Angular
https://angular.dev/

¿Qué es Angular?

Básicamente es un Framework de JS que te permite crear Single Page Applications (o SPA's).

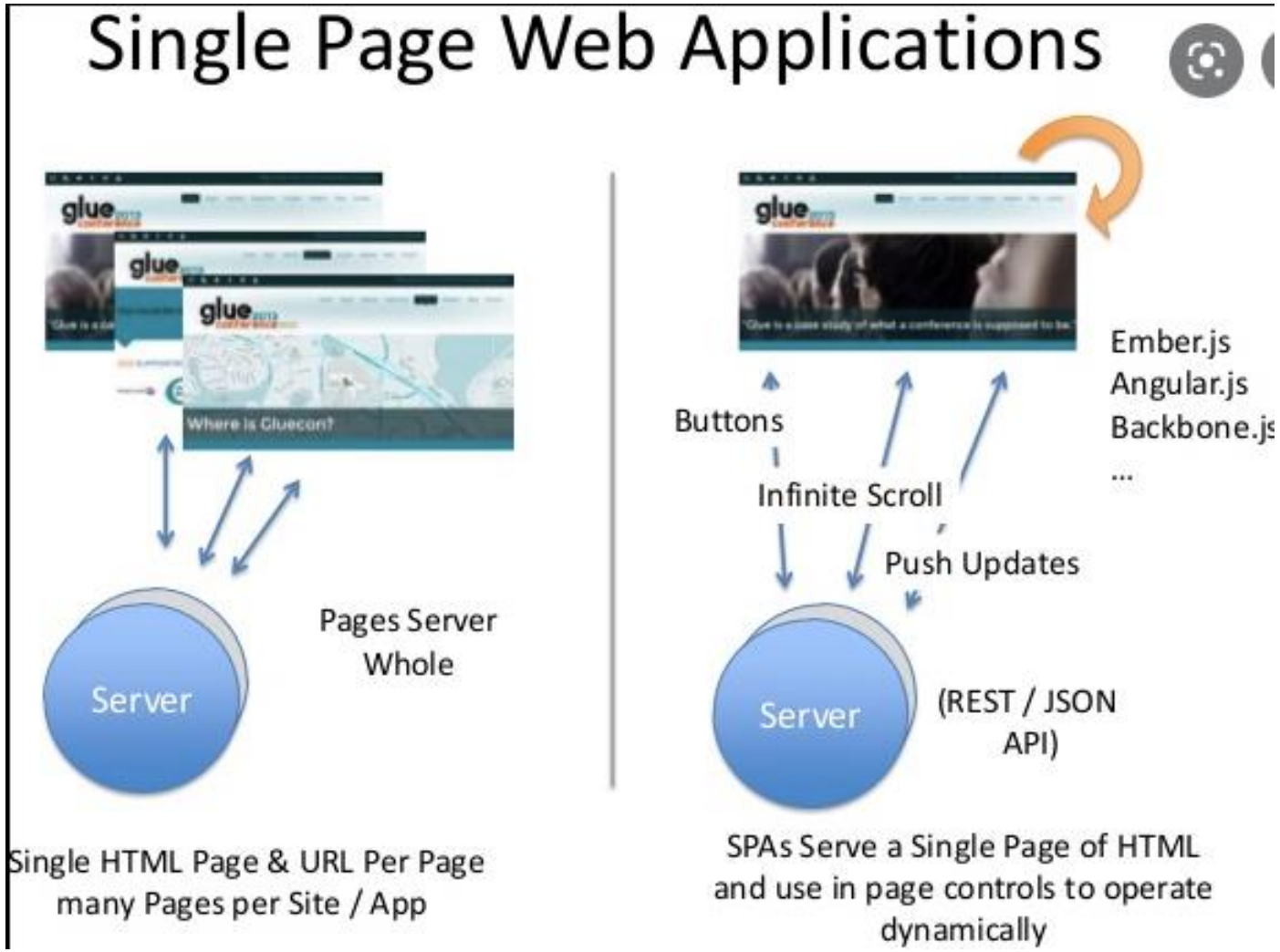
Angular utiliza el patrón de arquitectura MVC (Modelo-Vista-Controlador separando la lógica de la aplicación de la interfaz del usuario

Angular nació en 2010 (Google), era una librería.

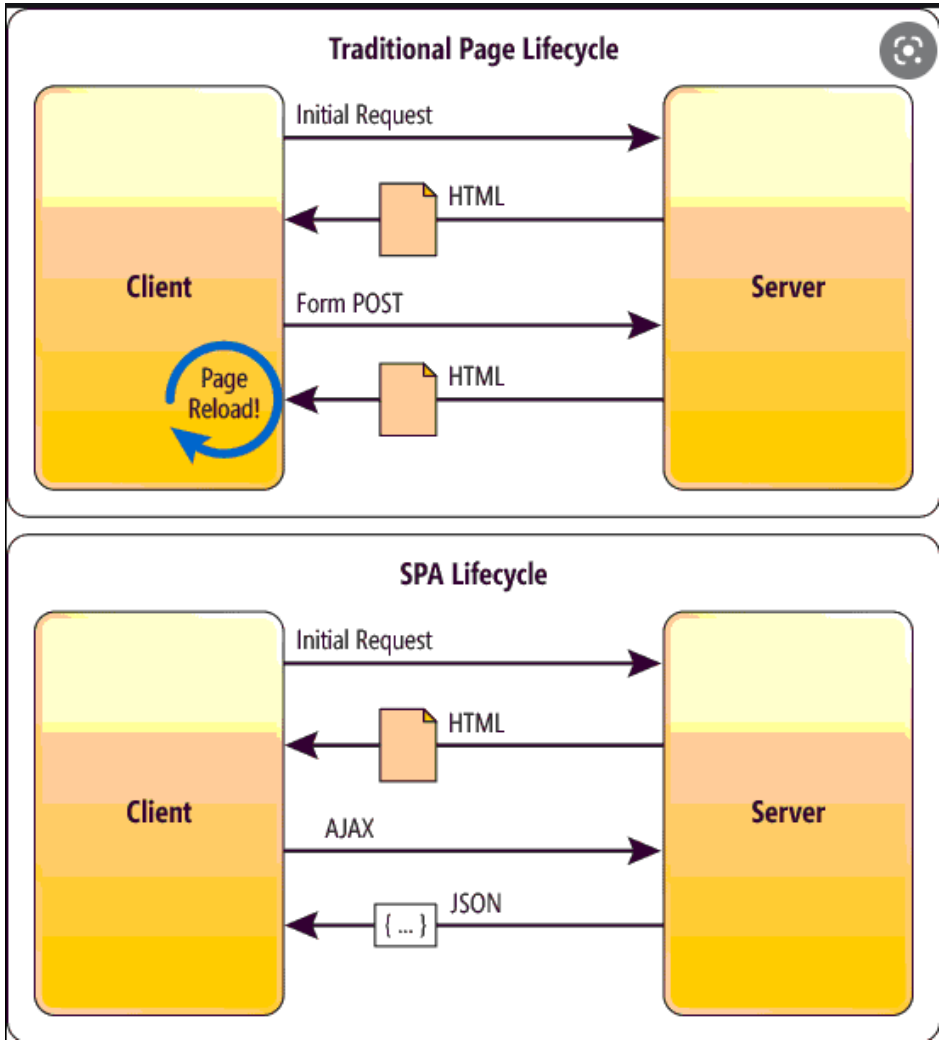
En el 2016 cuando salió la versión 2, el cambio fue radical y mucha gente migraron a otros framework.

Angular 5 y 6 salen novedades y mucha gente regresa.

SPA


Single Page Web Applications
Where is Gluecon?
Server
Pages Server
Whole
Single HTML Page & URL Per Page
many Pages per Site / App
Buttons
Infinite Scroll
Push Updates
Ember.js
Angular.js
Backbone.js
...
Server
(REST / JSON
API)
SPAs Serve a Single Page of HTML
and use in page controls to operate
dynamically

SPA


Traditional Page Lifecycle
Initial Request
HTML
Client
Server
Form POST
Page Reload!
HTML
SPA Lifecycle
Initial Request
HTML
Client
Server
AJAX
JSON
{ ... }

Separa frontend y backend de la aplicación

Simplifica el código

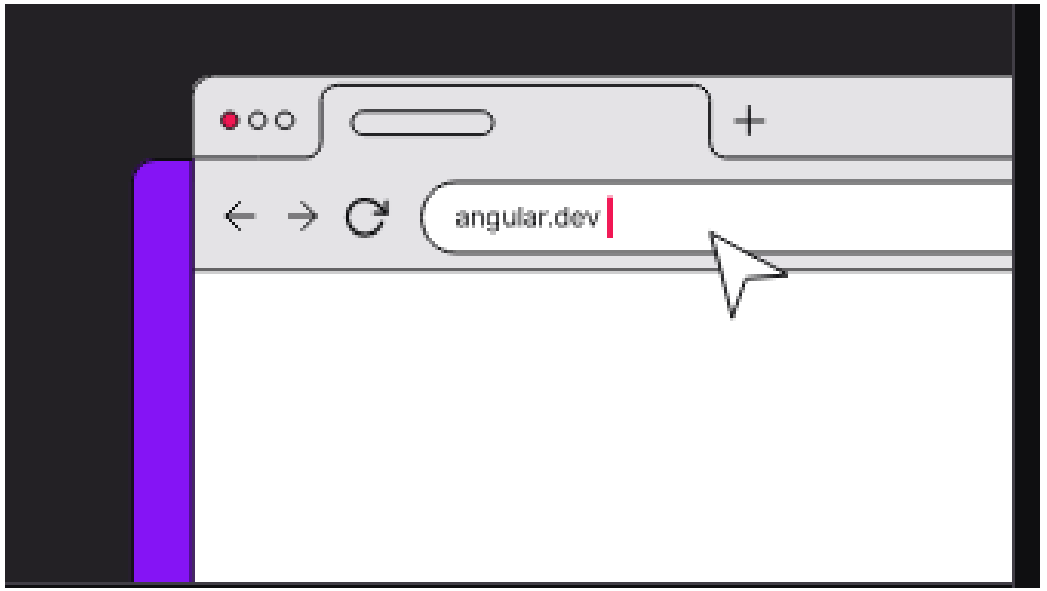
Sigue el patrón MVC

Basado en componentes

Es de código abierto

¿Qué lenguaje utiliza? Typescript

Se asocia de forma natural a un servidor node.js



Razones de su éxito

Crear aplicaciones **perfectamente modulares** (se basan en reglas para administrar los módulos Node.js).

Extender el lenguaje HTML gracias a **directivas** muy potentes, principalmente designando los componentes por medio de etiquetas.

Asociar operaciones a los flujos de datos **asíncronos** (por medio de programación **reactiva**).

Activar las diferentes vistas de interfaz, gracias a un **router** extremadamente perfeccionado, que vincula la selección de rutas a la activación de componentes.

Enlace de datos bidireccional: Los cambios realizados en la interfaz de usuario se reflejan automáticamente en el modelo de datos y viceversa

Elementos de una aplicación angular

Componentes con un decorador (plantilla html y css) y una clase.

Servicios son componentes de “utilidades” (se pueden utilizar en varios componentes, y no tienen decorador)

Módulos implementan las funcionalidades de la aplicación
Submódulos : se utilizan para organizar la aplicación

Router que dirige la navegación a través de rutas asociadas a componentes.

