***Crear base de datos "Pymes" (SqlServer Local)***

* Crear la base “Pymes”
* Ejecutar el script de creación del proyecto Pymes
* Testear base con la consulta:
  + select \* from Articulos
  + select \* from ArticulosFamilias

Nota: EF maneja relaciones entre tabla, por medio de clave foráneas, (ej Clientes.IdDepartamento -> Departamentos.IdDepartamento). Sin embargo en Artículos y ArticulosFamilias se obvió dicha relación para simplificar el modelo.

***Crear Backend "PymesBackend" (Visual Studio Local)***

* Crear proyecto ‘PymesBackend’, tipo “Aplicación web asp (.net framework)”, opciones: vacío + “ApiWeb” + “configurar para https”
* A la solución: Agregar nuevo proyecto "Datos", tipo “Biblioteca de Clases (.net framework)”
  + Referenciar el proyecto Datos desde "PymesBackend”

Click derecho sobre el proyecto “PymesBackend”: Agregar referencia, pentaña proyectos, marcar al proyecto Datos.

* Nota: Se debe establecer “PymesBackend” como proyecto de inicio (Click Derecho sobre el proyecto -> Establecer como proyecto de inicio), sino cuando iniciemos la ejecucion con F5, se ejecutaría el proyecto actualmente seleccionado, y en el caso de “Datos”, por ser una biblioteca de clases, no se podría ejecutar.
* Desde el administrador de paquetes NUGET, agregar a ambos proyectos el paquete “EntityFramework”, para acceder a las librerias de Entity Framework
* Tambien desde NUGET, agrega el paquete Microsoft.AspNet.WebApi.Cors, para habilitar Cors (Cross-origin resource sharing), solo al proyecto PymesBackend
  + configurar la funcionalidad en el archivo WebApiConfig, agregando siguiente código al inicio del método “Register”:

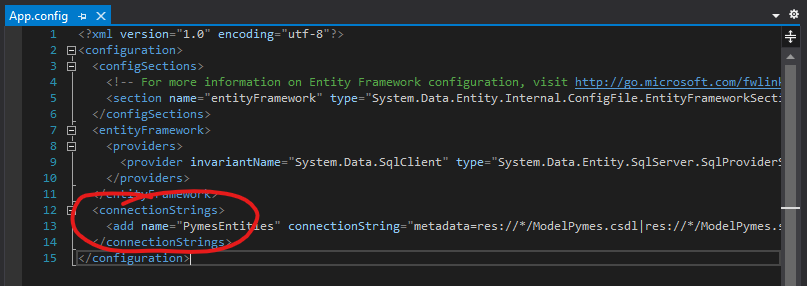
var cors = new System.Web.Http.Cors.EnableCorsAttribute("\*", "\*", "\*"); config.EnableCors(cors);

Nota: Se pueden buscar archivos, clases o métodos, fácilmente en una solución con el atajo “Ctrl+,”

* En el proyecto "Datos" agregar nuevo elemento: “ADO.NET Entity Data Model” que es el asistente para configurar Entity Framework. Configurarlo de la siguiente manera:
  + Nombre: “PymesModel”
  + EF Designer desde base de datos
  + Conexion a SQL server donde esta la base de datos pymes
    - sql: localhost\SQLEXPRESS (o el servidor que corresponda)
    - base: pymes
    - guardar en web.config como “pymesEntities”
  + Version: 6.x
  + incluir tabla: al menos ArticulosFamilias y Articulos, o todas las tablas para ver las relaciones entre las mismas
    - Espacio de nombre: “PymesModel”

Una vez completo el asistente:

* + Observe los modelos generados: Articulo, ArticuloFamilia (autogenerados: partial class)
  + Observe dbcontext: “PymesModel”
  + Observe cadena de conexión que se generó y guardo en app.config, tenemos que copiarla y llevarla al web.config del proyecto “PymesBackend” que es que configura toda la aplicación al ejecutarse la misma.
    - Copiar la etiqueta <connectionStrings> del App.config del proyecto Datos



