Practice 7 – Implement DTO's

Paso 1. Crea los DTO que nos servirán para trasportar la información entre capas.

- Abre Visual Studio y selecciona el proyecto AnimalSpawn.Domain.
- Crea una clase con el nombre AnimalResponseDto.cs dentro de la carpeta DTOs del proyecto, esta clase contendrá los atributos necesarios para devolver la información al cliente.

```
public class AnimalResponseDto
{
        public int Id { get; set; }
        public int SpeciesId { get; set; }
        public int FamilyId { get; set; }
        public int GenusId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public bool Gender { get; set; }
        public DateTime CaptureDate { get; set; }
        public string CaptureCondition { get; set; }
        public float Weight { get; set; }
        public float Height { get; set; }
        public int EstimatedAge { get; set; }
}
```

- Crea una clase con el nombre AnimalRequestDto.cs dentro de la carpeta DTOs del proyecto, esta clase contendrá los atributos necesarios para el registro.

```
public class AnimalRequestDto
{
        public int SpeciesId { get; set; }
        public int FamilyId { get; set; }
        public int GenusId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public bool Gender { get; set; }
        public DateTime CaptureDate { get; set; }
        public string CaptureCondition { get; set; }
        public float Weight { get; set; }
        public float Height { get; set; }
        public int EstimatedAge { get; set; }
}
```

Paso 2. Implementa el retorno y recepción de los DTOs en los controladores

- En la clase `AnimalController.cs` cambia el tipo de retorno del método `Get`, ahora este método debe devolver un listado de objetos `AnimalResponseDto.cs`; para ello también es necesario implementar un mecanismo de transformación de un objeto de tipo `Animal.cs` a `AnimalResponseDto.cs`.

```
public async Task<IActionResult> Get()
{
            var animals = await _repository.GetAnimals();
            var animalsDto = animals.Select(animal => new AnimalResponseDto
            {
                CaptureCondition = animal.CaptureCondition,
                CaptureDate = animal.CaptureDate ?? DateAndTime.Now,
                Description = animal.Description,
                EstimatedAge = animal.EstimatedAge ?? 0,
                FamilyId = animal.FamilyId,
                GenusId = animal.GenusId,
                Id = animal.Id,
                Gender = animal.Gender ?? false,
                Height = animal.Height ?? 0,
```

```
                Name = animal.Name,
                SpeciesId = animal.SpeciesId,
                Weight = animal.Weight ?? 0
            });

            return Ok(animalsDto);
}
```

- En la clase `AnimalController`, modifica el método `Get(int id)`, ahora este método debe devolver un objeto de tipo `AnimalResponseDto.cs`; para ello también es necesario implementar un mecanismo de transformación de un objeto de tipo `Animal.cs` a `AnimalResponseDto.cs`.

```
public async Task<IActionResult> Get(int id)
{
            var animal = await _repository.GetAnimal(id);
            var animalDto = new AnimalResponseDto
            {
                CaptureCondition = animal.CaptureCondition,
                CaptureDate = animal.CaptureDate ?? DateAndTime.Now,
                Description = animal.Description,
                EstimatedAge = animal.EstimatedAge ?? 0,
                FamilyId = animal.FamilyId,
                GenusId = animal.GenusId,
                Id = animal.Id,
                Gender = animal.Gender ?? false,
                Height = animal.Height ?? 0,
                Name = animal.Name,
                SpeciesId = animal.SpeciesId,
                Weight = animal.Weight ?? 0
            };

            return Ok(animalDto);
}
```