Instalación

Software necesario

Instalación Node.js lts

- Entorno JavaScript en tiempo de ejecución del lado del servidor. Sobre todo para poder utiliza el npm
- https://nodejs.org/en/download

Npm (Node package manager)

- Administrador de paquetes de Node.js (Maven), es instalado por defecto cuando se instala node.js

Visual Studio Code

- IDE
- https://code.visualstudio.com/download

Angular

- Angular CLI (Command Line Interface)

Node.js

- No se utiliza como back-end, sólo para el gestor de paquetes, descargar librerías, etc.
- Node.js, en la consola validamos que tenemos instalado con el comando:

  *node –v

  *npm -v
- De no ser así nos lo descargamos de la pagina oficial:
- https://nodejs.org/en/download

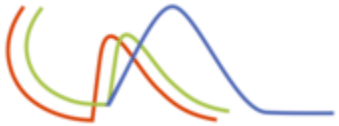
Pasos de instalación

Pagina oficial angular

1. Angular.dev
2. https://angular.dev/overview
3. En visual studio instalar la terminal bash

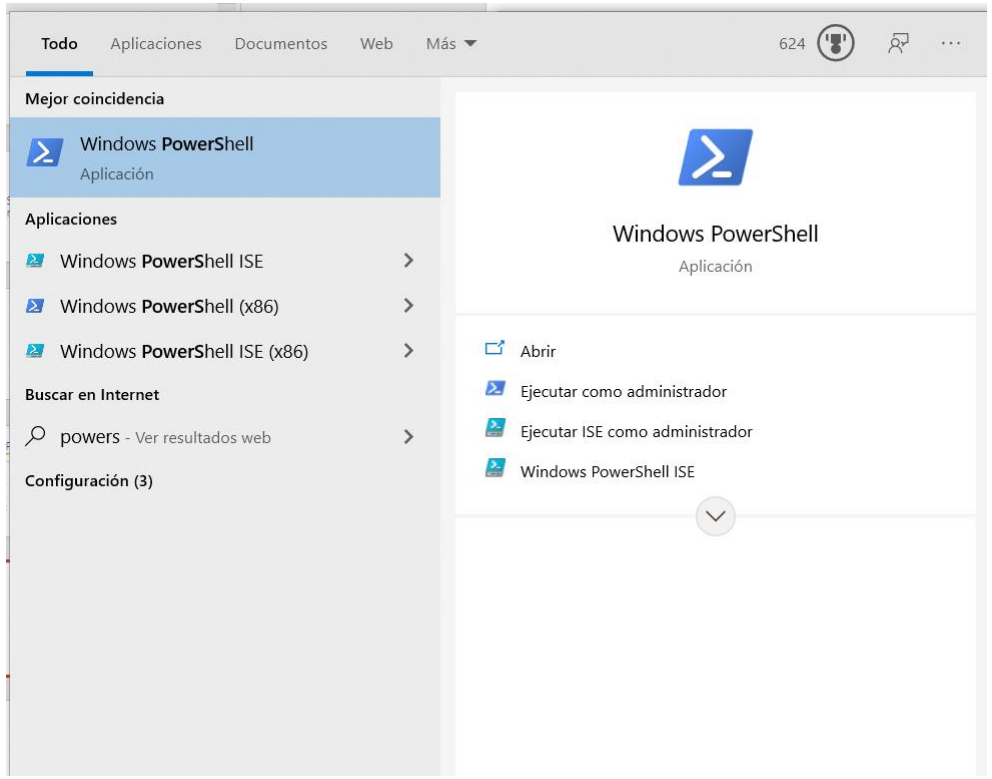
1. Abrir una ventana de consola como Administrador – instalar angular npm(Maven)
   1. `npm install -g @angular/cli`
   2. `Verificar que todo está ok`
   3. `ng version`
2. Crear una carpeta para los proyectos
3. Crear una carpeta para el primer proyecto
   1. ng new <nombre> (si, no, css)
   2. Ng serve --open (levantar el servidor) --port=<numero-puerto>
   3. <ctrl> <c> (Apagar el servidor)

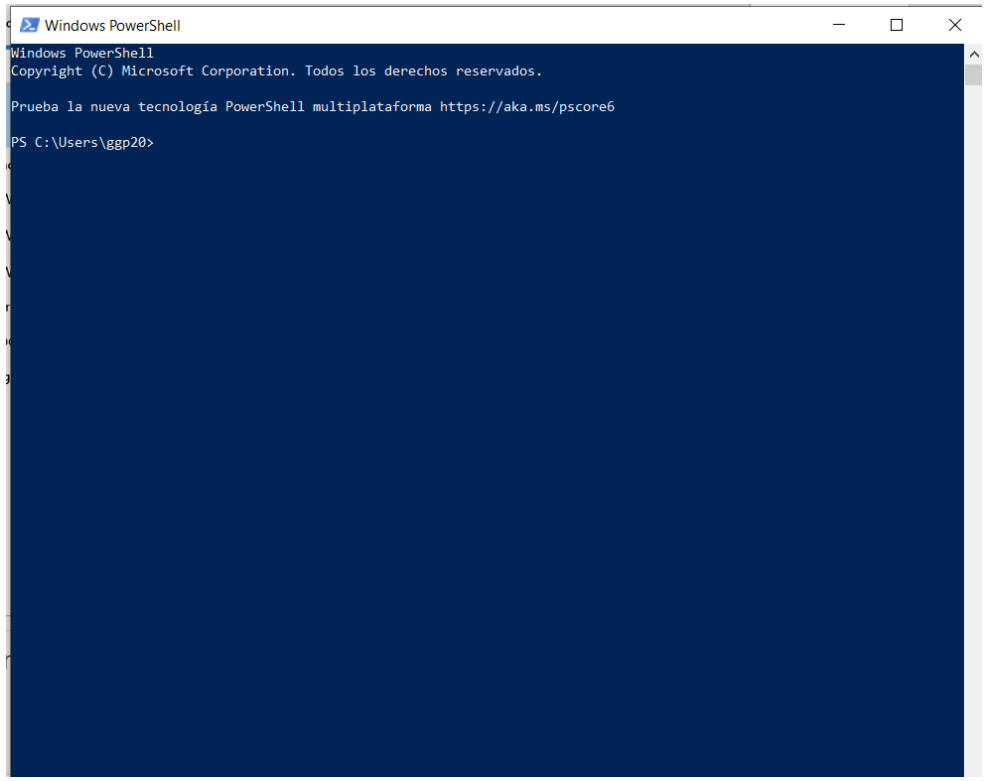
Extensiones



Error al ejecutar

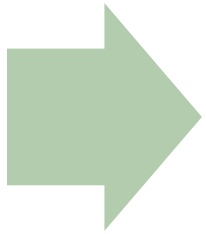
- Cuando sale error de seguridad
  - Buscar powerShell (como administrador)
  - Ejecutar el comando Set-ExecutionPolicy Unrestricted y dar y



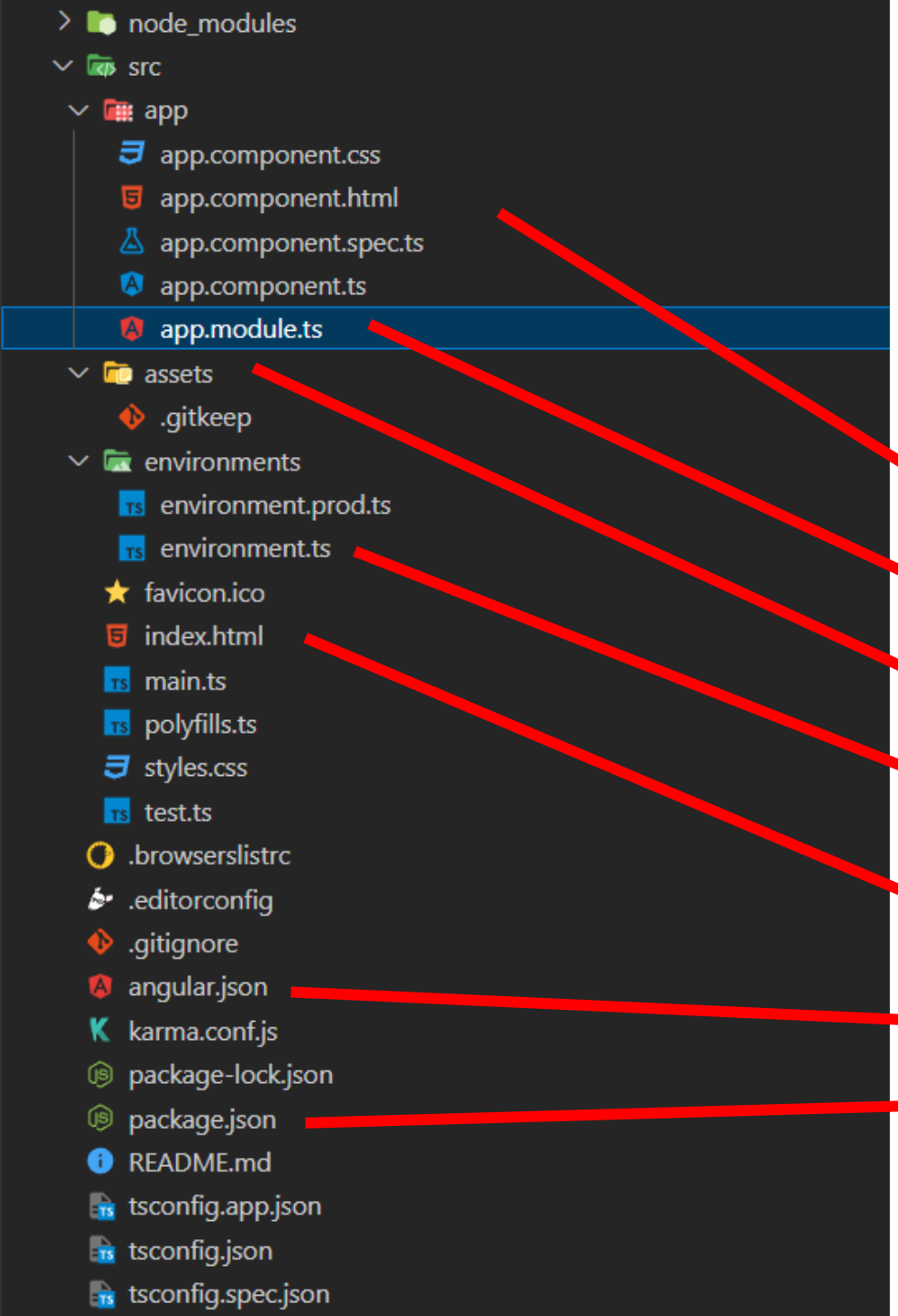


Angular vs Angular cli (línea de comandos)

Framework Angular (Ejemplo Spring)



Línea de comandos para facilitar su uso (Spring boot)



Estructura del proyecto

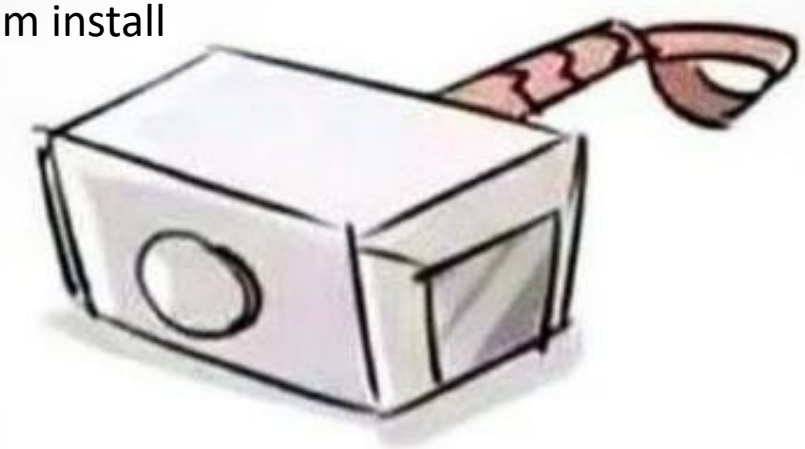
- Componente principal, con su ts, html y css
- Configuración de los módulos, las importaciones y proveedores.
- Recursos estáticos como imágenes, pdfs, etc.
- Definición de variables globales (en spring-application.properties).
- Única página (SPA)
- Propiedades de compilación
- Paquétes que se necesitan (en spring – pom.xml)

