**1.- Instalación**

Se debe instalar desde el Installer de visual studio el componente de Node.js y crear un proyecto Angular. Hay 3 opciones de proyecto (se usará la primera):

Instalar Angular Cli desde línea de comandos con:

npm install -g @angular/cli

ASP NetCore con Angular (un solo proyecto), Proyecto independiente de Typescript Angular y Angular y ASP NET Core en proyectos independientes

No instalar Microsoft.TypeScript.MSBuild ni ESLint que sugiere el IDE

En clientApp->src->app se trabaja la parte de Angular con una carpeta para cada componente

Angular es Single Page y la pagina principal es index.html, el componente principal es app.conmponent

Cada componente se conforma de un archivo .html, .css y .ts

Se pueden hacer cambios a los componentes, se guardan los cambios y se refleja automáticamente en el explorador

En app.module.ts se definen los routes

El archivo tsconfig.json es para la configuración de typescript en el proyecto

Angular es reactivo, los cambios se reflejan automáticamente en front

Para agregar un componente realizar lo siguientes en la terminal desde la carpeta ClientApp

**ng generate component <nombre componente> --skip-import**

**2.- Construcción de un componente**

a) Crear una carpeta nueva con el nombre del componente en minúsculas

b) Agregarle sus tres archivos html, ts y css con la siguiente nomenclatura:

<nombre-componente>.component.<extension>

**.html**

<div>

Hola **{{**nombre**}}**

</div>

<input type="button" value="Presiona" (click)="AsignarNombre()" />

**.ts**

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

@Component({

selector: 'app-chat-component',

templateUrl: './chat.component.html'

})

export class ChatComponent {

public nombre = "";

public AsignarNombre() {

this.nombre="Oscar"

}

}

**App.module.ts (Ruteo)**

import { ChatComponent } from './chat/chat.component';

. . .

@NgModule({

declarations: [

. . .

ChatComponent

],

. . .

RouterModule.forRoot([

. . .

{ path: 'chat', component: ChatComponent }

])

**Menu.html**

<li class="nav-item" [routerLinkActive]="['link-active']">

<a class="nav-link text-dark" [routerLink]="['/chat']">Chat</a>

</li>

**3.-Sintaxis Typescript**

Declaración de variables y constantes

const name: string = "Strider";

let numero: number = 0;

let bool: boolean = false;

let arr: string[] = ["pedro","maria","jesus"];

No usar el tipo de datos any

Declaración de interfaces y objetos que las implementan

interface Persona {

nombre: string;

edad: number;

genero?: string // opcional

MuestraMensaje: (mensaje :string) => string;

}

const per: Persona={

nombre: 'oscar',

edad: 41,

MuestraMensaje: function (mensaje: string): string {

return "Hola " + mensaje;

}

}

this.nombre = per.MuestraMensaje("Oscarillo");

per.edad.toString();

Métodos

public Suma(a: number, b: number): number {

return a + b;

}

let res: number = this.Suma(1, 2);

public Saluda(p: Persona): string {

return p.MuestraMensaje("Hola a todos");

}

Desestructuraciones en Objetos

Tomando la siguiente estructura se puede obtener solo los valores requeridos

interface SuperHero {

name: string,

age: number,

address: direccion

}

interface direccion {

calle: string,

pais: string,

ciudad: string

}

public superHeroe: SuperHero = {

name: 'Spiderman',

age: 30,

address: {

calle: 'Main St',

pais: 'USA',

ciudad: 'NY'

}

}

const { name: minombre } = this.superHeroe;

const { calle: micalle } = this.superHeroe.address;

Desestructuraciones en Arrays

const [p1, p2, p3]: string[] = ["e1", "e2", "e3"];

Importaciones y exportaciones

Exportar es hacer publico un objeto/función/interface fuera del archivo typescript

export interface direccion {

calle: string,

pais: string,

ciudad: string

}

Para importar una funcionalidad desde otro archivo ts se realiza así:

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

Clases

export class ChatComponent {

constructor(private name:string,public address:string) { }

}

const obj = new ChatComponent("oscar", "arbolillo II");

Herencia. Aunque es mejor usar composicion

export class ChatComponent extends ClaseBase{}

Composición

Es tener una referencia a la clase padre utilizando un atributo, la clase padre generalmente es recibida como parámetro en el constructor

