Programa:	**Programa:** Análisis y Desarrollo de Software
Instructor:	Jairo Augusto Arboleda L.
RA:	
Criterios de evaluación	**USANDO MEJORES BUENAS PRACTICAS Y Fluent API** para configurar la base de datos.
Fecha:	
Tema:	Desarrollando **mejores** Apps con Fluent API y Mejorando la seguridad.

Continuando con el ejercicio anterior, debemos tener en cuenta que la información de la **conexión a la BD como usuario y contraseña queda expuesta** en el **program.cs** esto es mala práctica, por tanto, por **seguridad** debemos realizar el siguiente cambio.

1. **Abra el appSettings.json, originalmente está así:**

```
{} appsettings.json > ...
1   {
2     "Logging": {
3       "LogLevel": {
4         "Default": "Information",
5         "Microsoft.AspNetCore": "Warning"
6       }
7     },
8     "AllowedHosts": "*"
9   }
```

Corte la conexión

2. Vaya a **Program.cs** a la línea 8 donde está la conexión y córtela de allá, Como se muestra en la siguiente imagen.

```
8   builder.Services.AddSqlServer<TareasContext>("");
```

3. Péguela en el siguiente archivo: **appsettings.json** y haga los cambios hasta quedar así:

```
{} appsettings.json > ...
1   {
2     "Logging": {
3       "LogLevel": {
4         "Default": "Information",
5         "Microsoft.AspNetCore": "Warning"
6       }
7     },
8     "AllowedHosts": "*",
9     "ConnectionStrings": {
10      "cnTareas": "Data Source=DESKTOP-P58V141\\SQLEXPRESS;Initial Catalog=TareasDb;Integrated Security=True"
11    }
12  }
```

4. Debes acceder a la conexión del **appsettings.json** desde el **Program.cs usando: cnTareas. Con este String de conexión se debe conectar de forma segura.**

```
8   builder.Services.AddSqlServer<TareasContext>(builder.Configuration.GetConnectionString("cnTareas")
9   var app = builder.Build();//App corriendo y escuchando peticiones
```

Probar Postman…

```
PS C:\projectef> dotnet build
MSBuild version 17.6.3+07e294721 for .NET
  Determinando los proyectos que se van a restaurar...
  Todos los proyectos están actualizados para la restauración.
  projectef -> C:\projectef\bin\Debug\net7.0\projectef.dll

Compilación correcta.
    0 Advertencia(s)
    0 Errores
```

5. **Nota: quizás pueda generarle estos errores al correr o compilar la APP,**

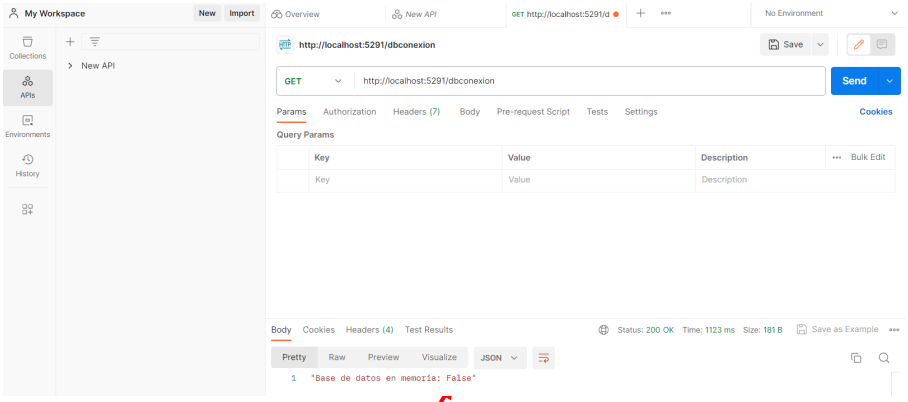
```
PS C:\projectef> dotnet run
Compilando...
Unhandled exception. System.IO.IOException: Failed to bind to address http://127.0.0.1:5291: address already in use.
 ---> Microsoft.AspNetCore.Connections.AddressInUseException: Solo se permite un uso de cada dirección de socket (pro
de red/puerto)
 ---> System.Net.Sockets.SocketException (10048): Solo se permite un uso de cada dirección de socket (protocolo/direc
o)
   at System.Net.Sockets.Socket.UpdateStatusAfterSocketErrorAndThrowException(SocketError error, Boolean disconnectOn
allerName)
```

Solo basta con cerrar la App y abrir de nuevo.

