**Curso de MEAN**

****

* **JavaScript // JSON**
* **Jquery**
* **TypeScript**
* **Angular**
* **Node.js**
* **Bongo.DB**

**Curso** MEAN.

**Sección** JavaScript Básico moderno.

Modo estricto. Para utilizar el modo estricto debemos poner en nuestro script.

**'use strict'**

Var & Let.

* var: Actúa de manera global

**var nombre = "Luis";**

**if(true) {**

**var nombre = "Enrique";**

**}**

**document.write(nombre);**

El resultado es que se imprimira ENRIQUE.

* **Let :** Actua a nivel de bloque.

**var nombre = "Luis";**

**if(true){**

**let nombre = "Enrique";**

**document.write(nombre);**

**}**

**document.write(nombre);**

El resultado al final sera LUIS y dentro del if sera ENRIQUE.

Constantes.

* **const :** para un valor que no puede ser cambiado.

**const ip = 28382929;**

Metodos.

|  |  |
| --- | --- |
| Metodo. | Descripción |
| varName.toString() | Convierte un numero a string |
| parseInt(varName) | Tranforma un String a entero o float (parseFloat) |
| Typeof varName | Nos devuelve el nombre del tipo de dato de la variable |
| isNaN(varName) | Nos dice si el valor de una variable no es un numero. |
| Metodo de texto. | Descripción |
| varName.toUpperCase() | Convierte un texto a mayusculas |
| varNameArray.length() | Nos indica el tamaño de un Array o de un texto. |
| varName.concat(" " + varName2) | Nos concatena dos textos previamente creados. |
| Metodo de busqueda. | Descripción |
| varName.search("word") | Retorna la posición del caracter donde se encuentra dicha palabra. |
| varName.includes("word") | Retorna true si se encuentra la palabra. |

Ventanas emergentes.

* **confirm("texto");** => Manda una ventana de confirmación al usuario.
* **prompt("dato");** => Manda una pregunta al usuario

**var confirmacion = confirm("¿Quieres que te preguntemos tu edad?");**

**if (confirmacion == true) {**

**var edad = prompt("Dinos cual es tu edad");**

**document.write("Tu edad es " + edad);**

**} else {**

**document.write("No quieres que te preguntemos");**

**}**

Parametros REST y SPREAD.

Estos estilos de parametros nos permiten guardar muchos valores en un arreglo.

* **Rest :** Nos pemite seguir guardando parametros en el final del arreglo.

**function frutas(fruta1, fruta2, ...resto){}**

* **Spread :** Nos permite meter un arreglo como en los parámetros.

**'use strict'**

**function frutas(fruta1, fruta2, ...resto) {**

**console.log("Fruta1 : ", fruta1);**

**console.log("Fruta2 : ", fruta2);**

**console.log("Lo demas es ", resto);**

**console.log("///////////////////////");**

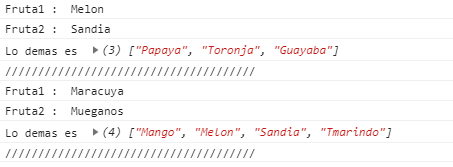
**}**

**frutas("Melon", "Sandia", "Papaya", "Toronja", "Guayaba");**

**var exoticas = ["Maracuya", "Mueganos", "Mango"];**

**frutas(...exoticas, "Melon", "Sandia", "Tmarindo");**

Resultado en consola...



Funciones anónimas & CALLBACKS.

**EJEMPLO DE FUNCION ANONIMA.**

* **FUNCION ANÓNIMA :** Es aquella que no tiene nombre y se guarda en una variable que puede ser invocada como una función.

**var nombre = function(minutos) {**

**document.write("Soy la funcion anonima y tengo " + minutos + " de haber sido creada <br><br>");**

**}**

**nombre(5);**

**EJEMPLO DE CALLBACK.**

* **CALLBACKS :** Es aquella funcion que se ejecuta en otra pero en el parametro por medio del uso de funciones anónimas.

