Sales.

Arquitectura para programación Distribuida

1. Como es costumbre subirlo al gitlab
2. Creamos el proyecto seleccionamos Croos-platform
3. Luego otro proyecto MVC con autenticación Windows Sales.Backend
4. Luego otro proyecto con API con autenticación Windows Sales.API
5. Luego otro proyecto de clase library net. Framework Sales.Domain
6. Luego otro proyecto de clase library .net framework Sales.Common

Luego Creamos el cuerpo de la Base de Datos **Vacia**  en este caso anubis

HACEMOS LA CONEXIÓN EN SALES.BACKEND EN EL WEBCONFIG Y AHÍ AGREGAMOS LA CONEXIÓN

<connectionStrings>

<add name="DefaultConnection"

connectionString="Data Source=DESKTOP-QGTILJS;

Initial Catalog=anubis;User=sa;Password=020986;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

y esta conexión va en el API y el Backend.

/\*vamos a correr por defecto el bakcend para poder continuar\*/

Para configurar el tipo de contraseña en el archivo IdentityConfig.cs podemos cambiar los requerimientos para la contraseña.

// Configure validation logic for passwords

manager.PasswordValidator = new PasswordValidator

{

RequiredLength = 6,

RequireNonLetterOrDigit = true,

RequireDigit = true,

RequireLowercase = true,

RequireUppercase = true,

};

/\*AHORA VAMOS A REFERENCIAS LOS PROYECTO PARA QUE QUEDEN UNIDOS SEGÚN LA PROGRAMACION DISTRIBUIDA\*/ (ADD REFERENCE)

PROYECTO DOMAIN----USA---COMMON

PROYECTO API-------USA---COMMON Y A DOMAIN

PROYECTO BACKEND---USA----COMMON Y A DOMAIN

/\*AHORA NOS VAMOS AL NUGGET MANAGE PACKAGE Y LE INSTALAMOS A TODOS LOS SIGUIENTES PAQUETES\*/

NEWTONSOFT.JSON

/\*MODIFICANDO EL COMMON\*/

Eliminamos la clase que nos crea por default

AGREGAMOS UNA CARPETA DE NOMBRE **Models** luego lo que hacemos es crea una clase a la que llamaremos Product.cs y ahí sus atributos

/\*MODIFICANDO EL DOMAIN\*/

HA ESTE LE VAMOS AGREGAR EL NUGGET ENTITY FRAMEWORK Y VAMOS A ELIMINAR LA CLASE QUE NOS CREA POR DEFAULT, LUEGO UNA CARPETA DE NOMBRE **Models** y dentro de esta una clase de nombre **DataContext.cs**

public class DataContext:DbContext

{

public DataContext():base("DefaultConnection")

{

}

}

Donde DefaultConnection es el nombre de la conexión que hay dentro del webconfig del api y backend.

/\*AGREGANDO EL LOCAL DATA CONTEXT\*/

EN EL PROYECTO BACKEND VAMOS A GREGAR UN MODELO QUE LE LLAMAREMOS **LocalDataContext y dentro de este**

using Sales.Domain.Models;

public class LocalDataContext:DataContext

{

}

Esta clase va heredar de la referencia DOMAIN el context de conexión, una vez echo esto corremos nuevamente el Backend y posteriormente creamos nuestro primer controlador

Model = Sales.Common.Model.Product

ContextClass = LocalDBContext //este ya tiene la conexión del domain

Modo Asyncrono

Y Creamos.

Como prueba agregamos algunos productos

/\*\*\*\*\*\*RECUERDA REVISAR LA EVOLUCION DE LA BASE DE DATOS POR FAVOR\*\*/

/\*TRABAJANDO CON LA API\*/

GET EXTRAE INFORMACION

POST CREA COSAS

PUT MODIFICAR COSAS

DELETE BORRAR COSAS

/\*probamos los api con postman\*/

**/\*TRABAJANDO CON SALES (PARA MOVILES)\*/**

**OJO AQUÍ TAMBIEN AGREGAMOS COMO REFERENCIA EL COMMON PARA LA CLASE RESPONSE**

Creamos una carpeta que se llame **Services**

Dentro de ella clase **ApiService.cs**

**Ahora en el Common agregamos la clase Response**

**AQUÍ REVISA LA CLASE SERVICE DE ESTE PROYECTO**

**/\*Consumiendo los servicios con moviles\*/**

**/\*OJO PARA PROBAR LA API LOCALMENTE PODRAS VER EN EL BACKEND QUE AGREGUE EL CONTROLLADOS PAR API SIN EMBARGO NO ES NECESARIO ESTE EN CAMPO\*/**

**Width=”2\*”**

**Width=”\*”**

**Migraciones de la base de datos**

**Para la clase products agregamos para poder agregar una imagen,también las características asi como la integración de datta anotations**

public class Product

{

[Key]

public int ProductId { get; set; }

[Required]

[StringLength(50, ErrorMessage =

"The field {0} can contain maximun {1} and minimum {2} characters",

MinimumLength = 3)]

public string Description { get; set; }

[Display(Name = "Remarks")]

[DataType(DataType.MultilineText)]

public string Remarks { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}", ApplyFormatInEditMode = false)]

public Decimal Price{ get; set; }

[Display(Name ="Is Available")]

public bool IsAvailable { get; set; }

[Display(Name = "Publish On")]

[DataType(DataType.Date)]

public DateTime PublishOn { get; set; }

/\*agregando una imagen\*/

[Display(Name ="Image")]

public string ImagePath { get; set; }

//Esto lo hacemos para devolver un resultado directo

public override string ToString()

{

return this.Description;

}

}

Para poder actualizar la base de datos

En El proyecto del backend abriendo la consola de Nuggets, cargo este comando.

Enabled-Migrations –ContextTypeName aquí el contexto –EnabledAutomaticMigrations –Force

En este caso LocalDataContext es el nombre del contexto.

Nos va crear una carpeta en el backend de nombre Migrations dentro de esta una clase de nombre Configurations.cs y ahí en el constructor agregamos esta línea AutomaticMigrationDataLossAllowed = true;

Luego nos vamos al Global.asx y dentro del Application\_start() agregamos

Database.SetInitializer(new MigrateDatabaseToLatestVersion<Models.LocalDataContext, Migrations.Configuration>()); donde “LocalDatacContext” es el contexto en cuestion

**Cuando iniciamo el backend nos manda error hay que elimnar la tabla para que se vuelva a crear, si ya contiene datos exportar los datos y después volverlos a cargar.**