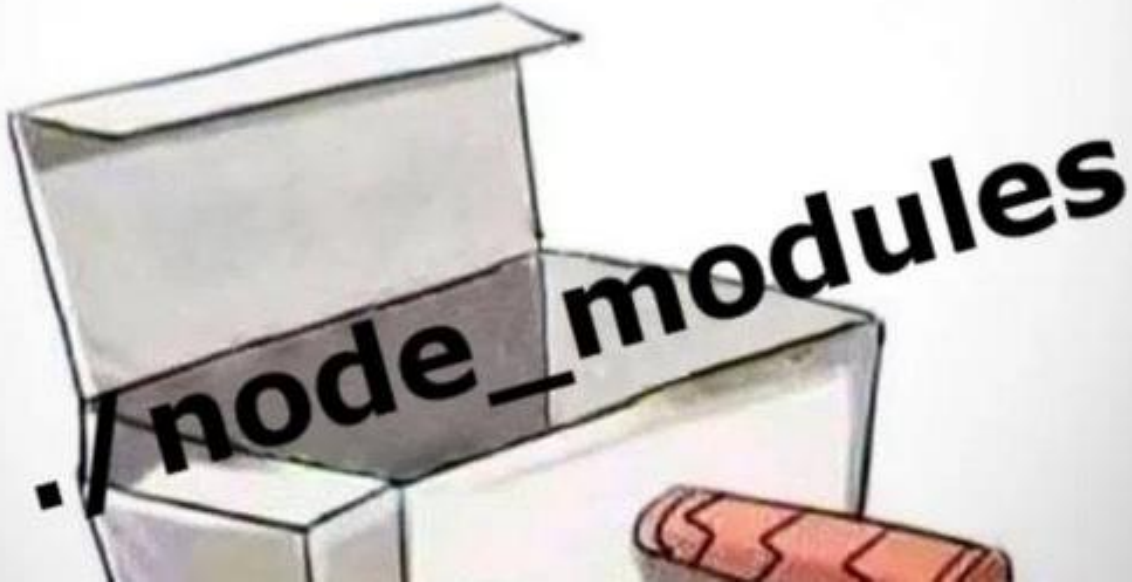
a simple
web page

2 GB
node_modules

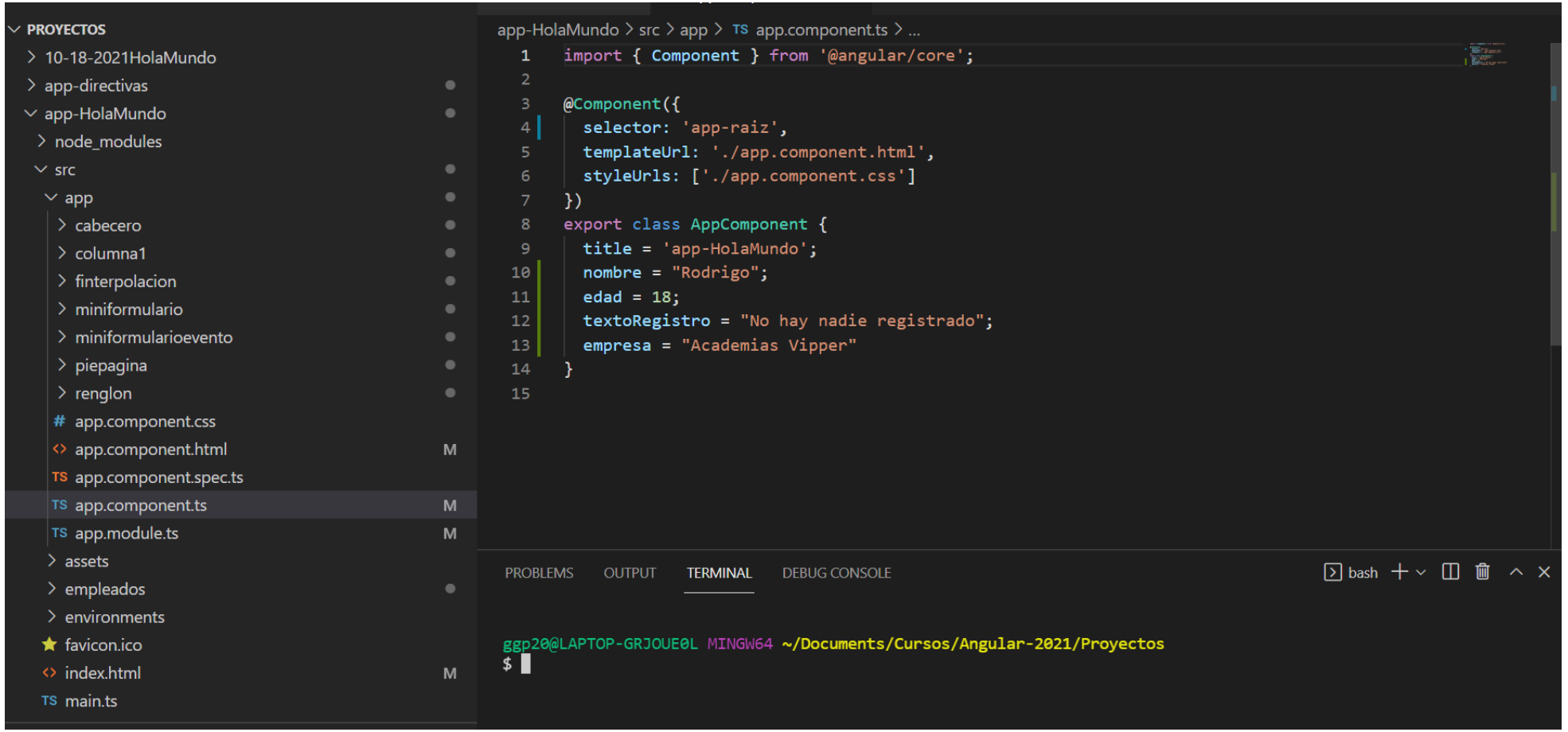
The secret behind Thor's hammer

npm install

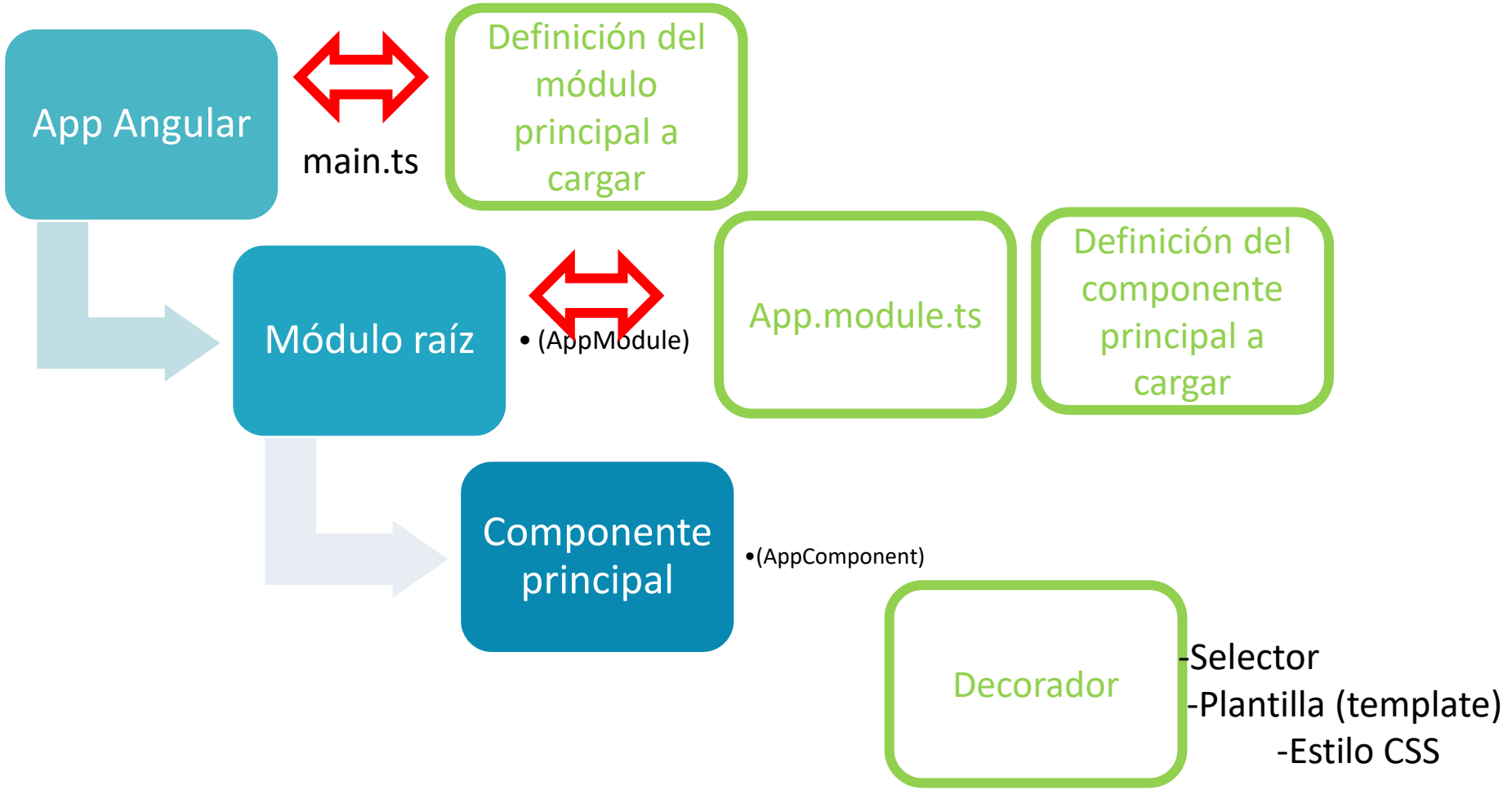




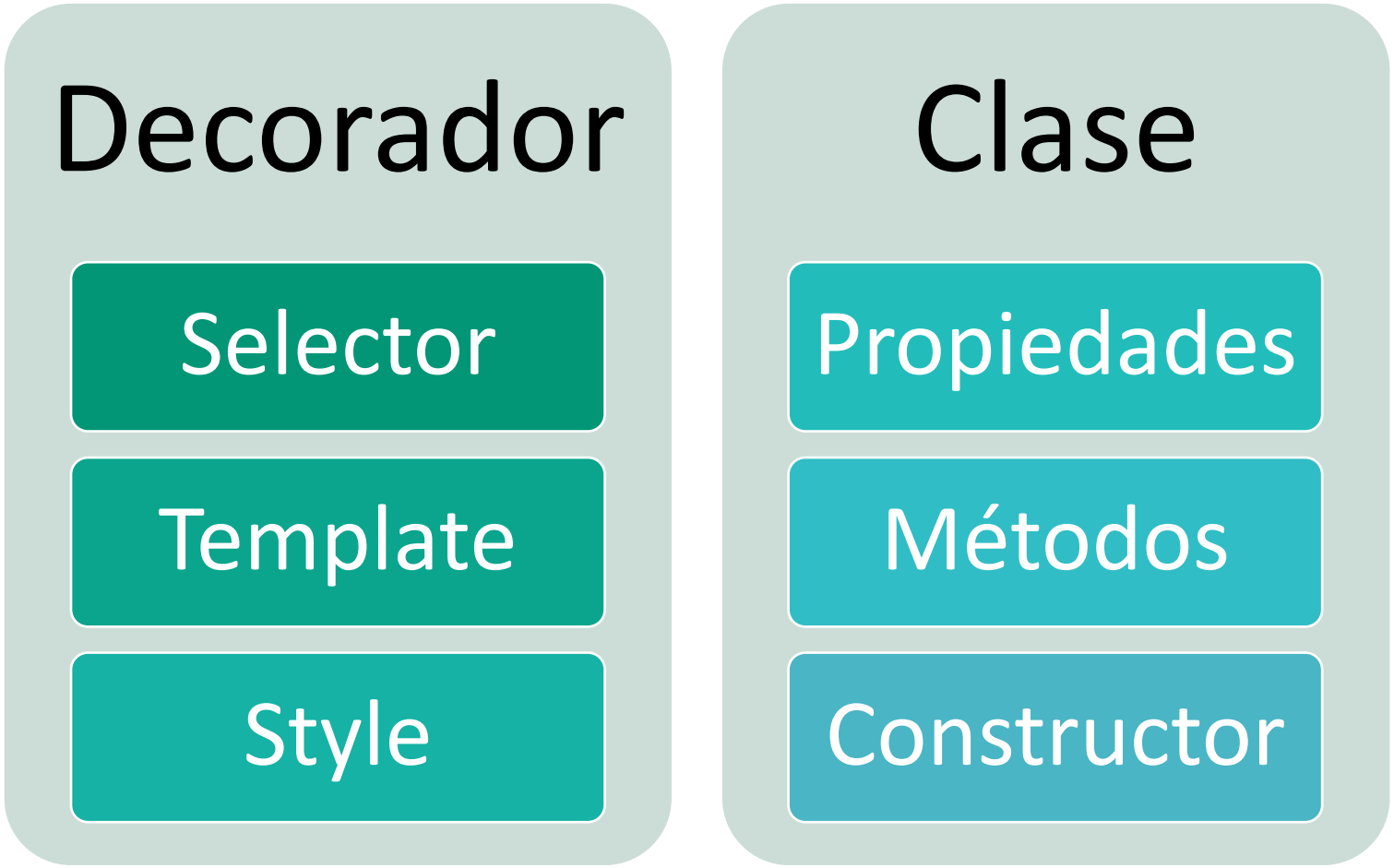
Tipos de terminales https://git-scm.com/downloads



Estructura y flujo



Componentes

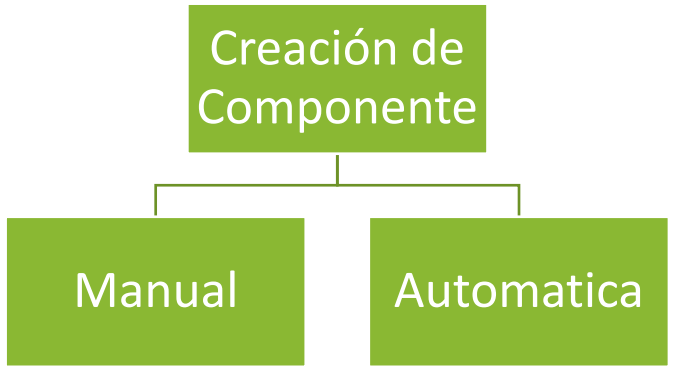


Comandos útiles

- Iniciar y apagar el servidor
  - Ng serve –open / ng serve -o
  - <ctrl><c>
- Crear un proyecto
  - Ng new <nombre proyecto>
- Abrir un proyecto en VS Code
  - File – open folder
- npm install bootstrap
  - Cambiar angular.json

```
],
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [


  "node_modules/bootstrap/dist/js/bootstrap.min.js"

]
```