**var num1 = parseInt(prompt("Dime el primer valor"));**

**var num2 = parseInt(prompt("Dime el segundo valor"));**

**function sumatoria(numero1, numero2, showSum, showMulTwo) {**

**//showSum y showMulTwo son funciones (CALLBACKS)**

**document.write("El primer valor es " + numero1);**

**document.write("<br> El segundo valor es " + numero2);**

**document.write("<br> El resultado de la suma de los dos valores es " + showSum);**

**document.write("<br> El producto es " + showMulTwo);**

**}**

**//Suma es una funcion ANONIMA**

**var suma = function(numero1, numero2) {**

**return numero1 + numero2;**

**}**

**//MUl es una funcion anonima**

**var mul = function(numero1, numero2) {**

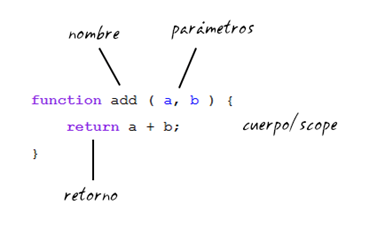
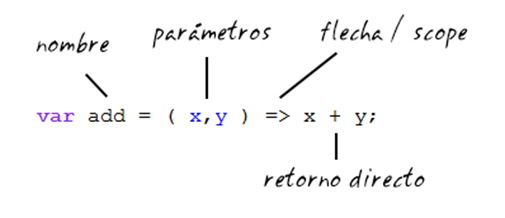
**return numero1 \* numero2;**

**}**

**sumatoria(num1, num2, suma(num1, num2), mul(num1, num2));**

Funciones de FLECHA.

Sirve para hacer más sencilla la estructura de la funciones una flecha.



Plantillas.

Por medio del uso de ${varName} y uso de comillas invertidas `` podemos hacer mucho más fácil la impresión de los textos.

**'use strict'**

**var nombre = prompt("Dame tu nombre");**

**var apellido = prompt("Dame tu apellido");**

**var texto = `**

**<h1>Bienvenido</h1>**

**<h3>Tu nombre es: ${nombre}</h3>**

**<h3>Tu apellido es: ${apellido}</h3>**

**`;**

**document.write(texto);**

Métodos para poder agregar elementos en los array.

|  |  |
| --- | --- |
| Metodo | Descripción |
| arrayName.push(); | Agrega un elemento a nuestro array. |
| arrayName.pop(); | Elimina el ultimo elemento de un array |
| varName.split(); | Convierte el texto a un array.  **var texto = "texto1, texto2, texto3";**  **var nuevoArray = texto.split(", ");** |
| arrayName.sort(); | Ordena los elementos de un arreglo de manera alfabetica. |
| arrayName.reverse(); | Ordena pero de manera inversa. |

FOR IN (EXAMPLE)

**'use strict'**

**var newArrayLengua = ["HTML", "JNFSNSF", "JBJBFJN", "NDJN"];**

**for (let lenguaje in newArrayLengua) {**

**document.write("<br> \* " + newArrayLengua[lenguaje]);**

**}**

Metodos para busqueda de elementos en un array.

Podemos utilizar .find o .findIndex para encontrar elementos en nuestros arreglos.

**var lenguajes = ["PHP", "JS", "HTML", "Perl", "R", "F#", "C#"];**

**var busqueda = lenguajes.findIndex((elemento) => elemento = "PHP");**

**document.write(busqueda);**

Parametros por defecto.

En la parte de nombre el valor por defecto es “No especificado”

**function usuario(nombre = “No especificado”, edad){**

**}**

**usuario(undefined, 12);**

Destructurar arreglos.

Lo que nos va a dar es cada valor del arreglo en una variable // lo que estamos haciendo es generar nuevas variables.

**let persona = [“Luis”, 34, “Chavarria”, “Desarrollador”];**

**let [nombre, edad, apellido, profesión = “No definida”] = persona;**

**console.log(persona);**

Lo que nos va a dar en esta sección es solo los tres valores que le estamos indicando a excepción del que dejamos en blanco.

**let persona = [“Luis”, 12, “Chavarria”, “Desarrollador”];**

**let [nombre, , país, profesión];**

**console.log(persona);**

Destructurar objetos.