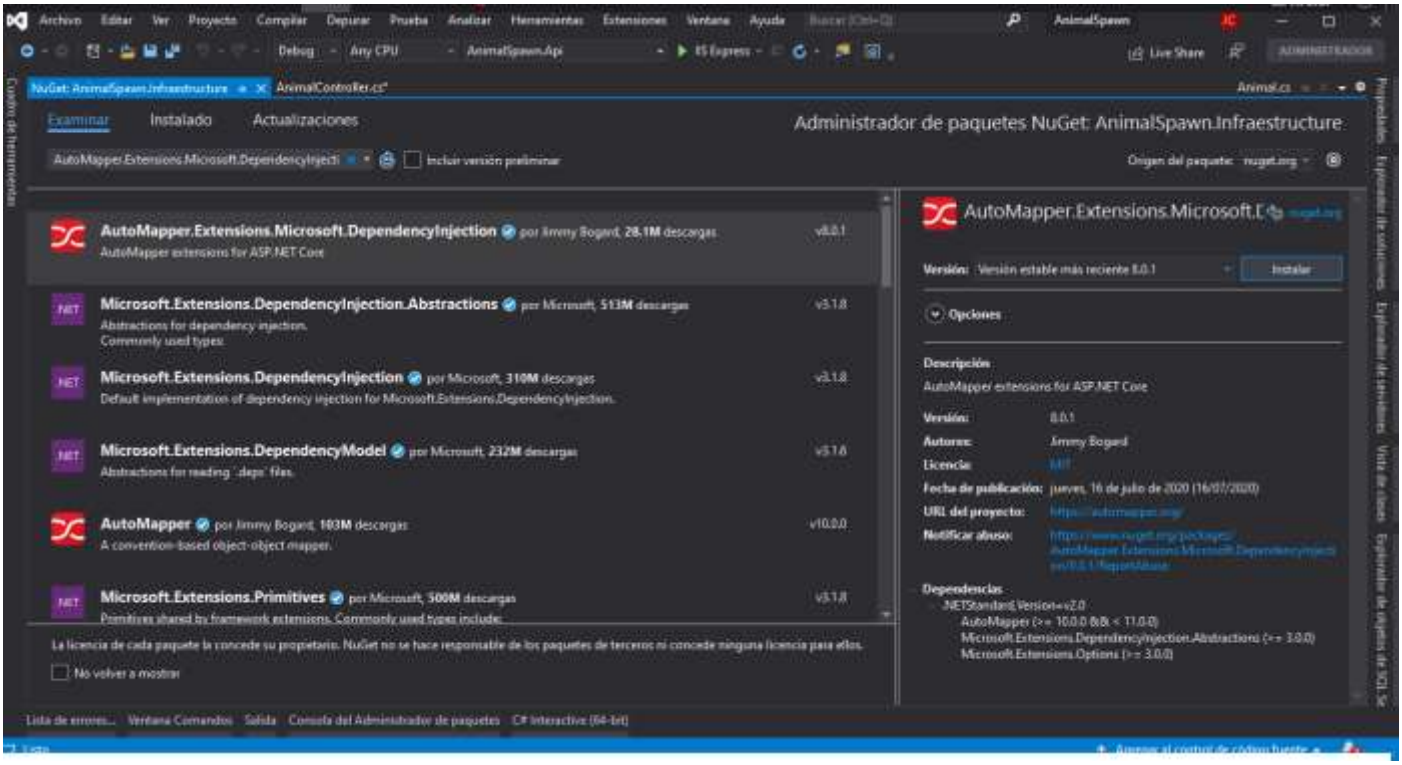
- En la clase `AnimalController`, modifica el método `Post`, ahora este método debe recibir un objeto de tipo `AnimalRequestDto.cs`; para el registro es necesario implementar un mecanismo de transformación de un objeto de tipo `AnimalRequestDto.cs` a `Animal.cs`.

```
[HttpPost]
public async Task<IActionResult> Post(AnimalRequestDto animalDto)
{
            var animal = new Animal {
                CaptureCondition = animalDto.CaptureCondition,
                CaptureDate = animalDto.CaptureDate,
                Description = animalDto.Description,
                EstimatedAge = animalDto.EstimatedAge,
                FamilyId = animalDto.FamilyId,
                GenusId = animalDto.GenusId,
                Gender = animalDto.Gender,
                Height = animalDto.Height,
                Name = animalDto.Name,
                SpeciesId = animalDto.SpeciesId,
                Weight = animalDto.Weight,
                CreateAt = DateAndTime.Now,
                CreatedBy = 3,
                Status = true,
            };
            await _repository.AddAnimal(animal);
            return Ok(animal);
}
```

- Con fines de comprobación prueba tu aplicación, pero recuerda que este "*Approach*" solo es temporal, y con la finalidad de comprender mejor la utilidad de los DTOs.

