styles.css">
```

## Selectores

El selector indica a **qué elementos** se aplicarán los estilos.

```
p { color: blue; }  
#titulo { font-size: 24px; }  
.destacado { font-weight: bold; }
```

Tipos de selectores:

- por **etiqueta** (p)
- por **clase** (.destacado)
- por **id** (#titulo)



## Propiedades básicas

Algunas de las más comunes:

```
body {  
  color: #333;  
  background-color: #f9f9f9;  
  font-family: Arial, sans-serif;  
  font-size: 16px;  
}
```

- color → color del texto
- background-color → fondo
- font-family → tipografía
- font-size → tamaño de texto

## Colores en CSS

Se pueden definir de varias formas:

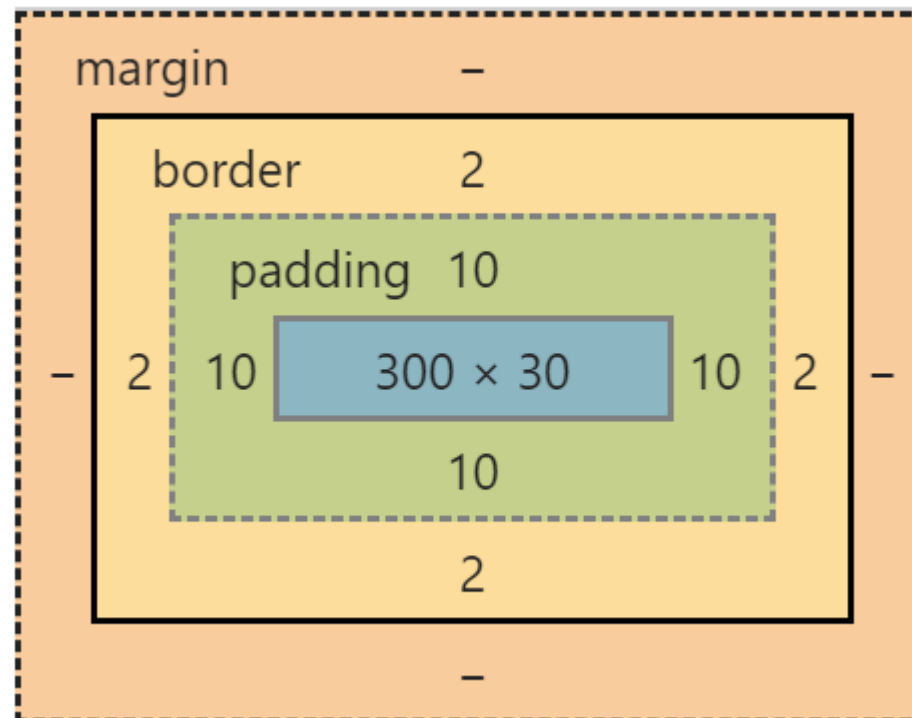
- Nombre: red, blue
- Hexadecimal: #ff0000
- RGB: rgb(255, 0, 0)
- RGBA (con transparencia): rgba(255, 0, 0, 0.5)

Ejemplo:

```
h1 { color: #007acc; }
```

## Modelo de Caja (Box Model)

Cada elemento HTML se comporta como una “caja”:

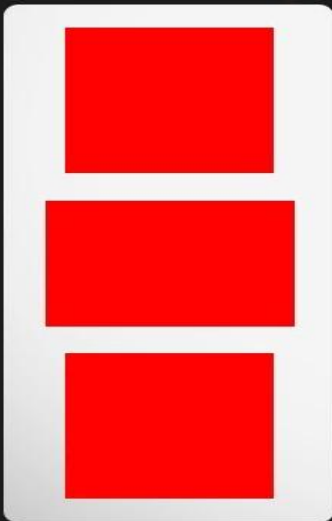


## Display y alineación

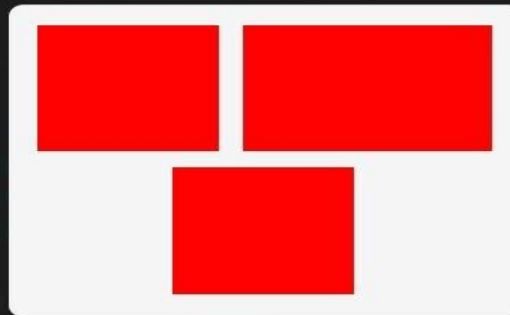
Propiedad display:

- block → ocupa todo el ancho
- inline → ocupa solo su contenido
- inline-block → mezcla ambos
- flex → distribuye y alinea elementos

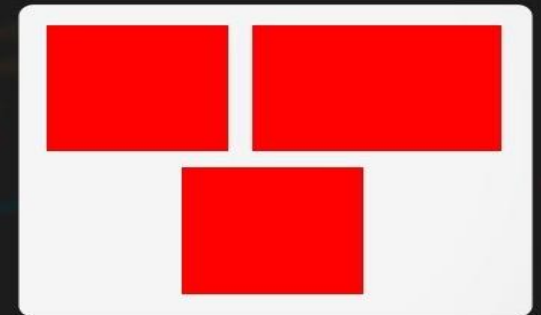
**block**



**inline**



**inline-block**



## Buenas prácticas CSS

- ✓ Separar estructura (HTML) de estilos (CSS)
- ✓ Usar clases en lugar de id para estilos repetidos
- ✓ Comentar el código
- ✓ Mantener consistencia en unidades (px, rem, %)
- ✓ Usar variables CSS (cuando corresponda)
- ✗ No usar estilos en línea
- ✗ No abusar de !important

## Desarrollo de Apps Web con React

### JavaScript ES6+

- Variables (let, const), tipos y operadores
- Funciones y arrow functions
- Arrays y objetos
- Desestructuración, spread/rest
- Manipulación del DOM
- Promesas, async/await y manejo de errores

## JavaScript ES6+

### ¿Qué es JavaScript?

- Lenguaje interpretado, multiplataforma.
- Se ejecuta en el navegador y en el servidor (Node.js).
- Controla la interactividad y la lógica de las páginas.

## Javascript - Tipos de Datos

Tipos primitivos:

- string → texto
- number → números
- boolean → verdadero o falso
- null → ausencia intencional de valor
- undefined → valor no asignado
- symbol, bigint (avanzados)

Tipos de referencia:

- Objetos, arrays y funciones
- JS: tipado dinámico / TS: tipado estático



## Ejemplos tipos de datos

JavaScript:

