

Clase 1:

instalacion de node JS

instalacion de NPM , ver la version

instalacion de TS : <https://www.typescriptlang.org/>

tsc -v : nos indica la version de ts

`npm install -g typescript`

para compilar un archivo ts utilizamos : `tsc nombreArchivo.ts`

instalamos VisualStudioCode : <https://code.visualstudio.com/>

de la pagina de VSCode debemos bajar la extensions VSCode-Icons

Instalamos Angular CLI : cli.angular.io (esta herramienta es muy util y facilita el trabajo)

`ng -v` vemos a version que instalo

Que es TypeScript?

ECMAScript v6

Primero un poco de historia. En 1995 (hace 20 años!) [Brendan Eich](#) crea un lenguaje llamado **Mocha** cuando trabajaba en **Netscape**. En Septiembre de ese año lo renombra a *LiveScript* hasta que le cambiaron el nombre a *JavaScript* debido a una estrategia de marketing, ya que Netscape fue adquirida por *Sun Microsystems*, propietaria del lenguaje **Java**, muy popular por aquel entonces.

En 1997 se crea un comité (TC39) para estandarizar JavaScript por la *European Computer Manufacturers' Association*, ECMA. Se diseña el estándar del DOM (*Document Object Model*) para evitar incompatibilidades entre navegadores. A partir de entonces los estándares de JavaScript se rigen por ECMAScript.

En 1999 aparece la 3a versión del estándar ECMAScript, que se mantendría vigente hasta hace pocos años. Hubo pequeños intentos de escribir la versión 4, pero hasta 2011 no se aprobó y se estandarizó la versión 5 (ES5) que es la que usamos hoy en día.

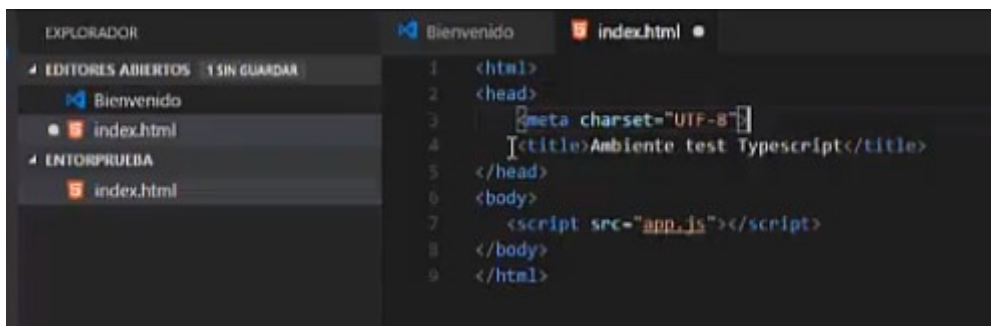
En junio de 2013 quedó parado el borrador de la versión 6, pero en Diciembre de 2014 se aprobó al fin y se espera su estandarización a partir de Junio de 2015.

Que es typescript ?

- TypeScript es un lenguaje que nos permite tener una orientación a objetos más limpia y potente en JavaScript, además añade un tipado fuerte y es el lenguaje que utilizamos para programar aplicaciones web con Angular que es uno de los frameworks más populares para desarrollar aplicaciones modernas y escalables en el lado del cliente.
- TypeScript es un lenguaje libre desarrollado por Microsoft y es un superconjunto de JavaScript que gracias a las ventajas que ofrece está siendo cada vez más utilizado.
- TypeScript es un lenguaje “compilado”, es decir, nosotros escribimos código TypeScript y el compilador (transpilador) lo traduce a código JavaScript que el navegador podrá interpretar.

mi primer hola mundo con typescript

- a.creamos una carpeta
- b. seleccionamos la carpeta y la abrimos con VSCode
- c.creamos un index.html con la estructura simple :



```
1 <html>
2 <head>
3   <meta charset="UTF-8">
4   <title>Ambiente test Typescript</title>
5 </head>
6 <body>
7   <script src="app.js"></script>
8 </body>
9 </html>
```

- d.creo el archivo app.ts con la linea : `console.log('hola mundo ');`
- e.abrimos la consola y ejecutamos : `tsc app.ts`
- f.miramos la ejecucion del index.html y mostrar el archivo traspilado
- g.agregamos un : `alert('pasamos por aqui');` en el app.ts y volvemos a compilar y ejecutar el TS
- h. `tsc -w*.ts` este comando activa un escuchador o autocompilador

tsconfig.json

Este archivo indica que es un proyecto TS y se genera ejecutando el comando “tsc -init” esto generaria el archivo de conf por defecto para el proyecto con algunos datos como por ej:

targer : indica la version de ECMAScript

module : inidica el modulo de generacion utilizado para trabajar en el prpoyecto

y algunas configuracion especiales de lo que es el proyecto

LET VAR Y CONST

var : es para globales

let : variables locales al bloque

Let / var / Const

En javascript hay dos formas de declarar variables: `var` y `let`, `var` no tiene un ámbito de bloque mientras que `let` sí.

`var`

```
var foo = 123;
if (true) {
  var foo = 456;
}
console.log(foo); // 456
```

`let`

```
let foo = 123;
if (true) {
  let foo = 456;
}
console.log(foo); // 123
```

Constantes

Let / var / Const

Ha sido añadido en ES6 / TypeScript permitiendonos añadir variables inmutables también conocidas como constantes. El uso de `const` es una buena práctica de mantenimiento y legibilidad. **Las constantes deben ser declaradas y asignadas siempre.**

```
const foo = 123;  
foo = 456; // NO permitido
```

Las constantes también admiten objetos literales como por ejemplo:

```
const foo = { bar: 123 };  
foo = { bar: 456 }; // ERROR no se permite la modificación de objeto
```

Pero si se puede modificar el contenido de las variables que contiene el objeto literal, ejemplo:

```
const foo = { bar: 123 };  
foo.bar = 456; // Permitido  
console.log(foo); // { bar: 456 }
```

Tipo de Datos Primitivos

Boolean

true o false

```
let isDone: boolean = false;
```

Number

Datos numéricos

```
let decimal: number = 6;  
  
let hex: number = 0xf00d;  
  
let binary: number = 0b1010;  
  
let octal: number = 0o744
```

Tipo de Datos Primitivos

String

Cadenas de caracteres y/o textos

```
let color: string = "blue"; //  
color = 'red';
```

También se pueden utilizar *"Templates"* plantillas para concatenar strings como por ejemplo:

```
let fullName: string = `Bob Bobbington`;   
let age: number = 37;   
let sentence: string = `Hello, my name is ${ fullName }. I'll be ${ age + 1 } years old next month.`
```

Para poder utilizar esta sintaxis los string deben estar contenidos entre ```.

