

Ejercicios JSON

```
#region Lista Generica
[HttpGet("generica")]
public IActionResult ListaGenerica()
{
    var personas = new List<Persona>
    {
        new Persona { Id = 1, Nombre = "Alice", Edad = 30 },
        new Persona { Id = 2, Edad = 25 }
    };

    return Ok(JsonHelper.ToJson(personas));
}
#endregion
```

```
#region Lista Generica
[HttpGet("diccionario")]
public IActionResult Diccionarios()
{
    var diccionario = new Dictionary<string, string>
    {
        { "clave1", "valor1" },
        { "clave2", "valor2" }
    };

    return Ok(JsonHelper.ToJson(diccionario));
}
#endregion
```

```
#region dynamic (ExpandoObject)
[HttpGet("dinamico")]
public IActionResult ObjetoDinamico()
{
    dynamic objetoDinamico = new ExpandoObject();
    objetoDinamico.Nombre = "Carlos";
    objetoDinamico.Edad = 40;
    objetoDinamico.Activo = true;

    return Ok(JsonHelper.ToJson(objetoDinamico));
}
#endregion
```

```
#region IEnumerable
[HttpGet("ienumerable")]
```

```
public IActionResult IEnumerableEjemplo()
{
    IEnumerable<string> numeros = new List<string>
    {
        "Uno",
        "Dos",
        "Tres"
    };

    return Ok(JsonHelper.ToJson(numeros));
}
#endregion
```

```
#region Hashtable
[HttpGet("hashtable")]
public IActionResult HashtableEjemplo()
{
    var hashtable = new Hashtable
    {
        { "uno", 1 },
        { "dos", 2 }
    };
    // Convertir Hashtable a Dictionary para serializar
    var dictFromHashtable = hashtable
        .Cast<DictionaryEntry>()
        .ToDictionary(k => k.Key.ToString(), v => v.Value);

    return Ok(JsonHelper.ToJson(dictFromHashtable));
}
#endregion
```

```
#region Queue
[HttpGet("cola")]
public IActionResult QueueEjemplo()
{
    var cola = new Queue<string>();
    cola.Enqueue("Primero");
    cola.Enqueue("Segundo");
    cola.Enqueue("Tercero");

    return Ok(JsonHelper.ToJson(cola));
}
#endregion
```

```
#region Stack
[HttpGet("pila")]
public IActionResult StackEjemplo()
{

```

```
var pila = new Stack<int>();
pila.Push(100);
pila.Push(200);
pila.Push(300);

return Ok(JsonHelper.ToJson(pila));
}
#endregion
```

```
#region HashSet
[HttpGet("hash")]
public IActionResult HashEjemplo()
{
    var conjunto = new HashSet<string> { "uno", "dos", "tres", "uno", "dos" };

    return Ok(JsonHelper.ToJson(conjunto));
}
#endregion
```

```
#region Compleja
[HttpGet("anidada")]
public IActionResult AnidadaEjemplo()
{
    // Estructura anidada
    var personasGrupo1 = new List<Persona>
    {
        new Persona { Id = 1, Nombre = "Ana", Edad = 28 },
        new Persona { Id = 2, Nombre = "Luis", Edad = 35 }
    };

    var personasGrupo2 = new List<Persona>
    {
        new Persona { Id = 3, Nombre = "María", Edad = 22 },
        new Persona { Id = 4, Nombre = "Juan", Edad = 40 }
    };

    var estructuraCompleja = new List<Dictionary<string, List<Persona>>>
    {
        new Dictionary<string, List<Persona>> { { "GrupoA", personasGrupo1 } },
        new Dictionary<string, List<Persona>> { { "GrupoB", personasGrupo2 } }
    };

    return Ok(JsonHelper.ToJson(estructuraCompleja));
}
#endregion
```

Lista Síncrona

```
[Route("api/personas")]
```

```
[ApiController]
```

```
public class ListSincronoController : ControllerBase
```

```
{
```

```
    private static List<Persona> personas = new()
```

```
    {
```

```
        new Persona { Id = 1, Nombre = "Ana", Edad = 28 },
```

```
        new Persona { Id = 2, Nombre = "Luis", Edad = 35 }
```

```
    };
```

```
// GET: /personas
```

```
https://localhost:44361/api/personas
```

```
[HttpGet]
```

```
public ActionResult<IEnumerable<Persona>> GetAll()
```

```
{
```

```
    return Ok(personas);
```

```
}
```

```
// GET: /personas/{id}
```

```
https://localhost:44361/api/personas/1
```

```
[HttpGet("{id:int}")]
```

```
public ActionResult<Persona> GetById(int id)
```

```
{
```

```
    var persona = personas.FirstOrDefault(p => p.Id == id);
```

```
    return persona is not null ? Ok(persona) : NotFound();
```

```
}
```

```
// POST: /personas
```

```
https://localhost:44361/api/personas
```

```
{
```

```
"nombre": "Carlos",
```

```
"edad": 42
```

```
}
```

```
[HttpPost]
```

```
public ActionResult<Persona> Create(Persona nueva)
```

```
{
```

```
    nueva.Id = personas.Any() ? personas.Max(p => p.Id) + 1 : 1;
```

```
    personas.Add(nueva);
```

```
    return CreatedAtAction(nameof(GetById), new { id = nueva.Id }, nueva);
```

```
}
```

```
// PUT: /personas/{id}
```

```
https://localhost:44361/api/personas/4
```

```
{
```

```
"nombre": "Juan Jose",
```

```
"edad": 30
}

[HttpPut("{id:int}")]
public IActionResult Update(int id, Persona actualizada)
{
    var persona = personas.FirstOrDefault(p => p.Id == id);
    if (persona is null) return NotFound();

    persona.Nombre = actualizada.Nombre;
    persona.Edad = actualizada.Edad;
    return Ok(persona);
}
```

```
// DELETE: /personas/{id}
https://localhost:44361/api/personas/1
[HttpDelete("{id:int}")]
public IActionResult Delete(int id)
{
    var persona = personas.FirstOrDefault(p => p.Id == id);
    if (persona is null) return NotFound();

    personas.Remove(persona);
    return NoContent();
}
}
```

