

Module frontend with angular - class 2

Today's agenda - Feb 3rd

Table of contents

- **<u>** Typescript</u>**
- <u>**\\Pi**</u> Angular components
- Angular directives
- Let's review
- Thank you



Last class review



- 1. NodeJS
- 2. Javascript
- 3. Nvm
- 4. Arquitectura cliente servidor
- 5. Cloud
- 6. SPA
- 7. Angular

- Aplicación web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como si fuera una aplicación de escritorio
- 2. Arquitectura distribuida que permite el dar servicios a un cliente desde un servidor remoto.
- 3. Ambiente de ejecución multi-plataforma de javascript
- 4. Lenguaje de programación que se usa para crear sitios web interactivos y aplicaciones backend con entornos como Deno, Bun o nodeJA.
- Framework de JavaScript de código abierto escrito en TypeScript. Su objetivo principal es desarrollar aplicaciones de una sola página y fue creado y es mantenido por Google.
- 6. Node Version Manager, es una herramienta que permite gestionar versiones de Node.js.
- 7. Servicio para entregar servicio de cómputo a través del internet, que incluye servicios como almacenamiento, bases de datos, software para evitar comprar y administrar servidores localmente ahorrando así tiempo, dinero y otros recursos.

lai	Script es un lenguaje de programación desarrollado por Microsoft que es un superconjunto de JavaScript. Añade tipos icos y objetos basados en clases, lo que lo hace especialmente útil para el desarrollo de aplicaciones a gran escala. Tinado de datos
let	Tipado de datos string nombre: string = "TypeScript"; mensaje: string = `Hola \${nombre}`;
let let	number edad: number = 25; precio: number = 99.99; negativo: number = -10;
// k let	ooolean estaActivo: boolean = true; tienePermiso: boolean = false;
let	Null y Undefined nulo: null = null; indefinido: undefined = undefined;
let let	array listaNumeros: number[] = [1, 2, 3]; listaCadenas: Array <string> = ["a", "b", "c"];</string>
let let	uple - Arrays con un número fijo de elementos de diferentes tipos coordenada: [number, number] = [10, 20]; usuarioStatus: [string, boolean] = ["admin", true]; Enum - Conjunto de constantes nombradas
en L I	um DiaSemana { Lunes, Martes, Miercoles, ueves,
} let	/iernes dia: DiaSemana = DiaSemana.Lunes; Object - Para definir la estructura de objetos
let r	usuario: { nombre: string; edad: number; activo?: boolean; // El ? indica que es opcional
};	nombre: "Carlos", edad: 30
let no	noEstoySeguro: any = 4; EstoySeguro = "tal vez un string"; EstoySeguro = false; // okay, definitivamente un booleano
	Tipos personalizados
int r	nterface erface Persona { nombre: string; edad: number; saludar(): string;
r	empleado: Persona = { nombre: "Ana", edad: 28,
};	saludar() { return `Hola, soy \${this.nombre}`; .
typ >	Type De Coordenadas = { XI: number; XI: number;
// (coordenadas: Coordenadas = {x: 10.12, y: 11.34} Union Types - Permite que una variable sea de varios tipos De StringOrNumber = string number;
let ide	identificador: StringOrNumber = "abc123"; entificador = 123; // También válido Genéricos
r } let	return arr[0]; primerNumero = primeroDeArray([1, 2, 3]); // Tipo inferido: number
±t	primerTexto = primeroDeArray(["a", "b", "c"]); // Tipo inferido: string
/	Clases ss Persona { // Propiedades de la clase
r e	nombre: string; edad: number; // Constructor constructor(nombre: string, edad: number) {
	this.nombre = nombre; this.edad = edad; // Método saludar(): string {
} } //(return `Hola, soy \${this.nombre} y tengo \${this.edad} años`; Creando una instancia de la clase
	nst persona1 = new Persona("Ana", 25); nsole.log(persona1.saludar()); // Output: "Hola, soy Ana y tengo 25 años"
	Estructuras de control
if (e	edad: number = 18; edad >= 18) { console.log("Eres mayor de edad.");
}	lse { console.log("Eres menor de edad."); puntuacion: number = 85;
(} e	ountuacion >= 90) { console.log("Excelente"); lse if (puntuacion >= 70) { console.log("Bueno");