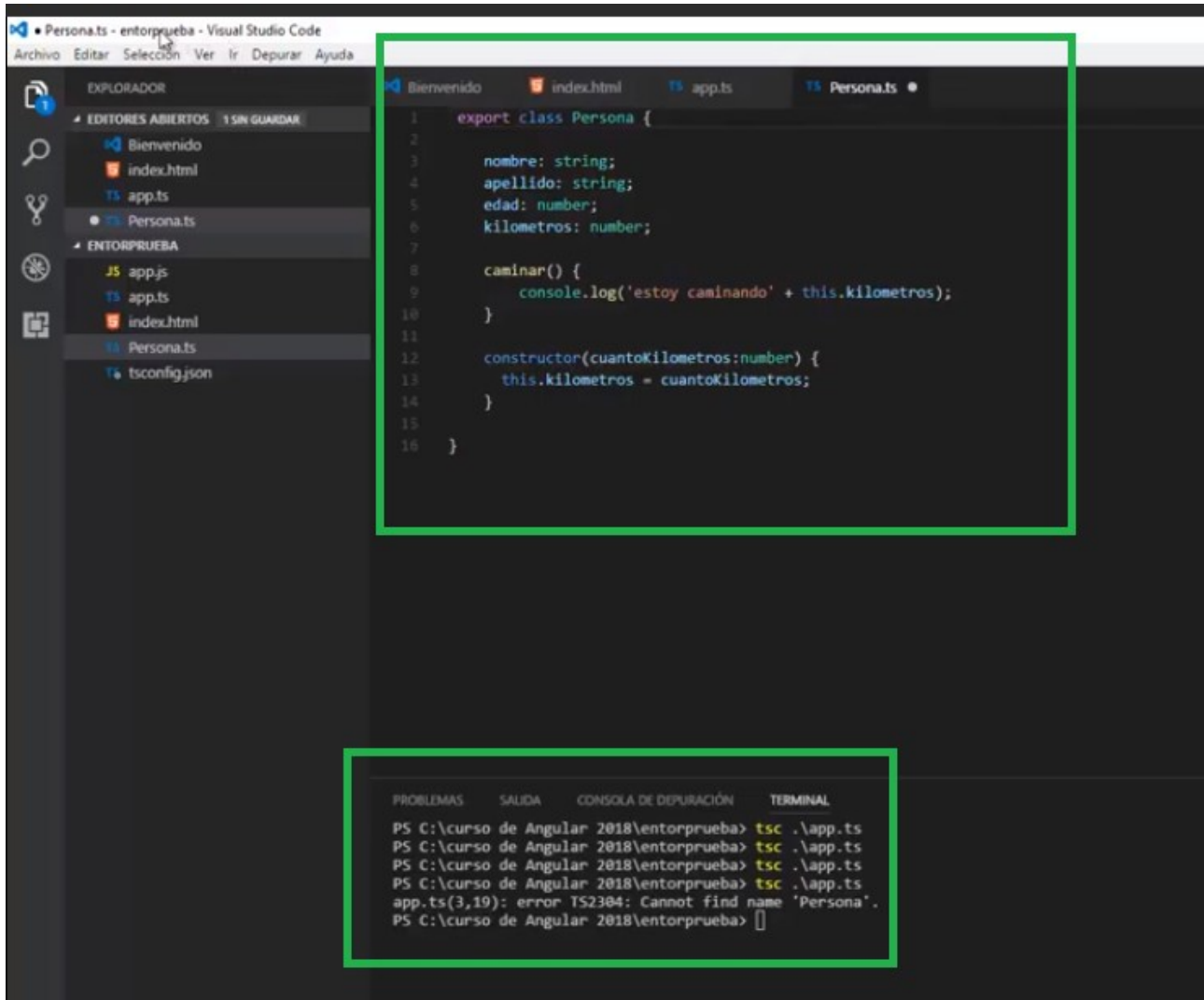
Este tipo de sintaxis es el equivalente a:

```
let sentence: string = "Hello, my name is " + fullName + "." + "I'll be " + (age + 1)   
+ " years old next month."
```

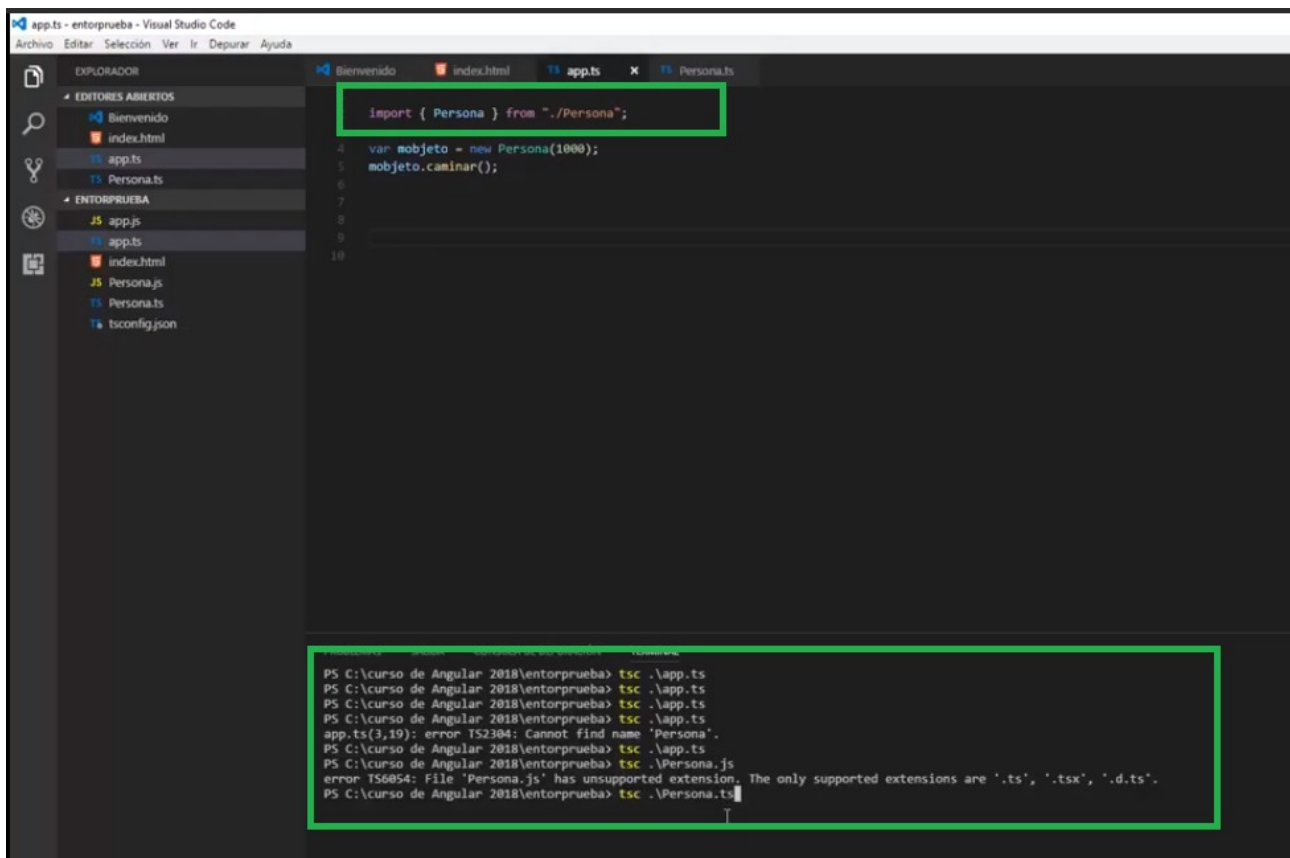
Modulos:

nos sirven para reutilizar las clases y componentes , para ellos debemos agregar la palabra export para que otro pueda importarlo y consumirlo ej :

Persona.ts : es exportable



app.ts : hace el import de Persona



The screenshot shows the Visual Studio Code interface. The Explorer on the left shows a project named 'entorprueba' with files like 'Bienvenido', 'index.html', 'app.ts', and 'Persona.ts'. The main editor shows 'app.ts' with the following code:

```
import { Persona } from './Persona';

var objeto = new Persona(1000);
objeto.caminar();
```

The terminal at the bottom shows the following commands and errors:

```
PS C:\curso de Angular 2018\entorprueba> tsc .\app.ts
PS C:\curso de Angular 2018\entorprueba> tsc .\app.ts
PS C:\curso de Angular 2018\entorprueba> tsc .\app.ts
PS C:\curso de Angular 2018\entorprueba> tsc .\app.ts
app.ts(3,19): error TS2304: Cannot find name 'Persona'.
PS C:\curso de Angular 2018\entorprueba> tsc .\app.ts
PS C:\curso de Angular 2018\entorprueba> tsc .\Persona.js
error TS6054: File 'Persona.js' has unsupported extension. The only supported extensions are '.ts', '.tsx', '.d.ts'.
PS C:\curso de Angular 2018\entorprueba> tsc .\Persona.ts
```

DECORADORES : ES COMO LA HERENCIA SIRVE PARA EXTENDER FUNCIONALIDAD DE ALGO QUE YA EXISTE , podemos decorar un atributo(validarlo, formatearlo), un método (modificaciones, validaciones),una clase (agregar funcionalidad) ,propiedad. siempre con el objetivo de agregar funcionalidad

Decoradores

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queremos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

Podemos considerar un decorador como la forma de aumentar las funcionalidades a ciertos tipos.