27/08/2025

- ☐ Clone del GIT
- ☐ Repaso de los ejercicios propuestos
- ☐ Que es un CRUD
- ☐ Ejercicio de listas Síncrono
- ☐ Tutorial Básico: Crear colecciones en Postman
- ☐ Conceptos de Try/Catch

29/08/2025

- ☐ Ejercicios: **listas Síncrono**
 - ☐ Conceptos de Try/Catch
 - ☐ Principios Solid
 - ☐ Estándares de código, Buenas prácticas
 - ☐ Que es ORM
 - ☐ Explicar que es Code First
 - ☐ Primer CRUD con Entity Framework
-

Primer CRUD con Entity Framework

1. Crear un CRUD Basico

a. Code First

- i. crear primero el modelo (entidades y contexto). La base de datos se genera a partir de nuestro código, usando migraciones.

b. Database First

- i. El modelo se genera a partir de una base de datos ya existente.

c. Crear una carpeta Models

d. Crear el archivo [Person.cs](#)

- i. Id, Name (required), Age (required)

```
public class Person
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    [Required]
    public string LastName { get; set; }

    public DateTime Birthday { get; set; }
}
```

ii. Propiedades de la clase

1. Get (Obtener el valor desde fuera de la clase)
2. Set (Modificar el valor desde fuera de la clase)
3. Nos permite controlar el acceso a los campos de la clase, de manera segura

e. Instalar paquetes

- i. Microsoft.EntityFrameworkCore
- ii. Microsoft.EntityFrameworkCore.SqlServer
- iii. Microsoft.EntityFrameworkCore.Tools

f. Crear la carpeta **Context**

i. Crear una clase **AppDbContext.cs**

1. Que es: Es una clase fundamental del EntityFramework

ii. Heredar de **DbContext**

iii. Crear un constructor

1. `public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)`

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }
}
```

```

    {
    }
}

public DbSet<Person> Persons { get; set; }

```

- iv. Configurar la clase DbSet
 1. public DbSet<Person> Persons { get; set; }
 2. Es nombrarla en plural según convención
- v. Que es el DbSet
 1. Representa una colección de entidades en el contexto de las bases de datos
- g. Registrar el servicio del Db Connection (Cadena de conexión)
 - i. Buscar el archivo **appsettings**
 - ii. "ConnectionStrings": {


```

          "Connection":
            "Server=.\SQLExpress;Database=DBPerson;Trusted_Connection=true;TrustServerCertificate=true;"
          }
          
```
 - iii. URL: 1:

<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/connection-string-syntax>
 - iv. URL 2:

<https://aspnetcoremaster.com/connectionstring/2019/02/27/cadenas-de-conexion-csharp.html>
- h. Usar la cadena de conexión en Program.cs
 - i. Crear variable para la cadena de conexión


```

var connectionString =
  builder.Configuration.GetConnectionString("Connection");
          
```
 - ii. Registrar el servicio para la conexión


```

builder.Services.AddDbContext<AppDbContext>(options =>
  options.UseSqlServer(connectionString));
          
```
- i. Migraciones
 - i. Herramientas > Administrador de paquetes Nuget > Consola del administrador de paquetes
 - ii. **Add-Migration Initial** -> Crear la carpeta Migrations con las instrucciones
 - iii. **Error:** Si sale algún error en propiedades de la solución, establecer en false el valor de "InvariantGlobalization"
 - iv.


```

          <PropertyGroup> <TargetFramework>net8.0</TargetFramework>
          <ImplicitUsings>enable</ImplicitUsings>
          <InvariantGlobalization>>false</InvariantGlobalization> </PropertyGroup>
          
```
 - v.
 - vi. **Update-database**
- j. Crear los controladores
 - i. Nuevo controlador > API > Controlador API con acciones que usan Entity Framework

Agregar Controlador de API con acciones que usan Entity Framework

Clase de modelo	Person (BackendTraining.Models)
Clase DbContext	AppDbContext (BackendTraining.Context) +
Proveedor de base de datos	Configurado a partir del DbContext seleccionado
Nombre de controlador	PersonController

Agregar Cancelar

Llamada a Puntos

Get: <https://localhost:44325/api/People>

Get Id: <https://localhost:44325/api/People/1>

Post: <https://localhost:44325/api/People>

