

Introducción al desarrollo **WEB con Angular**



Presentado por Alexander Díaz
www.consultec-ti.com



Introducción a **Angular**

Es un framework web para construir aplicaciones web y móviles de alta calidad y escalables.

Es de código abierto desarrollado por Google que se utiliza para construir aplicaciones web de una sola página (SPA) y aplicaciones móviles híbridas.



Introducción a **Angular**

Utiliza TypeScript, y se basa en el patrón de diseño de arquitectura (MVC) y (MVVM) para facilitar el desarrollo y la estructuración de las aplicaciones.

Día 1



Entendiendo las **bases de Angular**



¿Qué es programación orientada a componentes?

La programación orientada a componentes se centra en la modularidad, la reutilización y la mantenibilidad del software, ya que los componentes pueden ser fácilmente sustituidos o actualizados sin afectar a otras partes del sistema.





¿Qué son **templates/plantillas**?

En el contexto del desarrollo de software, una plantilla o template es una estructura predefinida que se utiliza como base para crear algo nuevo. En el desarrollo web, una plantilla se refiere a un archivo HTML que contiene una estructura básica y un diseño predefinido para una página web o una aplicación web.





¿Qué son **components** y **directivas** en Angular?

Los **componentes** encapsulan la interfaz de usuario y su lógica.

Las **directivas** se utilizan para manipular el DOM y agregar comportamiento a los elementos HTML existentes o crear nuevos elementos personalizados.





Inyección de dependencias

Es un patrón de diseño, utilizado en la programación orientada a objetos, que permite a los objetos comunicarse entre sí; sin tener que conocer los detalles de su implementación.

En Angular, la inyección de dependencias se utiliza para gestionar y compartir objetos entre diferentes componentes y servicios de una aplicación, lo que facilita la creación de aplicaciones escalables y mantenibles.





Singleton

Es una clase que solo se puede instanciar una vez durante toda la ejecución de una aplicación y proporciona un mecanismo para acceder a esa instancia única desde cualquier parte del código.

Este patrón de diseño se utiliza a menudo en situaciones en las que se necesita una única instancia de una clase para coordinar acciones en toda la aplicación.





TypeScript

Typescript



TypeScript es un lenguaje de programación, de código abierto basado en JavaScript, que proporciona una sintaxis más clara y concisa para la programación orientada a objetos y un sistema de tipado estático opcional para mejorar la seguridad y la calidad del código.

Es compatible con la mayoría de las bibliotecas y frameworks de JavaScript existentes. Se utiliza ampliamente para desarrollar aplicaciones web y móviles.



Propiedades y tipos de datos



boolean: Representa un valor verdadero o falso



number: Representa un número, ya sea entero o de punto flotante.



string: Representa una cadena de caracteres.



null y undefined: Representan valores nulos o indefinido.



any: Representa cualquier tipo de datos.



void: Representa la ausencia de un valor y se utiliza para indicar que una función no devuelve ningún valor.



object: Representa cualquier objeto JavaScript.

Propiedades y tipos de dato

```
let nombre: string = "Juan";
```

```
interface Persona {  
  nombre: string;  
  edad?: number;  
}
```

```
let persona: Persona = { nombre: "Juan" };
```



Clases y modelos de datos

```
export class Persona {  
  nombre: string;  
  edad: number;  
  
  constructor(nombre: string, edad: number) {  
    this.nombre = nombre;  
    this.edad = edad;  
  }  
  
  saludar(): void {  
    console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años.`)  
  }  
}
```





const, var y let

En general, se recomienda utilizar **const** siempre que sea posible, ya que esto ayuda a evitar errores y a escribir un código más seguro y confiable. Solo se debe utilizar **let** cuando se necesite reasignar una variable y se debe evitar el uso de **var** en su lugar.



Ejercicio guiado

nº01

Luego de seguir los pasos realizados por el instructor y crear un proyecto angular a partir de NG CLI, realizar un refactor del proyecto base e implementar mejores prácticas aprendidas en el curso de “Desarrollo WEB” al código autogenerado recientemente.