En el ejemplo lo que estamos haciendo es desectructurar un objeto en pequeñas variables.

**var usuario = {**

**nombre: “Luis”,**

**apellido: “Chavarria”,**

**correo: “ldkdnkn@gmail.com”**

**}**

**var {nombre = “No especificado”, correo} = usuario;**

**console.log(nombre);**

Promesas.

**const promesa = new Promise((resolve, reject) =>{**

**si la llamada AJAX es buena lo que ejecutaremos es resolve con .then // resolve();**

**si la llamada AJAS es falla ejecutamos reject con .catch // reject();**

**});**

**promesa.then(()=>{**

**alert(“Exito”);**

**});**

**promesa.catch(()=>{**

**alert(“Fallo”);**

**});**

**Curso** MEAN.

**Sección** DOM, BOM y eventos.

DOM.

Por medio del uso del DOM podemos acceder a cualquier tipo de elemento o clase que tengamos en nuestros documentos HTML.

* 1. Tip : como document.getElementById // ByClass // son muy viejos lo más recomendable es usar

**// varName = document.querySelector( "#idName" or ".className");**

* 1. Tip: para poder cambiar la clase de un element lo que tenemos que usar es:

**//varName.className = “className”;**

* 1. Tip: para poder modificar y acceder al contenido (texto) de cualquier elemento lo único que necesitamos hacer es usar

**//varName.innerHTML**

* 1. Tip: para poder modificar cualquier propiedad de cualquier objeto.

**//varName.style.cssProperty = “xpx o valor”;**

* 1. Tip: para poder seleccionar muchas clases al mismo tiempo requerimos el uso de:
     + **varName.document.getElementById**
     + **varName.document.getElementByTagName**

Se irán almacenando en un array.

* 1. Tip: Crear nuevos elementos lo único que debemos hacer es lo siguiente.

var padre = document.**getElementByTagName**(“section”);

var párrafo = document.**createNewElement**(“p”);

var texto = document.**createTextNode**(“Este es el nuevo texto”);

párrafo.**append** (texto);

padre.**append**(parrafo);

* **document.getElementByTagName //** nos permite el acceso al contenedor principal.
* **varName.createNewElement //** nos ayuda a crear el nuevo párrafo.
* **varName.createTextNode //** nos permite crear el texto que mostraremos en el párrafo.
* **varName.append //** nos da la posibilidad de poder meter en un contenedor padre un contenedor hijo.

BOM.

**\* varName = window.innerHeight;** // nos dira la altura de nuestra ventana

**\* varName = window.innerWidth;** // nos dira el ancho de nuestra ventana

**\* varName = screen.height;** // nos dira el alto de nuestra pantalla

**\* varName = screen.width;** // nos dira el ancho de toda la pantalla

Eventos.

Los eventos en la actualidad se ejecutan de la siguiente manera.

**var cambiaColor = () => {**

**var boton = document.getElementById(“boton”);**

**boton.style.background = “green”;**

**if(boton.style.background = “green”){**

**boton.style.background = “red”;**

**}else{**

**Boton.style.background = “green”;**

**}**

**}**

**boton.addEventListener(“click // mouseover // mouseout //”, function(){**

**cambiaColor();**

**});**

En la parte subrayada se aprecia como con **varName.addEventListener** podemos hacer que una acción se ejecute sobre el botón.

* En el caso de los inputs de tipo texto, requerimos usar el **focus (para cuando entra)** / **blur (para cuando** sale)

Timers. Con ellos podemos hacer que nuestro codigo se ejecute en un determinado tiempo.

**var intervalo = setInterval(function() {**

**var numero = "|";**

**document.write(numero);**

**}, 1000);**

Tips

**TIP**: para accede a cada valor en un formulario lo que necesitamos es poder entrar con …

**varName.value()**

Operador this.= This literalmente hace referencia a algun elemento antes mencionado.