```
{
  "Name": "Juan",
  "LastName": "Perez",
  "Birthday": "1984-12-13T00:00:00"
}
```

Put: <https://localhost:44325/api/People/1>

```
{
  "Id": 1,
  "Name": "Maria",
  "LastName": "Lopez",
  "Birthday": "1989-12-13T00:00:00"
}
```

Delete: <https://localhost:44325/api/People/1>

03/09/2025

- ☐ Primer CRUD con Entity Framework (CodeFirst)
- ☐

FLUENT API

Relación Uno a Uno (1:1)

Models

```
namespace AppTrainingBETeacher.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; } = null!;

        // Propiedad de navegación
        public UserProfile Profile { get; set; } = null!;
    }

    public class UserProfile
    {
        public int Id { get; set; }

        public string FullName { get; set; } = null!;
        public DateTime DateOfBirth { get; set; }

        // Clave foránea y navegación inversa
        public int UserId { get; set; }
        public User? User { get; set; }
    }
}
```

AppDbContext

```
using AppTrainingBE.Models;
//using AppTrainingBETeacher.Models;
using Microsoft.EntityFrameworkCore;

namespace AppTrainingBE.Context
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
        }

        public DbSet<Person> Persons { get; set; }

        #region Uno a Uno
        public DbSet<User> Users => Set<User>();
        public DbSet<UserProfile> UserProfiles => Set<UserProfile>();
        #endregion

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            #region Uno a Uno
            //// Configurar relación 1:1
            //modelBuilder.Entity<User>()
            //    .HasOne(u => u.Profile)
            //    .WithOne(p => p.User)
            //    .HasForeignKey<UserProfile>(p => p.UserId);

            //// Restricciones adicionales opcionales
            //modelBuilder.Entity<User>()
            //    .Property(u => u.Username)
            //    .IsRequired()
            //    .HasMaxLength(100);
            #endregion
        }
    }
}
```

```

    }
}
}

```

Modificar Json Reference en Program.cs

```

builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.ReferenceHandler =
        System.Text.Json.Serialization.ReferenceHandler.Preserve;
    });

```

Controladores

```

[Route("api/[controller]")]
[ApiController]
public class UsersController : ControllerBase
{
    private readonly AppDbContext _context;

    public UsersController(AppDbContext context)
    {
        _context = context;
    }

    // GET api/users
    [HttpGet]
    public async Task<IActionResult> GetAllUsers()
    {
        var users = await _context.Users
            .Include(u => u.Profile) // Traer relación uno a uno
            .ToListAsync();

        return Ok(users);
    }

    // GET api/users/1
    [HttpGet("{id}")]
    public async Task<IActionResult> GetUser(int id)
    {
        var user = await _context.Users
            .Include(u => u.Profile)
            .FirstOrDefaultAsync(u => u.Id == id);

        if (user == null)
            return NotFound();

        return Ok(user);
    }

    // POST api/users
    [HttpPost]
    public async Task<IActionResult> CreateUser([FromBody] User user)
    {
        _context.Users.Add(user);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetUser), new { id = user.Id }, user);
    }

    // PUT api/users/1
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateUser(int id, [FromBody] User
updatedUser)
    {
        var existingUser = await _context.Users
            .Include(u => u.Profile)
            .FirstOrDefaultAsync(u => u.Id == id);
    }
}

```

```

        if (existingUser == null)
            return NotFound();

        // Actualizar campos del usuario y perfil
        existingUser.Username = updatedUser.Username;
        existingUser.Profile.FullName = updatedUser.Profile.FullName;
        existingUser.Profile.DateOfBirth = updatedUser.Profile.DateOfBirth;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE api/users/1
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteUser(int id)
    {
        var user = await _context.Users
            .Include(u => u.Profile)
            .FirstOrDefaultAsync(u => u.Id == id);

        if (user == null)
            return NotFound();

        _context.Users.Remove(user);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}

```

POSTMAN

```

/* PERMISOS PARA CREAR DIAGRAMAS ENTIDAD RELACION */
USE BD_Test
GO
ALTER DATABASE BD_Test set TRUSTWORTHY ON;
GO
EXEC dbo.sp_changedbowner @loginame = N'sa', @map = false
GO
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO

```

POSTMAN

1:1

GET: https://localhost:44325/api/Users

POST: https://localhost:44325/api/Users

```

{
  "username": "jdoe",
  "profile": {
    "fullName": "John Doe",
    "dateOfBirth": "1990-05-10"
  }
}

```