6. **Si todo funciona bien, compile, ejecute y debe obtener esta pantalla.**

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5291
```

7. **Para probar de nuevo vaya al Postman y haga lo siguiente:**



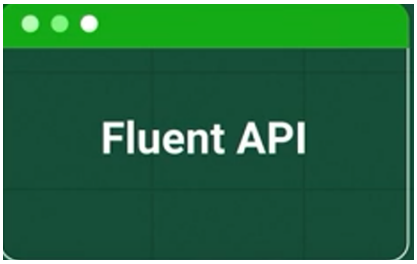
8. **Al ejecutar el Postman se devuelve un Script como este en la consola Powershell.**

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

        (SELECT *
         FROM [sys].[objects] o
         WHERE [o].[type] = 'U'
         AND [o].[is_ms_shipped] = 0
         AND NOT EXISTS (SELECT *
             FROM [sys].[extended_properties] AS [ep]
             WHERE [ep].[major_id] = [o].[object_id]
                 AND [ep].[minor_id] = 0
                 AND [ep].[class] = 1
                 AND [ep].[name] = N'microsoft_database_tools_support'
         )
    )
    SELECT 1 ELSE SELECT 0
```

PARTE 3

USANDO FLUENT API PARA CONFIGURAR O HACER EL DISEÑO DEL BD, EN ENTITY FRAMEWORK.

OTRA FORMA AVANZADA



Es una forma avanzada de configurar los modelos de Entity Framework sin utilizar atributos o data-annotations, permitiendo diseñar la base de datos considerando aspectos avanzados.

Se usan funciones de extensión anidadas para configurar tablas, columnas y especificar el mapeo de los datos.

Ejemplo

```
builder.Entity<Client>(entity =>
{
    entity.ToTable("Client");
    entity.HasKey(e => e.PersonId);
    entity.Property(e => e.ContactName).IsRequired(false).HasMaxLength(300);
    entity.Property(e => e.ContactPhone).IsRequired(false).HasMaxLength(50);
    entity.Property(e => e.Phone2).IsRequired(false).HasMaxLength(50);

    entity.HasOne(e => e.Person)
        .WithOne(e => e.Client)
        .HasForeignKey<Client>(b => b.PersonId)
        .HasConstraintName("FK_Client_Person")
        .OnDelete(DeleteBehavior.SetNull);
});
```

https://platzi.com/blog/comparativa-de-atributos-vs-fluent-api-entity-framework-net/

CREANDO EL MODELO DE CATEGORÍAS CON FLUENT API

9. Regresar a **TareasContext.cs** y observe el archivo original

```
TareasContext.cs > ...
1   using Microsoft.EntityFrameworkCore;
2   using projectef.Models;
3   namespace projectef;
    4 references
4   public class TareasContext:DbContext
5   { //aqui se importa la libreria  using...
6   //Representan las tablas dentro de la BD
    0 references
7   public  DbSet<Categoria> Categorias{get;set;}// Si diera error, importar using projectef.Models;
    0 references
8   public  DbSet<Tarea>Tareas{get;set;}
9
10  //Crear el método base del constructor, estas 11 lineas permiten hacer la  implementación
    0 references
11  public TareasContext (DbContextOptions<TareasContext>options):base(options){}
12  }
```

10. **Codifique las tablas y campos de la clase sí:**

```
7   public  DbSet<Categoria> Categorias{get;set;}// Si diera error, importar using projectef.M
    0 references
8   public  DbSet<Tarea>Tareas{get;set;}
9
10  //Crear el método base del constructor, estas 11 lineas permiten hacer la  implementación
    0 references
11  public TareasContext (DbContextOptions<TareasContext>options):base(options){}
    0 references
12  protected override void OnModelCreating(ModelBuilder modelBuilder)
13  {
14      modelBuilder.Entity<Categoria>(categoria=>
15      {
16          categoria.ToTable("Categoria");//Crear la tabla categoria
17          categoria.HasKey(p=> p.CategoriaId);// Crear el CategoriaId
18          //Regresa al modelo de categorias y comenta la linea [Key] ya no se requiere
19
20          });
```

11. Vaya a la clase **Categoria.cs** y elimine o comente la línea 7 en este caso:

```
7       // [Key]//Data anotation, Define la clave primaria de la tabla
```

Ya no se requiere.