Decorando una clase

```
4 function Bienvenida(target: Function): void {
5     target.prototype.saludo = function(): void {
6         console.log('¡Hola!');
7     }
8 }
9
10 @Bienvenida
11 class Saludar {
12     constructor() {
13         // Implementación va aquí...
14     }
15 }
16
17 var miSaludo = new Saludar();
18 miSaludo.saludo(); // Consola mostrará '¡Hola!'
19
```

Que es un Componente ?

Un componente es una **nueva etiqueta HTML** con una **vista** y una **lógica** definidas por el desarrollador

La **vista** es una plantilla (*template*) en HTML con elementos especiales

La **lógica** es una clase TypeScript vinculada a la vista

Un **estilo** para la vista

Tu **spec** archivo usado para las pruebas unitarias del componente

Mas claro ?



Pequeñas clases que cumplan funciones específicas.

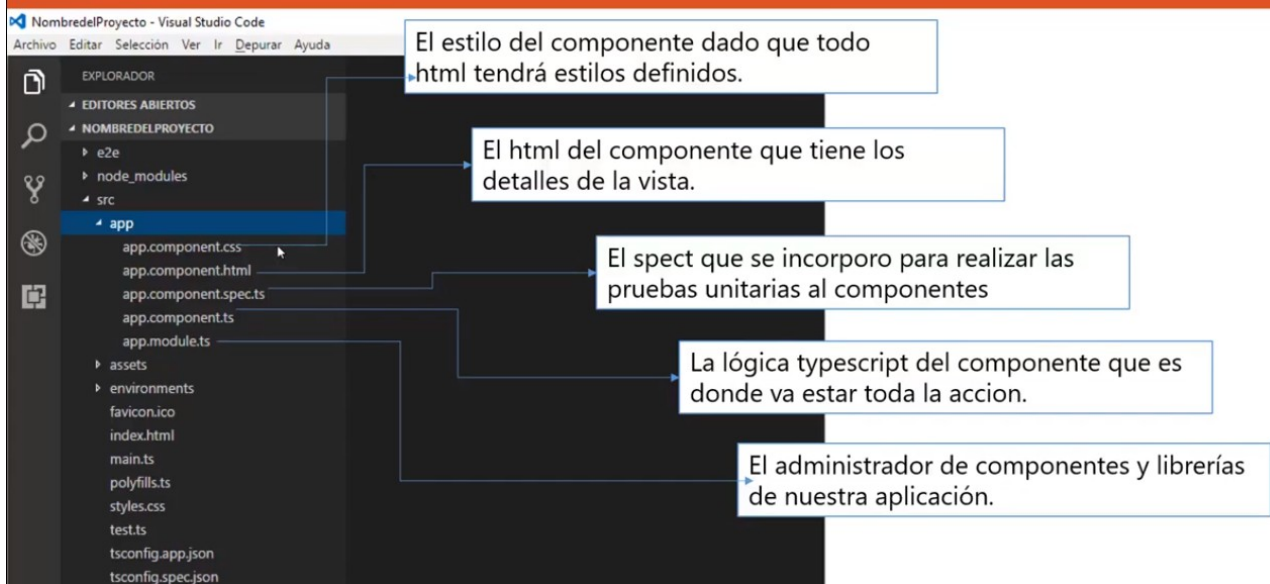
Es decir que una web en angular se encuentra formada por un conjunto de componentes que tienen eventos y propiedades que resguardan estado.

Que a su vez tienen un ciclo de vida que deben respetar.

Importante

- Todo componente tiene un decorador que permite asociar los elementos que lo conforman
- Todo componente para formar parte de una vista tendrá un selector <app-root>
- Todo componente esta formado por una vista , una lógica typescript y un estilo de la vista.
- Todo componente Importa Librerías que usara
- Todo componente Se exporta para que pueda formar parte del todo.

Entonces dentro de /app ??



```

index.html
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Nombre del Proyecto</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
15

```

```

app.component.ts
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'app';
10 }
11

```

```

app.component.html
1 <!-- The content below is only a placeholder and can be replaced. -->
2 <div style="text-align:center">
3   <h1>
4     Welcome to {{ title }}!
5   </h1>
6   
12   </li>
13   <li>
14     <a target="_blank" rel="noopener" href="https://github.com/angular/>
15   </li>
16   <li>
17     <a target="_blank" rel="noopener" href="https://blog.angular.io/">
18   </li>
19 </ul>
20
21

```

```

app.component.css
1

```

Pero quien gestiona los componentes y elementos de mi aplicación ?

```

app.module.ts
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4
5 import { AppComponent } from './app.component';
6
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19

```

Componentes declarados

Modulos Importados

Componentes Principales

Antes de importar cualquier módulo hay que definirlo. En Angular los módulos se declaran como clases de TypeScript, habitualmente vacías, decoradas con una función especial. Es la función `@NgModule()` que recibe un objeto como único argumento. En las propiedades de ese objeto es dónde se configura el módulo.

Este archivo es el encargado de entender qué componentes y dependencias tenemos en nuestra aplicación. Si el componente no fue declarado como por ejemplo `AppComponent` no puede ser utilizado.

El contexto de NgModule

NgModule es un decorador que recibe un objeto de metadatos que definen el módulo. Los metadatos más importantes de un **NgModule** son:

declarations: Las vistas que pertenecen a tu módulo. Hay 3 tipos de clases de tipo vista: componentes, directivas y pipes.

exports: Conjunto de declaraciones que deben ser accesibles para templates de componentes de otros módulos.

imports: Otros NgModules, cuyas clases exportadas son requeridas por templates de componentes de este módulo.

providers: Los servicios que necesita este módulo, y que estarán disponibles para toda la aplicación.

bootstrap: Define la vista raíz. Utilizado solo por el root module.

data binding de interpolación

Data Binding

Uno de los principales valores de Angular es que nos abstrae de la lógica pull/push asociada a insertar y actualizar valores en el HTML y convertir las respuestas de usuario (inputs, clicks, etc) en acciones concretas.

Escribir toda esa lógica a mano (lo que típicamente se hacía con JQuery) es tedioso y propenso a errores, y Angular 2 lo resuelve por nosotros gracias al **Data Binding**.

Simplifiquemos el template anterior para centrarnos en el *data binding*:

```
<div>{{todo.subject}}</div>
<todo-detail [todo]="selectedTodo"></todo-detail>
<div (click)="selectTodo(todo)"></div>
```


Data Binding

- **Interpolación:** (Hacia el DOM)

Al hacer `{{todo.subject}}`, Angular se encarga de insertar el valor de esa propiedad del componente entre las etiquetas `<div>` donde lo hemos definido. Es decir, evalúa `todo.subject` e introduce su resultado en el DOM.

- **Property binding:** (Hacia el DOM)

Al hacer `[todo]="selectedTodo"`, Angular está pasando el **objeto** `selectedTodo` del Componente padre a la propiedad `todo` del Componente hijo, en este caso de `TodoDetailComponent`. Recordemos de la [sección Componentes](#) que para esto el componente `TodoDetailComponent` habrá definido una propiedad `todo` con el decorador `@Input()`.

- **Event binding:** (Desde el DOM)

Al hacer `(click)="selectTodo(todo)"`, le indicamos a Angular que cuando se produzca un evento `click` sobre esa etiqueta `<div>`, llame al método `selectTodo` del Componente, pasando como atributo el objeto `todo` presente en ese contexto. (Aunque hemos simplificado el ejemplo, esto venía de un bucle que itera el array `todos` obteniendo la variable `todo`).