* + - Pegarla en el mismo lugar del archivo Web.config del Proyecto PymesBackend
* Grabar y compilar Solución (caso contrario va a fallar el paso siguiente)
* En proyecto “PymesBackend”: Agregar (sobre la carpeta Controllers) => Nuevo elemento con Scaffold... => Controlador de Web Api 2 con acciones que usan Entity Framework
  + clase de modelo: ArticulosFamilias
  + clase de contexto de datos: PymesEntitties
  + Nombre Controlador ArticulosFamiliasController
    - repetir para Articulos

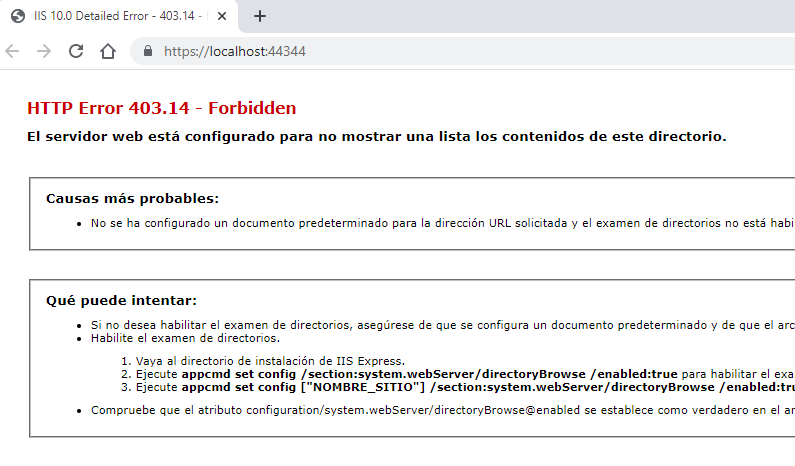
***Testear WebApi del Backend "PymesBackend"***

* Testear desde el browser las métodos get de las api:

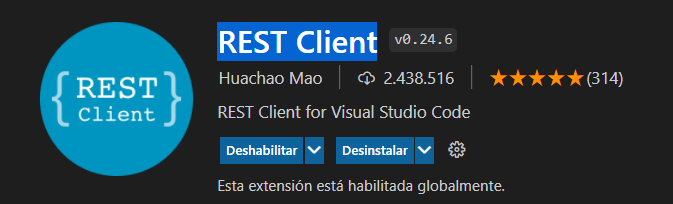
Por ej el recurso articulos (cambiar el número de puerto según corresponda)

* + <https://localhost:44345/api/articulos> (get todos los articulos)
  + [https://localhost:44345/api/articulos/1](https://localhost:44345/api/articulosfamilias/1) (get articulo segun su id)

Nota: observe que al iniciar el proyecto se visualiza una pantalla similar a la imagen siguiente, la cual indica que el sitio al no tener una página predeterminada para visualizar intenta mostrar una lista de los archivos disponibles y por cuestiones de seguridad la misma devuelve un error 403. Esto no significa que el sitio no puede responder a la url específica de los controladores creados anteriormente..



* Testear desde **Visual Code** (no el visual studio!) todos los métodos de webapi de los recursos, ya que desde la url del browser solo se invoca los métodos get
  + Instalar en Visual Code la extensión



* + Crear un archivo con extensión .http para el test
    - ej TestWebApi.http
  + Incluir el siguiente código, para las pruebas de todos los métodos.