Páginas WEB **dinámicas**



Páginas WEB **dinámicas**

En general, las páginas web dinámicas ofrecen una experiencia más interactiva y personalizada para el usuario, lo que puede mejorar la retención del usuario y la efectividad del sitio web en alcanzar sus objetivos.





Ventajas

- **Interactividad:** ofrecen una experiencia más interactiva y personalizada al usuario
- **Actualización en tiempo real:** pueden actualizar el contenido de la página sin necesidad de recargarla.
- **Personalización:** permiten la personalización del contenido para diferentes usuarios.
- **Mayor funcionalidad:** permiten la integración de funcionalidades más avanzadas, como formularios interactivos, chat en vivo, carritos de compra, entre otros.



Desventajas

- **Mayor complejidad:** pueden ser más complicadas de desarrollar y mantener que las páginas web estáticas.
- **Mayor costo:** pueden requerir más recursos y tiempo de desarrollo.
- **Problemas de compatibilidad:** pueden requerir el uso de tecnologías avanzadas.
- **Riesgo de seguridad:** pueden ser más vulnerables a los ataques de seguridad si no se implementan correctamente las medidas de seguridad necesarias.



[Contacte con nosotros](#)[Soporte ▾](#)[Español ▾](#)[Mi cuenta ▾](#)[Iniciar sesión](#)[Cree una cuenta AWS](#)[Productos](#)[Soluciones](#)[Precios](#)[Documentación](#)[Aprender](#)[Red de socios](#)[AWS Marketplace](#)[Habilitación para clientes](#)[Eventos](#)[Explorar más](#)

Nivel gratuito de AWS

[Información general](#)[Categorías de nivel gratuito ▾](#)[¿Cómo crear una cuenta?](#)[Ofertas destacadas para empresas ▾](#)[Preguntas frecuentes](#)[Términos y condiciones](#)

Nivel gratuito de AWS

Adquiera experiencia práctica y gratuita con la plataforma, los productos y los servicios de AWS

[Más información sobre la capa gratuita de AWS ⓘ](#)[Crear una cuenta gratuita](#)

DESTACADO

Las empresas emergentes pueden reunir los requisitos para recibir créditos de AWS

AWS Activate proporciona a las empresas emergentes elegibles una serie de recursos, que incluyen créditos de AWS gratis para gastar en servicios de AWS, y AWS Support.

[Regístrese hoy mismo en Activate »](#)

Tipos de ofertas

Explore los más de 100 productos y comience a crear en AWS con el nivel gratuito. Hay tres tipos diferentes de ofertas gratuitas disponibles en función del producto usado. Haga clic en el icono siguiente para explorar nuestras ofertas.



Pruebas gratuitas

Las ofertas de prueba gratuita a corto plazo se inician a partir de la fecha en la que se activa un servicio en particular



12 meses de uso gratuito

Disfrute de estas ofertas durante 12 meses después de su fecha de registro inicial en AWS



Gratis para siempre

Estas ofertas del nivel gratuito no caducan y están disponibles para todos los clientes de AWS



Programación **Asíncrona**

En un proceso asíncrono, las operaciones se realizan de manera independiente y en paralelo. Esto significa que una operación no tiene que esperar a que la anterior termine para comenzar.

En lugar de esperar a que la operación actual termine, se puede continuar con la siguiente operación, mientras se espera a que la operación anterior se complete.



Callbacks y Promesas

Los **callbacks** y las **promesas** son dos formas de manejar la asincronía en JavaScript.

Los **callbacks** son útiles para tareas simples, mientras que las **promesas** son más avanzadas y permiten manejar tareas más complejas y múltiples.



