**FRONTEND ANGULAR**

**Crear el proyecto**

1. Abrir la consola y ubicarse en la carpeta donde se va a crear el proyecto y utilizar el comando:

**ng new nombre-proyecto**

Aceptar **Angular routing** (ruteo de angular)

Seleccionar el estilo de formato para CSS

Esperar la descarga del proyecto

1. Ingresar a la carpeta del proyecto y abrir vscode con el comando **code .**

Compilar el proyecto y abrirlo en el navegador con la consola de vscode ejecutando:

**ng serve --o**

Seleccionar si se desea compartir datos del proyecto con el equipo de angular.

**Crear componentes y demás elementos**

1. Crear los componentes en src/app/components con el comando:

**ng g c components/nombre-componente**

Se crea una carpeta con 4 archivos en components/ para cada componente.

En components/app.modules.ts, en declarations se agrega el componente: **NombreComponenteComponent** en pascal case

1. Crear las siguientes carpetas dentro de src/app:

**interfaces**

**services**

**shared**

1. Crear servicios dentro de la carpeta services con el comando:

**ng g s services/nombre-servicio**

1. Ir a la carpeta src/app/interfaces y crear un archivo **nombre-inteface**

Agregar el siguiente código:

export interface NombreInterface{

id?: number,

nombre: string,

edad: number

}

1. Instalar Angular Material en el proyecto con el comando:

**ng add @angular/material**

Aceptar la instalación.

Seleccionar el tema o usar un tema personalizado.

Aceptar instalar la tipografía de angular.

Aceptar instalar las animaciones.

Ver la documentación en **material.angular.io**

1. Para usar los componentes creados ir al archivo src/app/components/app.components.html y escribir el nombre del selector del componente:

<nombre-componente></nombre-componente>

El nombre del selector del componente se encuentra en el archivo

Components/nombre-component.ts

Para cambiar el html del componente ir a la carpeta del componente y modificar el archivo nombre-componente.html

1. Para usar componentes de Angular Material, importarlo en el archivo src/app/app.modules.ts según la documentación:

import { **nombre-componente** } from @angular/material/...;

@NgModule ({

Declarations:[...],

imports: [

**NombreComponente**,

],

})

Copiar el código ts, html y css según la documentación en los respectivos archivos:

**nombre-componente.ts**

**nombre-componente.html**

**nombre-componente.css**

**Configurar las rutas y parámetros**

1. En el archivo src/app/index.html tenemos el routing activado con la siguiente línea de código:

<router-outlet></router-outlet>

1. Agregar las rutas en src/app/app-routing.modules.ts, importando el componente y editando la constante Routes:

import { NombreComponente1 } from ‘./components/...’;

import { NombreComponente2 } from ‘./components/...’;

const routes: Routes = [

{ path: ‘’, redirectTo: ‘otro-path’, pathMatch: ‘full’},

{ path: ‘nombreRuta’, component: NombreComponente1},

{ path: ‘nombreRuta/:id’, component: NombreComponente2},

{ path: ‘\*\*’, redirectTo: ‘otro-path’, pathMatch: ‘full’},

];

Un path vacío (‘’) y un path que no coindice con la lista (‘\*\*’) serán redireccionados al path especificado. Es importante que esta configuración siempre tenga este orden.

Los path con parámetros se especifican con /:nombre-parámetro

1. Para utilizar las rutas en los componentes se agrega el siguiente código a un elemento html, cualquiera de las dos opciones:

<etiqueta routerLink = ”/nombreRuta”></etiqueta>

<etiqueta [routerLink]= ”[‘/nombreRuta’]”></etiqueta>

Si se indica en la ruta un diagonal (/) quiere decir que la ruta es absoluta y se redirecciona así:

**Localhost:puerto/nombreRuta**

Si no se usa diagonal (/) en el nombre de la ruta, se indica que la ruta es relativa a la ruta actual y se redirecciona así:

**Localhost:puerto/rutaActual/nombreRuta**

Para utilizar parámetros se pueden usar estas dos formas:

<etiqueta routerLink = ”/nombreRuta/{{parámetro}}”></etiqueta>

<etiqueta [routerLink]= ”[‘/nombreRuta’, parámetro]”></etiqueta>

Para usar routerLink en el componente, agregar en los imports:

@Component({

...

imports: [

RouterModule,

...

],

})

1. Para utilizar rutas anidadas se configura así:

const routes: Routes = [

{

path: 'america',

component: AmericaComponent,

children: [

{

path: 'chile',

component: ChileComponent

},

{

path: 'argentina',

component: ArgentinaComponent

},

{

path: 'uruguay',

component: UruguayComponent

}

]

},

**Agrupar imports en un solo archivo**

1. Cortamos todos los imports de Angular Material que hemos agregado en el archivo src/app/app.module.ts

Creamos un nuevo módulo en la carpeta app/shared con el comando:

**ng g m shared**

Abrimos el archivo app/shared/shared.module.ts

Pegamos todos los imports de Angular Material

Hacemos lo mismo con los imports de Angular Material del arreglo @NgModule. Los debemos poner dentro del arreglo @NgModule del archivo shared.module.ts, tanto en imports como en exports así:

@NgModule ({

Declarations:[...],

imports: [

**ComponentesAngularMaterial**,

],

Exports: [

**ComponentesAngularMaterial**

],

})

Por último, importamos shared.module.ts en src/app/app.module.ts para poder usar los componentes de Angular Material, así:

import { SharedModule } from ‘./shared/shared.module’;

@NgModule ({

Declarations:[...],

imports: [

**SharedModule**,

],

})

**Conectar con el backend**

1. Crear el archivo entorno mediante el comando: **ng g environments**

Abrir el archivo src/enviroments/environment.ts (para producción o environment.development.ts para desarrollo) y agregamos una variable de entorno:

export const environment = {

production: false,

endpoint: ‘https://localhost/puerto-backend/’

};

El endpoint es la dirección y puerto del servidor.