```
}
```

PUT: <https://localhost:44325/api/Users/1>

```
{  
  "username": "jdoe_updated",  
  "profile": {  
    "id": 1,  
    "fullName": "Johnathan Doe",  
    "dateOfBirth": "1990-05-10",  
    "userId": 1  
  }  
}
```

DELETE: <https://localhost:44325/api/Users/6>

05/09/2025

- ☐ Completar, Primer CRUD con Entity Framework (Code-First)
- ☐ Primer CRUD con Entity Framework (FluentApi)
- ☐ Capas (Arquitectura limpia)
- ☐ Creación de los proyectos en visual studio
- ☐ Ejecutar los scripts de base de datos
- ☐ Primer CRUD con DataBase-First

10/09/2025

- ☐ Ejecutar los scripts de base de datos
- ☐ Primer CRUD con DataBase-First
- ☐ Get, Post

-
1. Crear una carpeta en la capa de **Infrastructure**
 - a. Crear **Data**
 2. Instalación de Paquetes en diferentes capas
 - a. Capa **Infrastructure**
 - i. Microsoft.EntityFrameworkCore.Tools
 - ii. Microsoft.EntityFrameworkCore.SqlServer
 - iii. Pomelo.EntityFrameworkCore.MySql
 - b. Capa **Api**
 - i. Microsoft.EntityFrameworkCore.Design

DataBase-First

- Ejecutar desde la capa de Infrastructure

Scaffold-DbContext

"Server=localhost;Port=3306;Database=DbSocialMedia;Uid=root;Pwd=Ucb.2025;"

Pomelo.EntityFrameworkCore.MySql -OutputDir Data -Context "SocialMediaContext" -Force

Scaffold-DbContext

```
"Server=DESKTOP-6RJ0EO8;Database=DbSocialMedia;Trusted_Connection=true;TrustServerCertificate=true;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Data -Context "SocialMediaContext" -Force
```

3. Crear una carpeta en la capa de Core “**Entities**”
4. Cortar las clases de: **Comment, Post, User** a la carpeta Entities
5. Crear una Carpeta dentro la capa: Infrastructure > Data > **Configurations**
 - a. Dentro de configurations crear archivos
 - i. PostConfiguration
 - ii. CommentConfiguration
 - iii. UserConfiguration

6. PostConfiguration

```
namespace SocialMediaTeacher.Infrastructure.Data.Configurations
{
    public class PostConfiguration : IEntityTypeConfiguration<Post>
    {
        public void Configure(EntityTypeBuilder<Post> builder)
        {
            builder.HasKey(e => e.Id).HasName("PK_Publicacion");

            builder.ToTable("Post");

            builder.Property(e => e.Date).HasColumnType("datetime");
            builder.Property(e => e.Description)
                .HasMaxLength(1000)
                .IsUnicode(false);
            builder.Property(e => e.Imagen)
                .HasMaxLength(500)
                .IsUnicode(false);

            builder.HasOne(d => d.User).WithMany(p => p.Posts)
                .HasForeignKey(d => d.UserId)
                .OnDelete(DeleteBehavior.ClientSetNull)
                .HasConstraintName("FK_Post_User");
        }
    }
}
```

7. CommentConfiguration

```
namespace SocialMediaTeacher.Infrastructure.Data.Configurations
{
    public class CommentConfiguration : IEntityTypeConfiguration<Comment>
    {
        public void Configure(EntityTypeBuilder<Comment> builder)
        {
            builder.HasKey(e => e.Id).HasName("PK_Comentario");

            builder.ToTable("Comment");

            builder.Property(e => e.Id).ValueGeneratedNever();
        }
    }
}
```

```

        builder.Property(e => e.Date).HasColumnType("datetime");
        builder.Property(e => e.Description)
            .HasMaxLength(500)
            .IsUnicode(false);

        builder.HasOne(d => d.Post).WithMany(p => p.Comments)
            .HasForeignKey(d => d.PostId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_Comment_Post");

        builder.HasOne(d => d.User).WithMany(p => p.Comments)
            .HasForeignKey(d => d.UserId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("FK_Comment_User");
    }
}
}

```

8. UserConfiguration

```

namespace SocialMediaTeacher.Infrastructure.Data.Configurations
{
    public class UserConfiguration : IEntityTypeConfiguration<User>
    {
        public void Configure(EntityTypeBuilder<User> builder)
        {
            builder.HasKey(e => e.Id).HasName("PK_Usuario");

            builder.ToTable("User");

            builder.Property(e => e.Email)
                .HasMaxLength(30)
                .IsUnicode(false);
            builder.Property(e => e.FirstName)
                .HasMaxLength(50)
                .IsUnicode(false);
            builder.Property(e => e.LastName)
                .HasMaxLength(50)
                .IsUnicode(false);
            builder.Property(e => e.Telephone)
                .HasMaxLength(10)
                .IsUnicode(false);
        }
    }
}

```

9. SocialMediaContext

```

namespace SocialMedia.Infrastructure.Data
{
    public partial class SocialMediaContext : DbContext
    {
        public SocialMediaContext()
        {

```

```

    }

    public SocialMediaContext(DbContextOptions<SocialMediaContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Comment> Comments { get; set; }
    public virtual DbSet<Post> Posts { get; set; }
    public virtual DbSet<User> Users { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
    }
}

```

10. Modificar la cadena de coneccion en appsettings.json

```

"ConnectionStrings": {
  "ConnectionString": "Server=Richard;Database=DbSocialMedia;Trusted_Connection=true;TrustServerCertificate=true;",
  "ConnectionString": "Server=localhost;Port=3306;Database=DbSocialMedia;Uid=root;Pwd=Ucb.2025;"
},
"DatabaseProvider": "SqlServer" // Opciones: "SqlServer" o "MySQL"

```

11. Modificar [Program.cs](#)