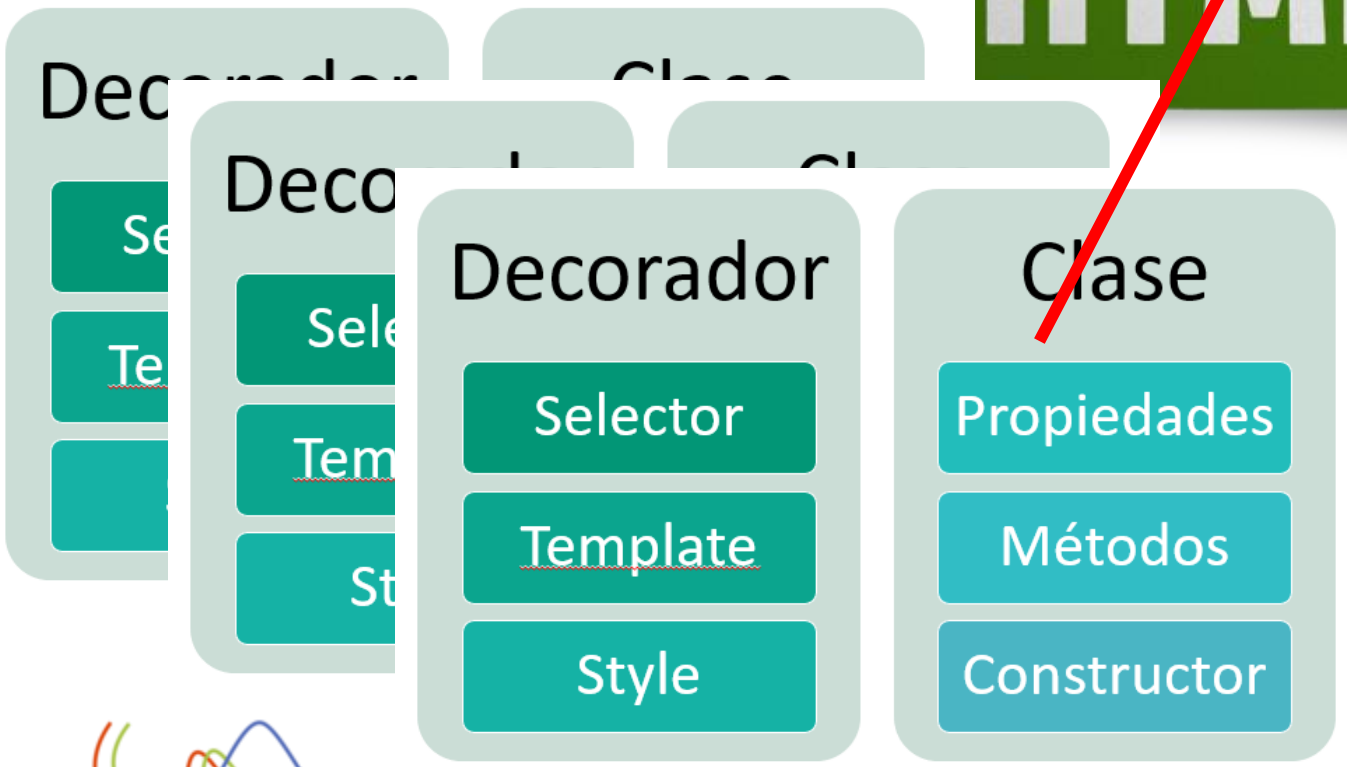
Creación de componentes



ng generate component <nombre>

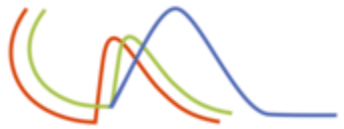
Comando para crear comando in line

ng generate component <nombre> -s –t
_ng g c <nombre> --skip-tests



Ejercicio

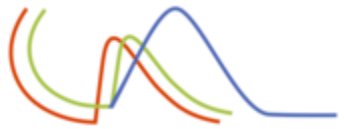
- Cambiar el selector del componente principal



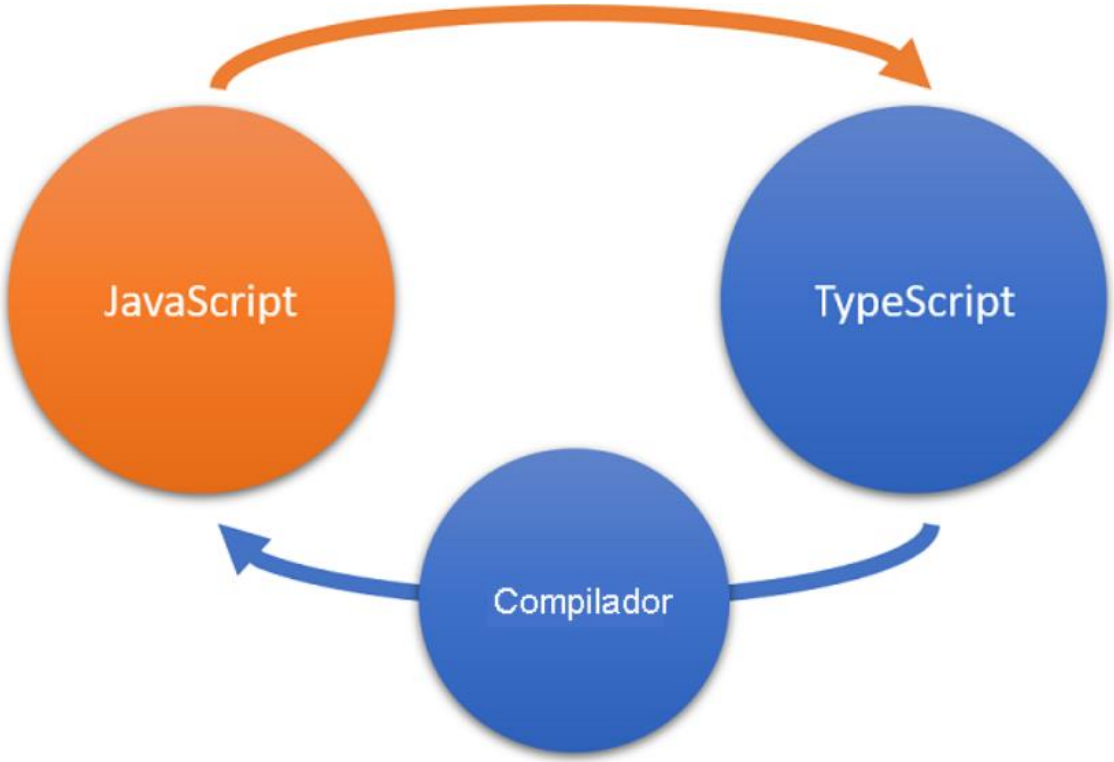

TypeScript - Microsoft

Que es TypeScript?

- TypeScript es un lenguaje de código abierto desarrollado por **Microsoft**. Se trata de un **supraconjunto** de JavaScript.
- La característica principal de TypeScript es su sistema de tipos. En TypeScript, puede identificar el tipo de datos de una variable o un parámetro mediante una *sugerencia de tipo*.
- Mediante la comprobación de tipos estáticos, TypeScript al principio del desarrollo detecta problemas de código que JavaScript normalmente no puede detectar hasta que el código se ejecuta en el explorador.



TypeScript



- Npm install –g typescript -> instala el compilador de typeScript
- Tsc –init -> genera fichero de configuración para la compilación
- Tsc -> compila todo el directorio
- Tsc <nombre-fichero> -> compila sólo el archivo
- Node .\build\<nombre-fichero>.js -> muestra el resultado en la consola.

Declaración de variables

- let x: number; //* Explicitly declares x as a number type
- let y = 1; //* Implicitly declares y as a number type
- let z; //* Declares z without initializing it

Declaración variables

Tipo boolean

- let flag: boolean;
- let yes = true;
- let no = false;

Tipos numéricos

- let x: number;
- let y = 0;
- let z: number = 123.456;
- let big: bigint = 100n;

cadena

- let s: string;
- let empty = "";
- let abc = 'abc';
- let firstName: string = "Mateo";
- let sentence: string = `My name is ${firstName}. I am new to TypeScript.`;
- console.log(sentence);
- My name is Mateo. I am new to TypeScript.

Enumeraciones

Enumeraciones
declaración

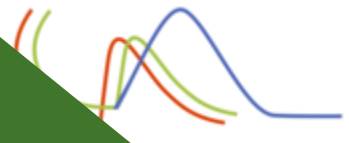
- enum ContractStatus { Permanent, Temp, Apprentice
- }

Uso de enumeraciones

- let employeeStatus: ContractStatus = ContractStatus.Temp;
- console.log(employeeStatus);

Asignar diferente valor

- enum ContractStatus {
- Permanent = 1,
- Temp,
- Apprentice
- }



Tipos any vs unknown

any

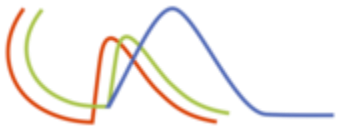
- let randomValue: any = 10;
- randomValue = 'Mateo'; // OK
- randomValue = true; // OK

unknown

- let randomValue: unknown = 10;
- randomValue = true;
- randomValue = 'Mateo';
-
- console.log(randomValue.name); // Error: Object is of type unknown
- randomValue(); // Error: Object is of type unknown
- randomValue.toUpperCase(); // Error: Object is of type unknown

Restricción de tipos

- let randomValue: unknown = 10;
-
- randomValue = true;
- randomValue = 'Mateo';
-
- if (typeof randomValue === "string") {
- console.log((randomValue as string).toUpperCase()); //* Returns MATEO to the console.
- } else {
- console.log("Error - A string was expected here."); //* Returns an error message.
- }



Matrices

Válido

- let list: number[] = [1, 2, 3];

Válido

- let list: Array<number> = [1, 2, 3];

Clases
class <nombre> { …}

Propiedades

```
_make: string;
_color: string;
_doors: number;
```

Constructores

Una clase puede contener como máximo una declaración constructor

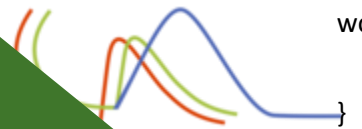
```
// Constructor
constructor(make: string, color: string, doors = 4) {
    this._make = make;
    this._color = color;
    this._doors = doors;
}
```

Descriptores de acceso

```
// Accessors
get make() {
    return this._make;
}
// Methods
set make(make) {
    this._make = make;
}
```