Tambien tenemos el two binding

Ejemplo de interpolacion:

1. creamos un proyecto con `ng new databinding`
2. ponemos a correr con `ng serve`
3. modificamos el `app.component.html` con esta linea

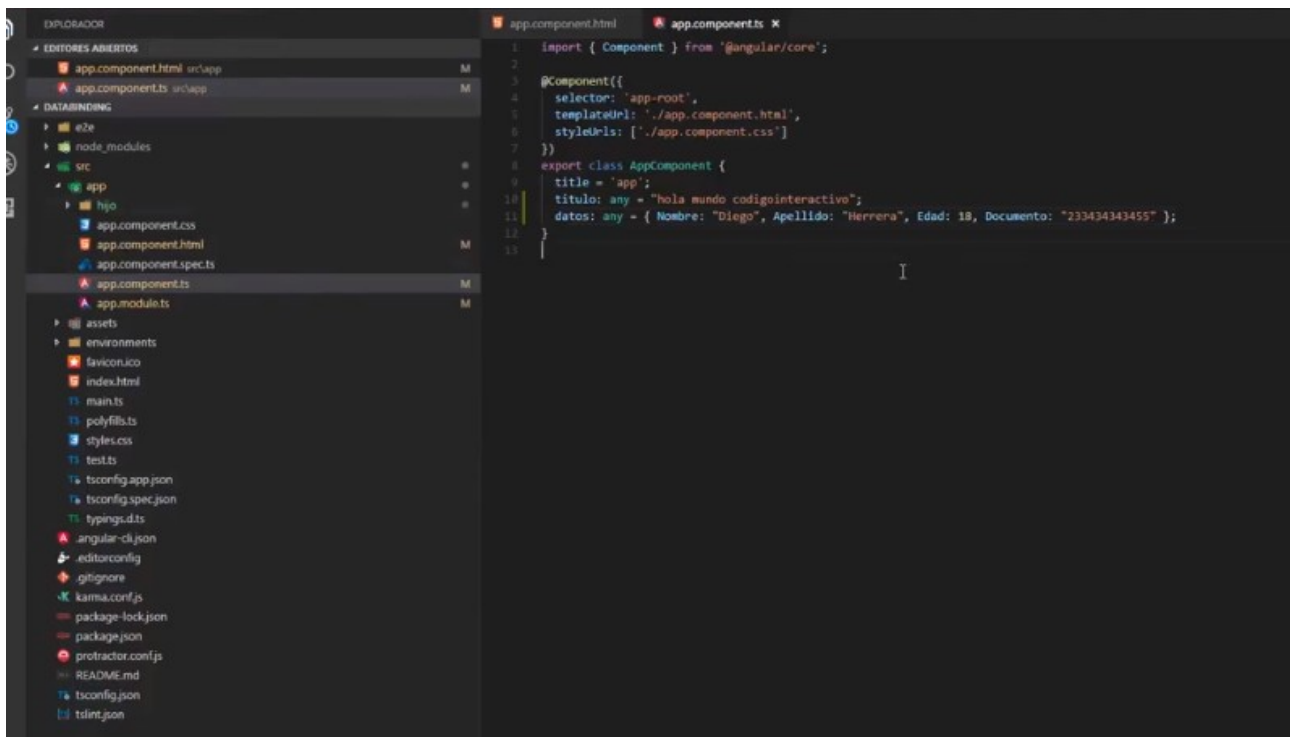
`<h1> titulo que se encuentra en la clase ts {{titulo}}</h1>`

4. modificamos `app.component.ts` agregando dentro de la clase la variable **`titulo:any ="hola mundo educacionit "`**;

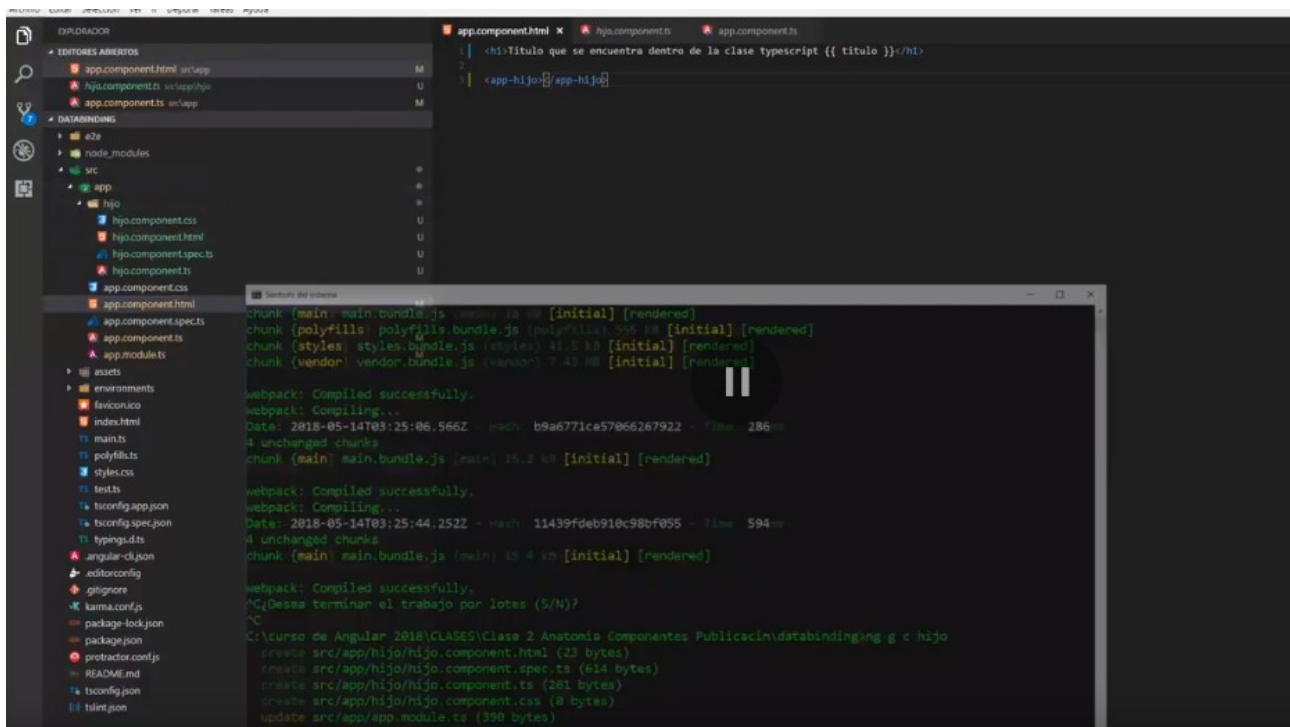
Ejemplo de Propertybinding:

demonstración para que un componente hijo pueda recibir el dato

1. dentro del proyecto creamos un componente llamado hijo (`ng g c hijo`)
2. nos vamos al componente padre (`appcomponent`) y agregamos una variable de tipo JSON llamada `datos`



3.no vamos al componente hijo y vemos cual es el selector (selector: app-hijo) para agregarlo en el componente padre (appcomponent.html)