Paso 3. Simplifica el código usando AutoMapper

- Selecciona el proyecto AnimalSpawn.Api e instala el paquete nuget: AutoMapper.Extensions.Microsoft.DependencyInjection.



- En la clase `AnimalController.cs`, cambia el contenido del método `Get`, para implementar el uso de AutoMapper

```
[HttpGet]
public async Task<IActionResult> Get()
{
            var animals = await _repository.GetAnimals();
            var config = new MapperConfiguration(mc => mc.CreateMap<Animal, AnimalResponseDto>());

            var _mapper = new Mapper(config);
            var animalsDto = _mapper.Map< IEnumerable <Animal>, IEnumerable
                        <AnimalResponseDto>>(animals);
            return Ok(animalsDto);
}
```

- En la clase `AnimalController.cs`, cambia el contenido del método `Get(int id)`, para implementar el uso de AutoMapper

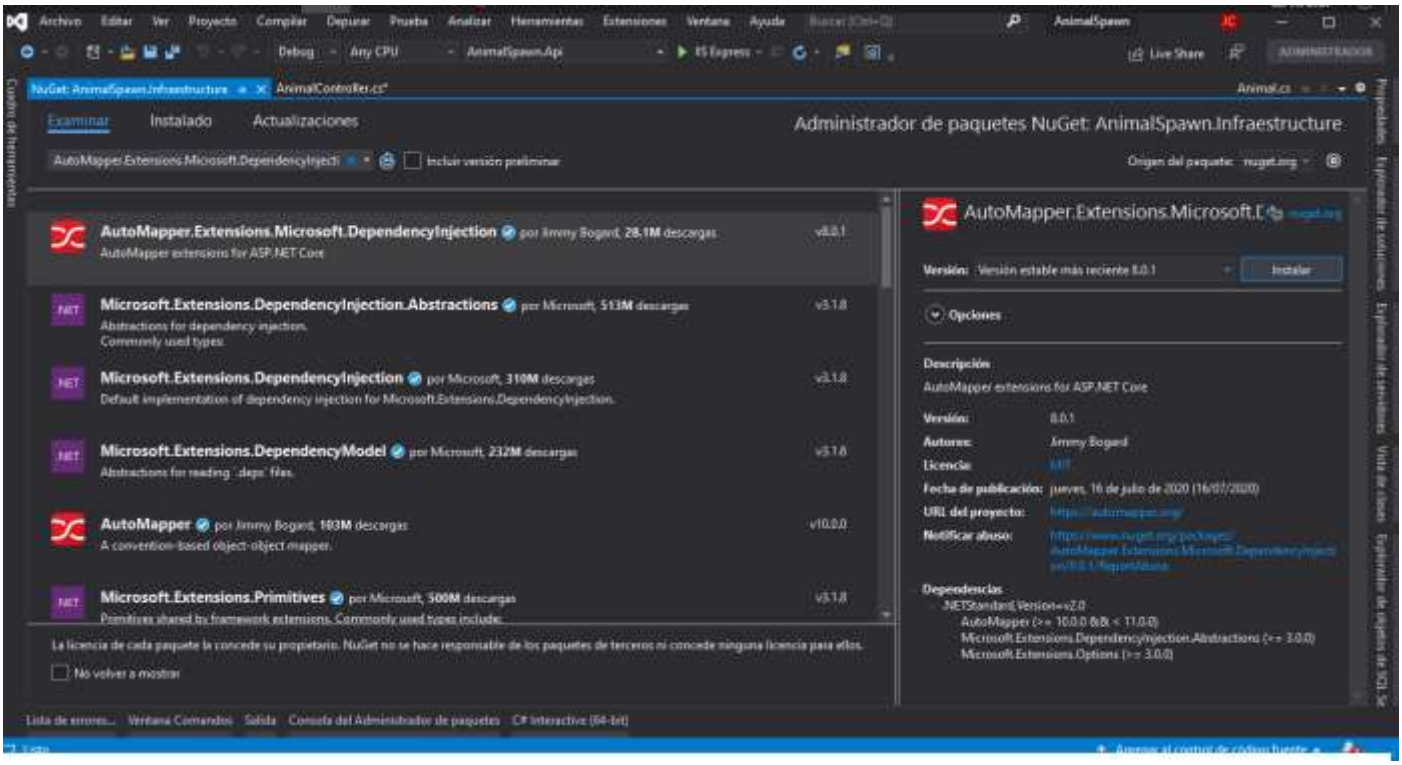
```
[HttpGet("{id:int}")]
public async Task<IActionResult> Get(int id)
{
            var animal = await _repository.GetAnimal(id);

            var config = new MapperConfiguration(mc => mc.CreateMap<Animal, AnimalResponseDto>());
            var _mapper = new Mapper(config);

            var animalDto = _mapper.Map<Animal, AnimalResponseDto>(animal);

            return Ok(animalDto);
}
```

