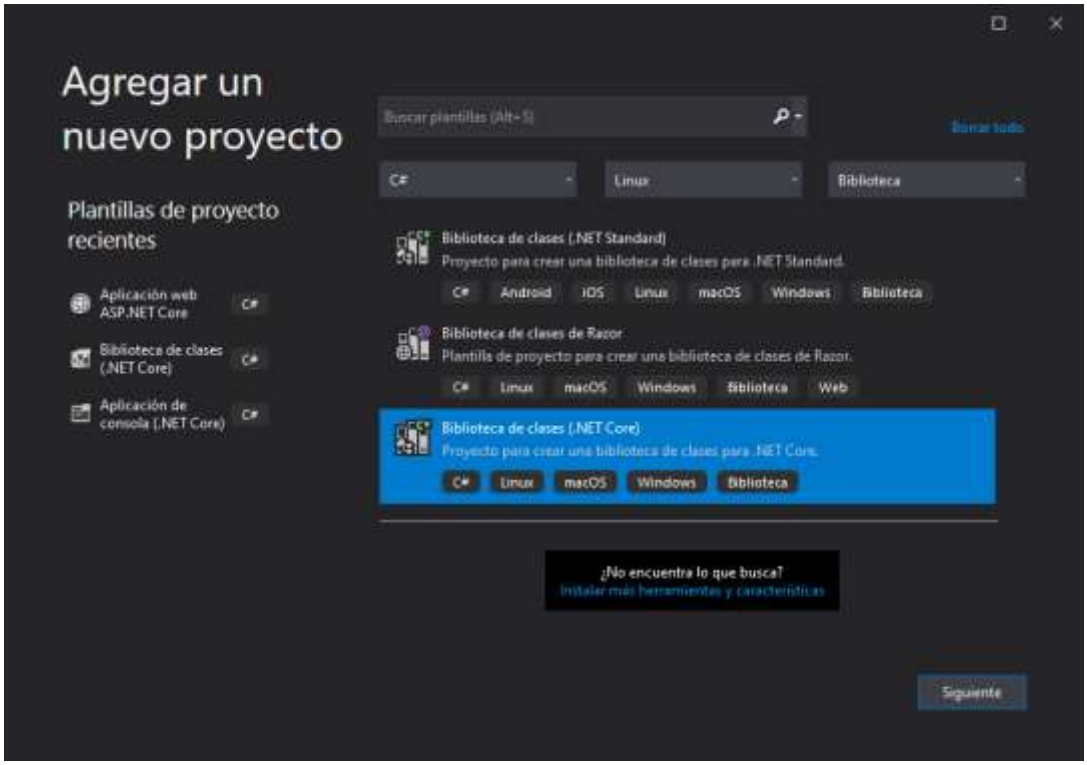
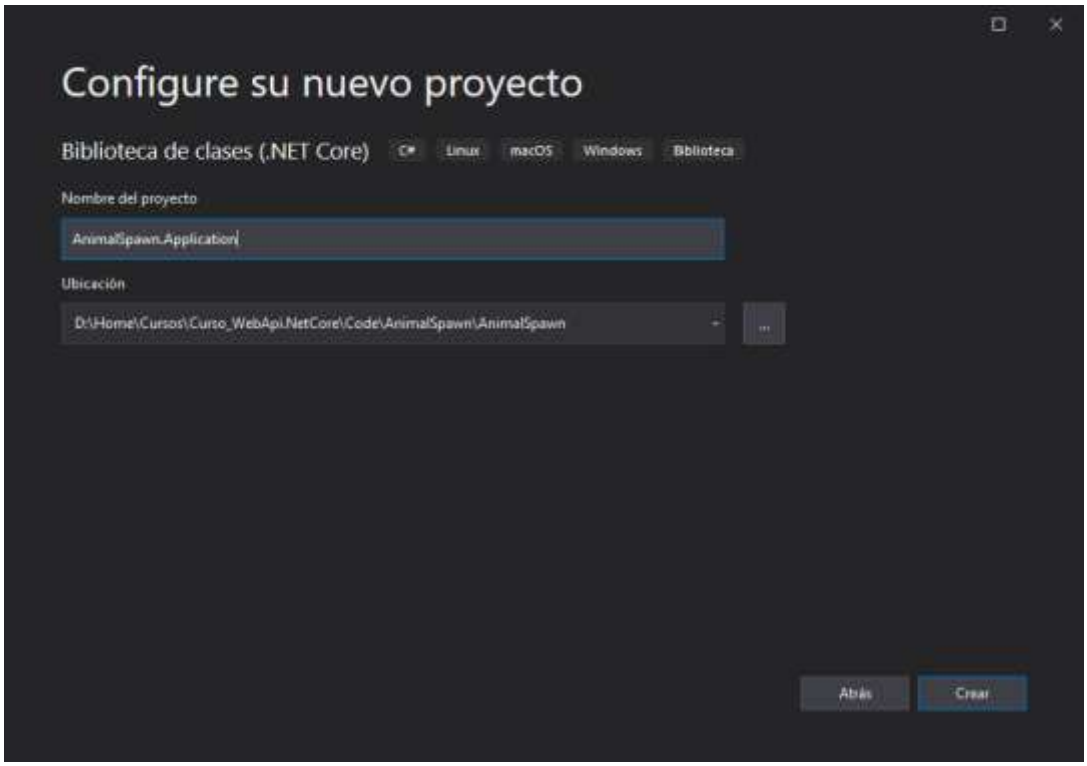
Practice 11 – To err is human, to forgive is not our policy.