) } e	lse if (puntuacion >= 50) { console.log("Aprobado"); lse { console.log("Reprobado");
	(let i: number = 0; i < 5; i++) { console.log(`Iteración número \${i}`);
let for	for of colores: string[] = ["rojo", "verde", "azul"]; (let color of colores) { console.log(color);
} // f let	for in persona = {
};	nombre: "Juan", edad: 30, ciudad: "Madrid" (let clave in persona) {
} // v	console.log(`\${clave}: \${persona[clave]}`); while contador: number = 0;
wh	tile (contador < 5) { console.log(`Contador: \${contador}`); contador++;
	do while contador: number = 0;
(} w	console.log(`Contador: \${contador}`); contador++; chile (contador < 5); cwitch case
let let sw	dia: number = 3; nombreDia: string; itch (dia) { case 1:
	nombreDia = "Lunes"; break; case 2: nombreDia = "Martes"; break;
	case 3: nombreDia = "Miércoles"; break; case 4: nombreDia = "Jueves"; break;
	case 5: nombreDia = "Viernes"; break; case 6: nombreDia = "Sábado";
	break; case 7: nombreDia = "Domingo"; break; default:
} coi	nombreDia = "Día no válido"; nsole.log(`El día es \${nombreDia}`);
for	oreak continue (let i: number = 0; i < 10; i++) { f (i === 5) { break; // Sale del bucle cuando i es 5
}	f (i % 2 === 0) { continue; // Salta la iteración si i es par
let let	ernary operator edad: number = 20; mensaje: string = edad >= 18 ? "Eres mayor de edad" : "Eres menor de edad";
_01	nsole.log(mensaje);
fur	Functions Function básica con tipos action saludar(nombre: string): string {
r } // F fur	return `Hola, \${nombre}!`; Función con múltiples parámetros nction sumar(a: number, b: number): number {
} // F	return a + b; Función flecha (arrow function) nst multiplicar = (a: number, b: number): number => a * b;
fur /	Parámetro opcional nction crearUsuario(nombre: string, edad?: number): string { // Si edad no se proporciona, usamos un valor por defecto const edadReal = edad ?? "desconocida";
} coi	return `Usuario: \${nombre}, Edad: \${edadReal}`; nsole.log(crearUsuario("Ana")); // "Usuario: Ana, Edad: desconocida" nsole.log(crearUsuario("Juan", 25)); // "Usuario: Juan, Edad: 25"
// F fur	Parámetros con valores predeterminados nction configurarServidor(host: string = "localhost", puerto: number = 3000) { return `Servidor iniciado en \${host}:\${puerto}`;
COI	nsole.log(configurarServidor()); // "Servidor iniciado en localhost:3000" nsole.log(configurarServidor("example.com", 8080)); // "Servidor iniciado en example.com:8080"
fur r }	nction sumarTodos(numeros: number[]): number { return numeros.reduce((total, num) => total + num, 0);
coi // F fur	nsole.log(sumarTodos(1, 2, 3)); // 6 nsole.log(sumarTodos(10, 20, 30, 40)); // 100 Función genérica nction primerElemento <t>(array: T[]): T undefined {</t>
} // [coi	return array.length > 0 ? array[0] : undefined; Las funciones genéricas pueden inferir el tipo Insole.log(primerElemento([1, 2, 3])); // 1 Insole.log(primerElemento(["a", "b", "c"])); // "a"
// F fur	nsole.log(primerElemento(["a", "b", "c"])); // "a" Función genérica con múltiples tipos nction combinar <t, u="">(a: T, b: U): [T, U] { return [a, b];</t,>
coi //	nst resultado = combinar("hola", 123); // ["hola", 123] nterface con función erface Calculadora {
s r }	erface Calculadora { sumar: (a: number, b: number) => number; restar: (a: number, b: number) => number; mplementación de la interface
COI	mplementación de la interface nst calculadora: Calculadora = { sumar: (a, b) => a + b, restar: (a, b) => a - b
	Type con función be Comparador <t> = (a: T, b: T) => number; nst compararNumeros: Comparador<number> = (a, b) => a - b;</number></t>
// T	Función asíncrona
// l typ coi // F asy	nc function obtenerDatos(id: string): Promise <any> { ry { // Simulamos una llamada a API</any>
// 1 typ coi	ry {
// 1 typ coi // F asy t // F asy	ry { // Simulamos una llamada a API const respuesta = await fetch(`https://api.ejemplo.com/datos/\${id}`); return await respuesta.json(); catch (error) { console.error("Error al obtener datos:", error);