- En la clase `AnimalController.cs`, cambia el contenido del método `Post`, para implementar el uso de AutoMapper

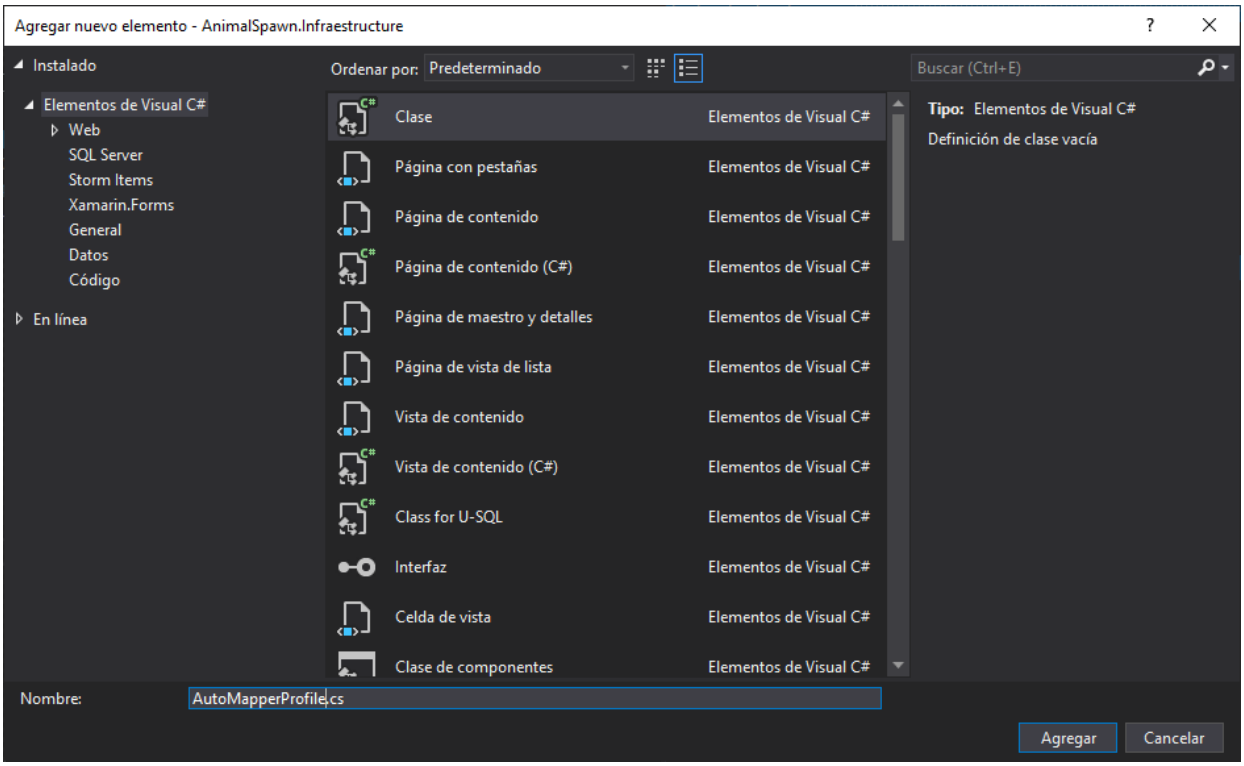
```
[HttpPost]
public async Task<IActionResult> Post(AnimalRequestDto animalDto)
{
            var config = new MapperConfiguration(mc => mc.CreateMap<AnimalRequestDto, Animal>()
            .AfterMap((source, destination) => {
                destination.CreateAt = DateTime.Now;
                destination.CreatedBy = 3;
                destination.Status = true;
            }));

            var _mapper = new Mapper(config);

            var animal = _mapper.Map<AnimalRequestDto,Animal>(animalDto);
            await _repository.AddAnimal(animal);

            config = new MapperConfiguration(mc => mc.CreateMap<Animal, AnimalResponseDto>());
            _mapper = new Mapper(config);
            var animalresponseDto = _mapper.Map<Animal, AnimalResponseDto>(animal);

            return Ok(animalresponseDto);
}
```

