Skriven av Micael Wollter
2020-05-22

Contents

Skriven av Micael Wollter
2020-05-22

HATEAOS-Guide

Guiden är till som hjälp att vidare implementera HATEOAS i MammalAPI projektet samt tankar om vidareutveckling.

Kodbasen bygger på följande tutorial:

https://baldbeardedbuilder.com/posts/adding-hateoas-to-an-asp-net-core-api/, men är anpassad och vidaremodifierad av mig (Micael Wollter) för att få den att passa för vårt API, samt att jag även lagt till lite extra kod om tankar för vidareutveckling.

Beskrivning av koden

Nedan följer en kort beskrivning av den kod som ni kommer att använda er av, jag kommer inte gå igenom vad allt gör, utan endast det ni kommer att behöva röra för att få länkarna att fungera, samt lite basic kunskap om kodbasen

HATEOAS Katalogen

Här finns 2/3 filer som baskoden består av.

```
namespace MammalAPI.HATEOAS
```

Link.cs

```
namespace MammalAPI.HATEOAS
{
    6 references
    public class Link
    {
        2 references
        public string Href { get; private set; }
        1 reference
        public string Rel { get; private set; }
        1 reference
        public string Type { get; private set; }

        1 reference
        public Link(string href, string rel, string type)
        {
            Href = href;
            Rel = rel;
            Type = type;
        }

        /// <summary>
        /// Possibly further implmentation the team can do with custom links
        /// without Rel & Type.
        /// Sometimes you just want a href link, without all the other fields.
        /// See Star Wars api for example.
        /// Wollter
        /// </summary>
        /// <param name="href"></param>
        0 references
        public Link(string href)
        {
            Href = href;
            throw new System.NotImplementedException();
        }
    }
}
```

Skriven av Micael Wollter
2020-05-22

I Link.cs finns den länkklass som används för att skapa varje länk genom tillhörande konstruktor. Har även lagt till en Overloadkonstruktor för vidareutveckling, där tanken är att man kanske vill slippa ha de andra parametrarna i länken, och endast en href länk.

```
public Link(string href)
{
    Href = href;
    throw new System.NotImplementedException();
}
```

Inspiration är hämtad från Starwars api:

HateoasLinkBase.cs

```
namespace MammalAPI.Controllers
```

En abstrakt klass som använder List<Link> Links för att lägga till länkarna i DTO:er.

```
namespace MammalAPI.HATEOAS
{
    1 reference
    public abstract class HateoasLinkBase
    {
        2 references
        public List<Link> Links { get; set; } = new List<Link>();
    }
}
```

För att använda länkarna måste era DTOer ärva denna klass. *Glöm heller inte heller att lägga till namespace:* ***MammalAPI.HATEOAS*** i klassen som ärver.

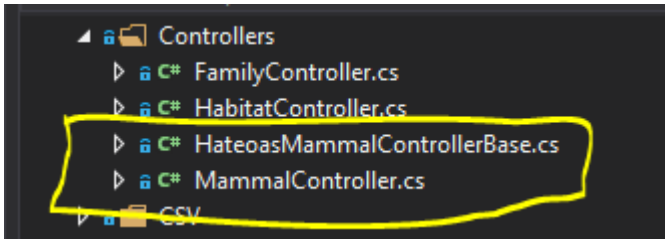
```
using MammalAPI.HATEOAS;

namespace MammalAPI.DTO
{
    18 references
    public class MammalDTO : HateoasLinkBase
    {
        2 references
        public int MammalID { get; set; }
        1 reference
        public string Name { get; set; }
```

Skriven av Micael Wollter
2020-05-22

ControllerBaseklassen

I controller ligger den 3e filen som slutar med ControllerBase.cs. Det är i denna klass, som själva länkarna länkas in i de olika get-metoderna osv från de olika controllersarna. För att hålla separation, och lätt administrera. Skapa en ny xxxControllerBase/controller.



För att göra detta på ett lätt sätt**, kopiera HateoasMammalControllerBase.cs,** döp om klassen till den controller nivill ha den till*, Exempelvis HateoasFamilyControllerBase.cs.*

Kod inuti HateoasControllerBaseklassen

Det finns en del kod innuti klassen, som sköter sig själv. Den behövs inte röras.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Abstractions;
using Microsoft.AspNetCore.Mvc.ActionConstraints;
using Microsoft.AspNetCore.Mvc.Infrastructure;

using MammalAPI.HATEOAS;
using MammalAPI.DTO;

namespace MammalAPI.Controllers
{
    2 references
    public class HateoasMammalControllerBase : ControllerBase
    {
        private readonly IReadOnlyList<ActionDescriptor> _routes;

        1 reference
        public HateoasMammalControllerBase(IActionDescriptorCollectionProvider actionDescriptorCollectionProvider)
        {
            _routes = actionDescriptorCollectionProvider.ActionDescriptors.Items;
        }

        2 references
        internal Link UrlLink(string relation, string routeName, object values)
        {
            var route = _routes.FirstOrDefault(f => f.AttributeRouteInfo.Name == routeName);
            var method = route.ActionConstraints.OfType<HttpMethodActionConstraint>().First().HttpMethods.First();
            var url = Url.Link(routeName, values).ToLower();
            return new Link(url, relation, method);
        }
```

Rör ej

VIKTIGT I den riktiga kontrollerklassen ersätts arvet Controllerbase mot HateoasMammalControllerBase, eller den controllerbase ni använder. HatoasMammalControllerBase ärver i sin tur ControllerBase som man kan se på bilden ovan.