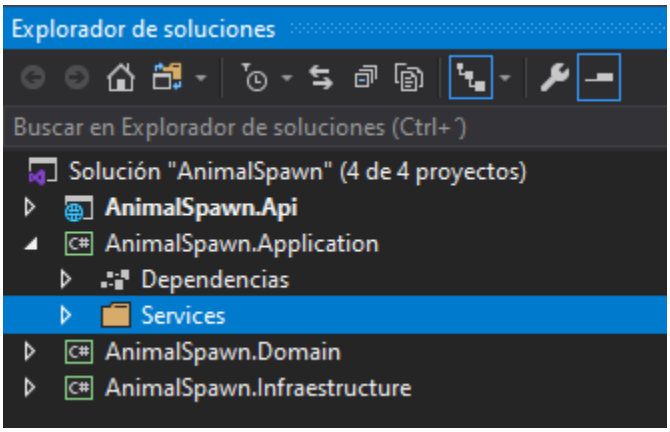
Paso 1. Agrega un nuevo proyecto para nuestra capa de regla de negocios.

- Abre Visual Studio y crea un nuevo proyecto de tipo biblioteca de clases con el nombre AnimalSpawn.Application; este proyecto corresponde a la capa de Application Services de la arquitectura de cebolla (Onion Architecture).