Genéricos

function whatsMyType<T>(argument: T): T {

return argument;

}

let mystring = whatsMyType<string>("Holilla");

let mynumber = whatsMyType<number>(10);

let myarray = whatsMyType<number[]>([0, 1, 2, 3]);

Decoradores

Cambia el comportamiento de nuestra clase para que se comporte como un servicio, componente, modulo. Ejemplo:

@Component({

selector: 'app-chat-component',

templateUrl: './chat.component.html'

})

export class ChatComponent {. . . }

Optional chaining

Es un if de una sola línea. Ejemplo:

Passenger.children?.length || 0;

**4.- Directivas**

\*ngFor: sirve para hacer bucles en el html. Ejemplo:

ts

export class ChatComponent {

public nombres: string[] = ["oscar", "juan", "pedro"];

}

html

<div \*ngFor="let elem of nombres">

**{{**elem**}}**

</div>

**5.- Invocación de componentes**

**Subcomponente html**

<div>

**{{**texto**}}**

</div>

**Subcomponente ts (se ocupa Input)**

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

@Component({

selector: 'app-message',

templateUrl: './message.component.html'

})

export class MessageComponent {

@Input() texto?: string;

}

**Invocador html**

<app-message \*ngFor="let elem of nombres" [texto]="elem">

</app-message>

**Invocador ts**

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

@Component({

selector: 'app-chat-component',

templateUrl: './chat.component.html'

})

export class ChatComponent {

public nombres: string[] = ["oscar", "juan", "pedro"];

}

**6.- Solicitud Http**

Controller

Se puede devolver la respuesta serializada o devolver una clase especifica que se tendrá que replicar en el cliente

[ApiController]

[Route("api/[controller]")]

public class BackendController : Controller

{

[HttpGet("[action]")]

public IActionResult ObtenerListaNombres()

{

List<string> list = new List<string>();

list.Add("melchor");

list.Add("gaspar");

list.Add("baltazar");

return Json(JsonConvert.SerializeObject(list) );

}

}

**.ts (se ocupa Httpclient, Inject y el método get)**

import { Component, Inject } from "@angular/core";

import { HttpClient } from '@angular/common/http';

@Component({

selector: "app-webservice",

templateUrl: "./webservice.component.html"

})

export class WebserviceComponent {

public nombres?: string[];

constructor(http: HttpClient, @Inject('BASE\_URL') baseUrl: string) {

http.get<string>(baseUrl + 'api/backend/ObtenerListaNombres').subscribe(result => {

this.nombres = JSON.parse(result);

}, error => console.error(error));

}

}

**Proxy.conf.js**

Se desactiva la seguridad para la url del servicio

"/api/backend"

**7.- Servicios**

Son proveedores de datos, aquí se pueden poner las conexiones al Core

Se crea una carpeta llamada services, se le agregan los servicios como archivos .ts con la nomenclatura <nombreservicio>.service.ts

Servicio .ts

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

@Injectable({

providedIn: 'root'

})

export class ChatService {

public mensaje: string = 'mensaje desde el servicio';

}

Se registra en el app.module

import { ChatService } from './services/chat.service';

. . .

providers: [ChatService],

. . .

Componente que lo invoca .ts. Se inyecta en el constructor y ya se puede usar

constructor(http: HttpClient, @Inject('BASE\_URL') baseUrl: string,

protected chatservice: ChatService

)

. . .

html

**{{**chatservice.mensaje**}}**

**8.- Observable**

Se ocupan para mejorar el rendimiento, ya que debido a que angular es reactivo puede ocupar muchos recursos todo el tiempo y con Observable solo reacciona ante cambios

Esto es semejante a programación asíncrona (async, task, await)

Servicio

import { Observable } from 'rxjs';

. . .

public ObtenerNombres():Observable<string[]> {

return this.http.get<string[]>(this.baseUrl + 'api/backend/ObtenerListaNombres');

}

Componente ts

. . .

public nombres?: Observable<string[]>;

constructor(protected chatservice: ChatService) {

this.nombres = chatservice.ObtenerNombres();

}

. . .

Componente html

<div \*ngFor="let elemento of nombres | async">

**{{**elemento**}}**

</div>

**9.- Post**