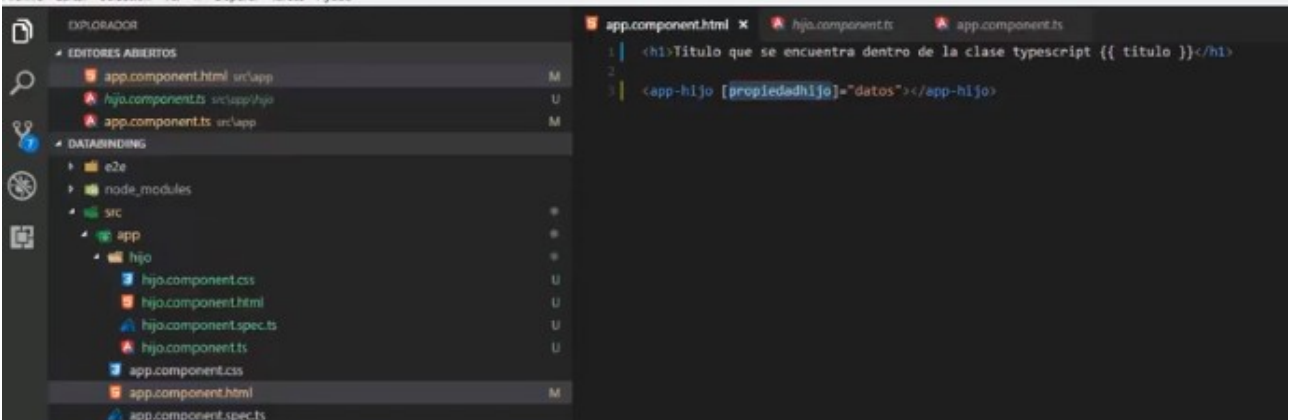


4. actualizamos y deberíamos ver el componente hijo

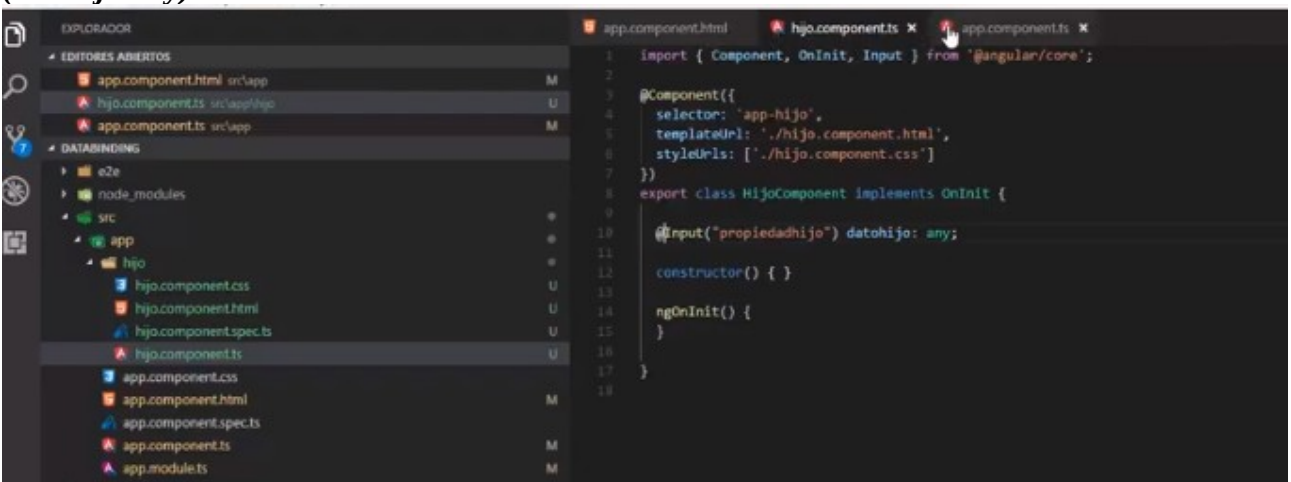


5. pasamos datos entre padre e hijo

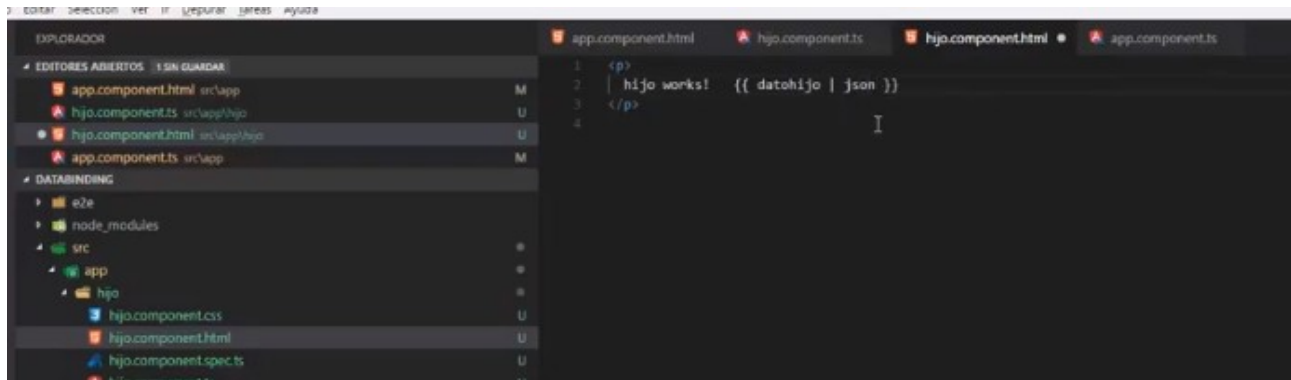
dentro del componente hijo hacemos lo siguiente : aca estamos pasando el objeto datos al hijo



6. en la clase hijo defino el decorador input ,lo importo y le indico el nombre de la propiedad input ("propiedadhijo") y lo voy asociar con → una propiedad de mi componente del tipo any (datohijo:any)



7. dentro de hijo.component.html utilizo la propiedad del padre

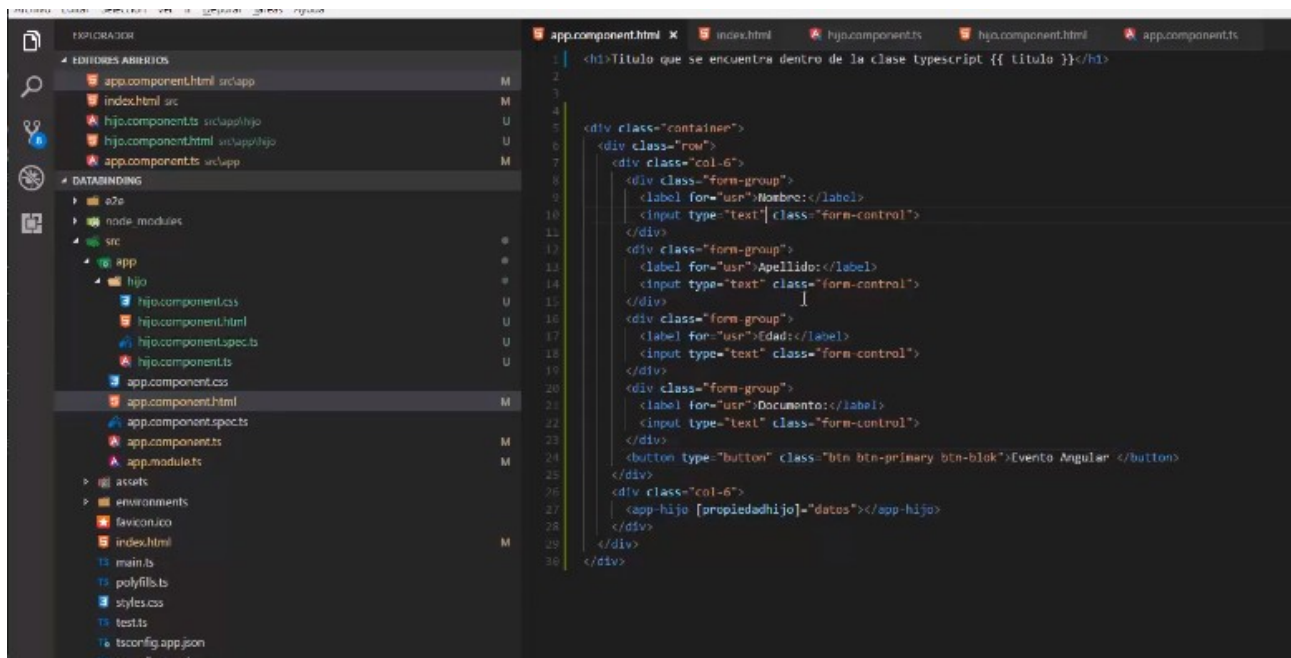


8. vemos el resultado del objeto json enviado desde un padre a un hijo en el navegador