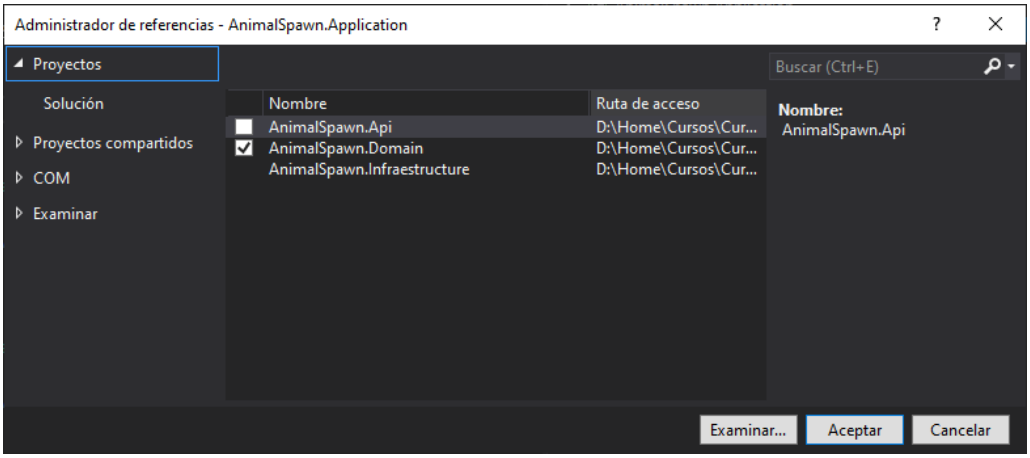




- Dentro del proyecto AnimalSpawn.Application crea una carpeta con el nombre `Services`, en esta carpeta vamos a agregar nuestras clases que nos permitan gestionar las reglas de negocio de la aplicación.

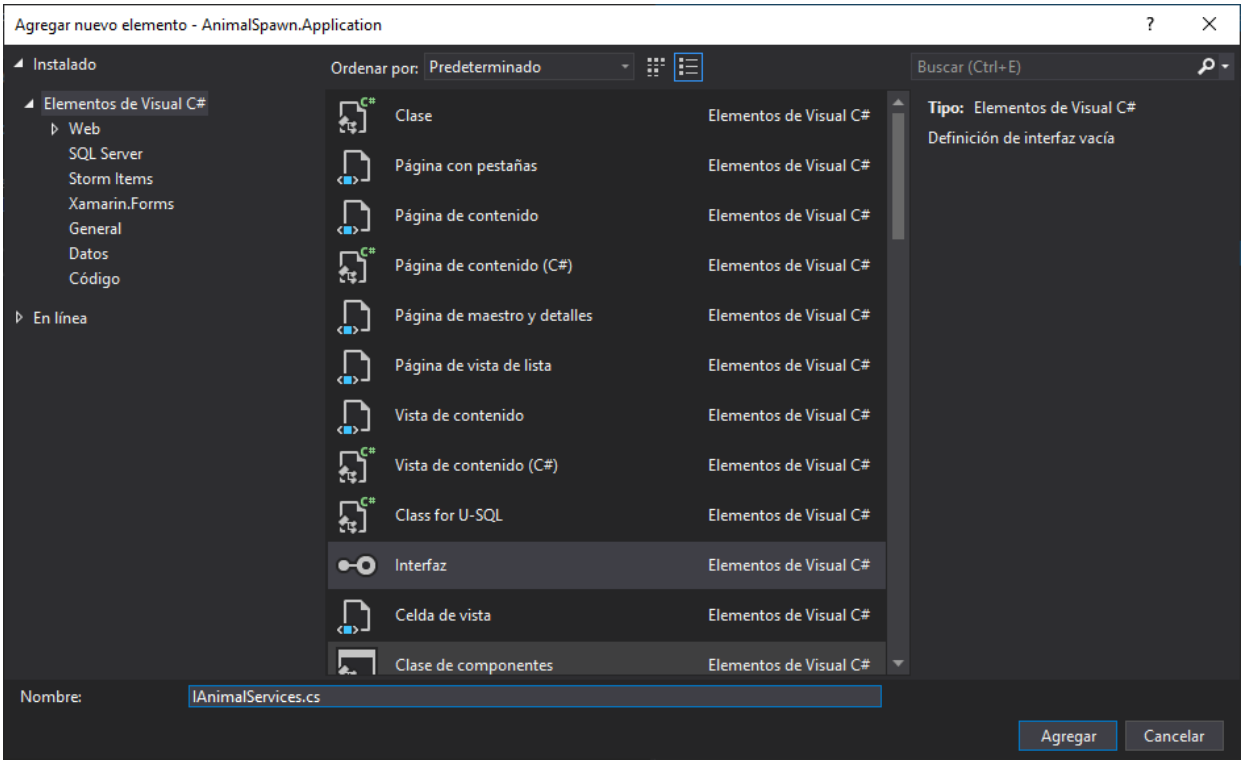


- Agrega la referencia al proyecto correspondiente a la capa de dominio (AnimalSpawn.Domain)



Paso 2. Crea el contrato para nuestro objeto que contendrá nuestras reglas de negocio

- Selecciona el proyecto AnimalSpawn.Domain y dentro de la carpeta `Interfaces` crea una nueva interfaz con el nombre `IAnimalService`.