**boton.addEventListener(“click”, function(){**

**boton = this**

**cambiaColor();**

**this.style.padding = “20px”;**

**});**

JSON. = Con JSON, podemos generar diferentes tipos de objetos (POO) para acceder a su información.

var primeraPeli = {

Aquí creamos nuestros objetos que almacenan películas.

titulo: "Batman vs Superman",

year: 2009,

pais: "Estados Unidos"

}

var segundaPeli = {

titulo: "La verdad duele",

year: 2016,

pais: "Francia"}

Almacenamos en un nuevo objeto los 2 anteriores

var peliculas = {

primeraPeli,

Imprimimos los títulos de nuestras pelis con un FOR IN.

Accedemos con OBJECT.property

segundaPeli

}

for (var index in peliculas) {

document.write(peliculas[index].titulo + " <br>");

}

Local Storage. Nos sirve para almacenar datos en la memoria del local Storage en la sección de aplications de nuestra consola.

En esta sección básicamente estamos creando el elemento LUIS y lo estamos almacenando en la memoria.

Con **localStorage.setItem(“Element”,value);** Y lo obtenemos con **getItem**

//Insertar un solo valor y recuperarlo.

localStorage.setItem("nombre", "Luis");

var nombre = localStorage.getItem("nombre");

document.write("Bienvenido " + nombre);

//Almacenar un objeto, convertirlo a string y borrarlo

En esta sección lo que estamos haciendo es poder **almacenar el objeto** en nuestra sección de Aplications y convertirlo en STRING el cual luego es convertido en objeto con JSON.parse() y ya podemos acceder como si fuera un objeto.

var usuario = {

nombre: "Luis Enrique",

apellido: "Chavarria Vazquez",

edad: "55"

}

localStorage.setItem("usuario", JSON.stringify(usuario));

var userObject = JSON.parse(localStorage.getItem(“usuario”));

localStorage.removeItem(“usuario”);

localStorage.clear();

Con **localStorage.removeItem** removemos un elemento del local storage y con **clear()** borramos todo de la memoria.

Fecha.

**var fecha = new Date();**

En esta parte lo que estamos hacienda es poder **acceder al objeto de la fecha**

**var year = fecha.getFullYear();**

**var mes = fecha.getMonth();**

**var dia = fecha.getDay();**

**var minutos = fecha.getMinutes();**

En todas las demás partes lo que estamos haciendo es **acceder a los elementos de DATE**

**var segundos = fecha.getSeconds();**

**Curso** MEAN.

**Sección** Jquery.

Selectores.

|  |  |
| --- | --- |
| Selectores. | Descripción |
| $(“#id”).method(); | Selecciona una id |
| $(“.class”).method(); | Selecciona una clase |
| $(“p”).method(); | Selecciona una etiqueta |

Métodos Clases.

|  |  |
| --- | --- |
| Selectores. | Descripción |
| $(“#id”).addClass(‘’); | Agrega una clase. |
| $(“#id”).removeClass(“”); | Remueve una clase. |
| $(“p”).hasClass(“”); | Comprueba si el nombre de dicha clase esta presente. |
| $(“p”).css({  “fontSize”: “30px”;  }); | Con esta propiedad lo que hacemos es que nuestros elementos adquieran características de tipo CSS3 |

Eventos: Los tenemos que llamar por medio del CALLBACK.

**caja.mouseover(function() {**

**$(this).css({ "background": "red" });**

**});**

|  |  |
| --- | --- |
| Eventos CALLBACK. | Descripción |
| varName.mouseOver(); | Cuando hacemos hover actua. |
| varName.mouseOut(); | Cuando salimos del hover actua. |
| VarName.click(); | Cuando demos un click se va a ejecutar. |
| varName.dblclick(); | Cuando demos doble click se ejecutara. |
|  |  |

|  |  |
| --- | --- |
| Eventos CALLBACK. | Descripción |
| $(“elementos”).show(); | Muestra un elemento oculto con hide(); |
| $(“elementos”).hide(); | Muestra un elemento mostrado con show(); |

Trucos JQUERY.

Cambiador de color personalizado.