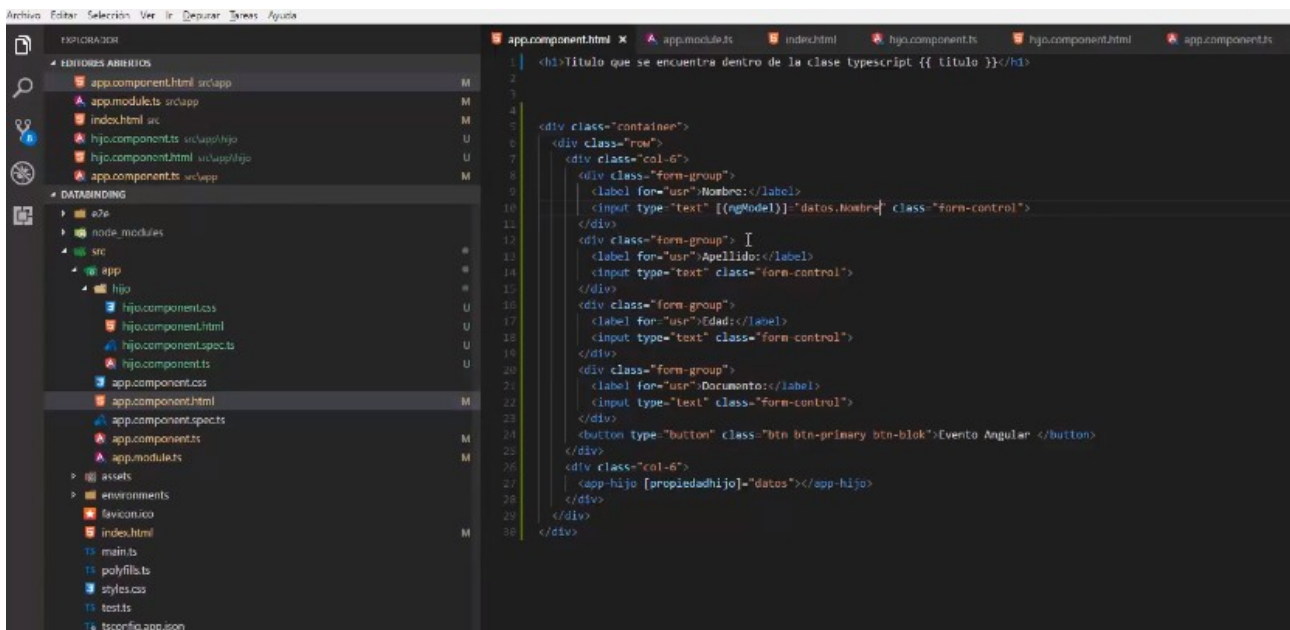


databinding doble

0. incorporamos los 3 cdn de bootstrap en index.html (tener cuidado el orden, primero jquery ,luego popper y por ultimo bootstrap)
 1. Creamos un div en app.component.html y dentro de este una serie de div que van a ser filas una fila para el formulario y otra para lo que son los botones
- form-group : sirve para agrupar labels y controles

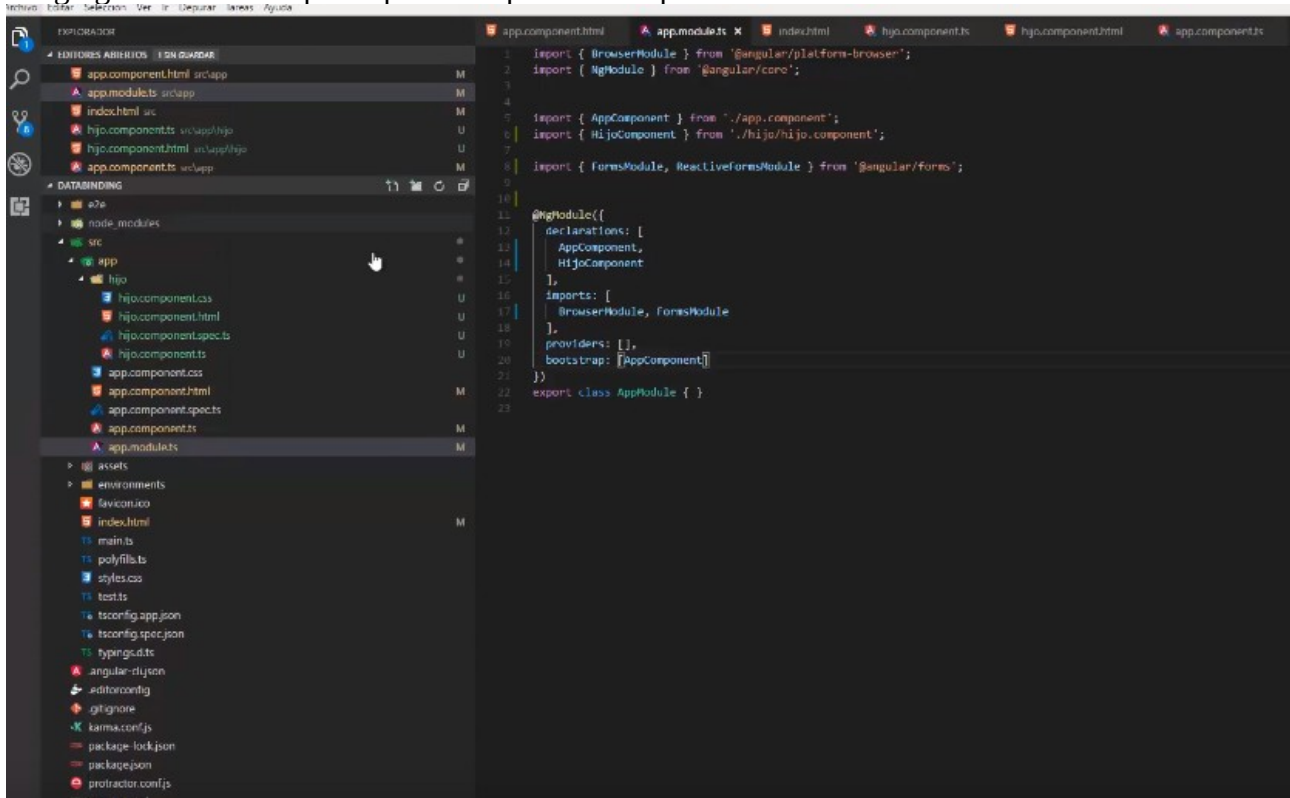


2. ahora implementamos el doble binding

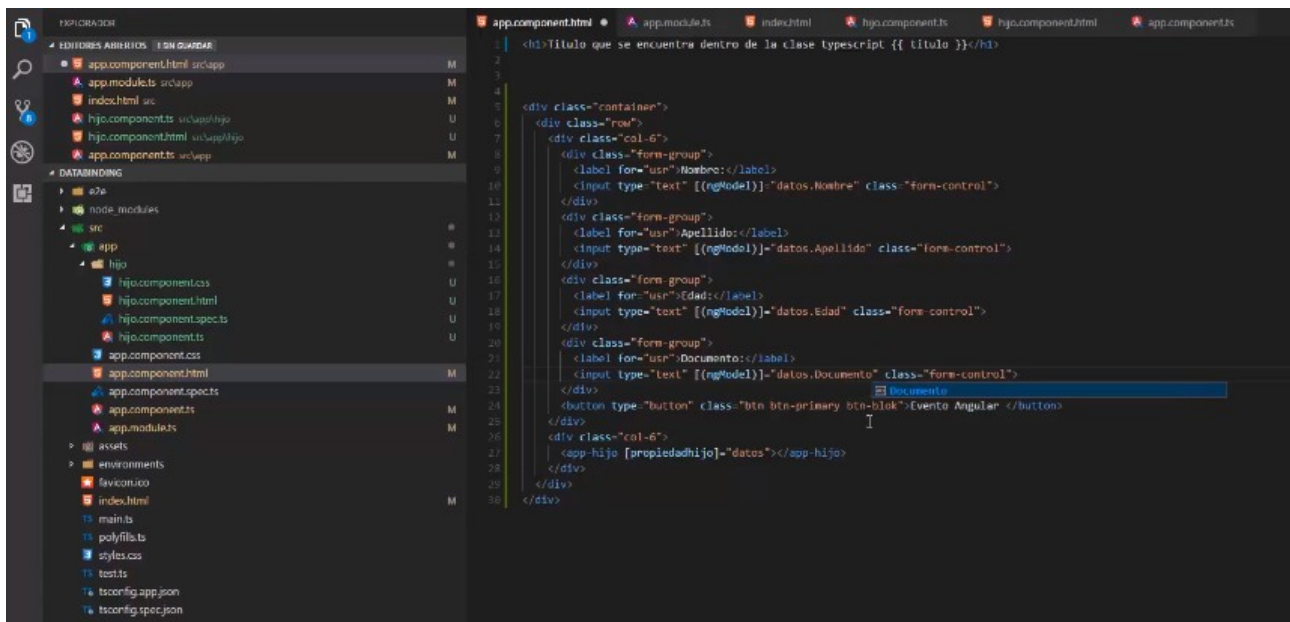


en app.module.ts incorporamos el import @angular/forms el objeto formssModule y reactive forms module

