Módulo 5
Desarrollo de aplicaciones Front-End con React

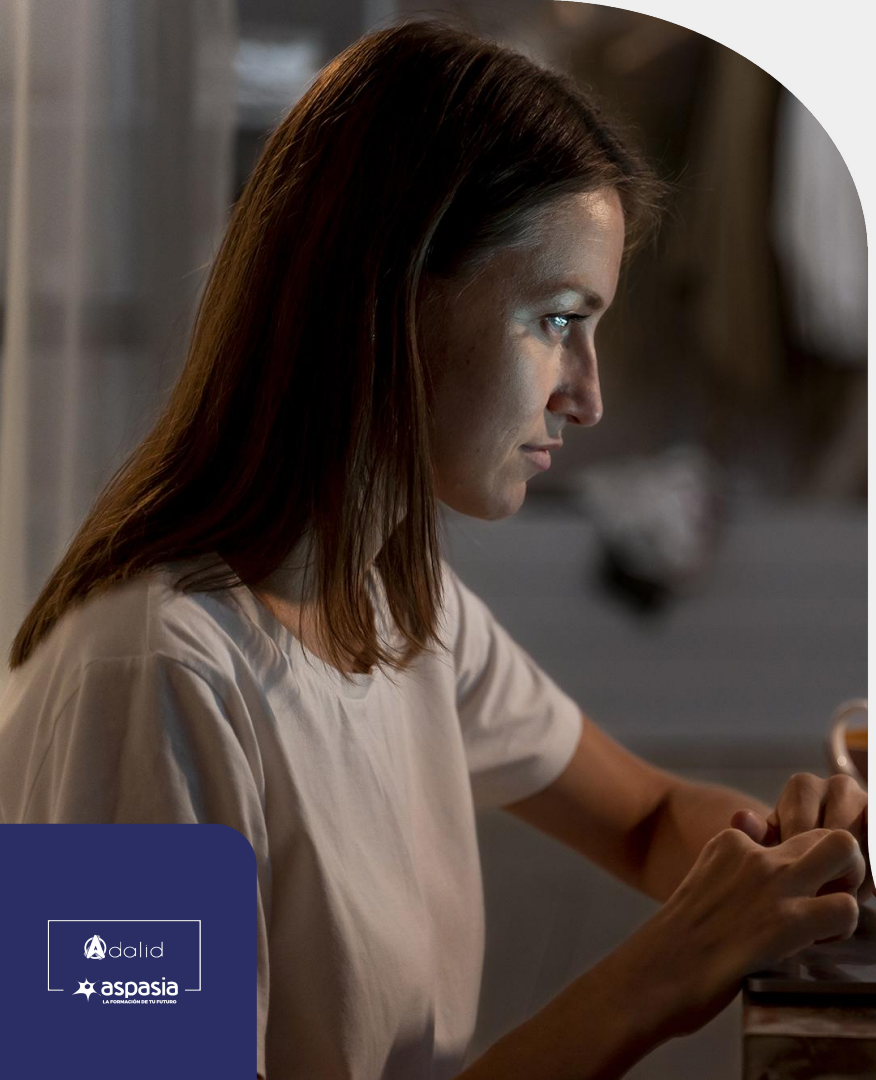
Introducción a TypeScript

Módulo 5

AE 2.2

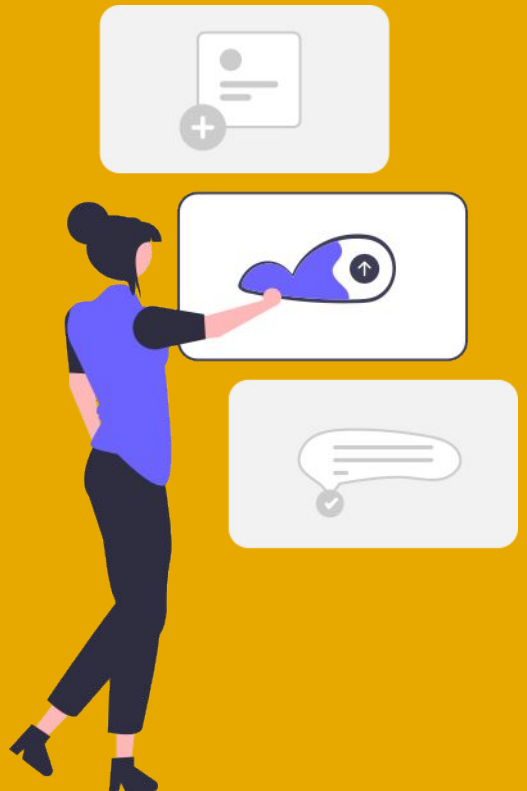
OBJETIVOS

Entender qué es TypeScript, cómo usarlo en proyectos React y su ventaja frente a JavaScript. Aprender a definir y componer tipos, usar interfaces y clases, e integrar TypeScript con frameworks como Next.js y Webpack.