<link id="theme" rel="stylesheet" type="text/css" href="css/green.css" />

// Selector de tema

Accediendo a la etiqueta LINK

var theme = $("#theme");

$("#to-green").click(function(){

En esta sección lo que hacemos es poder **intercambiar los archivos CSS previamente creados.**

theme.attr("href", "css/green.css");

});

$("#to-red").click(function(){

theme.attr("href", "css/red.css");

});

$("#to-blue").click(function(){

theme.attr("href", "css/blue.css");

});

Login falso.

**$("#login form").submit(function(){**

Recien enviado el elemento de nuestro formulario estamos accediendo al valor que se tenia en el input de nombre y almacenándolo en el local STORAGE.

**var form\_name = $("#form\_name").val();**

**localStorage.setItem("form\_name", form\_name);**

**});**

var form\_name = localStorage.getItem("form\_name");

Si el nombre esta bien, lo que estamos haciendo es mostrarlo en los párrafo y luego con la DOM lo imprimimos en la pantallas.

if(form\_name != null && form\_name != "undefined"){

var about\_parrafo = $("#about p");

about\_parrafo.html("<br><strong>Bienvenido, "+form\_name+"</strong> ");

about\_parrafo.append("<a href='#' id='logout'>Cerrar sesión</a>");

$("#login").hide();

El formulario se oculta una ves enviado.

$("#logout").click(function(){

localStorage.clear();

Cuando hace LOGOUT entonces la variables se limpian.

location.reload();

});

}

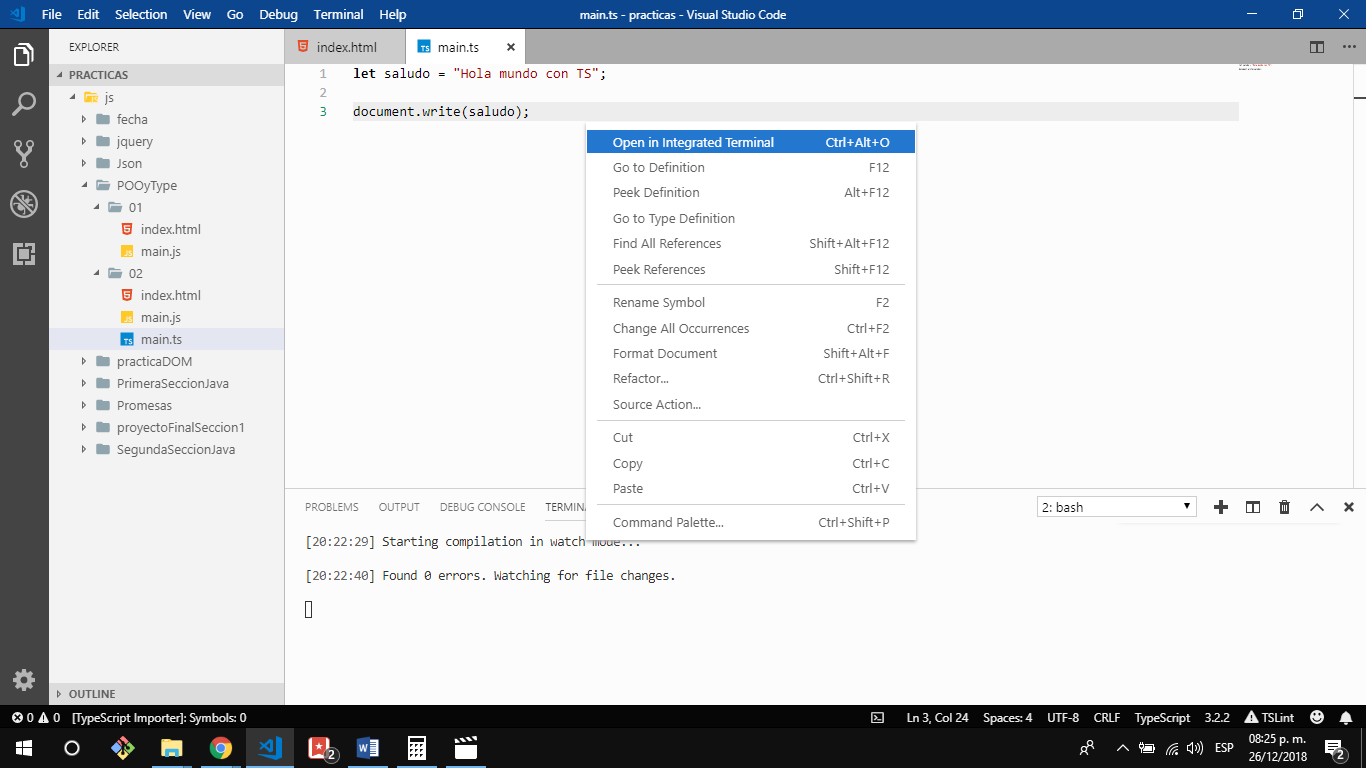
**Curso** MEAN.