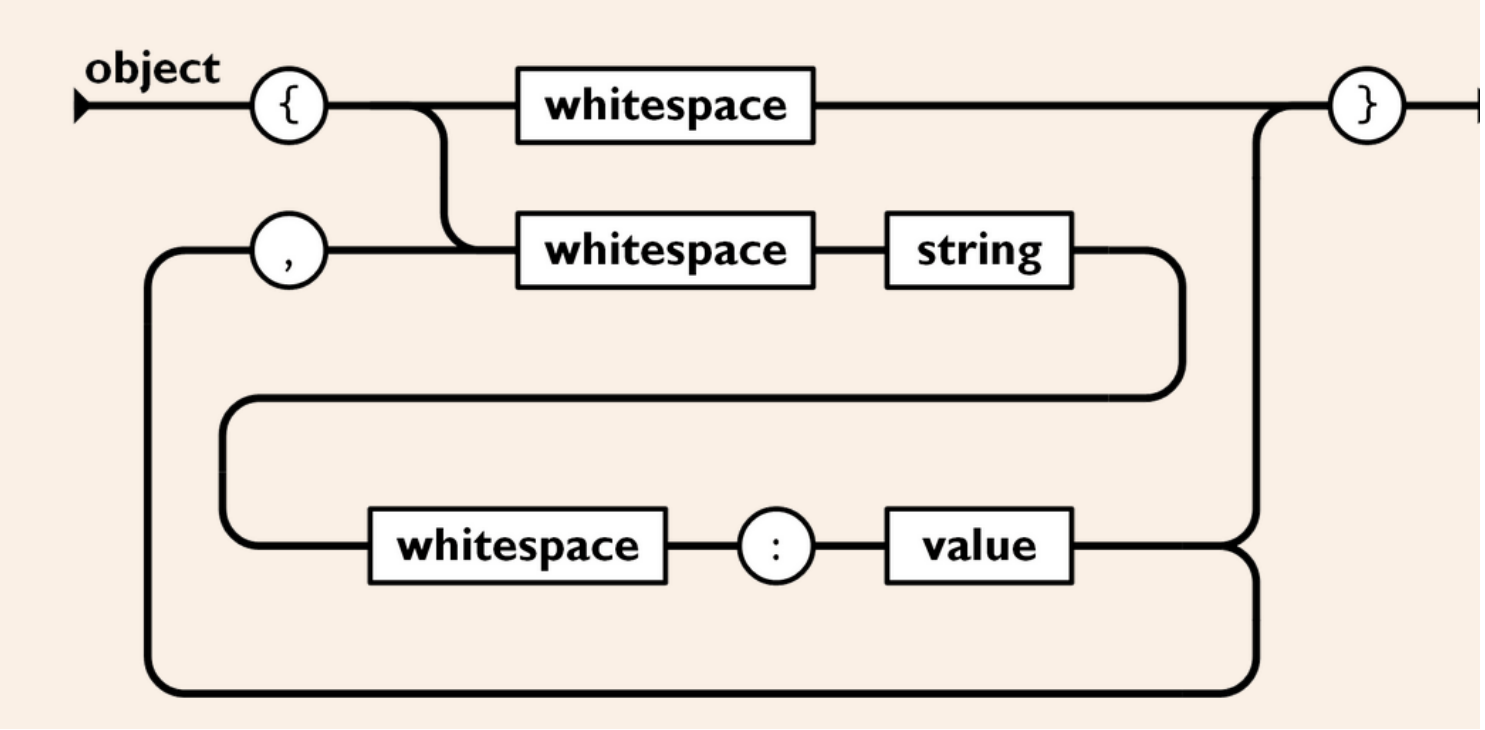
Métodos

```
brake(): string {
    return `${this.worker()} is braking with the standard braking system.`
}
turn(direction: 'left' | 'right'): string {
    return `${this.worker()} is turning ${direction}`;
}
// This function performs work for the other method functions
worker(): string {
    return this._make;
}
```



Json

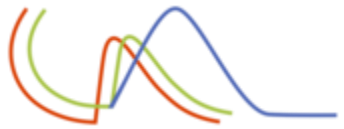
- Json.org



JavaScript

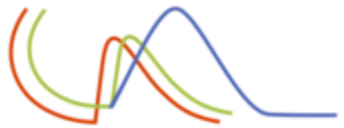
Un poco de historia …

- Creado en 1995 por Brendan Eich trabajando para la sociedad Netscape.
- Su objetivo original fue crear scripts (programas interpretados por el navegador) para manipular los datos del DOM (Document Object Model)
- Ha evolucionado:
  - Funcionalmente permitiendo el acceso asíncrono a los datos proporcionados por el servidor.
  - Del lado del servidor, con el entorno Node.js
- Programación funcional y POO



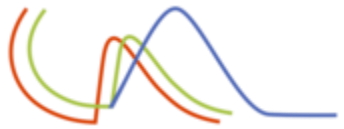
Utilización de JavaScript (no exhaustivo)

- Del lado del cliente
  - Gestionar eventos relacionados con una página HTML (controlar el contenido del formulario).
  - Acceder a los elementos del DOM, si es necesario, modificarlos con el API DOM, jQuery
  - Gestionar los flujos de datos asíncronos con el servidor a través de la arquitectura AJAX(Asynchronous JavaScript and XML, librería jQuery).
  - Gestionar una comunicación bidireccional y full-dúplex entre clientes y servidores.
  - Crear interfaces gráficas
  - Implementar una almacenamiento de datos en el sistema de archivos locales a través de los objetos JavaScript sessionSotrage y localStorage.



Librerías y frameworks

- Librerías
  - JQuery: permite acceder y modificar el DOM de manera muy sencilla, importar datos desde el servidor de manera asíncrona (a través de la arquitectura AJAX) y propone un buen conjunto de widgets gráficos (pestañas, desplegar acordeones, drag and drop).
  - D3.js y dc.js crear interfaces gráficas en 2D.
  - Three.js permite crear escenas 3D.
  - Bootstrap permite gestionar el web responsivo
- Frameworks
  - Polymer
  - React
  - Riot
  - Angular (Tiene una curva de aprendizaje grande)
- Sigue el estándar ECMAScript



Fundamentos Angular

Interpolación

- Combinar texto dinámico y texto estático
- Definir el valor en el componente

- ¿Cómo se utiliza?

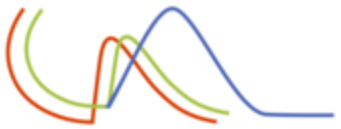
src / app / app.component.ts

```
currentCustomer = 'Maria';
```

Utilice la interpolación para mostrar el valor de esta variable en la plantilla de componente correspondiente:

src / app / app.component.html

```
<h3>Current customer: {{ currentCustomer }}</h3>
```

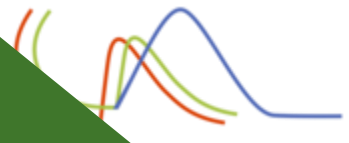


Ejemplo

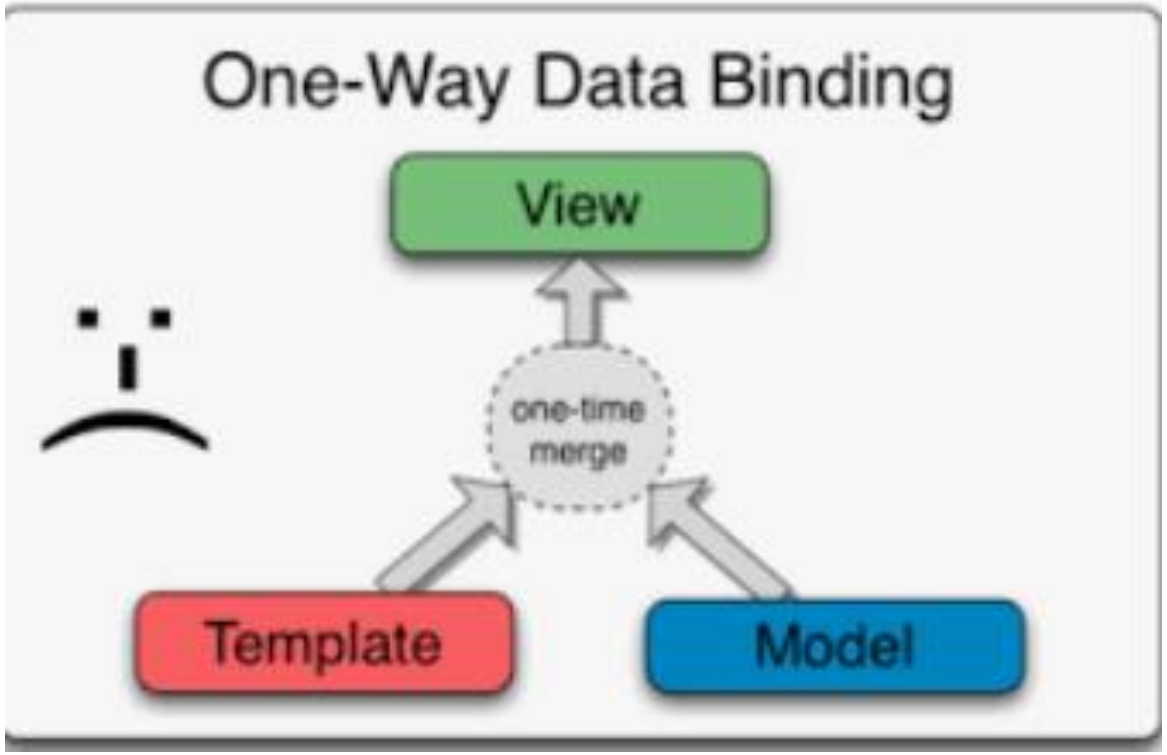
```typescript
export class FInterpolacionComponent implements OnInit {

  titulo:string="Hola mundo";

  constructor() { }

  ngOnInit(): void {
  }

  mostrarMensaje(){
    console.log(this.titulo)
  }

}
```

Go to component

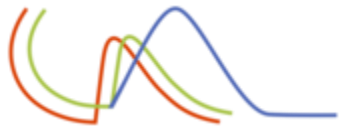
```html
<h1>PRINCIPIO DE INTERPOLACION</h1>

<p>Mostrar la información de un atributo o de un método en el HTML</p>
{{titulo}}
{{mostrarMensaje()}}
```



Property Binding



Sintaxis
[<propiedad>]=“nombre-variable”

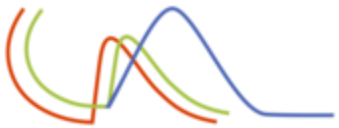


Ejemplo

```typescript
export class OneBindingComponent implements OnInit {

  marcado:boolean=true;

  constructor() { }

  ngOnInit(): void {
  }

  cambiarValor(){
    this.marcado=!this.marcado;
  }

}
```

Go to component

```html
<h1>Enlace de un solo sentido - One way binding</h1>
<p>Datos boolean : <input type="checkbox" [checked]="marcado"/> Acepta la politica de privacidad</p>
<p><button (click)="cambiarValor()">Haz clic para cambiar el valor del checkbox</button></p>
```



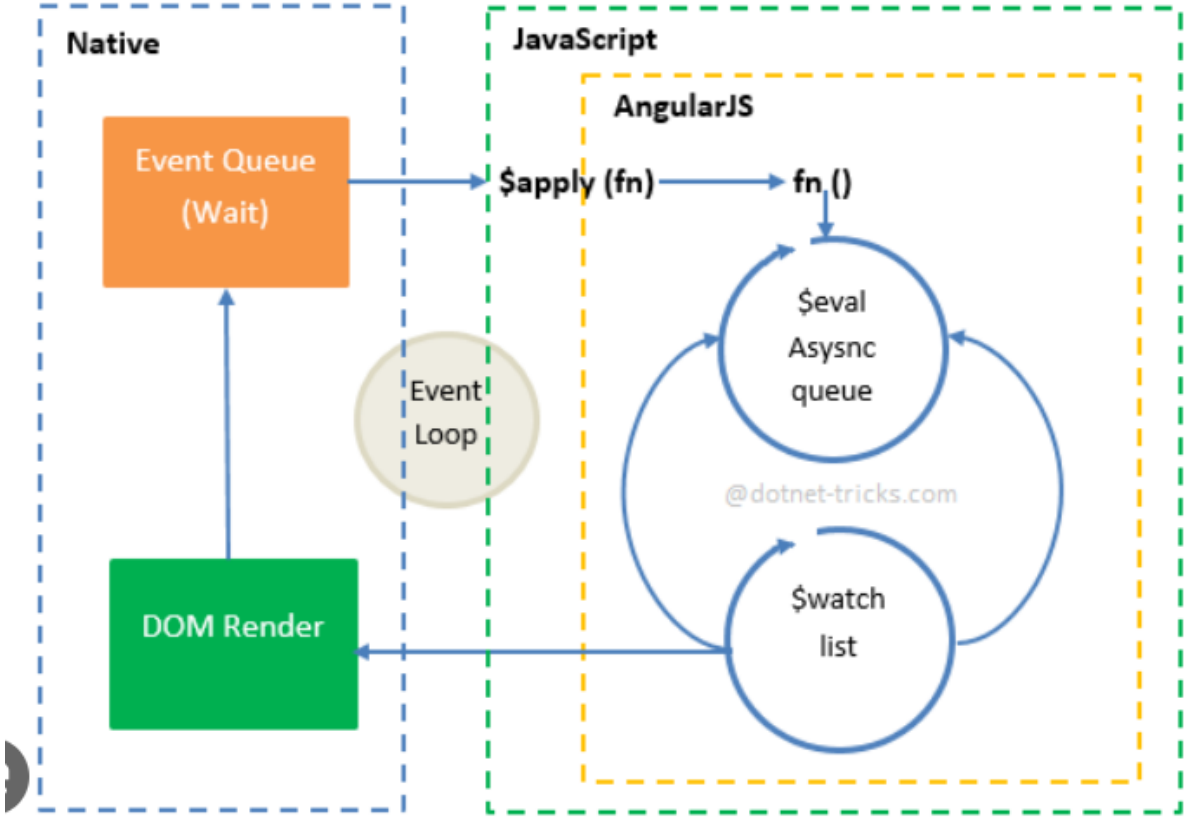
Eventos

- https://www.w3schools.com/angular/angular_events.asp

- `ng-blur`
- `ng-change`
- `ng-click`
- `ng-copy`
- `ng-cut`
- `ng-dblclick`
- `ng-focus`
- `ng-keydown`
- `ng-keypress`
- `ng-keyup`
- `ng-mousedown`
- `ng-mouseenter`
- `ng-mouseleave`
- `ng-mousemove`
- `ng-mouseover`
- `ng-mouseup`
- `ng-paste`