###

get https://localhost:44344/api/articulos

####

https://localhost:44344/api/articulos/1

### alta

post https://localhost:44344/api/articulos

content-type: application/json

{

"IdArticulo": 0,

"Nombre": "ARTICULO NUEVO",

"Precio": 299.00,

"CodigoDeBarra": "0779815559001",

"IdArticuloFamilia": 10,

"Stock": 329,

"FechaAlta": "2017-01-19T00:00:00",

"Activo": true

}

### get el articulo dado de alta anteriormente, o modificado mas adelante

https://localhost:44344/api/articulos/505

### modificacion

put https://localhost:44344/api/articulos/505

content-type: application/json

{

"IdArticulo": 505,

"Nombre": "ARTICULO NUEVO modif",

"Precio": 299.00,

"CodigoDeBarra": "0779815559001",

"IdArticuloFamilia": 10,

"Stock": 329,

"FechaAlta": "2017-01-19T00:00:00",

"Activo": true

}

### baja

delete https://localhost:44344/api/articulos/505

Observaciones: para testear cada uno de los metodos de la webapi, posicionese sobre la linea del verbo con la url y mediante boton derecho del mouse ejecute la opcion: “Send Request”

***Encapsular en el Proyecto “Datos” la funcionalidad de acceso a datos***

* Encapsular la funcionalidad de EF que está involucrada en el controlador “ArticulosController” llevándola a la clase Datos.GestorArticulos  
  + En el proyecto Datos creamos una nueva clase pública llamada GestorArticulos, para sea el responsable de implementar el acceso a datos preferido (EF, Ado.net, Hibernate, etc).
  + Usaremos métodos estáticos (para simplificar su acceso, ya que no mantienen estado)
  + Vamos a implementar los metodos según el grado de complejidad
    - **Metodo BuscarPorId**
      * observe: using, Find y FirstOrDefault

public static Articulo BuscarPorId(int sId)

{

using (pymesEntities db = new pymesEntities()) **//el using asegura el db.dispose() que libera la conexion de la base**

{

return db.Articulos.Where(x => x.IdArticulo == sId).FirstOrDefault();

//return db.Articulos.Find(sId);

}

}

* + - **Metodo Buscar (version inicial)**

**public static List<Articulo> Buscar()**

**{**

**using (pymesEntities db = new pymesEntities()) //el using asegura el db.dispose() que libera la conexion de la base**

**{**

**var Lista = db.Articulos.ToList();**

**return Lista;**

**}**

**}**

* + - **Metodo Grabar, para altas y modificaciones**
      * validaciones en el backend
      * concepto de Entry en la modificación.
      * error significativo para el usuario

public static void Grabar(Articulo Item)

{

// validar campos

string erroresValidacion = "";

if (string.IsNullOrEmpty(Item.Nombre))

erroresValidacion += "Nombre es un dato requerido; ";

if (Item.Precio == null || Item.Precio == 0)

erroresValidacion += "Precio es un dato requerido; ";

if (!string.IsNullOrEmpty(erroresValidacion))

throw new ApplicationException(erroresValidacion);

// grabar registro

using (pymesEntities db = new pymesEntities())

{

try

{

if (Item.IdArticulo != 0)

{

db.Entry(Item).State = EntityState.Modified; //vincula entidad y la marca como modificada

db.SaveChanges();

}

else

{

db.Articulos.Add(Item);

db.SaveChanges();

}

}

catch (Exception ex)

{

if (ex.ToString().Contains("UK\_ArticulosNombre"))

throw new ApplicationException("Ya existe otro Artículo con ese Nombre");

else

{ // loguearlo y devolver msj generico…

throw new ApplicationException("En estos momentos no podemos procesar su solicitud, intente nuevamente más tarde (ref xxxx)" );

}

}

}

}

* + - **Metodo: ActivarDesactivar**
      * Version Inicial: baja fisica

public static Articulo ActivarDesactivar(int id)

{

using (pymesEntities db = new pymesEntities())

{

// baja fisica.