lo agregamos en los imports para el soporte de lo que son los formularios



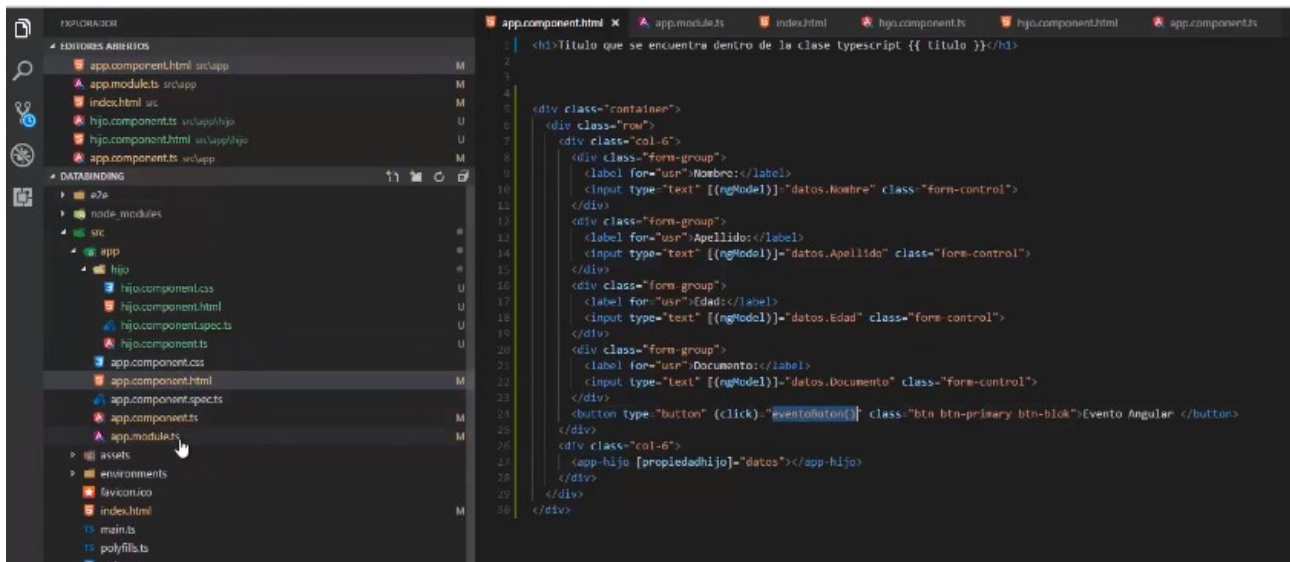
3. ahora lo aplicamos en el resto de los datos = nombre apellido edad documento ...



```
1 <h1>Titulo que se encuentra dentro de la clase typescript {{ titulo }}</h1>
2
3
4
5 <div class="container">
6   <div class="row">
7     <div class="col-6">
8       <div class="form-group">
9         <label for="usr">Nombre:</label>
10        <input type="text" [(ngModel)]="datos.Nombre" class="form-control">
11      </div>
12    </div>
13    <div class="form-group">
14      <label for="usr">Apellido:</label>
15      <input type="text" [(ngModel)]="datos.Apellido" class="form-control">
16    </div>
17    <div class="form-group">
18      <label for="usr">Edad:</label>
19      <input type="text" [(ngModel)]="datos.Edad" class="form-control">
20    </div>
21    <div class="form-group">
22      <label for="usr">Documento:</label>
23      <input type="text" [(ngModel)]="datos.Documento" class="form-control">
24    </div>
25    <button type="button" class="btn btn-primary btn-block">Evento Angular </button>
26  </div>
27  <div class="col-6">
28    <app-hijo [propiedadhijo]="datos"></app-hijo>
29  </div>
30 </div>
```

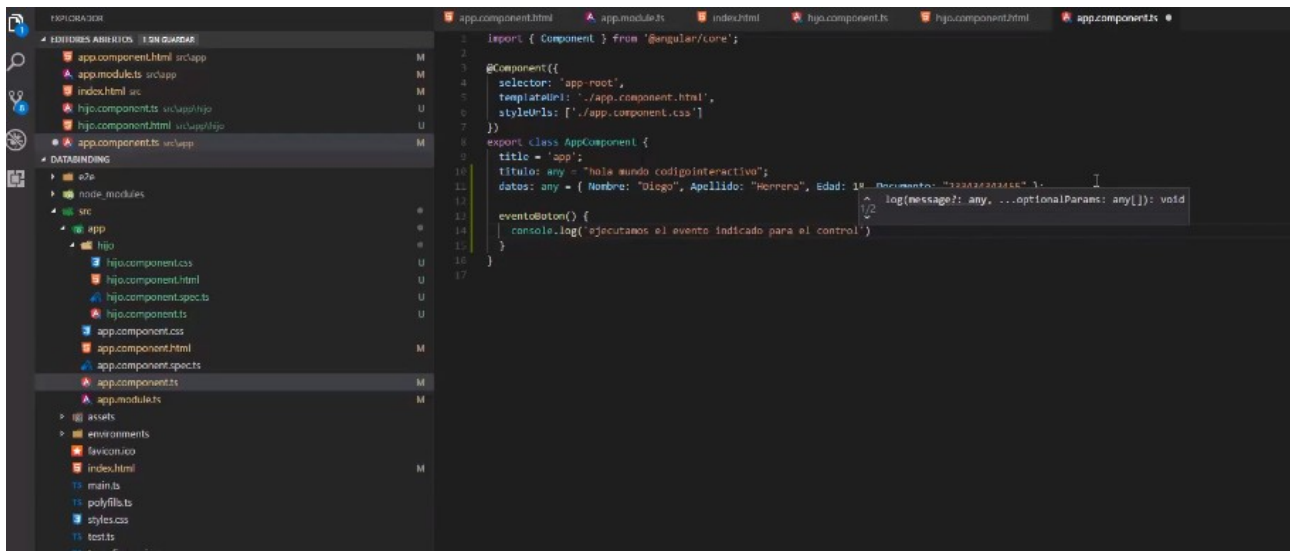
Event Binding:

1. creamos el evento click y lo igualamos al metodo llamado "eventoBoton "