```

#region Configurar la BD SqlServer
var connectionString = builder.Configuration.GetConnectionString("ConnectionString");
builder.Services.AddDbContext<SocialMediaContext> (options =>
options.UseSqlServer(connectionString));
#endregion

#region Configurar la BD MySQL
//var connectionString = builder.Configuration.GetConnectionString("ConnectionString");
//builder.Services.AddDbContext<SocialMediaContext>(options =>
// options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString)));
#endregion

```

12. En la capa **Infrastructure**

- a. Crear la carpeta **Repositories**
- b. Adicionar el Archivo PostRepository

```

public class PostRepository : IPostRepository
{
    private readonly SocialMediaContext _context;
    public PostRepository(SocialMediaContext context)
    {
        _context = context;
    }
}

```

```

public async Task<IEnumerable<Post>> GetAllPostsAsync()
{
    var posts = await _context.Posts.ToListAsync();
    return posts;
}

public async Task<Post> GetPostsByIdAsync(int id)
{
    var post = await _context.Posts.FirstOrDefaultAsync(x => x.Id == id);
    return post;
}

public async Task InsertPost(Post post)
{
    _context.Posts.Add(post);
    await _context.SaveChangesAsync();
}

public async Task UpdatePost(Post post)
{
    _context.Posts.Update(post);
    await _context.SaveChangesAsync();
}

public async Task DeletePost(Post post)
{
    _context.Posts.Remove(post);
    await _context.SaveChangesAsync();
}
}

```

13. En la capa **Core**

- a. Crear la carpeta **Interfaces**
- b. Adicionar el Archivo IPostRepository

```

public interface IPostRepository
{
    Task<IEnumerable<Post>> GetAllPostsAsync();
    Task<Post> GetPostsByIdAsync(int id);
    Task InsertPost(Post post);
    Task UpdatePost(Post post);
    Task DeletePost(Post post);
}

```

14. Adicionar “Inyectar las dependencias” en **Program.cs** **builder.Services.AddTransient<IPostRepository, PostRepository>();**

15. En la capa **API**

- a. Crear el controlador

```

public class PostController : ControllerBase
{

```



```

private readonly IPostRepository _postRepository;
public PostController(IPostRepository postRepository)
{
    _postRepository = postRepository;
}

#region Sin DTOs
[HttpGet]
public async Task<IActionResult> GetPosts()
{
    var posts = await _postRepository.GetAllPostsAsync();
    return Ok(posts);
}

[HttpGet("{id}")]
public async Task<IActionResult> GetPostById(int id)
{
    var post = await _postRepository.GetPostsByIdAsync(id);
    return Ok(post);
}

[HttpPost]
public async Task<IActionResult> InsertPost(Post post)
{
    await _postRepository.InsertPost(post);
    return Ok(post);
}
#endregion

#region Con DTOs
[HttpGet("dto")]
public async Task<IActionResult> GetPostsDto()
{
    var posts = await _postRepository.GetAllPostsAsync();
    var postsDto = posts.Select(p => new PostDto
    {
        Id = p.Id,
        UserId = p.UserId,
        Date = p.Date,
        Description = p.Description,
        Image = p.Imagen
    });
    return Ok(postsDto);
}

[HttpGet("dto/{id}")]
public async Task<IActionResult> GetPostByIdDto(int id)
{
    var post = await _postRepository.GetPostsByIdAsync(id);
    var postDto = new PostDto
    {
        Id = post.Id,

```

```

        UserId = post.UserId,
        Date = post.Date,
        Description = post.Description,
        Image = post.Imagen
    };
    return Ok(postDto);
}

```

```

[HttpPost("dto")]
public async Task<IActionResult> InsertPostDto(PostDto postDto)
{
    var post = new Post
    {
        Id = postDto.Id,
        UserId = postDto.UserId,
        Date = postDto.Date,
        Description = postDto.Description,
        Imagen = postDto.Image
    };
    await _postRepository.InsertPost(post);
    return Ok(post);
}

```

```

[HttpPut("dto/{id}")]
public async Task<IActionResult> UpdatePostDto(int id, [FromBody] PostDto postDto)
{
    if (id != postDto.Id)
        return BadRequest("El ID del post no coincide.");

    var post = await _postRepository.GetPostsByIdAsync(id);
    if (post == null)
        return NotFound("Post no encontrado.");

    // Mapear valores del DTO a la entidad
    post.UserId = postDto.UserId;
    post.Date = postDto.Date;
    post.Description = postDto.Description;
    post.Imagen = postDto.Image;

    await _postRepository.UpdatePost(post);

    return Ok(post);
}

```

```

[HttpDelete("dto/{id}")]
public async Task<IActionResult> DeletePostDto(int id)
{
    var post = await _postRepository.GetPostsByIdAsync(id);
    if (post == null)
        return NotFound("Post no encontrado.");

    await _postRepository.DeletePost(post);
}

```

```

        return NoContent(); // 204 sin contenido
    }
    #endregion
}

```

16. Obtener Post By Id

a. Repository

```

public async Task<Post> GetPostsByIdAsync(int id)
{
    var post = await _context.Posts.FirstOrDefaultAsync(x => x.Id == id);
    return post;
}

```

b. Api

