FRAME WORK ANGULAR

Contenido

INTRODUCCIÓN

Angular es un framework de desarrollo web front-end de código abierto mantenido principalmente por Google y una comunidad de desarrolladores individuales y corporaciones.

Utiliza el patrón **MVC** (Modelo vista controlador). Está diseñado para facilitar la creación de aplicaciones web de una sola página (SPA) y aplicaciones móviles a través de una arquitectura robusta y modular. Se asocia de forma natural a un servidor node.js.

Angular se utiliza a menudo junto con otras tecnologías como TypeScript (un supraconjunto de JavaScript), RxJS y varias herramientas de compilación. Es conocido por su robustez, escalabilidad y mantenibilidad, lo que lo convierte en una opción popular para crear aplicaciones web complejas.

CARACTERISTICAS

A continuación, se muestran algunas características y conceptos clave asociados con Angular:

1. **ARQUITECTURA BASADA EN COMPONENTES**: Las aplicaciones angulares se crean alrededor de componentes, que son fragmentos de interfaz de usuario reutilizables y autónomos que controlan una parte de la interfaz de usuario. Los componentes son los elementos básicos de una aplicación Angular.

2. CLI (INTERFAZ DE LÍNEA DE COMANDOS): Angular CLI es una poderosa interfaz de línea de comandos que ayuda a crear, construir y administrar aplicaciones Angular.

3. PLANTILLAS: Angular utiliza plantillas HTML para definir la interfaz de usuario de una aplicación. Estas plantillas pueden incluir una sintaxis específica de Angular para cosas como interpolación, enlace de datos y manejo de eventos.

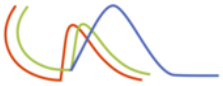
4. FORMULARIOS: Angular ofrece un conjunto de herramientas y técnicas para manejar formularios, incluidos formularios basados en plantillas y formularios reactivos.

5. ENLACE DE DATOS (Data Binding): Angular permite el enlace de datos bidireccional (two-way data binding), lo que significa que los cambios en el modelo se reflejan automáticamente en la vista y viceversa, así como one-way binding para propiedades y eventos.

5. **DIRECTIVAS**: Angular tiene un amplio conjunto de directivas como ngIf, ngFor y ngSwitch, @if{} - @else{}@for(..of …), que amplían HTML con funcionalidad adicional. Le permiten manipular el DOM según el estado de la aplicación.

6**. ENRUTAMIENTO**: Angular proporciona un sistema de enrutamiento sólido que le permite crear aplicaciones de varias páginas (MPA) dentro de una aplicación de una sola página (SPA).

7. **SERVICIOS**: los servicios son objetos únicos que se pueden utilizar para encapsular lógica, compartir datos entre componentes y realizar tareas que no pertenecen a ningún componente específico.



8. **INYECCIÓN DE DEPENDENCIAS**: Angular tiene un poderoso sistema de inyección de dependencias que le permite administrar eficientemente las dependencias entre diferentes partes de su aplicación.

9. **OBSERVABLES** y RxJS: Angular aprovecha RxJS, una poderosa biblioteca para manejar operaciones asincrónicas. Los observables son una parte clave de esto, ya que permiten la gestión de eventos y flujos de datos.

10**. HTTPCLIENT**: Angular incluye un módulo llamado HttpClient para realizar solicitudes HTTP, lo que le permite comunicarse con API y recuperar o enviar datos.

11. **MÓDULOS**: las aplicaciones angulares se organizan en módulos, que son contenedores lógicos para diferentes partes de una aplicación. Los módulos ayudan a organizar el código y gestionar las dependencias.

12. **PRUEBAS**: Angular tiene un marco de pruebas sólido y fomenta la escritura de pruebas unitarias para componentes, servicios y otras partes de una aplicación.

13. **INTERNACIONALIZACIÓN (I18N)**: Angular proporciona herramientas para crear aplicaciones que se pueden localizar fácilmente para diferentes idiomas y regiones.

14**. APLICACIONES WEB PROGRESIVAS (PWA):** Angular tiene soporte integrado para crear aplicaciones web progresivas, que son aplicaciones web que brindan una experiencia similar a una aplicación nativa

ARQUITECTURA WEB SPA CON REST API

ARQUITECTURA CLIENTE-SERVIDOR:El frontend (Angular) actúa como el cliente, y el backend (Spring con API RESTful) actúa como el servidor. El cliente se ejecuta en el navegador del usuario, y el servidor procesa las solicitudes del cliente, maneja la lógica de negocio, accede a la base de datos y devuelve los datos en formato JSON (o similar) a través de una API RESTful.

API RESTful: El backend en Spring expone una API REST (Representational State Transfer) que el frontend consume. Este patrón implica la comunicación a través de peticiones HTTP y el intercambio de datos en formato JSON, donde el frontend solicita recursos y el backend los entrega o los manipula según sea necesario.

SINGLE PAGE APPLICATION (SPA):En el frontend, Angular sigue el patrón de una SPA (Single Page Application). Esto significa que el contenido de la aplicación se carga dinámicamente en una sola página web sin necesidad de recargar la página entera, mejorando la experiencia de usuario y la rapidez de la navegación.

SEPARACIÓN DE RESPONSABILIDADES (Frontend/Backend): El frontend (Angular) se encarga de la presentación y la lógica de la interfaz de usuario. El backend (Spring) se encarga de la lógica de negocio, seguridad y acceso a la base de datos, proporcionando datos a través de una API REST.

ARQUITECTURA SPA

Características:

.