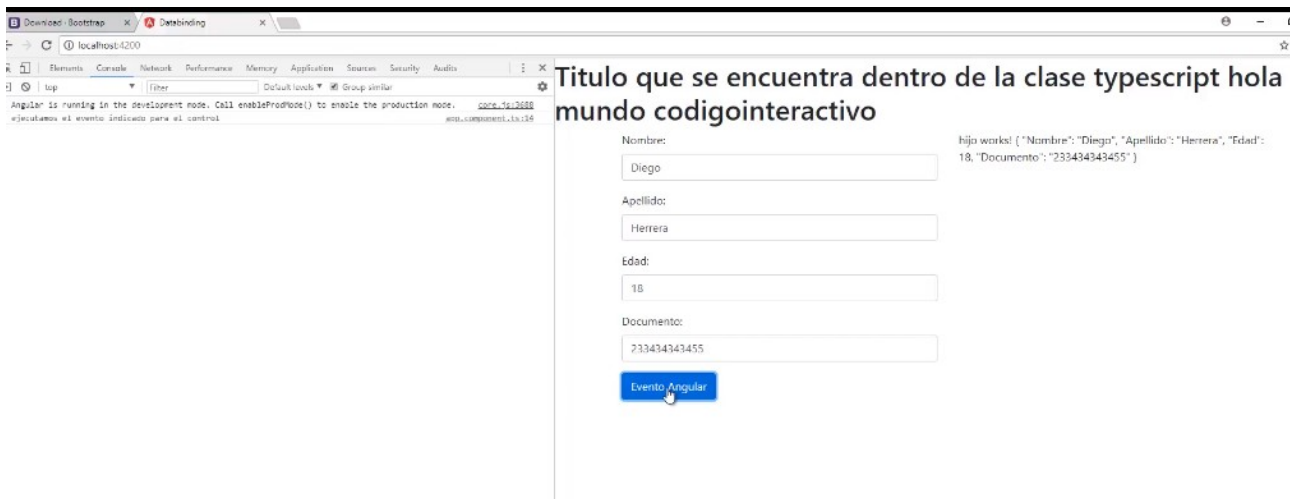
```
1 <h1>Titulo que se encuentra dentro de la clase typescript {{ titulo }}</h1>
2
3
4
5 <div class="container">
6   <div class="row">
7     <div class="col-6">
8       <div class="form-group">
9         <label for="usr">Nombre:</label>
10        <input type="text" [(ngModel)]="datos.Nombre" class="form-control">
11      </div>
12    </div>
13    <div class="form-group">
14      <label for="usr">Apellido:</label>
15      <input type="text" [(ngModel)]="datos.Apellido" class="form-control">
16    </div>
17    <div class="form-group">
18      <label for="usr">Edad:</label>
19      <input type="text" [(ngModel)]="datos.Edad" class="form-control">
20    </div>
21    <div class="form-group">
22      <label for="usr">Documento:</label>
23      <input type="text" [(ngModel)]="datos.Documento" class="form-control">
24    </div>
25    <button type="button" (click)="eventoBoton()" class="btn btn-primary btn-block">Evento Angular </button>
26  </div>
27  <div class="col-6">
28    <app-hijo [propiedadhijo]="datos"></app-hijo>
29  </div>
30 </div>
```

2.dentro del ts del padre app.component.ts vamos a construir el evento



```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'app';
10  titulo = "hola mundo codigointeractivo";
11  datos: any = { Nombre: "Diego", Apellido: "Herrera", Edad: 18, Documento: "233434343455" };
12
13  evento@button() {
14    console.log('ejecutamos el evento indicado para el control')
15  }
16 }
17
```

luego al dar click al boton vamos a salir por la consola



JSON

JSON (*Javascript Object Notation*) es un formato de intercambio de datos entre clientes y servidores, basado en la sintaxis de *Javascript* para representar estructuras en forma organizada y alto nivel, con el fin de acercar a una definición mucho más amigable por los desarrolladores.

JSON es una herramienta potente en el desarrollo de aplicaciones web, ya que facilita el desarrollo y comprensión de intercambio de datos.

Similar a la estructuración de datos primitivos y complejos en los lenguajes de programación, JSON establece varios tipos de datos: cadenas, números, booleanos, arrays, objetos y valores null.

El propósito es crear objetos que contengan varios atributos compuestos como pares clave-valor. Donde la clave es un nombre que identifique el uso del valor que lo acompaña.

```
{  
  "Id": 101 ,  
  "Nombre": "Carlos",  
  "EstaActivo": true,  
  "Notas": [ 2.3, 4.3,  
    5.0]  
}
```

Form Reactivos:

Primero lo Primero

Usando Form

Antes de cualquier cosa, debemos agregar las siguientes líneas en nuestro **app.module.ts**

```
//importar el modulo de form  
import { FormsModule } from '@angular/forms';  
  
//en la parte de @NgModule  
imports: [  
  BrowserModule,  
  FormsModule  
],
```

Que tenemos en formularios

El *double-binding* facilita mucho el desarrollo de formularios. Pero esa magia tienen un coste en escalabilidad; impacta en el tiempo de ejecución y además dificulta la validación y el mantenimiento de formularios complejos

Hay algunas clases en Angular 5, entre ellas FormGroup, FormBuilder y Validators, que son las clases más utilizadas al crear [formularios Reactivos](#).

- FormGroup: realiza el seguimiento del valor y el estado de validación de un control de formulario individual, en nuestro caso lo aplicaremos contra el formulario, así podremos controlar el email y password desde él.
- FormBuilder: crea un [AbstractControl](#) desde la configuración que le pasemos, nosotros utilizaremos el método group el cual recibe un FormGroup.
- Validators: por defecto trae una serie de validadores muy útiles para validar campos de formulario aunque es muy sencillo crear validadores personalizados.

Form Builder

Entra en acción el FormBuilder, un servicio del que han de depender los componentes que quieran desacoplar el modelo de la vista.

Se usa para construir un formulario creando un FormGroup, o grupo de controles, que realiza un seguimiento del valor y estado de validez de los datos.

```
1 import { FormBuilder, FormGroup } from '@angular/forms';
2 public form: FormGroup;
3 constructor(private formBuilder: FormBuilder) {}
4 public ngOnInit() {
5     this.form = this.formBuilder.group({});
6 }
```


Form Group

El formulario se define como un grupo de controles.

Cada control tendrá un nombre y una configuración. Esa definición permite establecer un valor inicial al control y asignarle validaciones.

```
1 this.name = 'ALBERTO';
2 this.form = this.formBuilder.group({
3   email: 'info@angular.io',
4   name: this.name.toLowerCase(),
5   registeredOn : new Date().toISOString().substring(0, 10)
6   password: ''
7 });
```

En mi Vista

```
1 <form [formGroup]="form">
2   <label for="email">E-mail</label>
3   <input name="email"
4     FormControlName="email"
5     type="email" />
6   <label for="name">Name</label>
7   <input name="name"
8     FormControlName="name"
9     type="text" />
10  <label for="registeredOn">Registered On</label>
11  <input name="registeredOn"
12    FormControlName="registeredOn"
13    type="date" />
14  <label for="password">Password</label>
15  <input name="password"
16    FormControlName="password"
17    type="password" />
```

usaremos dos directivas que vienen dentro del módulo reactivo son `[formGroup]="objetoFormulario"` para el formulario en su conjunto, y `FormControlName="nombreDelControl"` para cada control

Validaciones de Formulario

La validación es una pieza clave de la entrada de datos en cualquier aplicación. Es el primer **frente de defensa ante errores de usuarios**; involuntarios o deliberados.

Angular viene con un pequeño conjunto de validadores preconstruidos para que coincida con los que podemos definir a través de atributos HTML5 estándar, es decir, **required**, **minlength**, **maxlength** y **pattern**, a los que podemos acceder desde el módulo Validators.

```
1 this.form = this.formBuilder.group({
2   email: [
3     'info@angular.io',
4     [ Validators.required, Validators.email ]
5   ],
6   name: [
7     this.name.toLowerCase(),
8     Validators.required
9   ],
10  registeredOn : new Date().toISOString().substring(0, 10)
11  password: [
12    '',
13    [ Validators.required, Validators.minLength(4) ]
14  ]
15 });
```

Dmostracion de Form React:

- 1.creamos un proyecto
- 2.lo ejecutamos en el 4200
- 3.vamos a la pagina de bootstrap y nos copiamos le formulario luego lo pegamos en el componete principal

The screenshot shows the Bootstrap 4.1 documentation page for forms. On the left is a sidebar with navigation links like 'Getting started', 'Layout', 'Content', 'Components', and 'Form controls'. The main content area displays a form example with fields for 'Email address' and 'Password', a checkbox for 'Check me out', and a 'Submit' button. Below the form, the corresponding HTML code is shown, highlighting the use of Bootstrap classes like 'form-group', 'form-control', and 'form-check'.

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail" aria-describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword" placeholder="Password">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

4.modificamos el formulario que copiamos d :