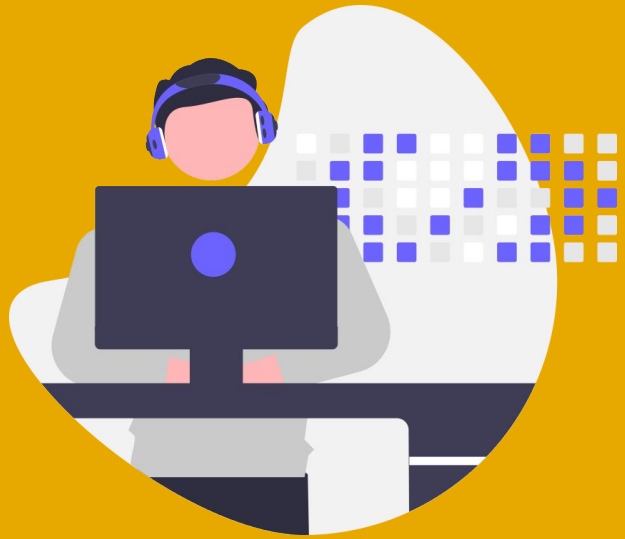
¿QUÉ VAMOS A VER?

- Introducción a TypeScript.
- Qué es TypeScript.
- Para qué se utiliza TypeScript.
- TypeScript en React.js.
- TypeScript vs. Javascript.
- TypeScript y Webpack.
- Definiendo tipos.
- Tipos por inferencia.
- Componiendo tipos.
- Sistema de tipo estructural.
- Interfaces y Clases.
- Algunos Frameworks que soportan TypeScript (Next.js, Gatsby.js).



TypeScript

Pongamos a prueba lo aprendido 😊!!!



Adalid aspasia

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Este ejercicio guiado te ayudará a **integrar TypeScript** en un proyecto React usando Vite, aprender a definir tipos, y explorar conceptos como inferencia, composición y el uso de interfaces y clases. Al final, se explicará cómo frameworks como Next.js y Gatsby.js aprovechan TypeScript.

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 1: Crear un Proyecto React con TypeScript

Crea el proyecto con Vite:

```
npm create vite@latest ts-react-app --template react-ts
```

Navega al directorio del proyecto e instala las dependencias:

```
cd ts-react-app
npm install
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 2: Creación del Primer Componente

Crea un componente llamado **Button** para explorar el uso de TypeScript. Ubicación: **src/components/Button.tsx.**

```tsx
import React from 'react';

interface ButtonProps {
  label: string;
  onClick: () => void;
}

const Button: React.FC<ButtonProps> = ({ label, onClick }) => {
  return <button onClick={onClick}>{label}</button>;
};

export default Button;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 2: Creación del Primer Componente

Reemplaza el contenido en **App.tsx** por:

```tsx
import React from 'react';
import Button from './components/Button';
import './App.css';

function App() {
  const handleClick = () => {
    alert('Botón presionado');
  };

  return (
    <div>
      <h1>Ejemplo de TypeScript en React</h1>
      <Button label="Haz clic aquí" onClick={handleClick} />
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 3: Utilidad para Funciones Matemáticas

Crea un archivo de utilidades con una función tipada, en la carpeta **src/utils/math.ts.**

```
export function multiplicar(a: number, b: number): number {
  return a * b;
}
```

Agrega la función en App.tsx:

```
import React from 'react';
import { multiplicar } from './utils/math';

function App() {
  const resultado = multiplicar(3, 4);

  return (
    <div>
      <h1>Resultado: {resultado}</h1>
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 3: Utilidad para Funciones Matemáticas