1. **INTERACTIVIDAD MEJORADA**: Las SPAs pueden proporcionar una experiencia de usuario más fluida y rápida, ya que solo se cargan los datos necesarios en lugar de volver a cargar toda la página.
2. **NAVEGACIÓN** Sin Recargas: La navegación entre diferentes vistas o secciones de la aplicación se maneja a través de JavaScript, lo que evita la recarga completa de la página y mejora la velocidad de respuesta.
3. **CARGA DINÁMICA**: En lugar de cargar múltiples páginas, una SPA carga una única página HTML y actualiza el contenido dinámicamente a medida que el usuario interactúa con la aplicación. Esto se hace a través de solicitudes asíncronas, normalmente utilizando AJAX o Fetch API.
4. **USO DE FRAMEWORKS**: Muchas SPAs son desarrolladas usando frameworks o bibliotecas de JavaScript como Angular, React, o Vue.js. Angular, en particular, es un framework muy popular para construir SPAs debido a su robustez y características como el enlace bidireccional de datos y el manejo de rutas.

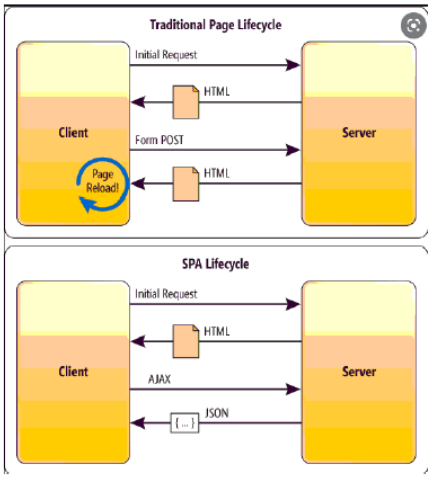


Ilustración 1:Arquitectura SPA

VENTAJAS

1. Extender el lenguaje HTML gracias a directivas muy potentes, principalmente designando los componentes por medio de etiquetas.
2. Enlace de datos bidireccional: Los cambios realizados en la interfaz de usuario se reflejan automáticamente en el modelo de datos y viceversa
3. Asociar operaciones a los flujos de datos asíncronos (por medio de programación reactiva).
4. Activar las diferentes vistas de interfaz, gracias a un router extremadamente perfeccionado, que vincula la selección de rutas a la activación de componentes.
5. Crear aplicaciones perfectamente modulares (se basan en reglas para administrar los módulos Node.js).

ELEMENTOS

Al crear un proyecto en Angular, se utilizan varios elementos y conceptos fundamentales que forman parte de la estructura y funcionalidad de la aplicación. Como:

COMPONENTES (Components): Los componentes son las piezas fundamentales de una aplicación Angular. Cada componente se compone de un archivo de clase TypeScript, un archivo HTML (plantilla) y un archivo CSS (estilos). Uso: Definen la vista y la lógica de una parte de la interfaz de usuario. Cada componente puede ser reutilizado en diferentes partes de la aplicación.

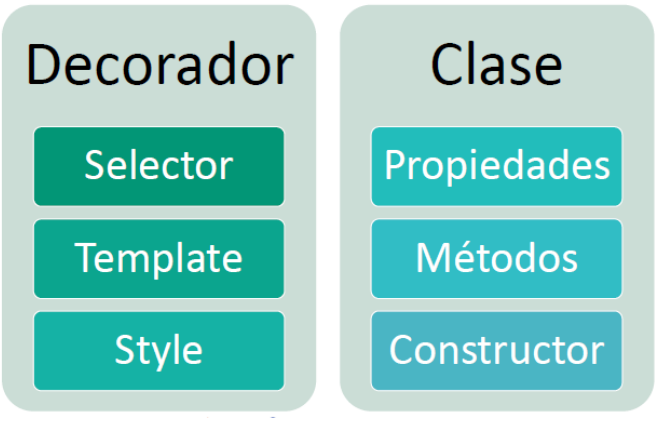


Ilustración 2-Elementos Componente

RUTAS (Routing) Descripción: El enrutamiento en Angular permite navegar entre diferentes vistas o componentes en una SPA. Se define a través del módulo de enrutamiento, donde se especifican las rutas y sus componentes asociados. Esto permite a los usuarios navegar por la aplicación sin recargar la página.

SERVICIOS (Services). Los servicios son clases que encapsulan la lógica de negocio y permiten compartir datos y funcionalidades entre diferentes componentes. Se utilizan para manejar tareas como la interacción con APIs, la gestión de datos, y la lógica de negocio, promoviendo la reutilización y separación de funcionalidades.

MÓDULOS (Modules): Un módulo en Angular es un contenedor para un conjunto de componentes, directivas, y servicios relacionados. Cada aplicación Angular tiene al menos un módulo, llamado AppModule, que es el módulo raíz. Facilitan la organización del código y la carga perezosa (lazy loading) de diferentes partes de la aplicación.