Callback

```
function sumar(a, b, callback) {  
  const resultado = a + b;  
  callback(resultado);  
}  
  
function imprimirResultado(resultado) {  
  console.log(`El resultado es: ${resultado}`);  
}  
  
sumar(5, 7, imprimirResultado);
```



Callback

```
function sumar(a, b) {  
  return new Promise((resolve, reject) => {  
    const resultado = a + b;  
    if(resultado >= 0) {  
      resolve(resultado);  
    } else {  
      reject(new Error('La suma es negativa'));  
    }  
  });  
}
```

```
sumar(5, 7)  
  .then(resultado => {  
    console.log(`El resultado es: ${resultado}`);  
  })  
  .catch(error => {  
    console.error(error);  
  });
```



Observables

Los Observables tienen tres partes principales:

1. **Creación:** Se crea un Observable para representar un flujo de datos.
2. **Suscripción:** Un observador se suscribe al Observable y recibe notificaciones cuando hay nuevos eventos disponibles.
3. **Terminación:** El Observable emite una señal de finalización cuando no hay más eventos disponibles.



Patrón Observer

El patrón Observer se compone de los siguientes elementos:

1. **Sujeto:** Es el objeto que mantiene una lista de observadores y notifica a los observadores cuando se produce un cambio en su estado.
2. **Observadores:** Son los objetos que se registran con el sujeto para recibir notificaciones de los cambios en su estado.
3. **Actualización:** Es el método que se llama en los observadores cuando el sujeto notifica un cambio en su estado.



Observable

```
import { Observable } from 'rxjs';

function sumar(a, b) {
  return new Observable(observer => {
    const resultado = a + b;
    observer.next(resultado);
    observer.complete();
  });
}

sumar(5, 7)
  .subscribe(resultado => {
    console.log(`El resultado es: ${resultado}`);
  });
```



Routing y Router Guards

Routing

```
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '**', component: NotFoundComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```




```
<router-outlet></router-outlet>
```



```
<nav>  
  <ul>  
    <li><a routerLink="">Home</a></li>  
    <li><a routerLink="about">About</a></li>  
    <li><a routerLink="contact">Contact</a></li>  
  </ul>  
</nav>
```



Router Guards

Los router guards en Angular son mecanismos que permiten proteger y controlar el acceso a las rutas de la aplicación. Estos guards son funciones que se ejecutan antes de navegar a una ruta determinada y permiten decidir si la navegación debe continuar o si se debe bloquear.


1. CanActivateFn
2. CanActivateChildFn
3. CanDeactivateFn
4. CanMatchFn



```
import { Injectable } from '@angular/core';
import { CanActivateFn, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivateFn {
  constructor(private authService: AuthService) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
    boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
    return this.authService.isLoggedIn();
  }
}
```