6 Made with Gamma

Typescript

DIIC Google Partner

Angular components



Son bloques fundamentales para construir la interfaz de usuario (UI). Cada componente controla una parte de la pantalla llamada **vista** y está compuesto por tres partes principales:

- 1. Clase TypeScript: Define la lógica del componente.
- 2. **Plantilla HTML**: Define la estructura del DOM que se renderiza.
- 3. Estilos CSS: Define el estilo visual del componente.

```
typescript

import { Component } from '@angular/core';

@Component({
    selector: 'app-mi-componente', // Selector para usar en el HTML
    template: `Hola, soy un componente`, // Plantilla HTML
    styles: [`p { color: blue; }`] // Estilos CSS (opcional)
})
export class MiComponente {
    // Lógica del componente
}
```

ng generate component nombre-del-componente

```
src/app/nombre-del-componente/

— nombre-del-componente.component.ts

— nombre-del-componente.component.html

— nombre-del-componente.component.css

— nombre-del-componente.component.spec.ts
```







Permiten crear interfaces dinámicas y reutilizables

```
// Directiva *nglf para renderizado condicional
@Component({
 template: `
  <div *ngIf="isVisible">
   Este contenido aparece solo si isVisible es true
  </div>
  <!-- También podemos usar else -->
  <div *ngIf="isLoggedIn; else loginTemplate">
   Bienvenido usuario
  </div>
  <ng-template #loginTemplate>
   Por favor, inicia sesión
  </ng-template>
})
export class MiComponente {
 isVisible = true;
 isLoggedIn = false;
}
```

```
@Component({
 template: `
  <!-- Iteración básica -->
  ul>
   {{ item }}
  <!-- Con índice y otras variables -->
  <div *ngFor="let user of users;</pre>
         let i = index;
         let isFirst = first;
         let isLast = last">
   Usuario {{i + 1}}: {{user.name}}
   <span *nglf="isFirst">(Primer usuario)</span>
   <span *ngIf="isLast">(Último usuario)</span>
  </div>
})
export class ListaComponent {
 items = ['Manzana', 'Naranja', 'Plátano'];
 users = [
  { name: 'Ana' },
  { name: 'Juan' },
  { name: 'María' }
];
}
```

```
// Directiva ngClass para manipular clases CSS
@Component({
 template: `
  <div [ngClass]="{'active': isActive,</pre>
            'error': hasError,
            'highlight': isHighlighted}">
   Contenido con clases dinámicas
  </div>
  <!-- ngStyle para estilos inline -->
  <div [ngStyle]="{'color': textColor,</pre>
            'font-size': fontSize + 'px',
           'background-color': bgColor}">
   Estilos dinámicos
  </div>
})
export class EstilosComponent {
 isActive = true;
 hasError = false;
 isHighlighted = true;
 textColor = 'blue';
 fontSize = 16;
 bgColor = '#f0f0f0';
```