ESTRUCTURA DEL PROYECTO

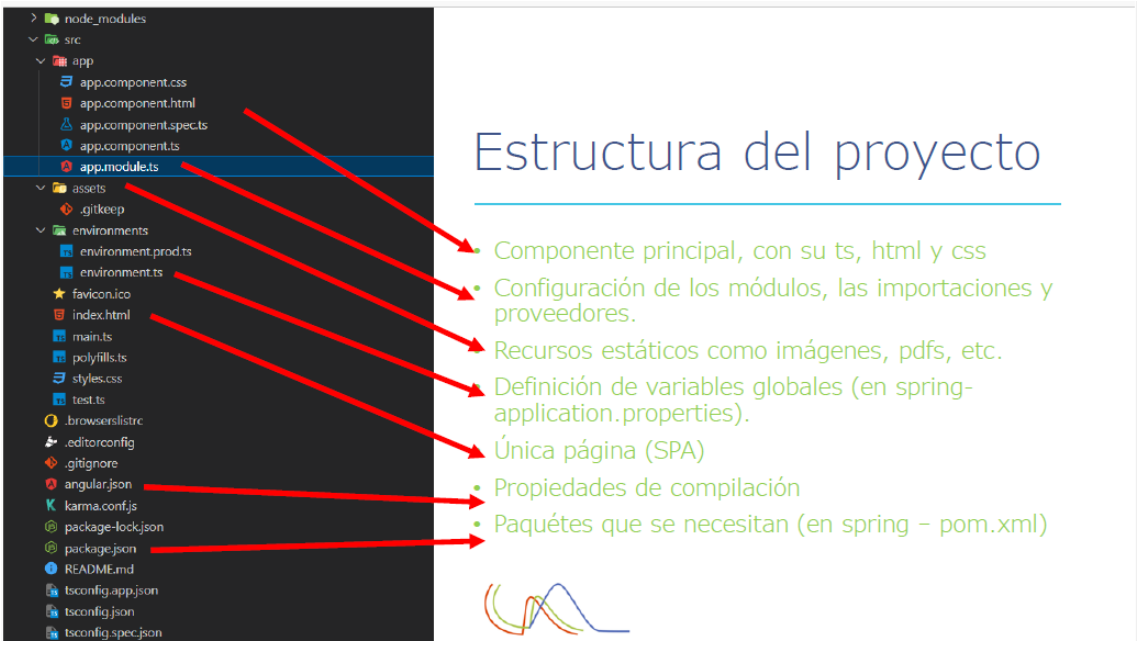


Ilustración 3 -Estructura del proyecto

INTERPOLACIÓN

La **interpolación** en Angular es una técnica que permite enlazar datos desde el componente a la vista HTML, insertando dinámicamente el valor de una propiedad en el contenido de la plantilla. Se utiliza mediante la sintaxis de dobles llaves {{ }}, donde se coloca una expresión que Angular evalúa y el resultado se muestra en el HTML. Esto es especialmente útil para mostrar valores dinámicos, como variables o el resultado de funciones dentro del componente, directamente en la interfaz de usuario.

La interpolación en Angular no solo admite la visualización de datos simples, como cadenas o números, sino que también permite evaluar expresiones complejas, como operaciones matemáticas o llamadas a métodos. Angular se encarga de detectar cambios en las propiedades enlazadas y actualizar automáticamente la vista cuando el modelo cambia, lo que garantiza una interfaz de usuario siempre sincronizada con los datos del componente.

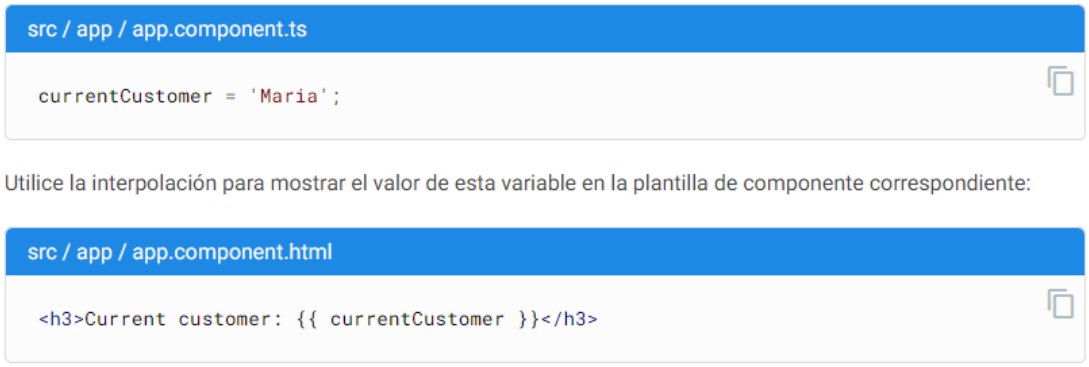


Ilustración 4 - Interpolación

ENLACE (BINDING)

BINDING en Angular es la técnica que permite sincronizar los datos entre el componente y la vista (HTML) de una aplicación. Existen varias formas de binding, dependiendo de la dirección de la comunicación de los datos:

1. one-way binding,
2. binding de eventos
3. two-way binding

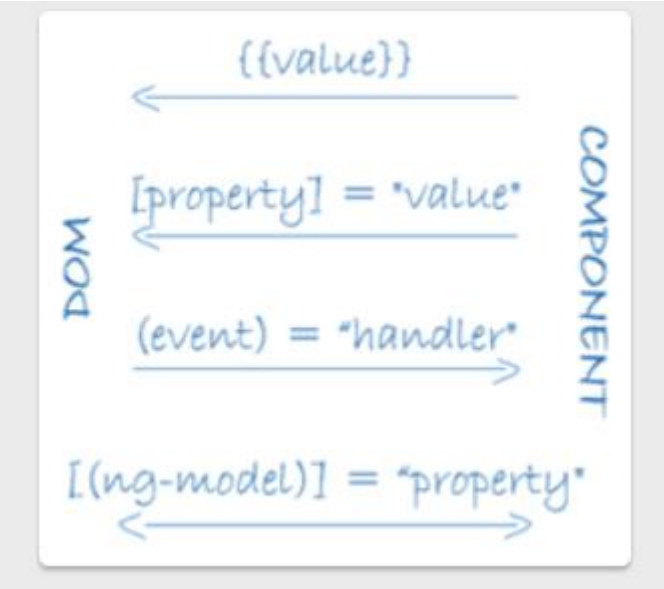
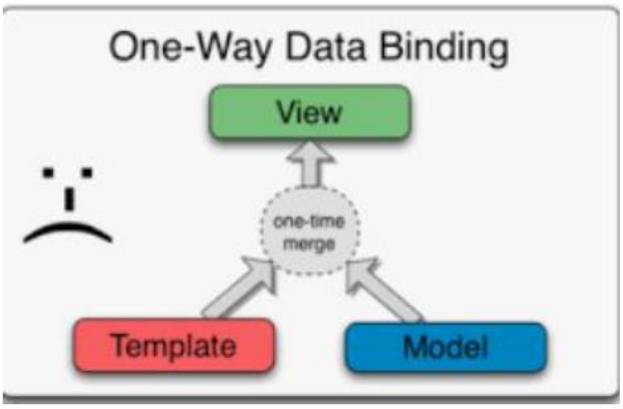