Paso 4. Mejora el nivel de acoplamiento usando AutoMapper

- Selecciona el proyecto AnimalSpawn.Infraestructure e instala el paquete nuget: AutoMapper.Extensions.Microsoft.DependencyInjection.



- En el proyecto AnimalSpawn.Infraestructure, crea una carpeta con el nombre Mappings esta carpeta la utiliaremos para guardar todas nuestras configuraciones de mapeo, dentro de la carpeta crea una clase con el nombre `AutoMapperProfile.cs`.



- La clase debe heredar de la clase `Profile`, para poder acceder a esta clase es necesario que agreguemos el `using` a la biblioteca de `AutoMapper`.

```
public class AutoMapperProfile : Profile{
       // …
}
```

- En el constructor de la clase debemos registrar nuestras conversiones de objeto a DTO y viceversa

```
public class AutoMapperProfile : Profile
{
        public AutoMapperProfile()
        {
            CreateMap<Animal, AnimalRequestDto>();
            CreateMap<Animal, AnimalResponseDto>();
            CreateMap<AnimalRequestDto, Animal>().AfterMap(
                ((source, destination) => {
                    destination.CreateAt = DateTime.Now;
                    destination.CreatedBy = 3;
                    destination.Status = true;
                }));
            CreateMap<AnimalResponseDto, Animal>();}
}
```

NOTA: Recuerda que usamos el método AfterMap, para agregar funcionalidad adicional una vez que AutoMapper termina la conversión del objeto; podemos consultar más configuraciones disponibles en: https://docs.automapper.org/en/stable/

Paso 5. Registra el uso de AutoMapper dentro del Midleware

- En la clase `Startup.cs` localiza el método `ConfigureServices` y agrega la configuración de los mapeos con AutoMapper usando el método `AddAutoMapper`, a este método le debes indicar la ruta o la clase en la cual se encuentran las configuraciones para la conversión de los objetos; para nuestro ejemplo usaremos la clase AppDomain para obtener la ruta donde se encuentran nuestros archivos de configuración

```
public void ConfigureServices(IServiceCollection services)
{
            services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

            services.AddControllers();

            services.AddDbContext<AnimalSpawnContext>(options =>
                options.UseSqlServer(Configuration.GetConnectionString("AnimalSpawnConnection"))
            );
            services.AddTransient<IAnimalRepository, AnimalRepository>();
}
```

Paso 6. Configura el uso de AutoMapper por medio de inyección de dependencias

- En el constructor de la clase `AnimalController.cs`, agrega un parámetro de tipo `IMapper` el cual será proporcionado por medio de inyección de dependencias (configuración en el paso previo).

```
public class AnimalController : ControllerBase
{
        private readonly IAnimalRepository _repository;
        private readonly IMapper _mapper;

        public AnimalController(IAnimalRepository repository, IMapper mapper)
        {
            _repository = repository;
            this._mapper = mapper;
        }

       // …
}
```

- En la clase `AnimalController`, modifica el comportamiento del método `Get` de la siguiente manera:

```
[HttpGet]
public async Task<IActionResult> Get()
{
            var animals = await _repository.GetAnimals();
            var animalsDto = _mapper.Map<IEnumerable<Animal>, IEnumerable<AnimalResponseDto>>(animals);
            return Ok(animalsDto);
}
```

- En la clase `AnimalController`, modifica el comportamiento del método `Get(int id)`de la siguiente manera:

