educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 9

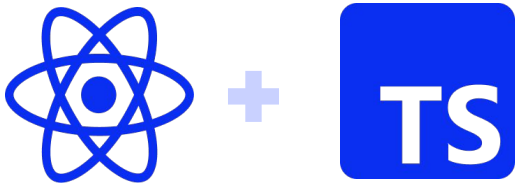
TypeScript React

educación IT
CODE YOUR FUTURE

TypeScript en ReactJS

En esta sección:

- **Aprenderemos a usar *TypeScript* con React**.
- Veremos **casos de uso comunes**.



Instalación (crear un nuevo proyecto)

1. Crea un nuevo proyecto con **`Create-React-App`** usando **la plantilla inicial para *TypeScript*** con el siguiente comando:

```
npx create-react-app <NOMBRE_APP_AQUI> --template typescript
```

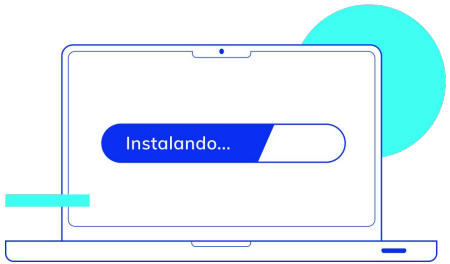
2. Una vez creado el proyecto, **navegamos hasta la carpeta creada**.
3. Ya tenemos el proyecto **configurado con *TypeScript***.
4. Podemos observar que **todos los archivos JS se han transformado en archivos TS**.
5. También tenemos archivos **TSX**, que tienen la misma funcionalidad que JSX pero con ***TypeScript* añadido**.

Instalación (crear un nuevo proyecto)

1. Crea un nuevo proyecto con **`Create-React-App`** usando **la plantilla inicial para *TypeScript*** con el siguiente comando:

```
npx create-react-app <NOMBRE_APP_AQUI> --template typescript
```

2. Una vez creado el proyecto, **navegamos hasta la carpeta creada**.
3. Ya tenemos el proyecto **configurado con *TypeScript***.



4. Podemos observar que todos **los archivos JS se han transformado en archivos TS.**

5. También **tenemos archivos TSX,** que tienen la **misma funcionalidad que JSX pero con TypeScript añadido.**

Instalación (agregar a un proyecto existente)

1. Toma un proyecto **ya existente**.
2. Instala los siguientes paquetes por **NPM:**

```
npm install --save typescript @types/node @types/react
@types/react-dom @types/jest
```

3. Si estabas usando el servidor de React (estabas corriendo *npm start*), **detén el servidor y vuelve a iniciarlo**.
4. Ya puedes usar *TypeScript* con React.

Instalación (agregar a un proyecto existente)

1. Toma un proyecto **ya existente**.
2. Instala los siguientes paquetes por **NPM:**

```
npm install --save typescript @types/node @types/react
@types/react-dom @types/jest
```

3. Si estabas usando el servidor de React (estabas corriendo *npm start*), **detén el servidor y vuelve a iniciarlo**.
4. Ya puedes usar *TypeScript* con React.

Casos de uso comunes

Con *TypeScript* podemos hacer que nuestros componentes React sean **más predecibles para quienes están desarrollando**. En otras palabras: podemos hacer que el código sea **fácilmente legible** y, en cierta forma, **autodocumentado**.

Al agregar *TypeScript*, contamos con una herramienta que permite definir con mayor precisión las diferentes estructuras y componentes que vayamos a usar. De esta forma, el código es más documentado y más fácilmente mantenible (sobre todo en grandes proyectos).

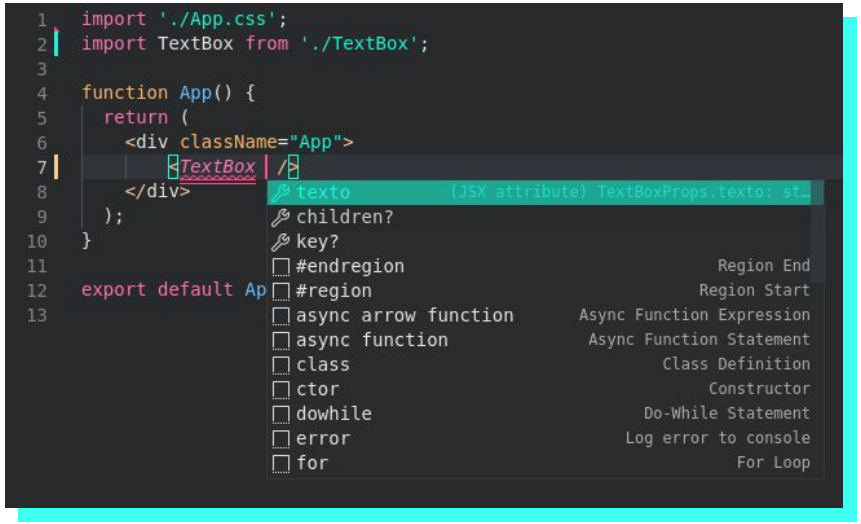
Conviene, además de exportar el componente, **exportar también sus *props* por si necesitamos usarlas en otro lado**.

