**TypeScript**

**¿Qué es FrontEnd?**

Todo aquel código desarrollado y ejecutado desde el cliente (browser). El browser entiende solamente JS.

**¿Qué es BackEnd?**

Todo aquel código que es desarrollado y ejecutado desde el servidor. Posee un mundo sin fin de lenguajes para programar (Python, Java, C, etc).

✔ JavaScript es un lenguaje no tipado, es decir, se tiene la libertad de poder asignar cualquier tipo de valor a nuestras variables.

✔ TypeScript es una librería que extiende la funcionalidad de JavaScript Vanilla mediante la inclusión de types dentro de nuestro código.

**¿Qué es Typescript?**

TypeScript es un lenguaje fuertemente tipado creado por Microsoft, aunque está muy relacionado con el framework Angular, TypeScript es un lenguaje independiente que puedes usar para proyectos de BackEnd con Express o FrontEnd ya sea con Vue, React o Angular. Para que TypeScript pueda correr en el navegador debe ser transpilado a JavaScript con herramientas como Babel.

**Ventajas TS:**

Nos brinda mayor control al no tener que reutilizar variables mediante diferentes valores

Dentro de la gran cantidad de IDE’s para programar, encontraremos varias extensiones que nos van a brindar una gran comodidad a la hora de programar. (TSLint, ESLint)

**Desventajas de TS:**

JS nos da total libertad para poder incluir cualquier tipo de variable con diferentes valores. Estas variables luego pueden ser reutilizadas sin importar el tipo de esos valores.



A diferencia de JavaScript, TypeScript es un lenguaje fuertemente tipado. Mientras que en JavaScript declaramos una constante de la siguiente manera:

**const a = 'hola'**

En TypeScript utilizando el carácter ‘:’ le asignamos un tipo de dato a la variable:

**const a: string = 'hola'**

En caso de que no le asignemos un tipo de dato a la variable, TypeScript automáticamente le va a asignar un tipo de dato, esto es el Type Inference.

Si le asignamos el tipo de dato any a una variable, esta variable va a poder almacenar cualquier tipo de dato, similar a JavaScript Vainilla.

Dentro de TypeScript contamos con la palabra reservada type que nos va a ser de utilidad cuando hagamos nuestros propios tipos de datos.

**Clases**

Dentro de TypeScript podemos realizar programación orientada a objetos y como su nombre lo dice vamos a programar objetos, pero para generar dichos objetos primero debemos crear su plantilla base que recibe el nombre de clase.

Una clase está integrada por propiedades y funciones, estas pueden ser privadas, públicas o protected.

A la variable que le asignamos el valor de un objeto debemos indicarle que su tipo de dato es igual a la clase.

Una característica de la programación orientada a objetos es que podemos crear clases padre que hereden sus propiedades y funciones a una clase hijo, esto es la herencia y dentro de TypeScript se representa con la palabra extends.

**Interfaces**

Una interfaz es un tipo abstracto que sirve como contrato para la estructura de un objeto y al igual que las clases puede ser utilizada como un tipo de dato. Para declarar una interfaz en TypeScript utilizamos la palabra clave interface.

Dentro de las interfaces en TypeScript podemos manejar propiedades opcionales añadiendo el signo de pregunta ‘?’ al final del nombre de la propiedad.

interface Materia {

nombre: string,

calificacion: number,

descripcion?: string

}

const materia: Materia = {

nombre: 'Algebra',

calificacion: 18,

}

En donde la propiedad “descripción” puede quedar vacía porque es opcional con el símbolo “?”

**Shapes**

es una forma de comparar dos clases y saber si tienen las mimas propiedades o si una exíste una relacion hijo-padre entre ellas. Por ejemplo, un pediatra es un médico mas no necesariamente un médico debe ser un pediatra

**Union Types**

Habrá ocasiones en las que necesitemos asignar variables con más de un tipo, para evitar el uso de any debemos usar union types. Utilizando el signo ‘|’ podemos indicarle a TypeScript que utilice uno u otro tipo de dato, por ejemplo:

type SumarParameter = string | number;

type SumarReturnType = string | number;

function Suma(num1: SumarParameter, num2: SumarParameter): SumarReturnType {

    return Number(num1) + Number(num2);

    //return String(num1) + num2;

}

**Intersection Types**

La principal diferente entre la union y la intersección es que la unión es un valor "o" el otro, en cambio la intersección nos indica que un valor no puede estar indistinto del otro.

Se puede utilizar en cualquier cantidad de interfaces, agregando si se desea una Interface3 para ver que si procede el código.

Lo importante es que TypeScript nos indica cuando se da este tipo de error, y así poder corregirlo y llevar una mejor gestión del código.

interface Interface1 {

    prop1: number;

}

interface Interface2 {

    prop2: number;

    // prop3: number; Si se incluyera ésta propiedad, nuevamente daría error.

}

type InterfaceMix1 = Interface1 & Interface2;

const interfaceMix1: InterfaceMix = {

    prop1: 2,

    prop2: 2,

    //prop3: 2,

**Function Types**

Los Functions types permiten definir la estructura de nuestra función, como por ejemplo los argumentos y retornos de la misma.

type AreaRectangulo = (altura: number, base: number) => number;

const getAreaRectangulo: AreaRectangulo = (altura: number, base: number ): number => {

    return altura \* base;

}

**Decorators**

Los decorators son muy utilizados en Angular, estos son una declaración que tiene TypeScript para poder extender la funcionalidad de distintos elementos ya sea una clase, un parámetro, una propiedad o una función.

En una función decorator, el parámetro target hace referencia al objeto que posee el decorador y el parámetro propertyKey, o key, hace referencia al elemento que extendemos.

Formas de usar el Decorator:

* **Decorators en Métodos**
* **Decorators en Clases**
* **Decorators en Propiedades**
* **Decorators en Parametros**

**Angular**

**¿Qué es Angular?**

Mas que un Framework, es una Plataforma.

Desarrollado por Google, Angular es más que un framework, es una plataforma que nos da la posibilidad de desarrollar aplicaciones web como aplicaciones mobile. Además, es un framework de estructura que nos va a brindar funcionalidades para extender el template de nuestra aplicación.

Algunas ventajas que trae Angular son:

Rapidez.

Mayor estructura y control del proyecto.

SPA

Gran comunidad que ayuda con cualquier problema.

En detalle:

Podemos decir que es un Framework de Estructura. El cual es aquel que nos va a brindar ciertas funcionalidades que podemos agregar a nuestro template para extender y brindar nueva lógica a nuestra app.

Posee Directivas, como:

ng-if

ng-if-else

ng-for

Las cuales dan una nueva funcionalidad a nuestros tags de nuestro html.

**Angular CLI**

Angular CLI es la interfaz de línea de comandos de Angular con la cual desde una terminal puedes crear aplicaciones, generar componentes, montar un servidor local para tu aplicación y testear tu proyecto.

Creación de Proyecto

Para preparar nuestro entorno de trabajo lo primero que debemos hacer es instalar Node.js. Una vez tenemos instalado Node en nuestro computador, debemos instalar el CLI de Angular mediante el comando:

**npm install -g @angular/cli**

Ya que tenemos listo el CLI, creamos nuestro proyecto con el siguiente comando y contestando las preguntas de configuración que nos haga:

**ng new typescript-platzi**

Corremos la aplicación con el Comando

**ng serve**