Ilustración 5 - Tipos de Enlace

ONE-WAY BINDING (ENLACE UNIDIRECCIONAL) Es cuando los datos fluyen en una sola dirección, desde el componente hacia la vista o viceversa:

- **Desde el componente hacia la vista**: Es el caso más común, donde los datos del componente (modelo) se muestran en la vista (plantilla) y no hay retroalimentación desde la vista al componente. La interpolación {{ }} es un ejemplo de **one-way binding** desde el componente hacia la vista. Otra forma es usar **property binding**, donde se enlaza directamente una propiedad del HTML a un valor del componente usando la sintaxis [propiedad]="valor".



Sintaxis
[<propiedad>]="nombre-variable"

Ilustración 6 - One way Binding

- **Desde la vista hacia el componente**: Esto se conoce como **event binding** y permite que los eventos en la vista, como un clic o una entrada de texto, envíen información al componente. La sintaxis es (evento)="método ()".

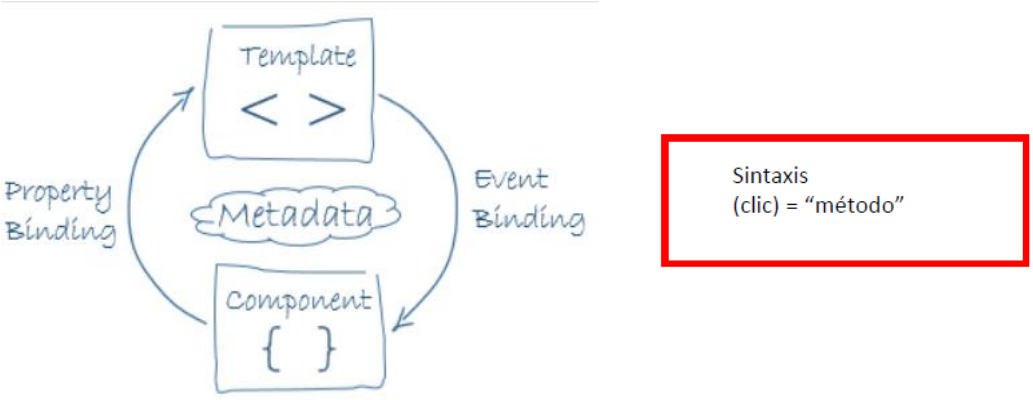


Ilustración 7 -Eventos

- **TWO-WAY BINDING** (Enlace Bidireccional).Permite la sincronización de datos en ambas direcciones: cuando un valor en el componente cambia, la vista se actualiza, y cuando un valor en la vista cambia (como en un input), el componente también se actualiza. En Angular, el two-way binding se logra con la directiva especial [(ngModel)].

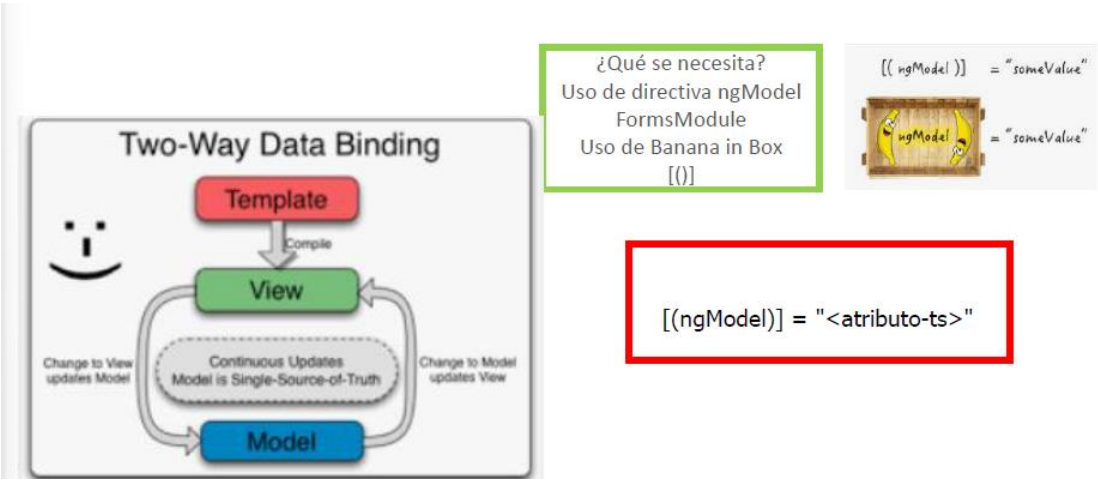


Ilustración 8 -Two way binding

DIRECTIVAS

En Angular, las directivas son instrucciones que extienden el comportamiento de los elementos HTML. Se dividen en tres grupos principales: directivas estructurales, directivas de atributos y directivas personalizadas.