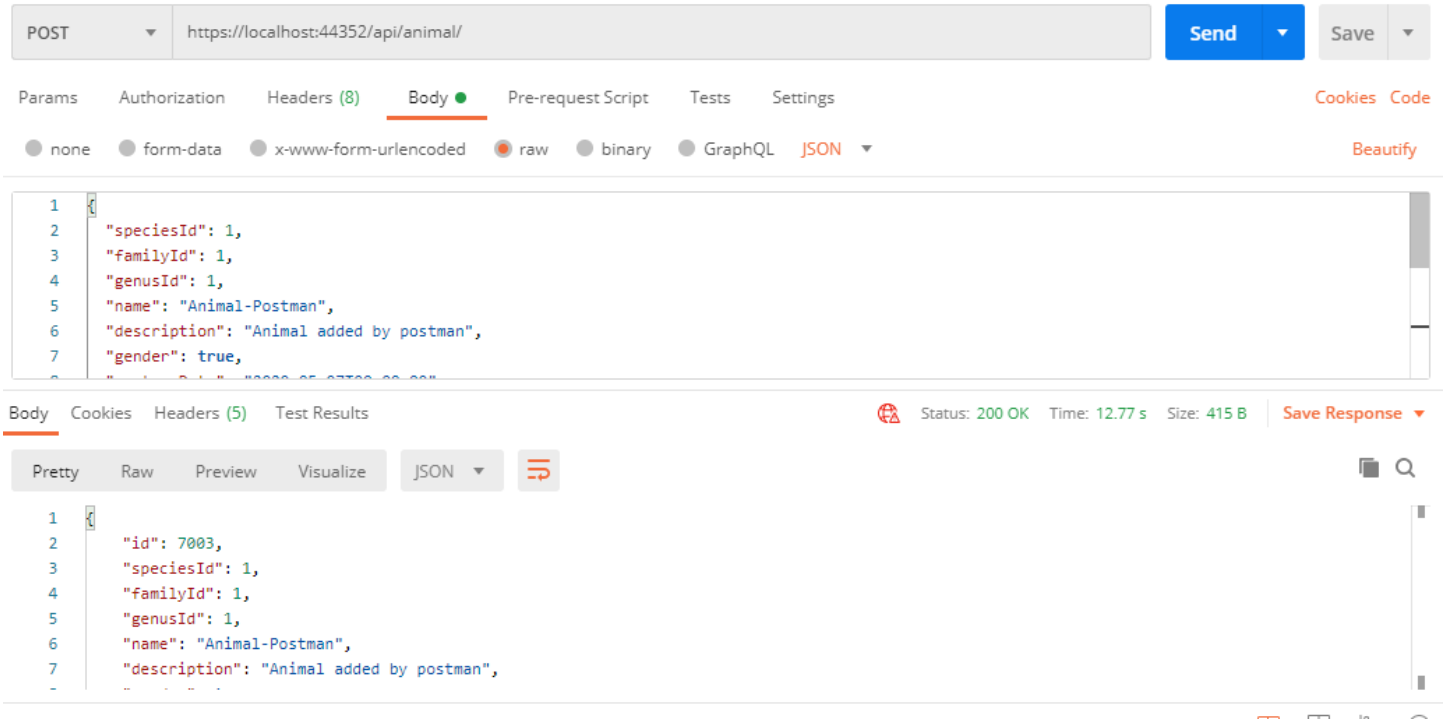
```
[HttpGet("{id:int}")]
public async Task<IActionResult> Get(int id)
{
            var animal = await _repository.GetAnimal(id);
            var animalDto = _mapper.Map<Animal, AnimalResponseDto>(animal);
            return Ok(animalDto);
}
```

- En la clase `AnimalController`, modifica el comportamiento del método post de la siguiente manera:

```
[HttpPost]
public async Task<IActionResult> Post(AnimalRequestDto animalDto)
{
            var animal = _mapper.Map<AnimalRequestDto,Animal>(animalDto);
            await _repository.AddAnimal(animal);
            var animalresponseDto = _mapper.Map<Animal, AnimalResponseDto>(animal);
            return Ok(animalresponseDto);
}
```

Paso 7. Prueba tu aplicación

- En el navegador de tu preferencia navegar por la ruta https://localhost:XXXXX/api/animal/, prueba consultar un animal por identificador y por último ingresa un nuevo animal a través de postman



- Para finalizar intenta repetir el proceso con otra de las entidades del sistema, éxito y felices compilaciones.