```

[HttpGet("{id}")]
public async Task<ActionResult> GetPostById(int id)
{
    var post = await _postRepository.GetPostsByIdAsync(id);
    return Ok(post);
}

```

17. Insert POST

a. Repository

```

public async Task InsertPost(Post post)
{
    _context.Posts.Add(post);
    await _context.SaveChangesAsync();
}

```

b. Api

```

[HttpPost]
public async Task<ActionResult> InsertPost(Post post)
{
    await _postRepository.InsertPost(post);
    return Ok(post);
}

```

c. Llamar desde Postman

- i. Convertir Class a Json
- ii. <https://csharp2json.azurewebsites.net/>

```

{
    "userId": 2,
    "date": "2025-01-01T00:00:00",
    "description": "Nuevo valor",
    "image": null
}

```

iii. Cambiar la clase Post

```
public partial class Post
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public DateTime Date { get; set; }
    public string Description { get; set; } = null!;
    public string? Image { get; set; }
    public virtual ICollection<Comment> Comments { get; set; } = new List<Comment>();
    public virtual User? User { get; set; }
}
```

12/09/2025

- ☐ Get, Post (PostController) Sin DTOs
- ☐ Usar DTOs
- ☐ Get, GetById, Insert, Update, Delete Post (PostController) Con DTOs
- ☐ Automapper

1. Mostrar Vulnerabilidad de insertar datos con usuarios

```
{
  "userId": 2,
  "date": "2025-01-01T00:00:00",
  "description": "Nuevo valor 2",
  "imagen": null,
  "user":
  {
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "jperez@gmail.com",
    "dateOfBirth": "2000-01-01",
    "telephone": "151616556",
    "isActive": false
  }
}
```

2. Usar DTOs

- a. En la capa Core, crear la carpeta **DTOs**
- b. En la capa API instalar el paquete:
 - i. Microsoft.AspNetCore.Mvc.NewtonsoftJson
 - ii. Ignorar el error Circular

```
builder.Services.AddControllers().AddNewtonsoftJson(options =>
{
    options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore;
});
```

- c. Dentro la carpeta DTOs crear el archivo **PostDto**

```

public class PostDto
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public DateTime Date { get; set; }
    public string Description { get; set; }
    public string? Image { get; set; }
}

```

d. En el controlador Convertidor a DTOs, realizar con todos los metodos

Payload de todos los metodos DTOSs

POST:

<https://localhost:7050/api/Post/dto>

```

{
    "userId": 2,
    "date": "2025-01-01T00:00:00",
    "description": "Nuevo valor 2",
    "image": null
}

```

UPDATE:

<https://localhost:7050/api/Post/dto/1>

```

{
    "id": 1,
    "userId": 14,
    "date": "1970-03-13T06:19:47.22",
    "description": "Actualizando el post desde API",
    "image": "https://lemi.hinto.hu/ithhait/to/hinallare/buter.png"
}

```

DELETE:

<https://localhost:7050/api/Post/dto/59>

AUTOMAPPER

1. En la capa de **Infrastructure**
 - a. descargar el paquete: **AutoMapper (Version=12.0.1)**
 - b. Crear una carpeta **Mappings**
 - i. Crear una clase **MappingProfile**

```

public class MappingProfile: Profile
{
    public MappingProfile()
    {
        CreateMap<Post, PostDto>();
        CreateMap<PostDto, Post>();
    }
}

```

```
}
```

2. En la capa de **API**

- a. descargar el paquete: **AutoMapper** y **AutoMapper.Extensions.Microsoft.DependencyInjection (Version=12.0.1)**
- b. En el archivo: [Program.cs](#)
 - i. **builder.Services.AddAutoMapper(typeof(MappingProfile));**
- c. En el controlador de **Post** modificar el código

#region Con Mapper

```
[HttpGet("dto/mapper/")]
```

```
public async Task<IActionResult> GetPostsDtoMapper()
```

```
{  
    var posts = await _postRepository.GetAllPostsAsync();  
    var postsDto = _mapper.Map<IEnumerable<PostDto>>(posts);  
    return Ok(postsDto);  
}
```

```
[HttpGet("dto/mapper/{id}")]
```

```
public async Task<IActionResult> GetPostByIdDtoMapper(int id)
```

```
{  
    var post = await _postRepository.GetPostsByIdAsync(id);  
    var postDto = _mapper.Map<PostDto>(post);  
    return Ok(postDto);  
}
```

```
[HttpPost("dto/mapper/")]
```

```
public async Task<IActionResult> InsertPostDtoMapper(PostDto postDto)
```

```
{  
    var post = _mapper.Map<Post>(postDto);  
    await _postRepository.InsertPost(post);  
    return Ok(post);  
}
```

```
[HttpPut("dto/mapper/{id}")]
```

```
public async Task<IActionResult> UpdatePostDtoMapper(int id, [FromBody] PostDto postDto)
```

```
{  
    if (id != postDto.Id)  
        return BadRequest("El ID del post no coincide.");  
  
    var post = await _postRepository.GetPostsByIdAsync(id);  
    if (post == null)  
        return NotFound("Post no encontrado.");  
  