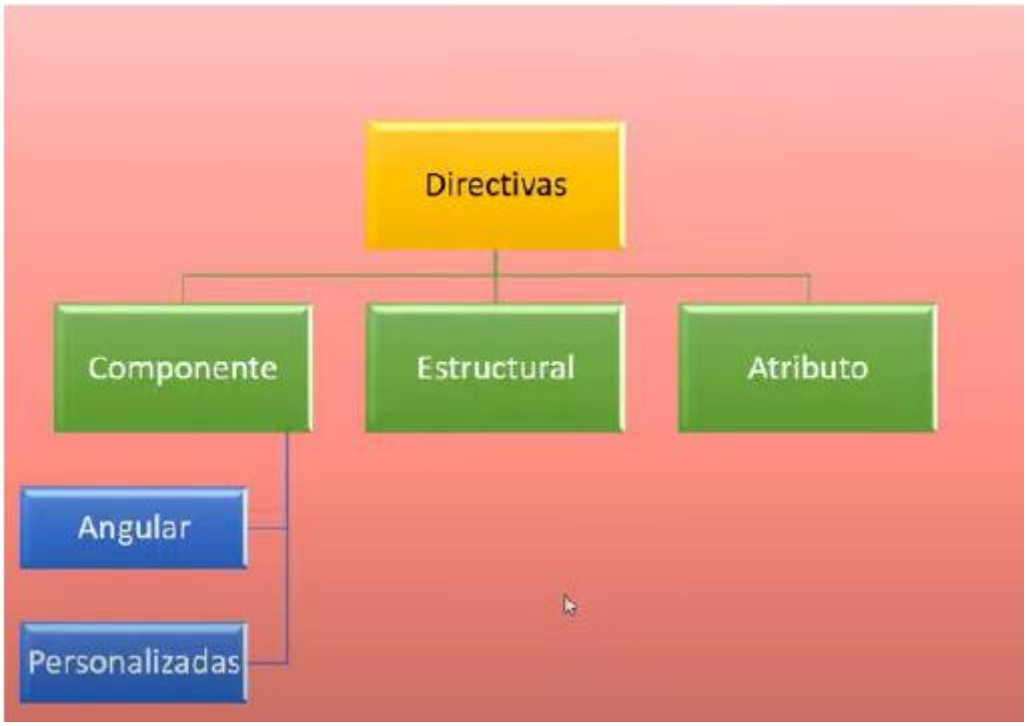


Ilustración 9 - Directivas

DIRECTIVAS DE COMPONENTE: Un componente en Angular es, de hecho, una directiva que controla una vista. Las directivas de componentes son el bloque fundamental de una aplicación Angular. Un componente define una clase en TypeScript, un template HTML para la vista y un conjunto de estilos. A diferencia de otras directivas, los componentes siempre tienen una plantilla asociada, que se renderiza en la interfaz de usuario. Los componentes manejan la interacción entre el modelo y la vista, y son esenciales para crear interfaces de usuario modulares y reutilizables.

DIRECTIVAS ESTRUCTURALES: Este tipo de directivas cambia la estructura del DOM agregando o eliminando elementos. Modifican el contenido de la página al insertar, eliminar o modificar secciones completas del DOM. Las directivas estructurales más comunes son *ngIf, *ngFor, y *ngSwitch. @If{}@else{}

Ejemplo:

*ngIf - @if{} @else{}muestra o oculta un bloque HTML basado en una condición.

*ngFor - @for(elemento of <colección>; track índice) repite un bloque de HTML para cada elemento de una colección.

*ngSwitch - @switch(variable){@case(valor){} @default{}} -evalúa una expresión y muestra uno de varios elementos posibles basados en el valor.

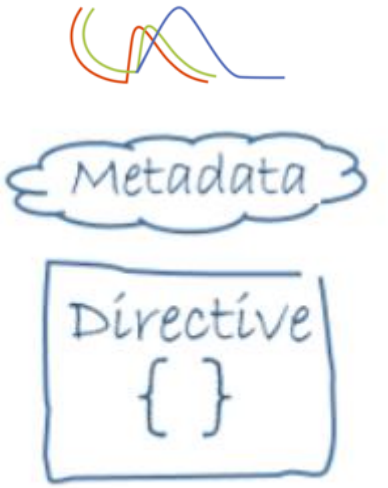


Ilustración 10 - Directivas

DIRECTIVAS DE ATRIBUTOS: Las directivas de atributos modifican el comportamiento o la apariencia de un elemento ya existente, sin alterar la estructura del DOM. Actúan como decoradores que añaden funcionalidades a los elementos en los que se aplican. Un ejemplo clásico es ngClass para gestionar clases CSS dinámicas y ngStyle para manejar estilos CSS dinámicamente.

Ejemplo:

ngClass aplica o elimina clases CSS basadas en condiciones.

ngStyle cambia estilos en línea del elemento de acuerdo con la evaluación de expresiones.

DIRECTIVAS PERSONALIZADAS: Las directivas personalizadas permiten crear comportamientos específicos que no están cubiertos por las directivas integradas. Los desarrolladores pueden definir sus propias directivas para modificar el comportamiento de cualquier elemento o componente de Angular, ya sea para añadir lógica de manipulación de eventos o modificar dinámicamente propiedades HTML.

Ejemplo: Puedes crear una directiva que cambie el color de fondo de un elemento cuando el mouse pase sobre él, o que implemente una funcionalidad personalizada según las necesidades de tu aplicación.

Estas directivas permiten que Angular sea muy flexible y extienda el comportamiento del HTML de manera dinámica y programática.

COMUNICACIÓN ENTRE COMPONENTES

En Angular, la comunicación entre componentes se realiza mediante la interacción entre componentes padre e hijo. Hay dos direcciones principales para esta comunicación: Padre a Hijo y Hijo a Padre.

1. **COMUNICACIÓN PADRE A HIJO**: El componente padre puede pasar datos al componente hijo utilizando el decorador @Input(). Esto permite que el componente hijo reciba valores desde el padre y los utilice dentro de su lógica o en su plantilla.