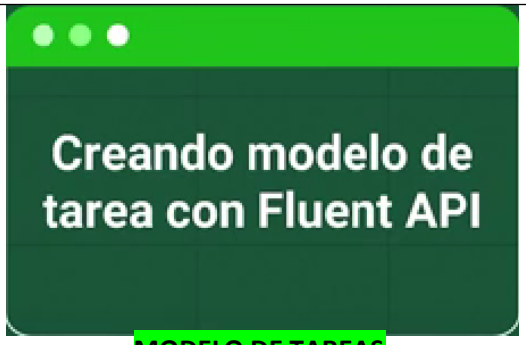
12. Regrese a **TareasContext.cs** y codifique los campos faltantes:

```
12  protected override void OnModelCreating(ModelBuilder modelBuilder)
13  {
14      modelBuilder.Entity<Categoria>(categoria=>
15      {
16          categoria.ToTable("Categoria");//Crear la tabla categoria
17          categoria.HasKey(p=> p.CategoriaId);// Crear el CategoriaId
18          //Regresa al modelo de categorias y comenta la linea [Key] ya no se requiere
19          categoria.Property(p=> p.Nombre).IsRequired().HasMaxLength(150);//Crear y Validar el campo categoria
20          categoria.Property(p=>p.Descripcion);
21
22          });
23  }
24
25  }
```

13. Vaya a **Categoría.cs** y comente las líneas 9 y 10 en este caso. Ya no se requieren y podrían entrar en conflicto. Ya esto quedó resuelto en **TareasContext.cs.**

```
8       public Guid CategoriaId{get;set;}
9       // [Required]
10      //[MaxLength(150)]
```

Fin de la configuración del modelo CATEGORIAS usando Fluent API.



MODELO DE TAREAS

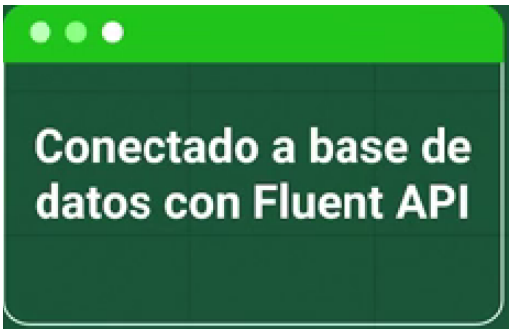
1. Vaya a la clase **Tarea.cs**
Observe qué cambios debemos hacer en la clase.

2. Regrese a **TareasContext.cs y** agregue **la entidad Tarea**

```
        categoria.HasKey(p=> p.CategoriaId);// Crear el CategoriaId
        //Regresa al modelo de categorias y comenta la linea [Key] ya no se requiere
        categoria.Property(p=> p.Nombre).IsRequired().HasMaxLength(150);//Crear y Validar el campo Nombre
        categoria.Property(p=>p.Descripcion);

        });
    modelBuilder.Entity<Tarea>(tarea=> //Crear la entidad Tarea
    {
        tarea.ToTable("Tarea"); //Crear la tabla Tarea
        tarea.HasKey(p=>p.TareaId); // Crear la Pk dela tabla
        tarea.HasOne(p=>p.Categoria).WithMany(p=>p.Tareas).HasForeignKey(p=>p.CategoriaId);//Clave foránea
        tarea.Property(p=>p.Titulo).IsRequired().HasMaxLength(200);
        tarea.Property(p=>p.Descripcion);
        tarea.Property(p=>p.PrioridadTarea);
        tarea.Property(p=>p.FechaCreacion);

    });

    }
}
```

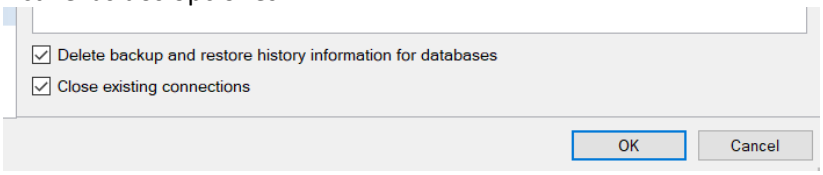
3. Regrese a la clase **Tarea.cs** y comente las líneas: 21 [**NotMapped**]
   YA QUEDÓ CONFIGURADO EL MODELO USANDO Fluent API