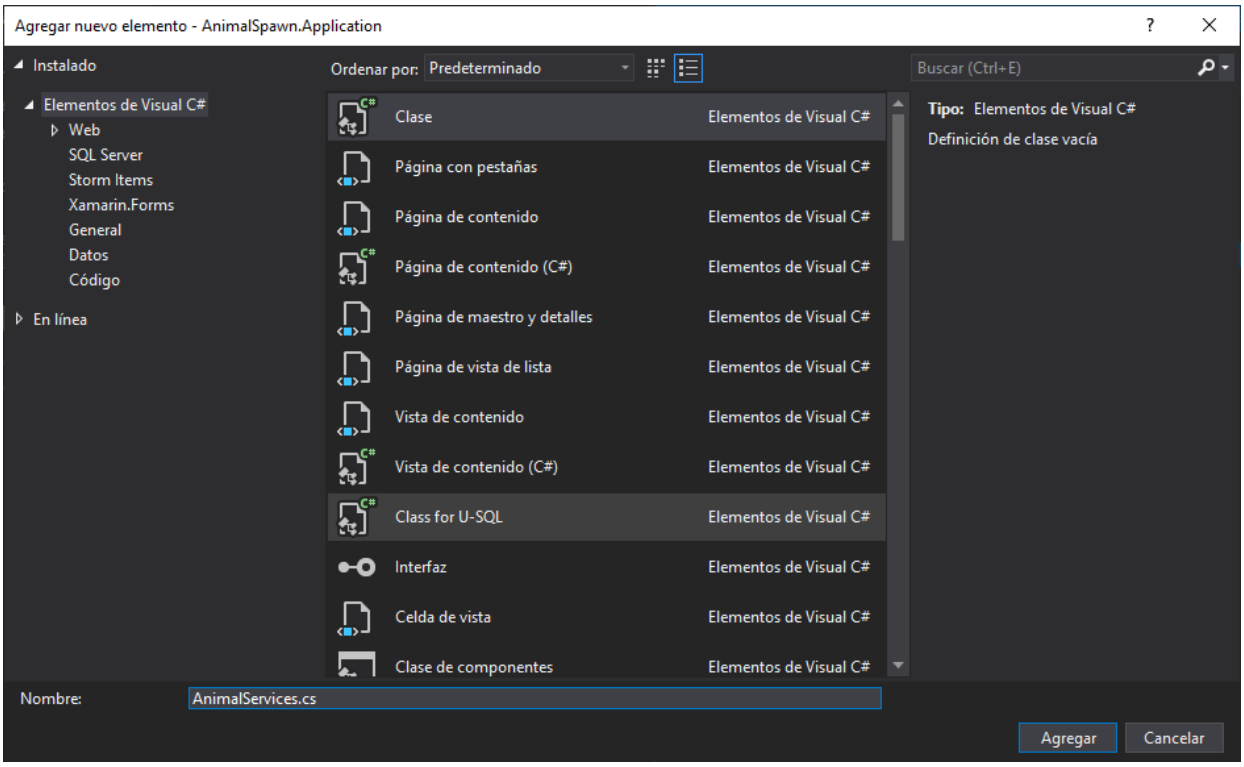


- Agrega a la interfaz `IAnimalService` los métodos correspondientes a las operaciones CRUD, estas operaciones puedes copiarlas de la interfaz `IAnimalRepository`.

```
public interface IAnimalService
{
        Task AddAnimal(Animal animal);
        Task<bool> DeleteAnimal(int id);
        Task<IEnumerable<Animal>> GetAnimals();
        Task<Animal> GetAnimal(int id);
        Task<bool> UpdateAnimal(Animal animal);
}
```

Paso 3. Implementa el contrato de nuestro servicio en la capa de aplicacion

- Selecciona el proyecto AnimalSpawn.Application y dentro de la carpeta `Services` crea una nueva clase con el nombre `AnimalService`.