Event Binding

- Capturar ese objeto $event

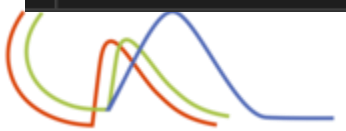


Sintaxis
(clic) = “método”

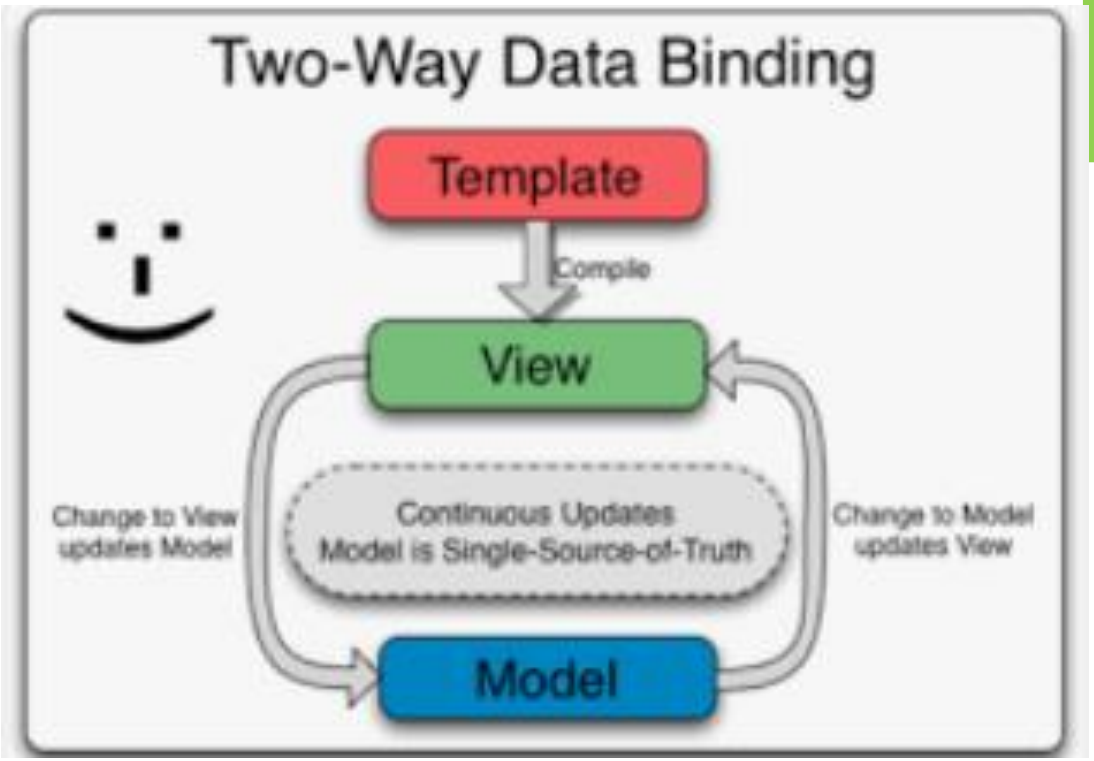
Ejemplo

```html
<h1>Eventos</h1>

<p>Recoger el valor de una caja de texto</p>
<p>DNI : <input type="text" placeholder="DNI" (keyup)="recogerValor($event)"></p>
<p><button (click)="mostrarValor()">Mostrar valor recibido </button></p>
```

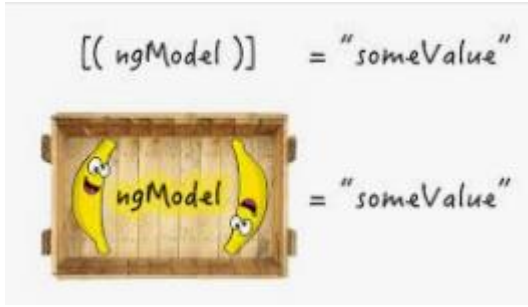
```typescript
export class EventosComponent implements OnInit {

  dato:string="";

  constructor() { }

  ngOnInit(): void {
  }

  recogerValor(e:any){
    console.log(e.target.value);
    this.dato = e.target.value;
  }

  mostrarValor(){
    alert("El valor recibido es "+ this.dato);
  }
```



Binding Bidireccional (Two way binding)



¿Qué se necesita?
Uso de directiva ngModel
FormsModule
Uso de Banana in Box
[()]



[(ngModel)] = "<atributo-ts>"

Ejemplo

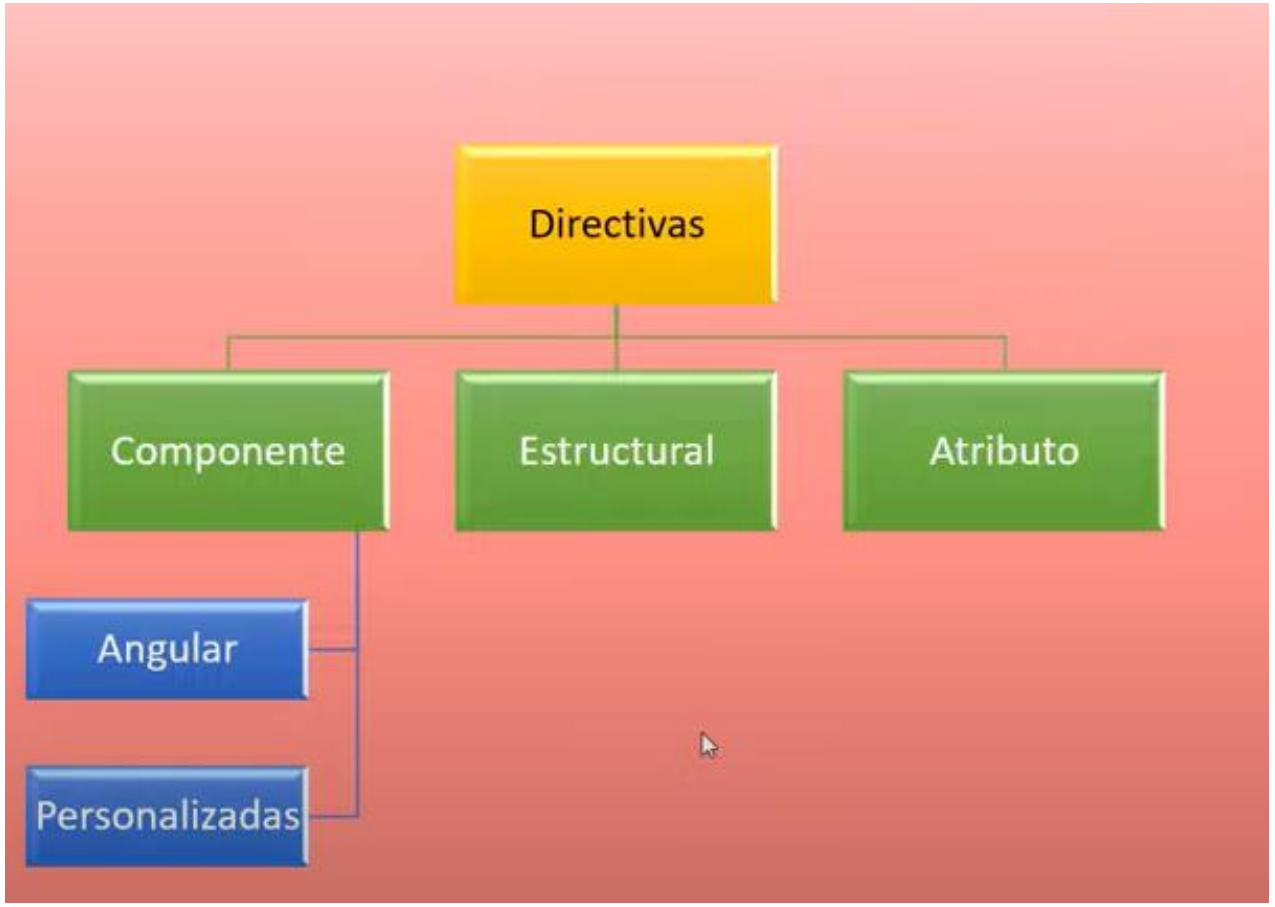
```ts
export class TwoWayBindingComponent implements OnInit {

  dni:string="";

  constructor() { }

  ngOnInit(): void {
  }
  mostrarValor(){
    alert("El valor es -> " + this.dni)
  }

}
```

```ts
@NgModule({
  declarations: [
    AppComponent,
    FInterpolacionComponent,
    EntradaDatosComponent,
    OneBindingComponent,
    EventosComponent,
    CalculadoraComponent,
    TwoWayBindingComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
```

```html
<h1>Two way binding</h1>

<p>Dato de entrada Two-way-binding <input type="text" [(ngModel)]="dni"/></p>

<p>Comprobación</p>
<button (click)="mostrarValor()">Mostrar valor recibido</button>
<p>Por interpolacion {{dni}}</p>
```

Directivas

- ¿Qué son?
- Elementos que se aplica a etiquetas HTML que añaden funcionalidad a la etiqueta donde se aplican
- ¿Para qué?
- Para modificar la estructura del DOM y apariencia



Directiva estructural

Componente Angular	• @Component • @Module
Estructural	• ngIf – else / ngFor / ngSwitch /ngPlural / ngTemplate /ngComponentOutlet
Atributos	• ngStyle / ngClass / ngModel

@if{}-@else{}

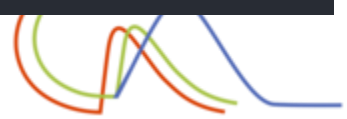
```ts
@Component({
  selector: 'app-usuario',
  standalone: true,
  imports: [RegistroComponent],
  templateUrl: './usuario.component.html',
  styleUrl: './usuario.component.css'
})
export class UsuarioComponent {

  estaLogeado: boolean = false;
  nombreUsuario: string ='Juan Lopez';

}
```

```html
<h1>Login</h1>
@if(estaLogeado){
    <p>Bienvenido a nuestra web {{nombreUsuario}}</p>
    <img src="https://github.com/TitaRuiz.png" alt="foto">
}@else {
    <p>Inicie sesion !!</p>
    <button (click)="estaLogeado=true">Iniciar Sesion</button>
    <app-registro></app-registro>
}
```

@for(elemento of <colección>;track índice)

```ts
@Component({
  selector: 'app-administrador',
  standalone: true,
  imports: [],
  templateUrl: './administrador.compon
  styleUrl: './administrador.component
})
export class AdministradorComponent {
  usuarios =[
    {id:1,nombre:'Laura Flores'},
    {id:2,nombre:'Mar Gonzalez'},
    {id:3,nombre:'Luis García'}
  ]

}
```

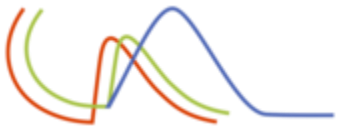
```html
<h1>Administrador</h1>
<h2>Usuarios que dependen de este administrdor</h2>
<ul>
    @for(elemento of usuarios; track elemento.id){
        <li>{{elemento.id}}   {{elemento.nombre}}</li>
    }@empty {
        <p>No hay datos que mostrar</p>
    }

</ul>
```



@switch(variable){ @case(valor){} @default{}}

```ts
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, UsuarioCompo
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  perfilUsuario:string = 'usuario';
}
```

```
@switch(perfilUsuario){
    @case('usuario'){
        <app-usuario></app-usuario>
    }
    @case('administrador'){
        <app-administrador></app-administrador>
    }
    @default {
        <p>Usuario sin perfil</p>
    }

}
```



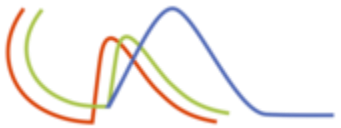
Ejemplo

```html
<h1>Directivas</h1>
<h2>Directiva *ngIf</h2>
<div *ngIf="incluido">
    <p>Este es un párrafo</p>
</div>
<button (click)="incluir()">Incluir / no incluir</button>

<h2>Directiva *ngFor</h2>
<div *ngFor="let elemento of tipoViviendas; let i = index">
    <p>{{elemento}} es el elemento {{i}}</p>
</div>

<h2>Directiva *ngSwitch</h2>
<div [ngSwitch]="estadoCivil">
    <p *ngSwitchCase="1">Su estado civil es casado</p>
    <p *ngSwitchCase="2">Su estado civil es divorciado</p>
    <p *ngSwitchCase="3">Su estado civil es soltero</p>
    <p *ngSwitchCase="4">Su estado civil es viudo</p>
    <p *ngSwitchDefault>Valor invalido</p>
</div>
```

```typescript
export class DirectivasComponent implements OnInit {
  incluido:boolean=false;
  tipoViviendas:string[]=["chalet Individual" , "chalet pareado", "piso", "atico"];
  estadoCivil:number=5;

  constructor() { }

  ngOnInit(): void {
  }

  incluir(){
    this.incluido=!this.incluido;
  }
```