```
const routes: Routes = [  
  {  
    path: 'dashboard',  
    component: DashboardComponent,  
    canActivate: [AuthGuard]  
  }  
];
```



Día 2



Pruebas Unitarias

- Ayudan a detectar errores temprano en el ciclo de desarrollo, lo que reduce el coste y tiempo de corrección.
- Facilitan la refactorización del código, ya que permiten asegurarse de que los cambios realizados no afectan al comportamiento esperado de las unidades de código.
- Mejoran la calidad del software, ya que garantizan que el código se comporta correctamente y cumple con los requisitos del sistema.



Test-Driven Development

1. **Escribir una prueba:** Se escribe una prueba automatizada para la funcionalidad que se desea implementar.
2. **Escribir el código:** Se escribe el código necesario para que la prueba pase. El objetivo es escribir el mínimo código necesario para hacer pasar la prueba.
3. **Refactorizar el código:** Se refactoriza el código para mejorar su calidad y mantenibilidad, asegurándose de que la prueba siga pasando.



Herramientas de testing en Angular

1. **Jasmine:** Es un framework de pruebas de JavaScript que proporciona una sintaxis clara y fácil de leer para escribir pruebas.
2. **Karma:** Es un runner de pruebas que se integra con Jasmine y permite ejecutar las pruebas en diferentes navegadores y plataformas.
3. **Protractor:** Es una herramienta de pruebas end-to-end para Angular que se utiliza para simular interacciones del usuario en el navegador.



Mocks y Spies

Las funciones de mock y spy son herramientas que permiten simular comportamientos y espiar sobre métodos y objetos. Los mocks se utilizan para simular datos o comportamientos, mientras que los spies se utilizan para registrar y verificar llamadas a métodos.



 **Jasmine** 4.5.0

5 specs, 0 failures, randomized with seed 88227

LoremDirectiveDirective

- should create an instance

AppComponent

- should have as title 'ng-notebook'
- should render title
- should create the app

LoremComponentComponent

- should create



Reto

Escribe un programa que muestre por consola (con un print) los números de 1 a 100 (ambos incluidos y con un salto de línea entre cada impresión), sustituyendo los siguientes:

- Múltiplos de 3 por la palabra "fizz".
- Múltiplos de 5 por la palabra "buzz".
- Múltiplos de 3 y de 5 a la vez por la palabra "fizzbuzz".



Directivas y Componentes

Directivas

- **Directivas de atributo:** Se utilizan para modificar el comportamiento o la apariencia de un elemento HTML.
- **Directivas estructurales:** Se aplican a un elemento HTML utilizando una sintaxis especial y se utilizan para agregar o eliminar elementos del DOM, repetir elementos o cambiar la estructura de la plantilla.



Componentes

1. **Clase del componente:** Se define la lógica de la interfaz de usuario y se puede interactuar con servicios y otros componentes.
2. **Plantilla del componente:** Esta es la vista HTML que define la estructura y la apariencia del componente.
3. **Metadatos del componente:** Esta es una anotación que se utiliza para configurar el componente y definir sus propiedades.



Reto

Escribe una función que reciba dos palabras (String) y retorne verdadero o falso (Boolean) según sean o no anagramas.

- Un Anagrama consiste en formar una palabra reordenando TODAS las letras de otra palabra inicial.
- NO hace falta comprobar que ambas palabras existan.
- Dos palabras exactamente iguales no son anagrama.



Reto

Escribe un programa que, dado un número, compruebe y muestre si es primo, fibonacci y par.

Ejemplos:

- Con el número 2, nos dirá: "2 es primo, fibonacci y es par"
- Con el número 7, nos dirá: "7 es primo, no es fibonacci y es impar"



Data Binding

Data Binding

1. **Interpolación:** Permite mostrar valores de propiedades del componente en la plantilla del componente. Se utiliza la sintaxis de doble llave `{{ }}` para realizar la interpolación.
2. **Property binding:** Permite establecer valores de propiedades HTML en la plantilla del componente. Se utiliza la sintaxis de corchetes `[]` para realizar el property binding.
3. **Event binding:** Permite detectar eventos HTML en la plantilla del componente y ejecutar una función en el componente en respuesta al evento. Se utiliza la sintaxis de paréntesis `()` para realizar el event binding.



Two-way Data binding

Permite la actualización automática de los datos en la vista y el componente. Con el two-way data binding, un componente puede recibir datos de un usuario, actualizar su estado interno y, a su vez, actualizar la vista con el nuevo estado.



Reto

Escribe un componente NG que reciba un texto y como resultado te muestre del texto cuales palabras era anagramas.

- El diseño web es de libre elección
- El código de los anagramas debió ser resuelto en retos anteriores
- El componente a desarrollar debe utilizar una o mas técnicas de Data Binding según lo explicado anteriormente.



Eventos

- **(click)**: Se utiliza para capturar el evento de clic en un elemento de la plantilla.
- **(keyup)**: Se utiliza para capturar el evento de tecla levantada en un elemento de la plantilla.
- **(submit)**: Se utiliza para capturar el evento de envío de un formulario.
- **(focus)**: Se utiliza para capturar el evento de foco en un elemento de la plantilla.



Angular también proporciona la clase **EventEmitter** para crear eventos personalizados en los componentes.

La clase **EventEmitter** se puede utilizar para emitir eventos personalizados desde un componente hijo al componente padre.



Reto

Escribe un componente que reciba una palabra, y dentro del componente anterior busque y reemplace todas las ocurrencias por un nuevo texto o palabra que también será ingresado dentro de este componente.

- El componente a desarrollar debe utilizar una o mas técnicas de Eventos según lo explicado anteriormente.



Día 3



Servicios Angular y WEB Services

Servicios Angular

Son clases que se utilizan para compartir datos, lógica y funcionalidad entre componentes y otras partes de la aplicación. Los servicios se utilizan para proporcionar una única instancia de datos o funcionalidad a través de la aplicación, lo que evita la duplicación de código y mejora la modularidad de la aplicación.



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

@Injectable({
  providedIn: 'root'
})
export class MiServicio {
  datos: any[];

  constructor() {
    this.datos = ['dato1', 'dato2', 'dato3'];
  }

  obtenerDatos(): any[] {
    return this.datos;
  }

  agregarDato(dato: any): void {
    this.datos.push(dato);
  }
}
```



```
import { Component } from '@angular/core';
import { MiServicio } from '../mi-servicio.service';

@Component({
  selector: 'mi-componente',
  template: `
    <ul>
      <li *ngFor="let dato of datos">{{dato}}</li>
    </ul>
  `
})
export class MiComponente {
  datos: any[];

  constructor(private miServicio: MiServicio) {
    this.datos = miServicio.obtenerDatos();
  }

  agregarDato(dato: any): void {
    this.miServicio.agregarDato(dato);
  }
}
```