Crea un archivo de utilidades con una función tipada, en la carpeta **src/utils/math.ts.**

```
export function multiplicar(a: number, b: number): number {
  return a * b;
}
```

Agrega la función **en App.tsx**:

```
import { multiplicar } from './utils/math';

function App() {
  const resultado = multiplicar(3, 4);

  return (
    <div>
      <h1>Resultado: {resultado}</h1>
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 4: Componente con Tipos Compuestos

Crea un componente que utilice tipos compuestos para manejar diferentes formatos de entrada en **src/components/ProductCard.tsx**

```tsx
type ID = string | number;

interface Product {
  id: ID;
  name: string;
  price: number;
}

const ProductCard: React.FC<Product> = ({ id, name, price }) => {
  return (
    <div>
      <h3>{name}</h3>
      <p>ID: {id}</p>
      <p>Precio: ${price}</p>
    </div>
  );
};

export default ProductCard;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 4: Componente con Tipos Compuestos

Agrega la función **en App.tsx**:

```
import React from 'react';
import ProductCard from './components/ProductCard';

function App() {
  const producto = { id: 1, name: 'Laptop', price: 1200 };

  return (
    <div>
      <h1>Productos</h1>
      <ProductCard {...producto} />
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 5: Clases con TypeScript

Agrega una clase que implemente una interfaz. en **src/utils/ProductManager.ts**

```ts
interface Product {
  id: number;
  name: string;
  price: number;
}

class ProductManager {
  private products: Product[] = [];

  addProduct(product: Product): void {
    this.products.push(product);
  }

  getProducts(): Product[] {
    return this.products;
  }
}

export default ProductManager;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 5: Clases con TypeScript

Agrega la función **en App.tsx**:

```
import React, { useState } from 'react';
import ProductManager from './utils/ProductManager';

function App() {
  const manager = new ProductManager();
  manager.addProduct({ id: 1, name: 'Tablet', price: 500 });

  const productos = manager.getProducts();

  return (
    <div>
      <h1>Gestión de Productos</h1>
      {productos.map((producto) => (
        <div key={producto.id}>
          <h3>{producto.name}</h3>
          <p>Precio: ${producto.price}</p>
        </div>
      ))}
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Introducción a TypeScript en un Proyecto React

Paso 6: Exploración de Frameworks

En algunos proyectos, frameworks como **Next.js** y **Gatsby.js** ofrecen soporte completo para TypeScript.

Next.js:

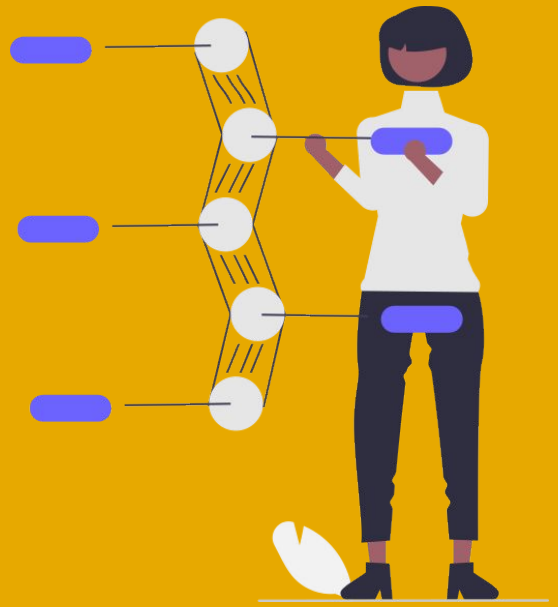
```
npx create-next-app my-next-app --typescript
```

Gatsby.js:

```
gatsby new my-gatsby-app
npm install typescript @types/react @types/node
```

Estos frameworks aprovechan **TypeScript** para mejorar la **experiencia del desarrollo** y la **escalabilidad** de los proyectos.

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- TypeScript mejora JavaScript añadiendo tipado estático, lo que reduce errores y facilita el desarrollo en proyectos complejos.
- En React, permite definir tipos para props, estados y funciones, mejorando la productividad y escalabilidad del código.
- Frameworks modernos como Next.js y Gatsby.js integran TypeScript para aplicaciones más robustas y mantenibles.
- Aprendiste a usar conceptos clave como definición de tipos, inferencia, interfaces y clases, además de integrar TypeScript con herramientas como Webpack.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO