```

using System.Text.Json;
using Microsoft.EntityFrameworkCore;
using S05_P04_MVC2.DbContexts;
using S05_P04_MVC2.Repositories;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

/*
I want to extract the repository-like functionality from the controller in a specific dedicated class.
The repository must be injected:
1) in the controller constructor and saved in a private field, so it can be used later in the methods
2) directly in the method, with the [FromServices] attribute.

When I want to use a type as a dependency of other types,
I must configure its lifecycle. I have 3 options:
AddSingleton(): a single instance is created for the whole life of the application
AddScoped(): a new fresh instance is created for every HTTP request, and inside the same request the same instance is used
AddTransient(): a new fresh instance is created for every class that needs that dependency.

I can configure these methods with the concrete class I want to use, for example:

    builder.Services.AddSingleton<InMemorySuperheroesRepository>();

The controller will be:

    public class SuperheroesController : ControllerBase
    {
        private readonly ISuperheroRepository _repo;

        public SuperheroesController(InMemorySuperheroRepository repo) {
            _repo = repo;
        }
    }

This way I have separated the responsibility, but the controller is tightly coupled with the repository in-memory.
I want to decouple the two classes, creating an abstraction of the repository, the ISuperheroRepository.
In this case I use the Add...() methods with 2 generics:

    Add...<TInterface, TImplementation>()

For the in-memory repo, I use UseSingleton, because the list of superheroes
```

```
must survive the different HTTP requests.
*/
builder.Services.AddSingleton<ISuperheroRepository, InMemorySuperheroesRepository>();

// If I use the SQL repo, I can configure it as Scoped:
builder.Services.AddScoped<ISuperheroRepository, SqlSuperheroesRepository>();

/*
The system throws an Exception if a class has a longer life-cycle than one of its dependencies.
Therefore, if the DbContext is Scoped, I cannot configure the repo to be Singleton.
*/

/*
And the controller becomes:

    public class SuperheroesController : ControllerBase
    {
        private readonly ISuperheroRepository _repo;

        public SuperheroesController(InMemorySuperheroRepository repo) {
            _repo = repo;
        }
    }

Therefore, if I want to use another repository implementation, I just change the Services configuration, the controller does not change.
*/

builder.Services.AddDbContext<AppDbContext>(options =>{
    options.UseSqlServer(GetConnectionString());
});

var app = builder.Build();

app.MapControllers();

app.Run();

static string GetConnectionString() {
    // Remember to add TrustServerCertificate=True to the connection string
    // if you use the IRES databases.
    var directory = Directory.GetCurrentDirectory();
    var fileName = "credentials.json";
    var filePath = Path.Combine(directory, fileName);
    var jsonCredentials = File.ReadAllText(filePath);
    var credentials = JsonSerializer.Deserialize<Credentials>(jsonCredentials);
    return credentials.ConnectionString;
}

class Credentials
{
    public string ConnectionString { get; set; }
}

```

```csharp

using System.Text.Json;
using Microsoft.EntityFrameworkCore;
using S05_P05_MVC3.DbContexts;
using S05_P05_MVC3.Repositories;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddScoped<ISuperheroesRepository, SqlSuperheroesRepository>();
builder.Services.AddScoped<IVillainsRepository, SqlVillainsRepository>();

// This is how I configure the DbContexts:
builder.Services.AddDbContext<AppDbContext>(options =>{
    options.UseSqlServer(GetConnectionString());
    // This configuration of the options is used to inject the options on the instances of the DbContexts (see AppDbContext)
});

// This is how I can override the behavior of the ApiControllers of the app:
// builder.Services.Configure<ApiBehaviorOptions>(config => {
//     config.InvalidModelStateResponseFactory = context => {
//         var firstError = context.ModelState.Values...
//          ...
//     };
// });

var app = builder.Build();
app.MapControllers();

app.Run();

static string GetConnectionString() {
    var directory = Directory.GetCurrentDirectory();
    var fileName = "credentials.json";
    var filePath = Path.Combine(directory, fileName);
    var jsonCredentials = File.ReadAllText(filePath);
    var credentials = JsonSerializer.Deserialize<Credentials>(jsonCredentials);
    return credentials.ConnectionString;
}

class Credentials
{
    public string ConnectionString { get; set; }
}