Articulo articulo = db.Articulos.Find(id);

db.Articulos.Remove(articulo);

db.SaveChanges();

return articulo;

}

}

* + - * Version Final: cambiar la baja fisica por logica y que tambien para que pueda revertirla: poner activo = true/false (concepto de activar/desactivar)

public static Articulos ActivarDesactivar(int id)

{

using (pymesEntities db = new pymesEntities())

{

// baja logica y proceso inverso

Articulo articulo = db.Articulos.Find(id);

articulo.Activo = !articulo.Activo;

db.SaveChanges();

return articulo;

}

}

* + - * Version final alternativa: se podria ejecutar un sql directo a la base, con parametros.

public static void ActivarDesactivar(int IdArticulo)

{

using (pymesEntities db = new pymesEntities())

{

//ref Entity Framework; ejecutar codigo sql directo

db.Database.ExecuteSqlCommand("Update Articulos set Activo = case when ISNULL(activo,1)=1 then 0 else 1 end where IdArticulo = @IdArticulo",

new SqlParameter("@IdArticulo", id)

);

}

}

* + - **Metodo Buscar (version final)**
  + paso 1 de 2: Modificar el controlador Articulos para que acepte los parámetros: Nombre, Activo y NumeroPagina

public IHttpActionResult GetArticulos(string Nombre = "", bool? Activo = null, int Pagina = 1)

{

//ref webapi tipos de retorno de los metodos; cambiamos la devolucion generica del metodo: IQueryable<Articulos> por IHttpActionResult para poder devolver tambien RegistrosTotal

int RegistrosTotal;

//ref c# var

var Items = Datos.GestorArticulos.Buscar(Nombre, Activo, Pagina, out RegistrosTotal);

return Ok(new { Items = Items, RegistrosTotal });

}

* + paso 2 de 2) modificar el gestor de datos para implementar el filtro y el paginado, devolver datos y cantidad de registros encontrados.

Nota: Para hacer skip y take tiene que estar ordenado por el nombre

public static List<Articulo> Buscar(string Nombre, bool? Activo, int? Pagina, out int RegistrosTotal)

{

//ref EF

using (pymesEntities db = new pymesEntities()) //el using asegura el db.dispose() que libera la conexion de la base

{

IQueryable<Articulo> consulta = db.Articulos;

// aplicar filtros

//ref LinQ

//Expresiones lambda, metodos de extension

if (!string.IsNullOrEmpty(Nombre))

consulta = consulta.Where(x => x.Nombre.ToUpper().Contains(Nombre.ToUpper())); // equivale al like '%TextoBuscar%' de sql

if (Activo != null)

consulta = consulta.Where(x => x.Activo == Activo);

RegistrosTotal = consulta.Count();

// ref EF; consultas paginadas

int RegistroDesde = 1;

int CantidadRegistros;

if (Pagina != null)

{

RegistroDesde = (Pagina.Value - 1) \* 10;

CantidadRegistros = 10;

}

else

CantidadRegistros = 1000;

var Lista = consulta.OrderBy(x => x.Nombre).Skip(RegistroDesde).Take(CantidadRegistros).AsNoTracking().ToList(); // la instruccion sql recien se ejecuta cuando hacemos ToList()

return Lista;

}

}

Nota: Testear el método buscar desde el browser, con la siguiente sintaxis:

https://localhost:44345/api/articulos?Nombre=a&Pagina=1

**Testear Backend desde aplicación local Frontend “Pymes”**

* cambiar la url de los servicio para que apunte a la webapi del servidor local.

**Testear Backend desde aplicación Frontend “Pymes2021” (StackBlitz online)**

* hacer un fork de la app
* cambiar la url del servicio para que apunte a la webapi del servidor local.
  + tener en cuenta que debe apuntar a la url de ***https*** del proyecto (no a la url de http)

**Bajar el proyecto de StackBlitz, para probarlo localmente, seguir estos pasos:**

* bajar proyecto y descomprimirlo
* ejecutar "npm install"
* Cambiar configuración del archivo tsconfig.json
* Grabar el Proyecto y ejecutarlo con "ng serve --open"
* Testear la aplicación