1. Abrir el archivo src/app/shared/shared.module.ts y agregar el módulo HttpClientModule:

import { HttpClientModule } from @angular/common/http;

...

imports: [

...

HttpClientModule

],

exports: [

...

HttpClientModule

]

1. Modificamos el archive creado en el punto 5, src/app/services/Nombre-servicio.service.ts y agregamos la variable de entorno con la url de la app:

import { HttpClient } from ‘@angular/common/http’;

import { environment } from ‘src/environment/environment’;

import { Observable } from 'rxjs';

...

Export class NombreService {

private myAppUrl: string = environment.endpoint;

private myRouterUrl: string = ‘api/NombreRuta-backend/’;

}

myRouterUrl es la ruta del controlador en el backend.

//inyectamos la dependencia HttpClient

constructor(private http: HttpClient){}

//Creamos un método para consultar la ruta del backend

getMyData(): Observable<**NombreInterface**[]>{

return this.http.get<**NombreInterface**[]>(

`${this.myAppUrl}${this.myRouterUrl}`

);

}

1. Inyectamos el servicio en el componente mediante inyección de dependencias. Vamos al archivo src/app/components/nombre-component/nombre-componente.ts y modificamos el siguiente código:

import { NombreService } from ‘src/app/services/nombre.service’;

...

constructor(private \_nombreService:NombreService){}

//Creamos una función para llamar a la función del servicio

getList(){

this.\_nombreService.**getMyData**().subscribe(data => {

console.log(data)

})

}

**BACKEND EN .NET CORE 6**

**Creación del proyecto**

1. Abrir visual studio y seleccionar nuevo proyecto/**ASP.NET Core Web API**

Seleccionar la carpeta y dejar las opciones por defecto.

1. Eliminar el controlador por defecto y la clase WeatherForecast.cs
2. Ir al menú herramientas/Administrador de paquetes NuGet/Administrar paquetes...

**Instalar dependencias**

1. Ir a Examinar e instalar los siguientes paquetes:

**Microsoft.EntityFrameworkCore**

**Microsoft.EntityFrameworkCore.Tools**

**Microsoft.EntityFrameworkCore.SqlServer**

**Crear modelos**

1. Seleccionar el proyecto y agregar una nueva carpeta llamada **Models**

Dentro de esta carpeta crear una nueva clase en Pascal case para generar los modelos correspondientes a cada tabla de la base de datos.

1. En el código del modelo podemos escribir **prop** y dar enter para que aparezca automáticamente el formato de propiedades:

public int MyProperty { get; set; }

Podemos agregar, por ejemplo:

public int Id { get; set; }

public string Nombre { get; set; }

public float Precio { get; set; }

public DateTime FechaCreacion { get; set; }

Consultar la documentación en:

https://www.entityframeworktutorial.net/efcore/entity-framework-core-console-application.aspx

1. Dentro de la carpeta Models crear una nueva clase llamada: **AplicationDbContext.cs**

Agregar el siguiente código:

using Microsoft.EntityFrameworkCore;

...

public class AplicationDbContext: DbContext

{

public AplicationDbContext(DbContextOptions<AplicationDbContext> options): base(options)

{

}

public DbSet<NombreModelo> NombreModelo { get; set; } = null!;

}

**Crear conexión a la base de datos**

1. Abrir el archivo appsettings.json en la raíz del proyecto y establecer la cadena de conexión a la base de datos:

...

“AllowedHosts”: “\*”,

“ConnectionStrings”: {

“Nombre-conexión”: “Server=.\\SQLExpress;Database=Nombre-base-datos;Trusted\_Connection=True; TrustServerCertificate=True”

}

El nombre del servidos se encuentra en SQL server. Reemplazar la cadena por el nombre de su servidor.

El parámetro TrustServerCertificate=True es opcional y en caso de que aparezca el error “La cadena de certificación fue emitida por una entidad en la que no se confía”.

1. Abrir el archivo Program.cs en la raíz del proyecto y agregar el contexto:

using NombreProyecto.Models;

using Microsoft.EntityFrameworkCore;

...

//Habilitar CORS

Builder.Services.AddCors(options => options.AddPolicy(“AllowWebapp”,

builder => builder.AllowAnyOrigin().AllowAnyHeader()

.AllowAnyMethod()));

builder.Services.AddDbContext<AplicationDbContext>(options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString(“Nombre-conexión”));

});

...

app.UseCors(“AllowWebapp”);

**Crear migraciones**

1. Abrir la consola del Administrador de paquetes en Herramientas/Administrador de paquetes NuGet, y ejecutar el siguiente comando:

**Add-Migration Nombre-migración**

La migración se crea automáticamente con los datos del Modelo. Ejecutamos el siguiente comando para crear la base de datos y realizar las migraciones:

**Update-database**

Automáticamente se crea la base de datos si no está creada con sus respectivas tablas.

**Crear controladores**

1. Ir a la carpeta controllers y crear nuevo controlador, seleccionar Controlador de API: en blanco.

El nombre del controlador debe terminar con la palabra Controller (NombreController).

Por defecto la ruta para acceder al controlador será:

**Localhost:puerto/api/NombreController**

Agregar las rutas en el controlador:

using Microsoft.EntityFrameworkCore;

using netcore6.Models;

...

public class ProductController : ControllerBase

{

private readonly AplicationDbContext \_context;

public ProductController(AplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Get()

{

try

{

//Consultar la base de datos

var productList = await \_context.Products.ToListAsync();

return Ok(productList);

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

}