Props predecibles

Podemos obligar a un componente a **recibir determinadas props y no otras**. Esto es muy útil en grandes aplicaciones donde **no hay tiempo como para ir a cada archivo de cada componente y ver las props que usa** ¿No sería mucho más sencillo tenerlas previamente documentadas y que el mismo editor te las sugiera?

Un caso especial de las `props`, como sabemos, es el `Children`. Se aconseja que el tipo de dato de la prop `children` sea **`React.ReactNode`** para contemplar mayores casos de uso.

Recuerda que un componente puede no tener hijos, por lo que `children` deberá ser una propiedad **opcional** (se marca con un **?** luego del nombre).




```tsx
import React from "react";

export interface TextBoxProps {
    children?: React.ReactNode;
    texto: string;
}

export default function TextBox(props: TextBoxProps) {
    return (
        <p>{props.texto}</p>
    );
}
```

Componentes funcionales predecibles

Así como especificamos las `props`, podemos, también, asegurarnos de que el componente **retorne un elemento `JSX`**. Esta verificación ayuda a que, si por error no lo hacemos, **el *Type Check* nos notifique**.

El tipo de dato de retorno es **`JSX.Element`**, ya que un componente React retorna un elemento JSX.

```
1  import React from "react";
2
3  export interface TextBoxProps {
4      children?: React.ReactNode;
5      texto: string;
6  }
7
8  export default function TextBox(props: TextBoxProps): JSX.Element {
9      return (
10         <p>{props.texto}</p>
11     );
12 }
```

Hooks predecibles

Podemos definir, también, **el tipo de dato que gestiona una variable de estado**. Esto se logra mediante **`useState`**. Este *hook*, al detectar *TypeScript*, **soporta una definición genérica**.

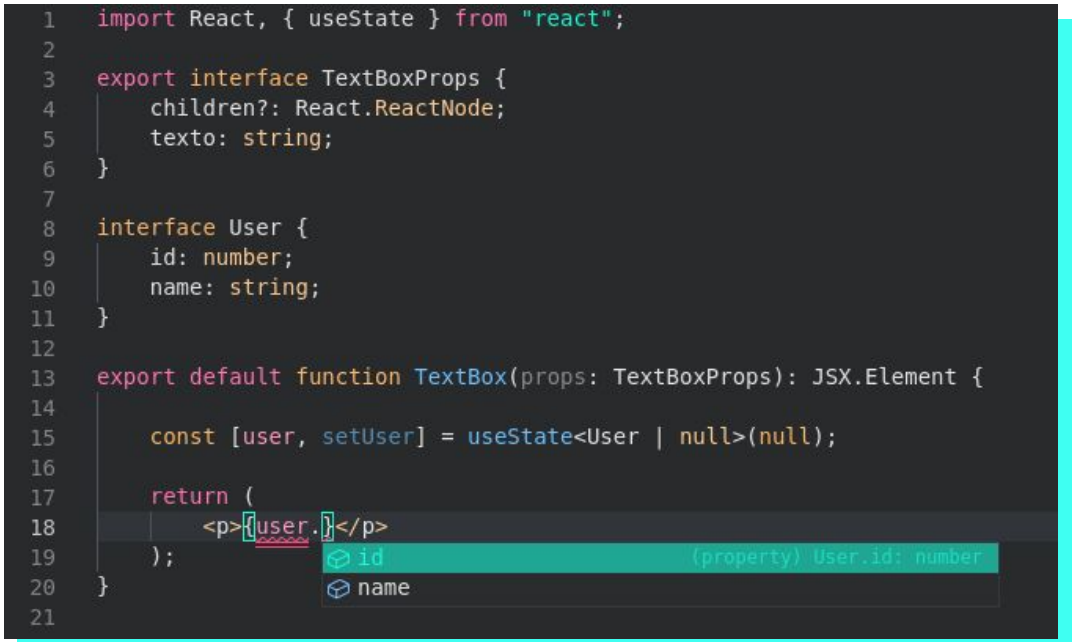
Si necesito contemplar el caso en que esta variable de estado no tenga ningún valor puedo usar el operador `|` para indicar que **puede tener un tipo de dato u otro**.

Por ejemplo, si pongo:

```
useState<number | null>(...)
```

significa que esa variable de estado puede ser del tipo `number` o puede estar vacía.

La misma lógica se puede aplicar, por ejemplo, a `React.createContext<>()`.


import React, { useState } from "react";
export interface TextBoxProps {
children?: React.ReactNode;
texto: string;
}
interface User {
id: number;
name: string;
}
export default function TextBox(props: TextBoxProps): JSX.Element {
const [user, setUser] = useState<User | null>(null);
return (
{user.}
);
}
id
(property) User.id: number
name

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!