Servicios WEB

Existen diferentes tipos de servicios web, como los servicios web **SOAP** (Simple Object Access Protocol) y los servicios web **REST** (Representational State Transfer), que se diferencian en la forma en que se realiza la comunicación y el intercambio de datos. Los servicios web **REST** son más simples y flexibles que los servicios web **SOAP**, y son ampliamente utilizados en el desarrollo de aplicaciones web modernas.



Axios

```
import axios from 'axios';

axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(function (response) {
    // manejar la respuesta exitosa
    console.log(response.data);
  })
  .catch(function (error) {
    // manejar el error
    console.log(error);
  });
```



HTTP Client

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataService {
  constructor(private http: HttpClient) { }

  getData(): Observable<any> {
    return this.http.get<any>('https://ejemplo.com/api/data');
  }
}
```



Reto

Escribe un servicio NG que agrupe la implementación de la lógica de los retos anteriores y este pase a ser el encargado de cumplir con esos requerimientos funcionales.



Pipes y Formularios

Pipes

- **date**: transforma una fecha en una cadena con un formato específico.
- **uppercase** y **lowercase**: convierte una cadena en mayúsculas o minúsculas.
- **decimal** y **currency**: da formato a un número como decimal o como moneda, respectivamente.
- **async**: se utiliza con datos asincrónicos para manejar la carga y la presentación de datos en tiempo real.



Formularios

- **Basados en plantillas:** Se crean utilizando plantillas HTML y se enlazan con el modelo de datos de la aplicación utilizando data binding.
- **Reactivos:** Se basan en la programación reactiva y se utilizan para crear formularios más complejos y dinámicos. Son más flexibles y potentes que los formularios basados en plantillas y permiten realizar validaciones avanzadas y reaccionar a los cambios de forma dinámica.



Reto

Escribe el código de un Pipe que reciba como entrada una palabra que deba ser buscada y reemplazada por mayúsculas cerradas dentro de un texto.



SOLID

- **S:** Principio de responsabilidad única (Single Responsibility Principle)
- **O:** Principio de abierto/cerrado (Open/Closed Principle)
- **L:** Principio de sustitución de Liskov (Liskov Substitution Principle)
- **I:** Principio de segregación de interfaces (Interface Segregation Principle)
- **D:** Principio de inversión de dependencias (Dependency Inversion Principle)



Contacto

E. adiaz@consultec-ti.com
T. + 507 6408 1387
Panamá

 info@consultec-ti.com

 [@consulteclatam](https://www.instagram.com/consulteclatam)

 [@consultec-ti](https://www.linkedin.com/company/consultec-ti)

 consultec-ti.com



GRACIAS

¡Nos vemos pronto!