- La clase debe implementar la interfaz `IAnimalService`, recuerda que es importante trabajar con abstracciones, de esta manera aseguramos que nuestras piezas de software mantengan un bajo acoplamiento.

```
public class AnimalService : IAnimalService
{
        public Task AddAnimal(Animal animal)
        {
            throw new System.NotImplementedException();
        }

        public Task<bool> DeleteAnimal(int id)
        {
            throw new System.NotImplementedException();
        }

        public Task<Animal> GetAnimal(int id)
        {
            throw new System.NotImplementedException();
        }
```

```
        public Task<IEnumerable<Animal>> GetAnimals()
        {
            throw new System.NotImplementedException();
        }

        public Task<bool> UpdateAnimal(Animal animal)
        {
            throw new System.NotImplementedException();
        }
    }
```

Paso 4. Invoca las operaciones CRUD definidas en el repositorio

- En la clase `AnimalService` agrega un constructor este debe recibir un objeto de tipo `IAnimalRepository`, recuerda crear un atributo de tipo lectura para el parámetro que se reciba (Patrón Dependency Injection).

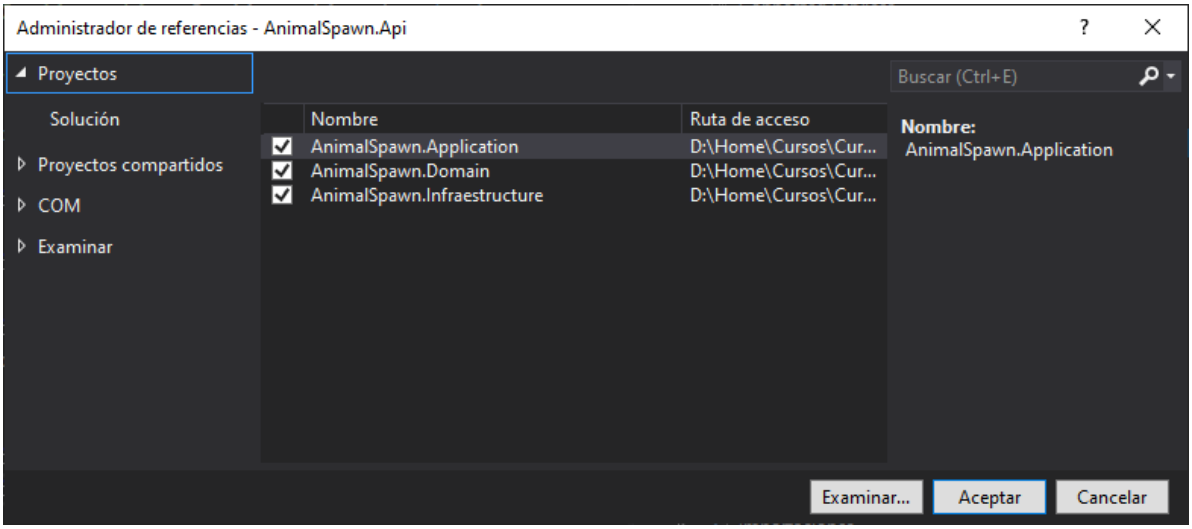
```
public class AnimalService : IAnimalService
{
        private readonly IAnimalRepository _repository;

        public AnimalService(IAnimalRepository repository)
        {
            this._repository = repository;
        }

       // …

}
```