Ejemplo: En el padre, puedes pasar datos al componente hijo en el HTML de esta forma:

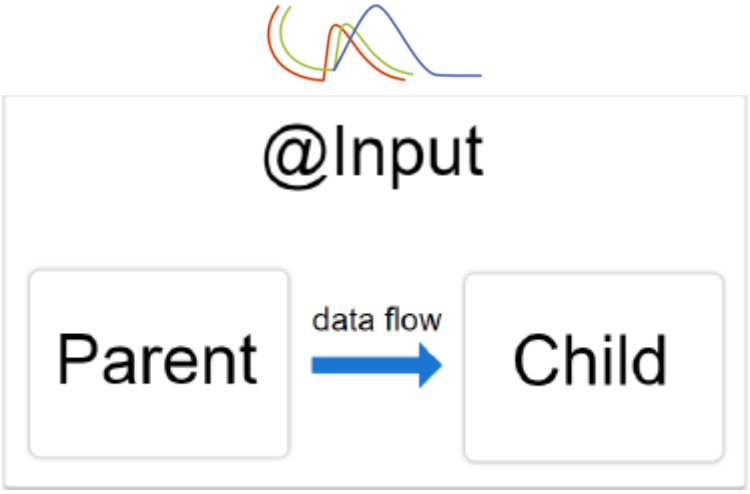


Ilustración 11- Comunicacion Padre-Hijo

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


}
```

```
<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>
```

Ilustración 12 - Programación en Padre

```
@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);
  }

}
```

```
<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>
```

Ilustración 13 - Programación en Hijo

COMUNICACIÓN HIJO A PADRE: El componente hijo puede enviar datos o eventos al componente padre usando el decorador @Output() junto con un EventEmitter. De esta manera, el hijo emite un evento que el padre escucha para recibir datos o notificaciones.

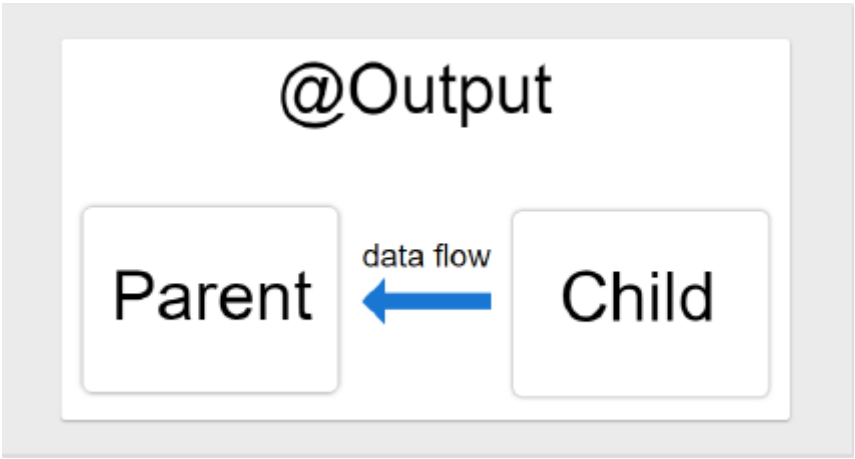


Ilustración 14 - Comunicación Hijo - Padre

En el hijo	En el padre

• 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 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.

Ilustración 15 - resumen programación

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

  asignaturas:string[]
  "SQL","JS","JSON","
  asignaturaSeleccion
  @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>
```

Ilustración 16 - Programacion Hijo

```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>
```

Ilustración 17 - Programacion Padre

Ejemplo: En el componente hijo, se utiliza @Output() para emitir un evento:

El routing en Angular es un mecanismo que permite la navegación entre diferentes componentes y vistas dentro de una aplicación de una sola página (SPA). A través del sistema de enrutamiento, los desarrolladores pueden definir rutas que mapean URLs específicas a componentes, facilitando una experiencia de usuario fluida y sin recargas de página. Esto se logra mediante el uso del **RouterModule**, que proporciona herramientas y servicios para gestionar la navegación y el estado de las rutas.

Para programar el routing en Angular, se define un arreglo de rutas en un módulo de enrutamiento, donde cada ruta especifica una URL y el componente correspondiente que debe cargarse cuando se accede a esa URL. Por ejemplo, se puede crear un archivo app-routing.module.ts con el siguiente contenido:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' }
];
```

Luego, se importa el RouterModule en el módulo principal y se utiliza el RouterOutlet en la plantilla HTML para indicar dónde deben renderizarse los componentes de las rutas. Los desarrolladores también pueden utilizar enlaces de navegación con la directiva routerLink, lo que permite cambiar de vista sin recargar la página. El sistema de routing de Angular es altamente configurable, permitiendo características avanzadas como rutas anidadas, parámetros de ruta, y guardias de ruta para controlar el acceso a diferentes secciones de la aplicación.

Pasos:

Añadir las rutas en el fichero app.routes.ts

•En el app.configs.ts añadir el parámetro withDebugTracing()

•Importar el RouterModule

•Usar el atributo routerLinken la etiqueta a (en lugar del href)

•Usar la etiquete router-outlet

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

];
```

```
export const appConfig: ApplicationConfig = {

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

};
```

Ilustración 18 -Routing

NAVEGACION PROGRAMATICA

En lugar de utilizar enlaces en la plantilla HTML. Esto permite a los desarrolladores redirigir a los usuarios a diferentes componentes basándose en eventos específicos o condiciones lógicas, lo que es útil en situaciones como la validación de formularios, la gestión de permisos, o después de completar una acción.