```
let nombre = "Nicolás";
```

```
let edad = 30;
```

```
let activo = true;
```

TypeScript:

```
let nombre: string = "Nicolás";
```

```
let edad: number = 30;
```

```
let activo: boolean = true;
```

## Operadores de comparación

`==` Igual (sin tipo)

`===` Igual (con tipo)

`!=` Distinto (sin tipo)

`!==` Distinto (con tipo)

`>`, `<`, `>=`, `<=` Comparaciones numéricas

Ejemplo:

```
console.log(5 == "5"); // true
```

```
console.log(5 === "5"); // false
```

## Null vs Undefined







undefined → variable declarada sin valor

null → ausencia intencional de valor

```
let telefono;  
console.log(telefono); // undefined
```

```
let direccion = null;  
console.log(direccion); // null
```

## Variables

Tipo	Reasignable	Alcance	Recomendado
<b>var</b>		Función/global	 No usar
<b>let</b>		Bloque	
<b>const</b>		Bloque	

## Estructuras de control

Condicionales:

if, else, else if

switch (casos múltiples)

Bucles:

for, while, do...while

## Ejemplo de estructuras de control

```
if (edad >= 18) {  
  console.log("Sos mayor de edad");  
} else {  
  console.log("Sos menor de edad");  
}
```

```
for (let i = 1; i <= 5; i++) {  
  console.log("Número: " + i);  
}
```

## Funciones en JavaScript

```
function saludar(nombre) {  
  console.log("Hola " + nombre + "!");  
}
```

```
function sumar(a, b) {  
  return a + b;  
}
```

Funciones flecha:

```
const multiplicar = (a, b) => a * b;
```

## Desarrollo de Apps Web con React

### Introducción a Vite Js






- ¿Qué es Vite y por qué reemplaza a Create React App?
- Instalación y configuración de Vite
- Creación de un proyecto base con React + TypeScript
- Estructura de carpetas y configuración inicial
- Ejecución, hot reload y build de producción
- Integración con VS Code y herramientas de desarrollo



## ¿Que es Vite?

- Herramienta moderna de desarrollo frontend.
- Creada por Evan You (autor de Vue.js).
- Se enfoca en velocidad, simplicidad y modularidad.
- Usa ES Modules en desarrollo y Rollup para producción.

## ¿Por qué usar vite?

-  Inicio instantáneo del servidor.
-  Recarga en caliente (HMR).
-  Configuración mínima.
-  Build optimizado con Rollup.
-  Soporte integrado para TypeScript, JSX y CSS.

## Plantillas en Vite

Vite ofrece varias plantillas (templates):

- React: react
- React + TS: react-ts
- Vue: vue / vue-ts
- Svelte: svelte / svelte-ts
- Vanilla JS: vanilla

## Instalación paso a paso

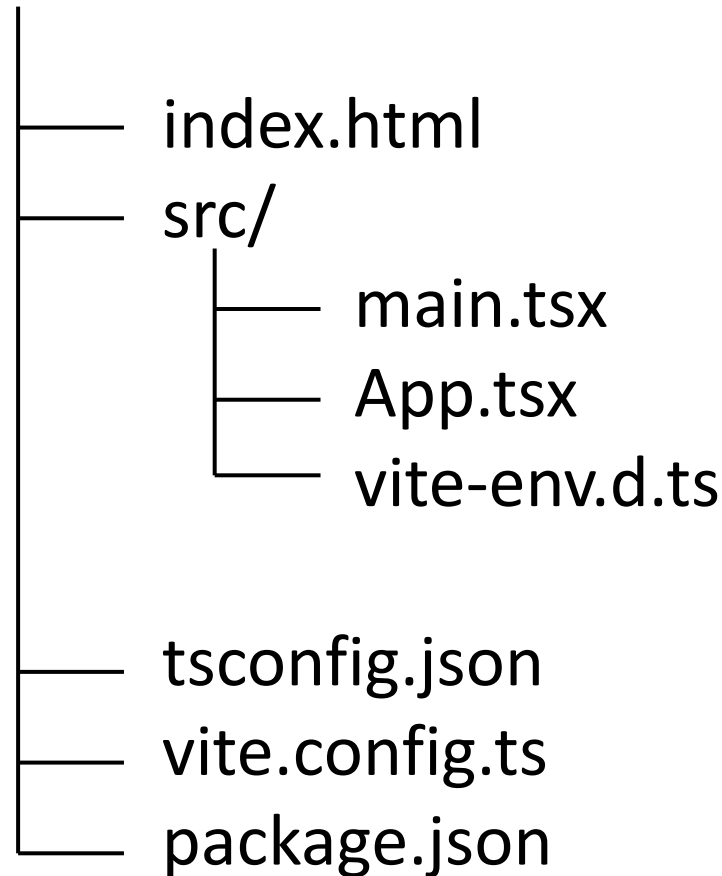
- `npm create vite@latest my-react-ts-app`
- Seleccionar React + TypeScript
- `cd my-react-ts-app`
- `npm install`
- `npm run dev`

# Estructura del proyecto