    _mapper.Map(postDto, post);  
  
    await _postRepository.UpdatePost(post);  
}
```

```

        return Ok(post);
    }

    [HttpDelete("dto/mapper/{id}")]
    public async Task<ActionResult> DeletePostDtoMapper(int id)
    {
        var post = await _postRepository.GetPostsByIdAsync(id);
        if (post == null)
            return NotFound("Post no encontrado.");

        await _postRepository.DeletePost(post);

        return NoContent(); // 204 sin contenido
    }
#endregion

```

3. Modificar el archivo **MappingProfile**

```

public MappingProfile()
{
    CreateMap<Post, PostDto>()
        .ForMember(dest => dest.Image, opt => opt.MapFrom(src => src.Imagen));
    CreateMap<PostDto, Post>()
        .ForMember(dest => dest.Imagen, opt => opt.MapFrom(src => src.Image)); ;
}

```

4.

24/09/2025

- ☐ Repaso General
- ☐ Completa el Automapper
- ☐ **Validaciones**

1. Instalar los paquetes En la capa de **Infrastructure**
 - a. **FluentValidation 12.0.0**
 - b. **FluentValidation.DependencyInjectionExtensions**
2. En la capa de **Infrastructure** Crear la carpeta **Validators** el archivo [PostDtoValidator.cs](#)

```

public class PostDtoValidator : AbstractValidator<PostDto>
{
    public PostDtoValidator()
    {
        RuleFor(x => x.UserId)
            .GreaterThan(0)
            .WithMessage("El IdUser debe ser mayor que 0"); RuleFor(x => x.Description)
            .NotEmpty().WithMessage("La descripción es requerida")
            .MaximumLength(500).WithMessage("La descripción no puede exceder 500 caracteres")
    }
}

```

```

        .MinimumLength(10).WithMessage("La descripción debe tener al menos 10 caracteres");

RuleFor(x => x.Imagen)
    .Must(UriImagenValida).When(x => !string.IsNullOrEmpty(x.Imagen))
    .WithMessage("La URL de la imagen no es válida")
    .MaximumLength(1000).WithMessage("La URL de la imagen es demasiado larga");

// Validar que la fecha pueda ser parseada desde el formato dd-MM-yyyy
RuleFor(x => x.Date)
    .NotEmpty().WithMessage("La fecha es requerida")
    .Must(BeValidDateFormat).WithMessage("La fecha debe tener el formato dd-MM-yyyy")
    .Must(BeValidDate).WithMessage("La fecha no es válida");
}

private bool UriImagenValida(string url)
{
    if (string.IsNullOrEmpty(url)) return true;

    return Uri.TryCreate(url, UriKind.Absolute, out var uriResult)
        && (uriResult.Scheme == Uri.UriSchemeHttp
            || uriResult.Scheme == Uri.UriSchemeHttps);
}

private bool BeValidDateFormat(string fecha)
{
    if (string.IsNullOrEmpty(fecha))
        return false;

    return DateTime.TryParseExact(fecha, "dd-MM-yyyy",
        CultureInfo.InvariantCulture, DateTimeStyles.None, out _);
}

private bool BeValidDate(string fecha)
{
    {
        if (DateTime.TryParseExact(fecha, "dd-MM-yyyy",
            CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))
        {
            return result != default(DateTime) &&
                result >= new DateTime(1900, 1, 1) &&
                result <= new DateTime(2100, 12, 31);
        }
        return false;
    }
}
}

```

1. En **Infrastructure** usar la carpeta **Validators** y crear el archivo **ValidationService**

```

public interface IValidationService
{
    Task<ValidationResult> ValidateAsync<T>(T model);
}

```

```

public class ValidationResult

```



```

{
    public bool IsValid { get; set; }
    public List<string> Errors { get; set; } = new();
}

public class ValidationService : IValidationService
{
    private readonly IServiceProvider _serviceProvider;

    public ValidationService(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public async Task<ValidationResult> ValidateAsync<T>(T model)
    {
        var validator = _serviceProvider.GetService<IValidator<T>>();

        if (validator == null)
        {
            throw new InvalidOperationException($"Validación No encontrada para el tipo {typeof(T).Name}");
        }

        var result = await validator.ValidateAsync(model);

        return new ValidationResult
        {
            IsValid = result.IsValid,
            Errors = result.Errors.Select(e => e.ErrorMessage).ToList()
        };
    }
}

```

2. En **Infrastructure** Crear la carpeta **Filters** y el archivo **ValidationFilter** (Mostrar diapositiva)

```

public class ValidationFilter : IAsyncActionFilter
{
    private readonly IValidationService _validationService;
    private readonly IServiceProvider _serviceProvider;

    public ValidationFilter(IValidationService validationService, IServiceProvider serviceProvider)
    {
        _validationService = validationService;
        _serviceProvider = serviceProvider;
    }

    public async Task OnActionExecutionAsync(ActionExecutingContext context,
        ActionExecutionDelegate next)
    {
        foreach (var argument in context.ActionArguments.Values)
        {

```

```

        if (argument == null) continue;

        var argumentType = argument.GetType();