4. Hemos creado los modelos **CATEGORIA** Y **TAREA ahora conectemos la BD mediante Fluent API.**

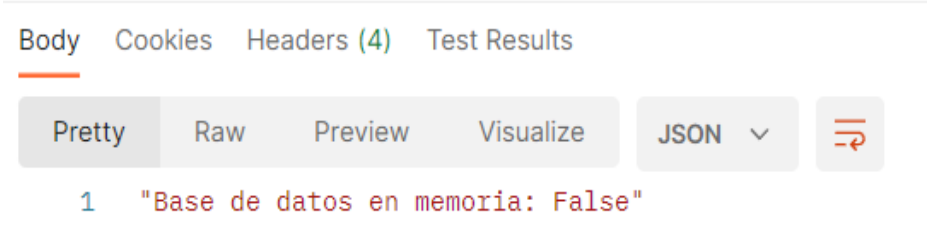


5. Antes debemos probar si nos estamos conectando con Postman aún

6. Vaya el Management Studio y Elimine la BD **TareasDB**
   Active las dos opciones

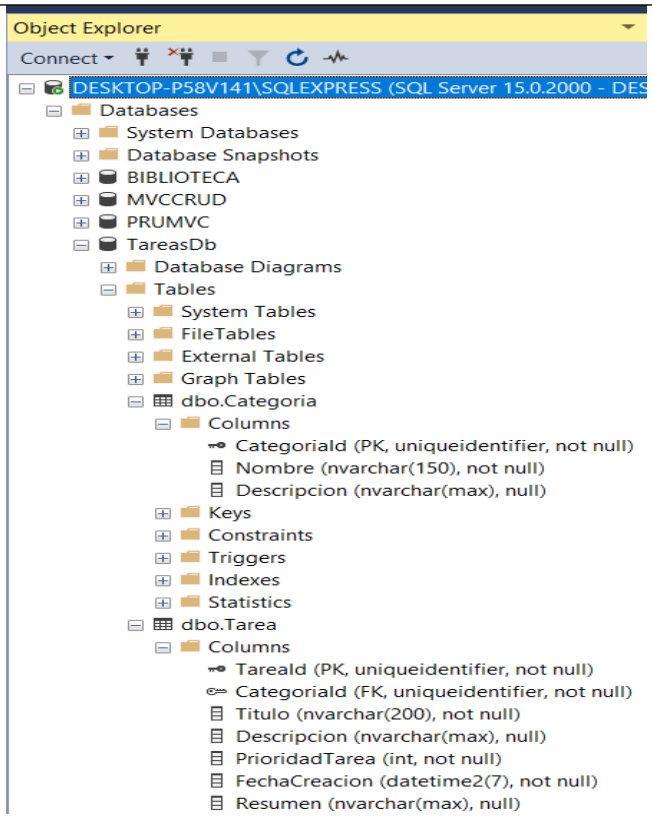


7. Ejecute el proyecto dotnet run , si da error cierre VSC y ábralo de nuevo
   Vaya al Postman y ejecute de nuevo el end point
   Le debería devolver este mensaje:



Significa que la BD de datos se creó con el nuevo modelo de Fluent API.

8. Verificar que efectivamente se creó la BD en SQL
   Si no aparece abra y cierre el SQL y listo.

9. 

10. Observar que **Resumen** no debía quedar almacenado en la Base de datos, por tanto, se debe evitar esto.
    ¿Cómo solucionarlo?
    - Vaya a **TareasContext.cs** y detenga la ejecución, luego adicione la última línea. Con esta función no se agregará ala BD cuando corra el Postman.

```
tarea.Property(p=>p.PrioridadTarea);
tarea.Property(p=>p.FechaCreacion);
tarea.Ignore(p=>p.Resumen);//No guardar en BD
```

11. Para probar, ejecute dotnet run luego elimine la BD desde el Management y luego ejecute Postman y analice si ya en la tabla no aparece el campo. Recuerde refrescar el SQL.

FIN