```
✓ 03-vite\my-react-ts-app
  > node_modules
  ✓ public
    vite.svg
  ✓ src
    > assets
      {} App.css
      {} App.tsx
      {} index.css
      {} main.tsx
    .gitignore
    eslint.config.js
    index.html
    package-lock.json
    package.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.node.json
    vite.config.ts
  README.md
```

# Estructura del proyecto

my-react-ts-app/



## Scripts importantes

- `npm run dev` → Inicia el servidor de desarrollo
- `npm run build` → Crea el build de producción
- `npm run preview` → Previsualiza el build localmente

## Ventajas Clave

- ⚡ Mayor velocidad y productividad
- 🧠 Soporte integrado de TypeScript
- 🔄 Recarga rápida (HMR)
- 🧱 Configuración limpia y modular
- 🌐 Comunidad activa



## Desarrollo de Apps Web con React

### Primeros pasos con React 19

- Introducción a la librería React y su ecosistema
- JSX/TSX: qué es y cómo funciona
- Componentes funcionales
- Props y estado (useState)
- Ejercicio práctico: crear un componente dinámico

## ¿Qué es React?

- Biblioteca de JavaScript para construir interfaces de usuario.
- Desarrollada por Facebook (Meta).
- Basada en componentes reutilizables.
- Utiliza JSX (JavaScript + HTML).

## Historia y Popularidad

- Lanzado en 2013.
- Usado por Facebook, Netflix, Instagram, Airbnb.
- Comunidad muy activa.
- Uno de los skills más demandados en el mercado.

## Componentes

- Piezas independientes y reutilizables de la interfaz.
- Definidos como funciones que retornan JSX.
- Cada componente puede tener su propio estado.

## JSX / TSX

- Permite escribir HTML dentro de JavaScript.
- Se convierte en llamadas a `React.createElement`.
- Más legible y expresivo.

Ejemplo:

```
function Hola() {  
  return <h1>¡Hola React!</h1>;  
}
```

## Virtual DOM

- Copia ligera del DOM real.
- Detecta los cambios y actualiza solo lo necesario.
- Mejora el rendimiento.

## Props

- Son datos que se envían desde un componente padre a uno hijo.
- Son inmutables (no se pueden modificar dentro del hijo).

Ejemplo:

```
function Saludo({ nombre }) {  
  return <h2>Hola {nombre}!</h2>;  
}
```

## Estado (State)

- Representa datos que pueden cambiar.
- React vuelve a renderizar el componente cuando el estado cambia.

Ejemplo:

```
const [contador, setContador] = useState(0);
```



## Hooks básicos

- `useState`: para manejar estado.
- `useEffect`: para ejecutar lógica al montar o actualizar un componente.
- Los hooks siempre comienzan con 'use'.

## Hooks personalizados

- Permiten crear lógica reutilizable a partir de otros hooks.
- Usan la convención 'use' al inicio del nombre (por ejemplo: useContador, useFetch).
- Ayudan a separar la lógica del negocio de los componentes.

Ejemplo:

```
function useContador(inicial = 0) {  
  const [contador, setContador] = useState(inicial);
```

## Ventajas de React

- Modularidad y reutilización.
- Gran rendimiento con Virtual DOM.
- Amplia comunidad y soporte.
- Compatible con TypeScript.
- Ecosistema enorme (React Router, Redux, etc.).