**Sección** TypeScript.

Compilación.

Debemos instalar primero NODE.JS y luego proceder a instalar en nuestra computadora TypeScript. Para compilar debemos usar **tsc –w \*.ts**

**Ejemplo:**



Tipado.

TypeScript tiene la característica de ser un lenguaje con tipado muy fuerte como en lenguaje de programación como JAVA o C.

**var nombre: string = “Luis Enrique”;**

**var numero: number = 34;**

**var boolean: boolean = false;**

**var any: any = 8284;**

**var numberString: number | string = “Luis”;**

**var array: Array<any> = [“Luis”, “Enrique”];**

POO en TypeScript.

Clases. Podrian considerarse como los moldes sobre los cuales crearemos nuestros objetos.

Metodos y atributos son el equivalente a las funciones y variables de nuestros objetos de forma respectiva.

**Las clases** son como los **moldes** en los cuales crearemos nuevos objetos.

**class Camiseta = {**

**public color: string;**

**public marca: string;**

**Las propiedades** variables internas de los objetos.

**public getColor(color){**

**Los métodos** nos ayudan a ejecutar funciones dentro de objetos.

**this.color = color;**

**}**

**}**

**var playera = new Camiseta();**

**Aquí ya hemos creado nuestro objeto**

**playera.color = “red”;**

**playera.getColor(“blue”);**

**playera.marca = “Nike”;**

* **Public.** Es accesible desde cualquier parte del codigo incluida cualquier tipo de clase.
* **Private.** Podemos acceder solo dentro de la clase y otras clases que estén relacionadas.
* **Protected.** Solo dentro de la misma clase.

Función constructora.

**class Tenis{**

**Función constructora.**

**let marca: string;**

**let precio: number;**

**constructor(){**

**this.marca = marca;**

**this.precio = precio;**

**}**

**Le pasamos los atributos a nuestro nuevo objeto.**

**}**

**var nike = new Tenis(“Nike”, 342);**

Herencia. Por medio de la herencia podemos pasarle todas las características de nuestro objeto anterior.

**class Camiseta{**

**public color: string;**

**Tenemos todas las características de nuestro objeto camiseta en el que estamos usando el método constructor.**

**public talla: string;**

**public precio: number;**

**constructor(color, talla, precio){**

**this.color = color;**

**this.talla = talla;**

**this.precio = precio;**

**Aquí heredamos los valores de nuestra clase padre.**

**}**

**}**

**class Sudadera extends Camiseta{**

**public capucha: boolean;**

**Lo que estamos haciendo es que cuando llamamos a este método nos retorna el valor de capucha.**

**public getCapucha(verdad):boolean{**

**this.capucha = verdad;**

**return this.capucha;**

**}**

**}**

**Le estamos pasando los atributos y estamos llamando a capucha.**

**var Sudadera\_nike = new Sudadera("red", "Small", 342);**

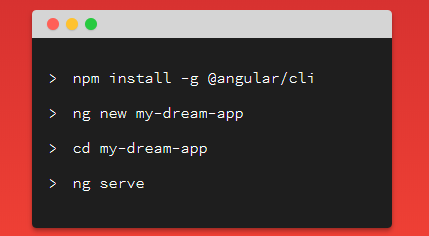
**Sudadera\_nike.getCapucha(true);**

**console.log(Sudadera\_nike);**

**Curso** Angular.

**Sección** Creación de aplicaciones en angular.

Instalación. En la página de **Angular CLI** encontramos todos los recursos necesarios.