- En la clase `AnimalService` agrega las llamadas a los métodos del repositorio correspondientes.

```
public class AnimalService : IAnimalService
{
        private readonly IAnimalRepository _repository;

        public AnimalService(IAnimalRepository repository)
        {
            this._repository = repository;
        }
        public async Task AddAnimal(Animal animal)
        {
            await _repository.AddAnimal(animal);
        }

        public async Task<bool> DeleteAnimal(int id)
        {
            return await _repository.DeleteAnimal(id);
        }

        public async Task<Animal> GetAnimal(int id)
        {
            return await _repository.GetAnimal(id);
        }

        public async Task<IEnumerable<Animal>> GetAnimals()
        {
            return await _repository.GetAnimals();
        }

        public async Task<bool> UpdateAnimal(Animal animal)
        {
            return await _repository.UpdateAnimal(animal);
        }
}
```

Paso 5. Agrega la llamada del servicio en la capa de presentación

- En el proyecto `AnimalSpawn.Api` agrega la referencia al proyecto `AnimalSpawn.Application`.



- En la clase `AnimalController` cambia el tipo de la variable `_repository` por el tipo `IAnimalService` y renombre la variable como `_service`. Recuerda que también tendrás que cambiar el tipo para el parámetro que se recibe en el constructor del controlador.

```
public class AnimalController : ControllerBase
{
        private readonly IAnimalService _service;
        private readonly IMapper _mapper;

        public AnimalController(IAnimalService service, IMapper mapper)
        {
            _service = service;
            this._mapper = mapper;
        }

       // …

}
```

NOTA: Recuerda que puedes renombrar un elemento de manera global usando la combinación de teclas CTRL+R+R

- Antes de poder usar el servicio debes registrar la clase `AnimalServices` correspondiente al servicio en el middleware (`startup.cs`), para poder utilizar el patrón dependency injection.

```
public void ConfigureServices(IServiceCollection services)
{
       // …
       services.AddTransient<IAnimalRepository, AnimalRepository>();
       services.AddTransient<IAnimalService, AnimalServices>();
       // …
}
```

Paso 6. Implementa las reglas de negocio para el registro

- Recordemos nuestras reglas de negocio para el registro de un animal:
    - El nombre del animal que se este registrando debe ser único en todo el sistema.
    - Cuando en el registro se especifica un valor para la edad mayor a cero, el valor de la altura y el peso también deben ser mayores a cero.
    - En el registro del animal, la fecha de captura solamente puede tener una antigüedad menor a 45 días.
    - Los tag de RFId solamente pueden estar asignados a un animal al mismo tiempo*.

- En el método `AddAnimal` comprueba que el nombre del animal es único en todo el sistema.

```
public async Task AddAnimal(Animal animal)
{
	var animals = await _repository.GetAnimals();

	if (animals.Any(item => item.Name == animal.Name))
		throw new Exception("This animal name already exist.");

	// …
}
```

- En el método `AddAnimal` comprueba que al proporcionar un valor de edad estimada mayor a cero, el paso y la altura también cuenta con un valor mayor a cero

```
public async Task AddAnimal(Animal animal)
{
	var animals = await _repository.GetAnimals();

	if(animal?.EstimatedAge > 0 && (animal?.Weight <= 0 || animal?.Height <= 0))
		throw new Exception("The height and weight should be greater than zero.");

	// …
}
```

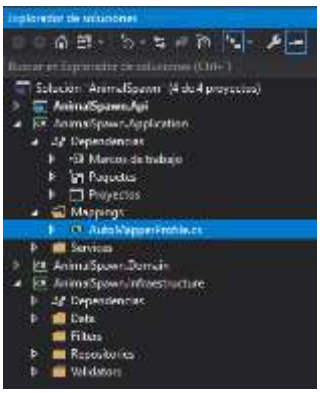
- En el método `AddAnimal` comprueba que al momento de intentar registrar el animal, la fecha de captura cuente con un valor con antigüedad menor a 45 días.

```
public async Task AddAnimal(Animal animal)
{
	var animals = await _repository.GetAnimals();

	var older = DateTime.Now - (animal?.CaptureDate ?? DateTime.Now);

	if (older.TotalDays > 45)
		throw new Exception("The animal's capture date is older than 45 days");

	// …
}
```

- Ahora que has visto como se interpretan las reglas de negocio en código, intenta realizar la última regla por ti mismo, éxito.