```

```
using Microsoft.EntityFrameworkCore;
using S05_P05_MVC3.Models;

namespace S05_P05_MVC3.DbContexts;

public class AppDbContext : DbContext
{
    // Typically this is the constructor to use,
    // so that the configuration are injected from the system:
    public AppDbContext(DbContextOptions options) : base(options)
    { }

    public DbSet<Superhero> Superheroes { get; set; }
    public DbSet<Villain> Villains { get; set; }
}

```

```csharp
using System.ComponentModel.DataAnnotations;

namespace S05_P05_MVC3.DataTransferObjects;

public class SuperheroEditDto
{
    // altri attributi da implementare x casa
    // [Required(AllowEmptyStrings = false)]
    // [MaxLength(100)]
    // public string Nickname { get; set; }
    [Required(AllowEmptyStrings = false), MaxLength(100)]
    public string Nickname { get; set; }
    public string SecretName { get; set; }
    [Range(1, 100)]
    public int Strength { get; set; }
    public bool CanFly { get; set; }
    public decimal? Assets { get; set; }
    public int Gender { get; set; }
    public string City { get; set; }

    public DateTime? Birth { get; set; }
}




```

```
using Microsoft.AspNetCore.Mvc;
using S05_P05_MVC3.DataTransferObjects;
using S05_P05_MVC3.Models;
using S05_P05_MVC3.Repositories;

namespace S05_P05_MVC3.Controllers;

/*
An Controller marked as ApiController has two behaviors:

1) the objects used as input parameters in the methods
are automatically filled from the body of the request.
Otherwise, an attribute [FromBody is required].

2) if the ModelState is invalid,
an automatic response 400 with the details of the validation errors
is sent back to the client, without even entering the controller's
  method.
You can override this behavior in the Program.cs through:
    builder.Services.Configure<ApiBehaviorOptions>(...)
(see Program.cs for an example)

*/

[ApiController]
public class SuperheroesController : ControllerBase
{
    private readonly ISuperheroesRepository _repo;
    private readonly ILogger<SuperheroesController> _logger;

    public SuperheroesController(
            ISuperheroesRepository repo,
            ILogger<SuperheroesController> logger) {
        _repo = repo;
        _logger = logger;
    }

    [HttpPost("/superheroes/add")]
    public void Add(Superhero model) {
        _repo.Add(model);
        _logger.LogInformation("Superhero successfully added.");
    }

    [HttpGet("/superheroes/all")]
    public List<SuperheroRowDto> GetAll() {
        var superheroes = _repo.GetAll();
        _logger.LogInformation("Superheroes successfully retrieved.");
        return superheroes;
    }

    [HttpPut("/superheroes/edit/{id}")]
    public void Edit(int id, SuperheroEditDto dto) {
        _repo.Edit(id, dto);
        _logger.LogInformation("Superhero successfully edited.");
    }

    [HttpDelete("/superheroes/delete/{id}")]
    public void Delete(int id) {
        if (_repo.Delete(id)) {
            _logger.LogInformation("Superhero successfully deleted.");
        } else {
            _logger.LogInformation("Superhero not found.");
        }
    }
}

```

```csharp
using Microsoft.AspNetCore.Mvc;
using S05_P05_MVC3.DataTransferObjects;
using S05_P05_MVC3.Repositories;

namespace S05_P05_MVC3.Controllers;

public class SuperheroesNonApiController : ControllerBase
{
    private readonly ISuperheroesRepository _repo;
    private readonly ILogger<SuperheroesController> _logger;

    public SuperheroesNonApiController(
            ISuperheroesRepository repo,
            ILogger<SuperheroesController> logger) {
        _repo = repo;
        _logger = logger;
    }

    // If there is no [ApiController] attribute on the controller,
    // the [FromBody] attribute must be used on the input object,
    // otherwise the dto is instatiated but not filled:
    [HttpPut("/superheroes-non-api/edit/{id}")]
    public object Edit(int id, [FromBody] SuperheroEditDto dto) {
        // If the model is invalid, the method is invoked,
        // but I can use the ModelState property
        // to check the validity of the inputs, for example:
        if (!ModelState.IsValid) {
            Response.StatusCode = 400;
            return ModelState;
        }
        _repo.Edit(id, dto);
        _logger.LogInformation("Superhero successfully edited.");
        return null;
    }
}

```