*ngIf - else

```
<p *ngIf="registrado; else sinRegistrar">{{mensaje}} con el nombre de {{nombre}} {{apellido}}</p>

<ng-template #sinRegistrar>

    <p>Nadie registrado</p>


</ng-template>
```



*ngStyle *ngClass

```css
.variosEstilos{
    color: blue;
    text-decoration: underline;
    font-weight: bolder;
}
```

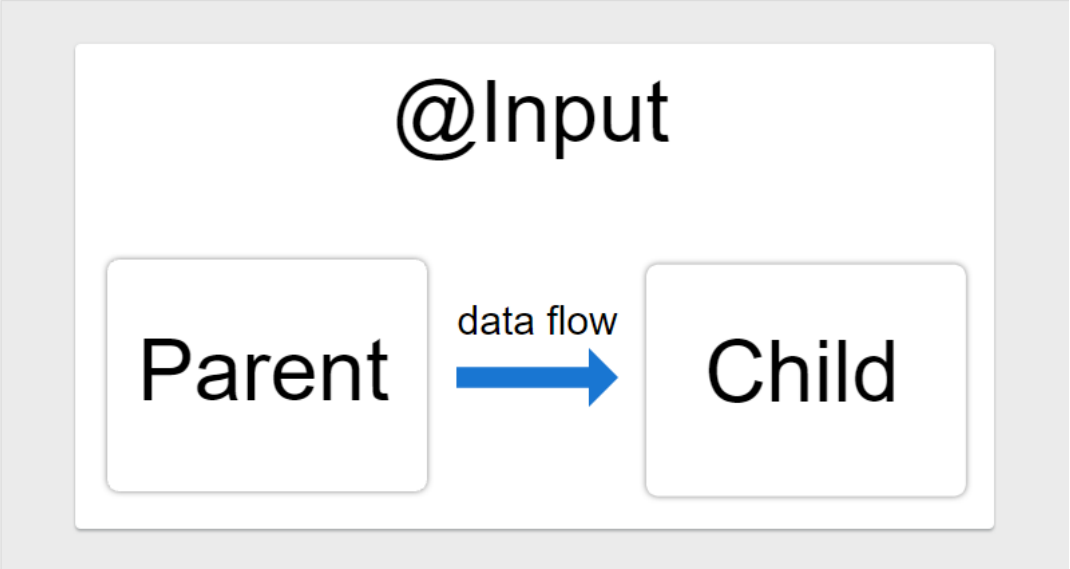
```ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-ej-directiva-ngif',
  templateUrl: './ej-directiva-ngif.component.html',
  styleUrls: ['./ej-directiva-ngif.component.css']
})
export class EjDirectivaNgifComponent implements
  titulo="Directivas de estructura - ngif "
  nombre="";
  contra="";
  mensaje="xxxxxx";
  mostrar=false;
  cargo="";

  loginUsuario(){
    this.mostrar=true;
    this.mensaje="Usuario registrado con exito,
  }

  constructor() { }

  ngOnInit(): void {
  }

}
```

```html
<label>Cargo : </label><input type="text" name="cargo" [(ngModel)]="cargo"/><br/>
<input type="submit" value="Registrarse" (click)= "loginUsuario()"/>

</form>
<br/>
<p *ngIf="mostrar; else sinRegistrar">{{mensaje}}</p>
<ng-template #sinRegistrar>
    <p>{{mensaje}}</p>
</ng-template>
<p>Uso de la directiva ngStyle</p>
<span [ngStyle]="{color:cargo=='director'?'red':'black'}">{{cargo}}</span>
<p>Uso de la directiva ngClass</p>
<span [ngClass]="{variosEstilos: cargo=='director'}">{{cargo}}</span>
```

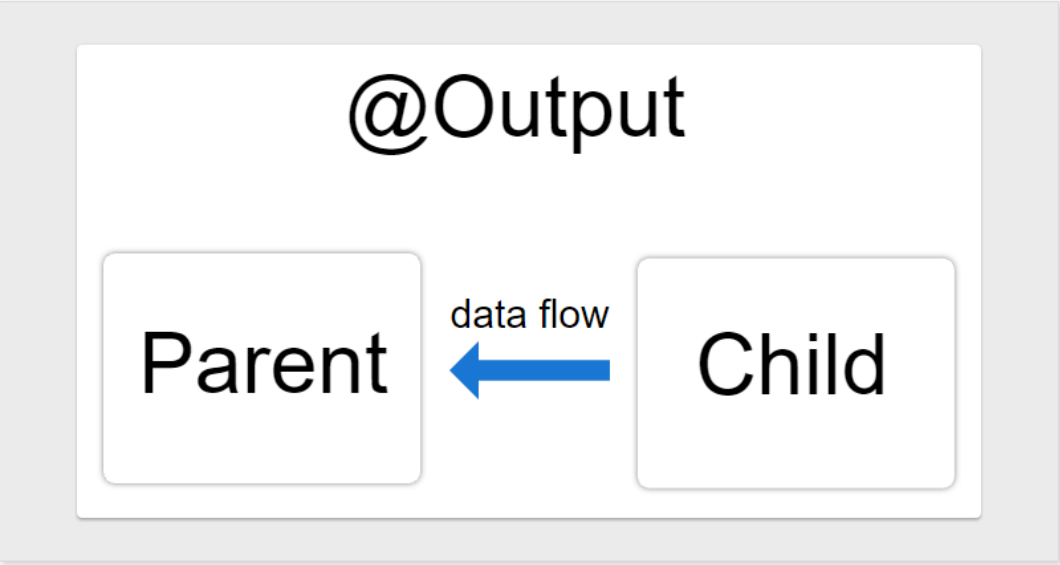
Comunicación entre componentes

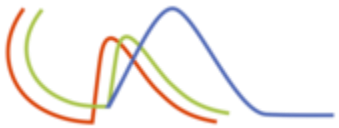
Comunicación entre componentes

- De padre a hijo con el decorador @Input



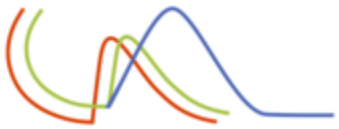
- De hijo a padre con el decorador @Output





Comunicación entre padre e hijo

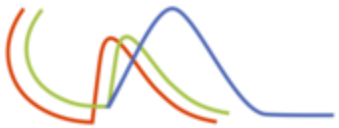
1. Declarar en la plantilla del padre dentro de la etiqueta del componente hijo la información que se va a comunicar
2. Declarar el decorador @Input() en la clase hija como any



Comunicación Padre-Hijo – (padre)

```ts
@Component({
  selector: 'app-padre',
  standalone: true,
  imports: [FormsModule,HijoComponent],
  templateUrl: './padre.component.html',
  styleUrl: './padre.component.css'
})
export class PadreComponent {
  dato:string='';


}
```

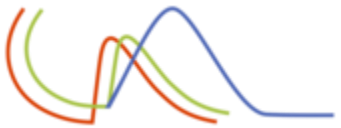
```html
<h1>Componente Padre</h1>
<div>
    <label >Dato a pasar al hijo</label>
    <input type="text" [(ngModel)]="dato">
    <br>
    <p>Dato escrito por el usuario</p>
    <app-hijo [datoHijo]="dato"></app-hijo>
</div>
```



Comunicación Padre-Hijo – (hijo)

```ts
@Component({
  selector: 'app-hijo',
  standalone: true,
  imports: [],
  templateUrl: './hijo.component.html',
  styleUrl: './hijo.component.css'
})
export class HijoComponent {

  @Input() datoHijo:any='';
  alumnos:string[]=[];

  agregar():void{
      this.alumnos.push(this.datoHijo);
  }

}
```

```html
<h2>Componente Hijo</h2>
<h2>Información recibida desde el padre</h2>
<button (click)="agregar()">Añadir alumno</button>
<p>{{datoHijo}} </p>
<h3>Datos del array</h3>
<ol>
@for(elemento of alumnos; track $index){
    <li>{{elemento}}</li>
}
</ol>
```



Comunicación entre hijo y padre

En el hijo

- En el TypeScript:
  - Crear un evento con el decorador @Output() de tipo EventEmitter
  - Crear un método que lance el evento, método emit()
- En el html
  - Un botón para que mande llamar el método en donde se crea el evento.

En el padre

- En el TypeScript
  - Método para recibir el dato
- En el html
  - Etiqueta del componente hijo con un evento(el creado en el hijo) y el método creado en el punto anterior.

Comunicación hijo - padre – (hijo)

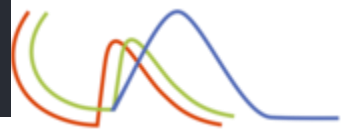
```
@Component({
  selector: 'app-hijo2
  standalone: true,
  imports: [FormsModul
  templateUrl: './hijo
  styleUrl: './hijo2.c
})
export class Hijo2Comp

  asignaturas:string[]
  "SQL","JS","JSON","Y
  asignaturaSeleccionada.string=this.asignaturas[0];
  @Output() eventoHaciaPadre= new EventEmitter();
  pasarPadre(){
    this.eventoHaciaPadre.emit(
      this.asignaturaSeleccionada);
  }
}
```

```
<div>
<label >Asignaturas</label>
<select class="form-select" [(ngModel)]="asignaturaSeleccionada">
    @for(elemento of asignaturas; track $index){
        <option >{{elemento}}</option>
    }
</select>

<button class="form-button" (click)="pasarPadre()">Seleccionar</button>
<br>
</div>
```

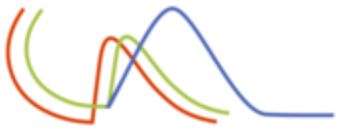
Comunicación hijo - padre – (padre)

```ts
@Component({
  selector: 'app-padre2',
  standalone: true,
  imports: [Hijo2Component],
  templateUrl: './padre2.component.html',
  styleUrl: './padre2.component.css'
})
export class Padre2Component {

  inscripcion: string[]=[];

  agregarAsignatura(elemento:string){

    this.inscripcion.push(elemento);
  }

}
```

```html
<h2>Inscripción al curso</h2>
<p>Asignaturas seleccionadas</p>
<ul>
    @for(elemento of inscripcion; track $index){
        <li>{{elemento}}</li>
    }
</ul>
<app-hijo2
(eventoHaciaPadre)="agregarAsignatura($event)">
</app-hijo2>
```

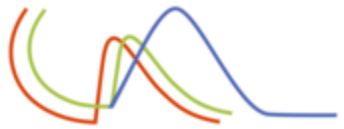


Servicios



Servicios

- No tienen decorador
- Se pueden utilizar desde cualquier componente
- Se inyectan en el constructor declarándolo como parámetro, así hace la inyección angular
- Se puede llamar un servicio dentro de otro servicio, con los mismos pasos.
  - Decorador @Injectable,
  - Inyectarlo en el constructor
  - usarlo

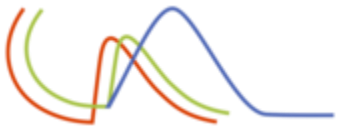


Ejemplo

```typescript
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ServicioMensajeService {

  muestraMensaje(mensaje:string){
    alert(mensaje);
  }

  constructor() { }
}
```

inyección

```typescript
import { Component, Input, OnInit } from '@angular/core';
import { ServicioMensajeService } from '../servicio-mensaje.service';
import { Empleado } from '../_modelo/empleado';

@Component({
  selector: 'app-hijo',
  templateUrl: './hijo.component.html',
  styleUrls: ['./hijo.component.css']
})
export class HijoComponent implements OnInit {

  constructor(private servicio:ServicioMensajeService) { }

  @Input() hEmpleado:any;
  @Input() hi:number=0;
  caracteristicas:string[] =[];
  elementos:string[] =[];

agregarEle to(nueva:string){
    this.servicio.muestraMensaje("Desde el servicio " + nueva)
    this.elementos.push(nueva);
  }
```

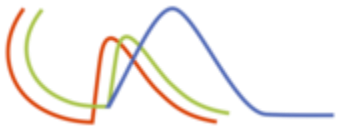



Routing

Routing

- ¿Qué es el routing?
  - Es el mecanismo que Angular tiene para la gestión de rutas internas.
  - Establece la correspondencia entre la ruta específica y un componente.