Paso 7. Asigna la responsabilidad del mapeo a la capa de aplicación

- Con la creación del proyecto `AnimalSpawn.Application`, también debemos asignar la responsabilidad de transformación de objetos a DTO y viceversa a la capa de aplicación



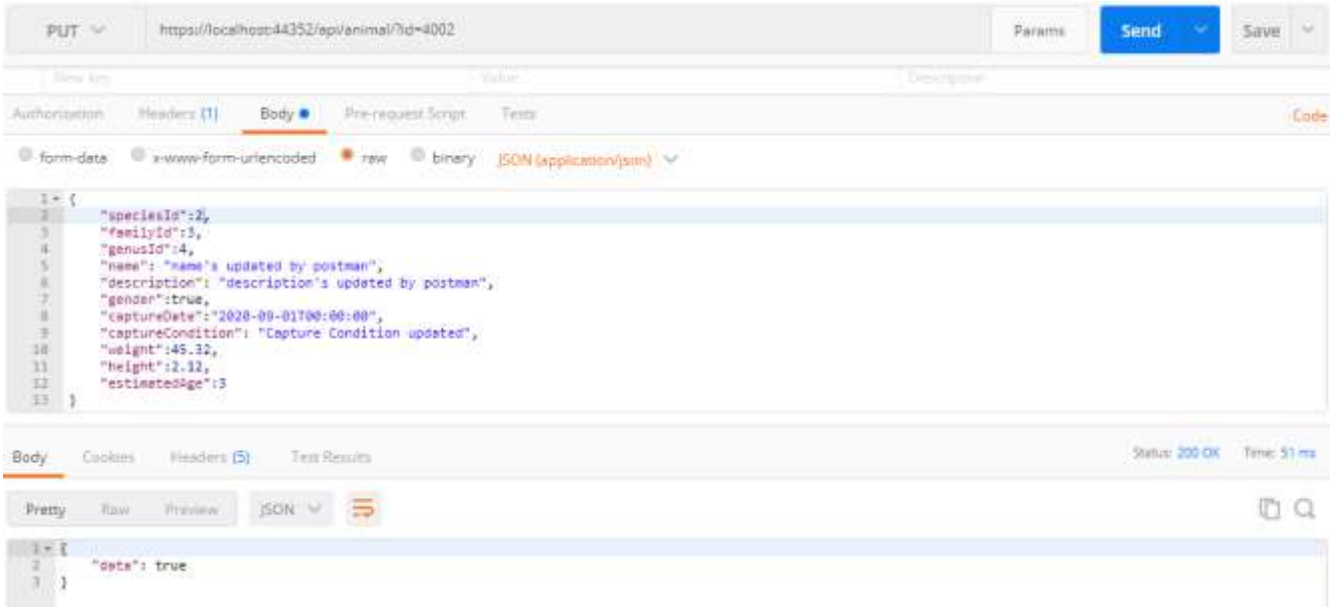
- Recuerda cambiar el espacio de nombre de la clase `AutoMapperProfile` al nuevo espacio correspondiente.

```
namespace AnimalSpawn.Application.Mappings
```

NOTA: Recuerda que para poder usar las clases correspondiente a AutoMapper es necesario instalar el paquete `AutoMapper.Extensions.Microsoft.DependencyInjection`.

Paso 8. Prueba tu aplicación

- Para probar que todo funciona de manera correcta abre postman y de la lista de verbos HTTP selecciona el verbo GET y accede a la ruta: https://localhost:XXXXX/api/animal



- Prueba el resto de métodos, como lo hemos visto en clases anteriores y una vez que has probado que tu aplicación funciona de manera correcta, intenta repetir el proceso con otra entidad, suerte y felices compilaciones.