// Componente con múltiples directivas

```
@Component({
 template: `
  <div class="task-list">
   <!-- Usar nglf para mostrar mensaje cuando no hay tareas -->
   <div *ngIf="tasks.length === 0" class="empty-message">
    No hay tareas pendientes
   </div>
   <!-- Lista de tareas con ngFor y múltiples directivas -->
   <div *ngFor="let task of tasks; let i = index"</pre>
      [ngClass]="{'completed': task.completed,
            'urgent': task.priority === 'alta'}"
      [ngStyle]="{'opacity': task.completed?'0.6':'1'}"
      appResaltado="lightyellow">
    <span class="task-number">{{i + 1}}.</span>
    <span class="task-title">{{task.title}}</span>
    <!-- Botones condicionales -->
    <button *nglf="!task.completed"</pre>
         (click)="completeTask(task)">
     Completar
    </button>
    <button *nglf="task.completed"</pre>
         (click)="undoComplete(task)">
     Deshacer
    </button>
   </div>
  </div>
})
export class TaskListComponent {
 tasks = [
  { title: 'Comprar víveres', completed: false, priority: 'normal' },
  { title: 'Pagar facturas', completed: false, priority: 'alta' },
  { title: 'Llamar al médico', completed: true, priority: 'alta' }
 ];
 completeTask(task: any) {
  task.completed = true;
 undoComplete(task: any) {
  task.completed = false;
}
@Component({
```

```
template: `
  <div [ngSwitch]="userRole">
   <div *ngSwitchCase=""admin"">
    Panel de administrador
   </div>
   <div *ngSwitchCase=""editor"">
    Panel de editor
   </div>
   <div *ngSwitchCase=""user"">
    Panel de usuario
   </div>
   <div *ngSwitchDefault>
    Acceso denegado
   </div>
  </div>
})
export class PanelComponent {
 userRole = 'admin';
}
```





```
{{ valor | uppercase }}: Convierte el texto a mayúsculas.
{{ valor | lowercase }}: Convierte el texto a minúsculas.
{{ valor | date:'dd/MM/yyyy' }}: Formatea una fecha.
{{ valor | currency:'USD' }}: Formatea un número como moneda.
{{ valor | json }}: Convierte un objeto a formato JSON.
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({ name: 'miPipe' })
export class MiPipe implements PipeTransform {
 transform(value: string): string {
  return value.split(").reverse().join(");
 }
}
{{ 'Hola' | miPipe }} <!-- Resultado: "aloH" -->
<div *ngIf="usuarios.length > 0">
 {{ usuario.nombre | uppercase }} - {{ usuario.fechaRegistro | date:'dd/MM/yyyy' }}
  </|i>
 </div>
@Component({
 template: `
  <!-- DatePipe - Formatea fechas -->
  Fecha: {{ fecha | date }}
  Fecha personalizada: {{ fecha | date:'dd/MM/yyyy' }}
  Hora: {{ fecha | date:'shortTime' }}
  <!-- UpperCasePipe y LowerCasePipe - Transformación de texto -->
  Mayúsculas: {{ nombre | uppercase }}
  Minúsculas: {{ nombre | lowercase }}
  <!-- DecimalPipe - Formato de números -->
  Número: {{ precio | number:'1.2-2' }}
  Moneda: {{ precio | currency: 'EUR' }}
  Porcentaje: {{ valor | percent:'1.0-2' }}
  <!-- SlicePipe - Recorta arrays o strings -->
  Texto recortado: {{ texto | slice:0:10 }}
  Array recortado: {{ array | slice:1:3 }}
  <!-- JsonPipe - Útil para debugging -->
  <{cobjeto | json }}</pre>
})
export class EjemploPipesComponent {
 fecha = new Date();
 nombre = 'Juan Pérez';
 precio = 23.5678;
 valor = 0.8543;
 texto = 'Este es un texto largo de ejemplo';
 array = [1, 2, 3, 4, 5];
 objeto = { nombre: 'Juan', edad: 25 };
// Pipe personalizado para filtrar una lista
@Pipe({
 name: 'filtrar'
})
export class FiltrarPipe implements PipeTransform {
 transform(items: any[], searchText: string): any[] {
  if (!items) return [];
  if (!searchText) return items;
  searchText = searchText.toLowerCase();
  return items.filter(item => {
   return item.nombre.toLowerCase().includes(searchText);
  });
// Componente que usa el pipe personalizado
@Component({
 template: `
  <input [(ngModel)]="busqueda" placeholder="Buscar...">
  ul>
   {{ item.nombre }}
   })
export class ListaComponent {
 busqueda = ";
 items = [
  { nombre: 'Manzana' },
  { nombre: 'Banana' },
  { nombre: 'Naranja' }
 ];
}
@Component({
 template: `
  <!-- Encadenamiento de pipes -->
  {{ texto | lowercase | titlecase }}
  {{ precio | currency:'EUR' | uppercase }}
  <!-- Pipes con parámetros -->
```

{{ fecha | date:'fullDate' | uppercase }}

export class PipesEncadenadosComponent {

texto = 'HOLA MUNDO';

precio = 99.99;

fecha = new Date();

})

}





Let's review

Did you learn?







Thank you

Thanks for paying attention!