        // Verificar si existe un validador para este tipo
        var validatorType = typeof(IValidator<>).MakeGenericType(argumentType);
        var validator = _serviceProvider.GetService(validatorType);

        if (validator == null) continue; // No hay validador, saltar

        try
        {
            // Llamar al servicio de validación con el tipo correcto
            var method = typeof(IValidationService).GetMethod("ValidateAsync");
            var genericMethod = method.MakeGenericMethod(argumentType);
            var validationTask = (Task<ValidationResult>)genericMethod.Invoke(_validationService,
new[] { argument });

            var validationResult = await validationTask;

            if (!validationResult.IsValid)
            {
                context.Result = new BadRequestObjectResult(new { Errors = validationResult.Errors });
                return;
            }
        }
        catch (Exception ex)
        {
            // Log the error but don't stop execution
            Console.WriteLine($"Error durante la validación: {ex.Message}");
        }
    }

    await next();
}
}

```

3. En el **controlador** inyectar el servicio mediante el constructor

```

private readonly IValidationService _validationService;
public PostController(IValidationService validationService)
{
    _validationService = validationService;
}

```

Probar únicamente en el insert del POST

```

#region Validaciones
// La validación automática se hace mediante el filtro
// Esta validación manual es opcional
var validationResult = await _validationService.ValidateAsync(postDto);

if (!validationResult.IsValid)

```

```

{
    return BadRequest(new { Errors = validationResult.Errors });
}
#endregion

```

4. Configurar dependencias en **Program**

a. Inyectar

```

//Validaciones
builder.Services.AddControllers(options =>
{
    options.Filters.Add<ValidationFilter>();
});

// FluentValidation
builder.Services.AddValidatorsFromAssemblyContaining<PostDtoValidator>();

// Services
builder.Services.AddScoped<IValidationService, ValidationService>();

```

5. Probar en ValidModel, muestra un error predeterminado para luego deshabilitarlo y hacer que llame a fluentvalidation

```

builder.Services.AddControllers().AddNewtonsoftJson(options =>
{
    options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore;
}).ConfigureApiBehaviorOptions(options =>
{
    options.SuppressModelStateInvalidFilter = true;
});

```

VALIDACIONES PARTE 2

1. Validar fechas
2. Cambiar el tipo de dato de DateTime a string en PostDTO
3. En el archivo **PostDtoValidator** adicionar el código:

```

// Validar que la fecha pueda ser parseada desde el formato dd-MM-yyyy
RuleFor(x => x.Date)
    .NotEmpty().WithMessage("La fecha es requerida")
    .Must(BeValidDateFormat).WithMessage("La fecha debe tener el formato dd-MM-yyyy")
    .Must(BeValidDate).WithMessage("La fecha no es válida");

```

```

private bool BeValidDateFormat(string fecha)
{
    if (string.IsNullOrEmpty(fecha))
        return false;

    return DateTime.TryParseExact(fecha, "dd-MM-yyyy",
        CultureInfo.InvariantCulture, DateTimeStyles.None, out _);
}

```

```

private bool BeValidDate(string fecha)
{
    if (DateTime.TryParseExact(fecha, "dd-MM-yyyy",
        CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))
    {
        return result != default(DateTime) &&
            result >= new DateTime(1900, 1, 1) &&
            result <= new DateTime(2100, 12, 31);
    }
    return false;
}

```

- Ahora que cambiamos de tipo de datos agregar en los DTOs el siguiente cambio para convertir de string a DateTime
 .ToString("dd-MM-yyyy")

VALIDACIONES PARTE 3

- Validar el ID en el get por by ID Post
- En **Core** Crear la carpeta **CustomEntities** y el archivo **GetByIdRequest**

```

public class GetByIdRequest
{
    public int Id { get; set; }
}

```

- En **Infrastructure** la carpeta **Validators** crear **GetByIdRequestValidator**

```

public class GetByIdRequestValidator : AbstractValidator<GetByIdRequest>
{
    public GetByIdRequestValidator()
    {
        RuleFor(x => x.Id)
            .NotNull().WithMessage("El ID es requerido")
            .GreaterThan(0).WithMessage("El ID debe ser mayor a 0")
            .LessThanOrEqualTo(1000000).WithMessage("El ID no puede ser mayor a 1,000,000")
            .Must(BeAValidIdFormat).WithMessage("El ID debe ser un número válido");
    }

    private bool BeAValidIdFormat(int id)
    {
        // Validaciones adicionales para el formato del ID
        return id.ToString().Length <= 7; // Máximo 7 dígitos
    }
}

```

- Agregar el código al controlador

```

[HttpGet("dto/mapper/{id}")]
public async Task<ActionResult> GetPostByIdDtoMapper(int id)
{
    #region Validaciones
    var validationRequest = new GetByIdRequest { Id = id };

```

```

var validationResult = await _validationService.ValidateAsync(validationRequest);

if (!validationResult.IsValid)
{
    return BadRequest(new
    {
        Message = "Error de validación del ID",
        Errors = validationResult.Errors
    });
}
#endregion

var post = await _postRepository.GetPostsByIdAsync(id);
var postDto = _mapper.Map<PostDto>(post);
return Ok(postDto);
}

```

5. Configurar el [Program.cs](#) con FluentValidation

// FluentValidation

```

builder.Services.AddValidatorsFromAssemblyContaining<PostDtoValidator>();
builder.Services.AddValidatorsFromAssemblyContaining<GetByIdRequestValidator>();

```