Para implementar la navegación por TypeScript, primero se debe inyectar el servicio **Router** en el componente donde se desea realizar la navegación. Esto se hace mediante la inyección de dependencias en el constructor del componente. Por ejemplo:

typescript

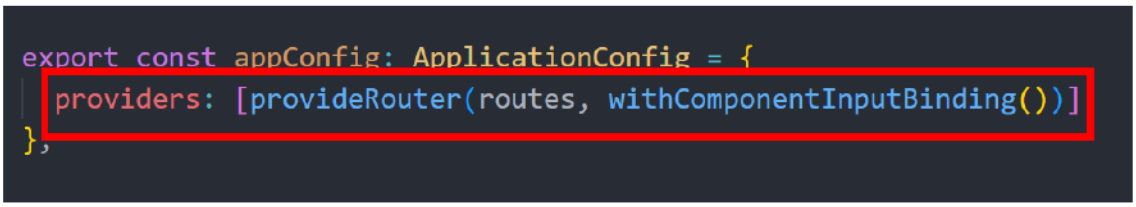
Ilustración 19 - Router programatico

También se pueden pasar parámetros y opciones adicionales al método navigate(), lo que permite un control más detallado sobre la navegación. Además, Angular permite el uso de router.navigateByUrl() para navegar a una URL completa. La navegación programática es especialmente poderosa cuando se combina con lógica de negocio, ya que permite a las aplicaciones reaccionar a eventos de manera dinámica y ofrecer una experiencia de usuario más fluida y coherente.

PASO DE VALOR POR LA URL

El paso de valores por la URL en Angular se realiza mediante parámetros de ruta y consultas de parámetros (query parameters). Esta funcionalidad permite que los desarrolladores envíen información adicional en la URL al navegar entre diferentes componentes de la aplicación, lo que es útil para situaciones como la carga de datos específicos, la filtración de resultados o la gestión del estado de la aplicación.

Como primer paso se debe configurar el ApplicationConfig



PARÁMETROS DE RUTA: Los parámetros de ruta son partes de la URL que se definen en las rutas del enrutador. Por ejemplo, si tienes una ruta que muestra detalles de un usuario, puedes definir la ruta como /user/:id, donde :id es un parámetro que representa el identificador del usuario.

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

Navegación con Parámetros:

Para navegar a esta ruta con un valor específico, puedes usar el método navigate del servicio Router y proporcionar un objeto con los parámetros:

```
<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.  ferencia }}</td>

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

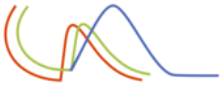
```
export class DetalleInmuebleComponent implements OnInit{
  @Input() id:number = 0;

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

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

O de forma programática

this.router.navigate(['/user', userId]);

Acceso a Parámetros en el Componente:



En el componente UserDetailComponent, puedes acceder al parámetro id usando el servicio ActivatedRoute:

```
import { ActivatedRoute } from '@angular/router';


constructor(private route: ActivatedRoute) {}


ngOnInit() {
  this.route.params.subscribe(params => {
    const userId = params['id'];
    // Usa el userId para cargar datos
  });
}
```

2. Query Parameters:

Los parámetros de consulta son valores que se agregan al final de la URL con un símbolo de interrogación `?`. Se utilizan comúnmente para enviar información adicional que no es parte de la estructura principal de la URL.

Navegación con Parámetros de Consulta:

Para agregar parámetros de consulta al navegar, puedes usar el objeto `queryParams`:

```
this.router.navigate(['/products'], { queryParams: { category: 'electronics', sort: 'asc' } });
```

Acceso a Parámetros de Consulta en el Componente:

En el componente correspondiente, puedes acceder a los parámetros de consulta de la siguiente manera:

```
ngOnInit() {
  this.route.queryParams.subscribe(params => {
    const category = params['category'];
    const sort = params['sort'];
    // Usa los parámetros de consulta para cargar datos
  });
}
```

```
<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']);
```

RUTAS DE ERROR

Las rutas de errores en Angular son una característica que permite manejar situaciones en las que un usuario intenta acceder a una ruta que no existe o que no está configurada correctamente, mejorando así la experiencia del usuario al proporcionar una respuesta clara ante errores de navegación. Angular permite definir una ruta especial para manejar errores mediante la configuración del enrutador, donde se puede especificar un componente de "página no encontrada", típicamente utilizando el path '**' para capturar cualquier ruta que no coincida con las rutas definidas. Esto permite que, en lugar de mostrar un mensaje de error genérico del servidor, la aplicación redirija al usuario a un componente específico que explique el error y ofrezca opciones para volver a navegar a rutas válidas, como un botón que redirija al usuario a la página principal o a una lista de rutas disponibles. Este enfoque no solo mejora la usabilidad, sino que también facilita el manejo de errores en el lado del cliente, contribuyendo a una experiencia de usuario más robusta y coherente.

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

En Angular, los servicios son clases que contienen lógica o funcionalidades que pueden ser reutilizadas a lo largo de la aplicación. Su propósito principal es permitir que diferentes componentes compartan datos o lógica de negocio sin duplicar código, promoviendo así la modularidad y el reuso. Los servicios suelen manejar tareas como el acceso a APIs externas, la manipulación de datos, el almacenamiento en caché, o cualquier tipo de lógica compleja que no debería estar directamente en los componentes, lo que ayuda a mantener estos últimos más simples y enfocados en la presentación.

Para utilizar un servicio en Angular, se suele crear una clase y decorarla con @Injectable(), lo que permite inyectar dicho servicio en cualquier componente o en otros servicios mediante Inyección de Dependencias (Dependency Injection). Esto asegura que Angular gestione la instancia de dicho servicio y pueda compartirla entre componentes. Los servicios son esenciales cuando necesitamos funcionalidades que deben ser consistentes o compartidas a lo largo de toda la aplicación, como la gestión de sesiones, la autenticación, o la comunicación con un servidor backend mediante HTTP requests.