Para mayor eficiencia en proceso de compilación es recomendable su ejecución en el símbolo del sistema.

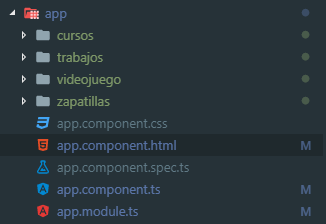
Una vez aplicado el comando **ng serve** requerimos entrar en **localHost:4200** para ver nuesta aplicación.

Componentes. Son los elementos que serán mostrados en nuestra aplicación.

* Para generar un nuevo componente en nuestra aplicación requerimos poner lo siguiente en nuestra consola.

**ng generate component nameComponent**

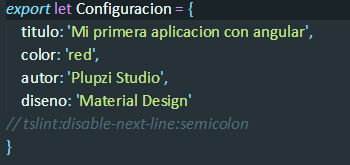
Una ves generado el componente tendremos que enlazarlo con nuestro componente en la aplicación principal.



**Debemos llamar nuestros componentes como si de etiquetas se tratara.**

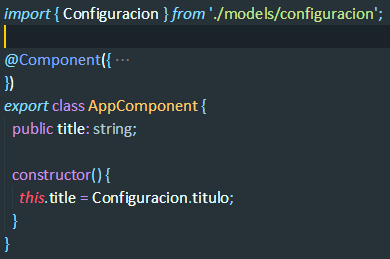
**Ex. <trabajos></trabajos>**

**Estos son nuestros componentes.**

Modelos. Son configuración o datos que guardamos del usuario y que después podemos emplear.



**En nuestro archivo de configuración lo que tenemos es un objeto exportado**



**Después estamos importando nuestra configuración**

**Accediendo a las variables de nuestra configuración para usarlas en el HTML con {{ title }}**

Directivas y eventos. Las directivas nos ayudan a llevar a cabo procesos de manera rápida como…

* **\*ngIf** //Es un if
* **\*ngFor** //Es un for
* **[(ngModel)]="input\_dato"** //almacena los datos de un input en la variable input\_dato.
  + **Para usarlo requerimos importarlo de la siguiente manera en app.module.ts**

**import { FormsModule } from ‘@angular/forms’;**

**imports:[**

**FormsModule**

**]**

**EX:**

**<input name=”user” [(ngModel)]=”user”></input>**

**<p>{{user}}</p>**

* **(click)=functionName();** //nos permite ejecutar alguna función cuando hacemos click.
* **(submit)=functionName(input.value);** //Donde nuestro codigo en TS debe tener una función que nos retorne false.
  + **Ex**

**functionName(inputValue){**

**return false;**

**}**

* **Almacenaje de inputs //**Nos guarda todos los atributos de input y para acceder a su texto lo único que debemos poner es **newInput.value**
  + **Ex**

**Almacenamos nuestro input** completo y lo pasamos a nuestra función.

**<form (submit)=addDato(newInput)>**

**<input type=”text” #newInput>**

**<button>Enviar</button>**

**</form>**

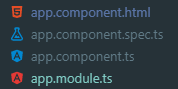
**functionName(newInput){**

**let valor = newInput.value;**

**return false;**

**}**

Configuración de router. Nos sirve para generar rutas…



* Debemos colocar en esta zona

**Estamos llamando al enroutador de angular.**

**import { RouterModule, Route} from '@angular/router';**

**const routes: Route[] = [**

**{path: '', component: DatosUsuarioComponent}, //Index**

**{path: 'about', component: AboutComponent}**

**];**

**Colocamos los componenetes a vincular asignadoles una ruta y el componente.**

**...**

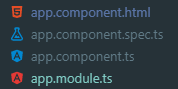
**imports: [**

**BrowserModule,**

**Estamos colocando el INDEX en esta zona,**

**RouterModule.forRoot(routes)**

**], ...**



<a routerLink="/">Home</a> <br>

<a routerLink="/about">About</a>

<router-outlet></router-outlet>