- ¿Cómo se crea ?
  - Añadir las rutas en el fichero app.routes.ts
  - En el app.configs.ts añadir el parámetro withDebugTracing()
  - Importar el RouterModule
  - Usar el atributo routerLink en la etiqueta a (en lugar del href)
  - Usar la etiquete router-outlet



Ejemplo paso 1 y 2

```
export const routes: Routes =
[
    {path:'arma', component:ArmaPCComponent},
    {path:'reserva', component:ReservaVuelosComponent},
    {path:'empleado', component:EmpleadoDinamicoComponent}

];
```

```
export const appConfig: ApplicationConfig = {

  providers: [provideRouter(routes, withDebugTracing())]

};
```

Paso 3.

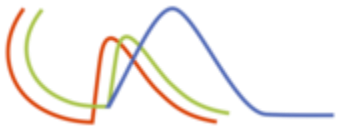
App.component.html

```ts
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterModule, RouterOutlet, E
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'htmldirectivas';
}
```

```html
<h1>Angular Router</h1>
<nav>
  <a routerLink="/arma"
  routerLinkActive="active"
  ariaCurrentWhenActive="page">Arma tu PC</a>
  <a routerLink="/reserva" routerLinkActive="a
  <a routerLink="/empleado" routerLinkActive="
</nav>
<router-outlet></router-outlet>
```

Función para navegar

```
agregar(){
  // this.servicio.muestraMensaje("Info : " + this.id);
  this.simulacion.agregarDesdeServicio(new Empleado(this.id, this.nombre, this.cargo, thi
  this.rutas.navigate([""])
}

constructor(private rutas:Router, private simulacion:SimulacionAccesoBBDDService) { }
```

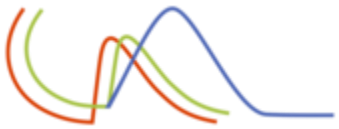


Pasar valor por la url

1. Preparar la ruta en el app.routing.ts
    1. {path:"quienes/:id", component:QuienesComponent},
2. Incluir en el app.config.ts el parámetro withComponentInputBinding()
3. Importar el RouterModule y El enlace debe tener la siguiente Propiedad
    1. <a [routerLink]="['/quienes',hi]“
4. Para recibir el valor hay que declararlo como Input()

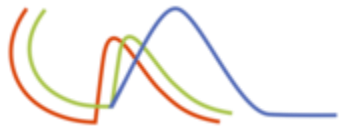
Paso de parámetro (sólo uno) .. Paso 1

```
export const routes: Routes = [
    {"path": "inmuebles", component:InmueblesComponent},
    {"path": "detalle/:id", component:DetalleInmuebleComponent},
    {"path": "", redirectTo: "inmuebles", pathMatch: "full"},
    {"path": "**", component:InmueblesComponent}
];
```



Paso de parámetro (sólo uno) .. Paso 2

```
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes, withComponentInputBinding())]
};
```



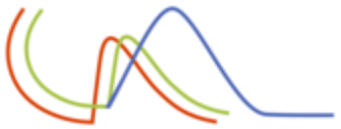
Paso de parámetro (sólo uno) .. Paso 3

```
<div>
    <table>
      <tr>
        <th>Inmuebles disponibles</th>

      </tr>
      @for(elemento of inmuebles; track elemento.referencia) {
      <tr>
        <a
          [routerLink]="['/detalle', elemento.referencia]"
        >
clic para agregar un punto de interrupción.  eferencia }}</td>

          <td><img  src="{{ elemento.img }}" alt="" /></td>
          <td>{{ elemento.descripcion }}</td>
        </a>
```

Paso de parámetro (sólo uno) .. Paso 4

```
export class DetalleInmuebleComponent implements OnInit{
  @Input() id:number = 0;

  ngOnInit(): void {
    console.log('id recibido ->'+ this.id)
  }
}
```

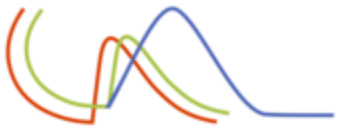
```
            <td>{{ inmuebles[id-1].referencia }}</td>
            <td width="50%"><img  src="{{ inmuebles[id-1].img }}" /></td>
            <td>{{ inmuebles[id-1].descripcion }}</td>
        </tr>
    </table>
```



Paso queryParams

```
<div class="form-group col-md-2">
    <a [routerLink]="['/quienes',hi]" [queryParams]="{accion:'1'}"> Actualizar</a>
</div>
```

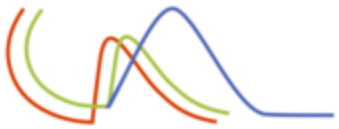
```
ngOnInit(): void {
  this.indice = this.activarRutas.snapshot.params['id'];
  let empleado: Empleado = this.simulacion.obtenerEmpleado(this.indice);
  this.acc = parseInt(this.activarRutas.snapshot.queryParams['accion']);
```



Ruta de error

```
const appRouters:Routes=[
{path:"", component:PadreComponent},
{path:"proyectos", component:ProyectosComponent},
{path:"quienes/:id", component:QuienesComponent},
{path:"**", component:ContactoComponent}
```

Capa presentación

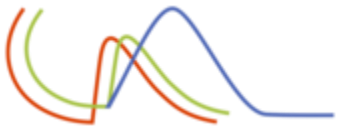


Opciones de framework de presentacion

Bootstrap

- Npm install bootstrap –save
  - Cambia el pakage.json automáticamente.
- Hay que incluir las librerías en el angular.json

```
,
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [


  "node_modules/bootstrap/dist/js/bootstrap.min.js"

]
```



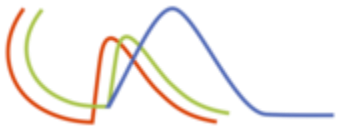
Prime ng

https://www.primefaces.org/primeng/calendar

1. Instalación
   - npm install primeng --save
   - npm install primeicons –save

2. En angular.json

```
"styles": [
    "node_modules/primeicons/primeicons.css",
    "node_modules/primeng/resources/themes/lara-light-blue/theme.css",
    "node_modules/primeng/resources/primeng.min.css",
    ...
]
```



Badge

A small value indicator that can be overlaid on another object.

Button toggle

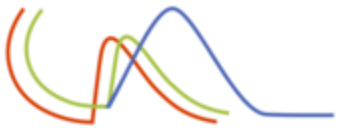
A groupable on/off toggle for enabling and disabling options.

AngularMaterial

https://material.angular.io/

¿Qué es Angular material?

- **Angular Material** es una librería de estilos (como Bootstrap) basada en la guía de diseño de **Material** Design, realizado por el equipo de **Angular** para integrarse perfectamente con **Angular**.
- https://material.angular.io/
- **Material Design ->** Material design es una normativa de diseño enfocado en la visualización del sistema operativo Android, además en la web y en cualquier plataforma. Fue desarrollado por Google y anunciado en la conferencia Google I/O celebrada el 25 de junio de 2014.
  - https://material.io/design

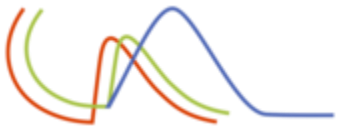


Cómo integrar un componente en el proyecto

Consultar la documentación

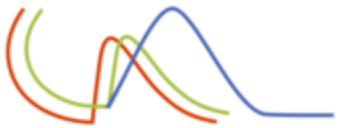
Copiar la api

Incluirla en el app.module

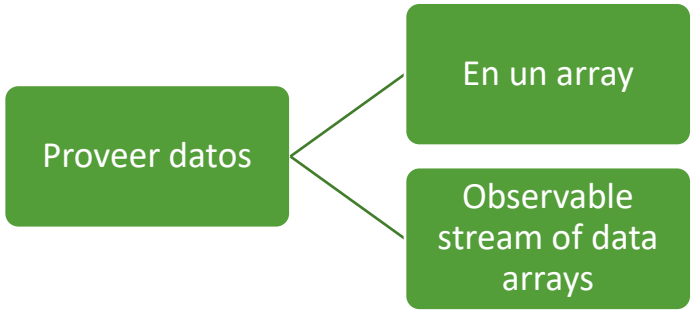


Angular material

- Para su instalación
  - ng add @angular/material



Table



https://material.angular.io/components/table/overview


Definir tabla y dataSource

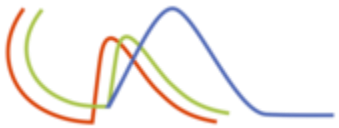

Definir las columnas (posible uso de template)


Paginacion


Filtrado

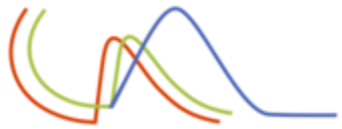

Clasificar


Seleccion



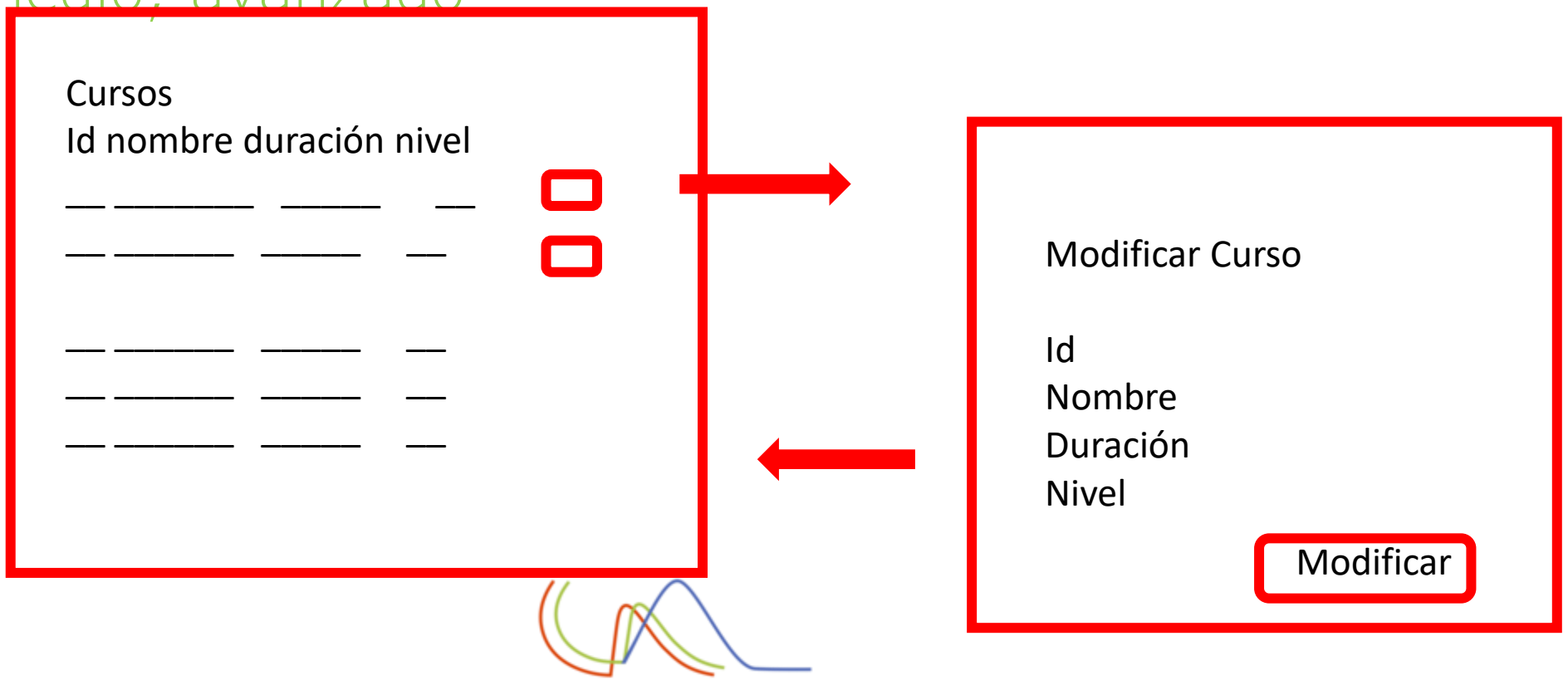
Objetivos del 19 de Abril

- Servicios en Angular
- Entregable es una CRUD – Factura
  - Idfactura
  - IdCliente
  - Totalfaltura
  - Iva
- Routing
  - Inicio – Quienes somos – Contactar – CRUD facturas



Objetivos del 20 de Abril

- Realizar un proyecto usand routing con paso de parámetros: servicio ,nivel una enumeración – iniciación, intermedio, avanzado



Objetivos del 20 de Abril

- Realizar usando Bootstrap su pagina web
  - Carrusel
  - Habilidades técnicas y suaves
  - Enlace al blog
  - Formulario de contacto
  - Dos artículos del blog