PROGRAMACIÓN

- No tienen decorador de componente, tiene un @Injectable
- 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.

Ilustración 20 - Inyección de servicio

La programación reactiva en Angular es un paradigma que se centra en la gestión de flujos de datos asíncronos y la propagación de cambios, permitiendo a los desarrolladores construir aplicaciones más dinámicas y responsivas. Este enfoque permite que las aplicaciones respondan de manera eficiente a eventos como interacciones del usuario, respuestas de servidores o cambios en datos en tiempo real. Utilizando bibliotecas como **RxJS**, Angular ofrece un conjunto de herramientas potentes que facilitan la creación de **Observables**, que representan flujos de datos que pueden ser observados y manipulados a través de una variedad de operadores. Esto no solo mejora la legibilidad y la mantenibilidad del código, sino que también permite a los desarrolladores gestionar de manera más efectiva la complejidad de las aplicaciones modernas, asegurando que la interfaz de usuario se mantenga sincronizada con el estado del modelo de datos de manera reactiva y eficiente.

OBSERVABLES

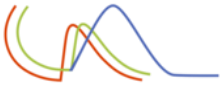
Los Observables en Angular son una parte fundamental de la programación reactiva y se utilizan ampliamente para manejar eventos asíncronos, flujos de datos y la interacción con APIs. Un Observable es un objeto que representa una colección de valores o eventos que pueden llegar en el tiempo, lo que permite a los desarrolladores suscribirse para recibir esos valores cuando estén disponibles. A diferencia de las **promesas**, que devuelven un único valor, los Observables pueden emitir múltiples valores a lo largo del tiempo, lo que los hace ideales para situaciones como la gestión de eventos del usuario, respuestas de una API o cualquier fuente de datos que pueda actualizarse continuamente.

En Angular, los Observables son proporcionados por el módulo **RxJS** y se integran de manera natural con el **HttpClient** y otros servicios de Angular. Al utilizar Observables, los desarrolladores pueden aprovechar una serie de operadores que permiten transformar, filtrar y combinar flujos de datos de manera eficiente. Por ejemplo, al realizar una llamada HTTP, el método devuelve un Observable que se puede suscribir para manejar la **respuesta asíncrona**. Al **suscribirse**, se puede definir cómo manejar los datos recibidos y los errores potenciales. Esto no solo facilita el manejo de la lógica asíncrona, sino que también mejora la capacidad de respuesta de la aplicación, permitiendo a los desarrolladores construir interfaces de usuario más dinámicas y fluidas.

SEÑALES

Las **señales** en Angular son una característica relativamente nueva que forma parte de la programación reactiva en el framework. Son un mecanismo que permite a los desarrolladores crear flujos de datos reactivos de una manera más sencilla y directa que los Observables. A diferencia de los Observables, que se suscriben y se manejan de forma más compleja, las señales se centran en la reactividad de los valores y permiten a los componentes y servicios reaccionar automáticamente a los cambios sin la necesidad de suscribirse manualmente. Esto mejora la legibilidad y la eficiencia del código, ya que las señales simplifican el proceso de seguimiento de los cambios en los datos.

Las señales son ideales para manejar estados que cambian frecuentemente y para crear aplicaciones que necesitan una alta capacidad de respuesta a las interacciones del usuario. Con las señales, los desarrolladores pueden definir un valor reactivo que se puede leer y escribir, y cualquier componente que dependa de esa señal se



actualizará automáticamente cuando el valor cambie. Esto se traduce en una experiencia más fluida y una menor cantidad de lógica de gestión de suscripciones. Al facilitar la creación de aplicaciones reactivas, las señales en Angular representan un paso importante hacia la simplificación del desarrollo de interfaces de usuario dinámicas y eficientes.

En Angular, el manejo de HTTP es clave para que nuestras aplicaciones interactúen con servidores backend, enviando y recibiendo datos de manera **asíncrona**. Angular proporciona el módulo **HttpClient** dentro de **@angular/common/http**, que facilita realizar solicitudes HTTP como **GET, POST, PUT, DELETE** a una API o servidor remoto. Este módulo está diseñado para trabajar con la arquitectura RESTful y manejar las interacciones del cliente con los recursos del servidor.

El **HttpClient** simplifica las peticiones y el manejo de respuestas, devolviendo un **Observable** que nos permite trabajar con las respuestas de forma asíncrona, lo que es crucial en aplicaciones web modernas. Para usarlo, primero se debe importar el **HttpClientModule** en el módulo principal (app.module.ts), y luego inyectar **HttpClient** en el servicio o componente donde se necesita realizar la petición.

Por ejemplo, para hacer una petición **GET** y obtener datos de una API:

typescript

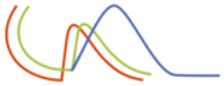
constructor(private http: HttpClient) {}

getData() { return this.http.get('https://api.example.com/data'); }

Este método devuelve un **Observable** con los datos de la API. Para suscribirse a estos datos y utilizarlos, se emplea el método subscribe:

this.getData().subscribe(data => console.log(data));

Además, **HttpClient** gestiona automáticamente tareas importantes como la codificación JSON, el manejo de errores, y permite agregar **headers** o parámetros de forma sencilla. Esto lo convierte en una herramienta poderosa y versátil para la comunicación entre el frontend Angular y un servidor backend.



ENLACES

Versión 17: https://angular.dev/

Json: https://www.json.org/json-en.html

Curso de TypeScript:

https://learn.microsoft.com/es-es/training/browse/?terms=typescript

RxJS: https://rxjs.dev/guide/overview