```
namespace MammalAPI.Controllers
{
    [ApiController]
    [Route("api/v1.0/[controller]")]
    1 reference
    public class MammalController : HateoasMammalControllerBase
    {
```

Följande kod kan modifieras

Koden som kan modifieras är följande två metoder. Dessa finns även beskrivna i kommentarer vad de gör. **HateoasSideLinks, är en vidareutveckling** av mig, tänkt på att man kanske inte vill visa alla länkar för varje i exemplets fall: Mammal, utan man kanske vill gruppera och visa andra länkar på olika getmetoder eller andra metoder. Detta är för vidareutveckling för er om ni kommer på något.

```
/// <summary>
/// Use this method for your mainlinks. Eg CRUD, just add
/// more links to your CRUD methods inside
/// Wollter
/// </summary>
/// <param name="mammal"></param>
/// <returns></returns>
2 references
internal MammalDTO HateoasMainLinks(MammalDTO mammal)
{
    MammalDTO mammalDto = mammal;

    mammalDto.Links.Add(UrlLink("all", "GetAll", null));
    mammalDto.Links.Add(UrlLink("_self", "GetMammalAsync", new { id = mammalDto.MammalID }));

    return mammalDto;
}

/// <summary>
/// Custom extention method
/// Use this method for possible further implementation sidelinks the team come up with.
/// Can also duplicate this method to new methods with tailored links to be grouped.
/// Wollter
/// </summary>
/// <param name="mammal"></param>
/// <returns></returns>
0 references
internal MammalDTO HateoasSideLinks(MammalDTO mammal)
{
    MammalDTO mammalDto = mammal;

    throw new System.NotImplementedException();

    //return mammalDto;
}
```

Eftersom det blir en ny fil för varje controller, som ni ser i ovanstående metoder, måste man skicka in och returnera den aktuella DTOn. Så gör man en HateoasFamilyController, så måste tex MammalDTO ändras till FamilyDTO.

Namnet på getmetoderna måste även ändras till respektive controller getmetodnamn. Samma med Update, Delete osv. Dessa är dock inte implementerade i detta exempel, då det enast var ett proof-of concept.

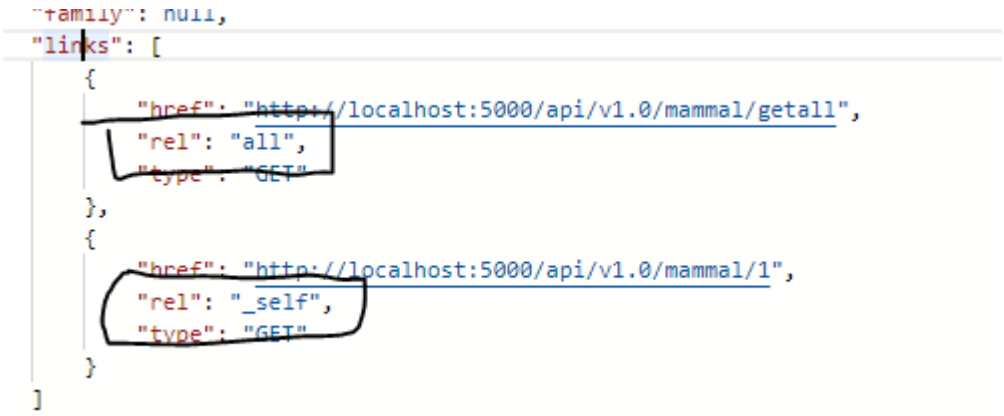
```
mammalDto.Links.Add(UrlLink("all", "GetAll", null));
mammalDto.Links.Add(UrlLink("_self", "GetMammalAsync", new { id = mammalDto.MammalID }));
```

”GetAll”, ”GetMamalAsync” är namn i på GetAll() och GetMammalById i MammalControllern. HateoasControllerBase, letar efter dessa namn för att lägga till länkarna resp. metod och DTO:

```
[HttpGet("GetAll", Name ="GetAll")]
0 references
public async Task<IActionResult> Get()
{
```

```
[HttpGet("{id:int}", Name = "GetMammalAsync")]
0 references
public async Task<IActionResult> GetMammalById(int id)
{
```

”all” och ”_self” i URL:en kan även ändras, då det är strängtext som visas:



Vad händer med koden?

Följande metod, letar igenom hela MammalControllerklassen efter routes. I routes (tex getmetoder) letar den efter det namn, som är angett i getmetoden i controllern och om det matchar det namn som står **i HateoasMainLinks -metoden**

```
2 references
internal Link UrlLink(string relation, string routeName, object values)
{
    var route = _routes.FirstOrDefault(f => f.AttributeRouteInfo.Name == routeName);
    var method = route.ActionConstraints.OfType<HttpMethodActionConstraint>().First().HttpMethods.First();
    var url = Url.Link(routeName, values).ToLower();
    return new Link(url, relation, method);
}
```

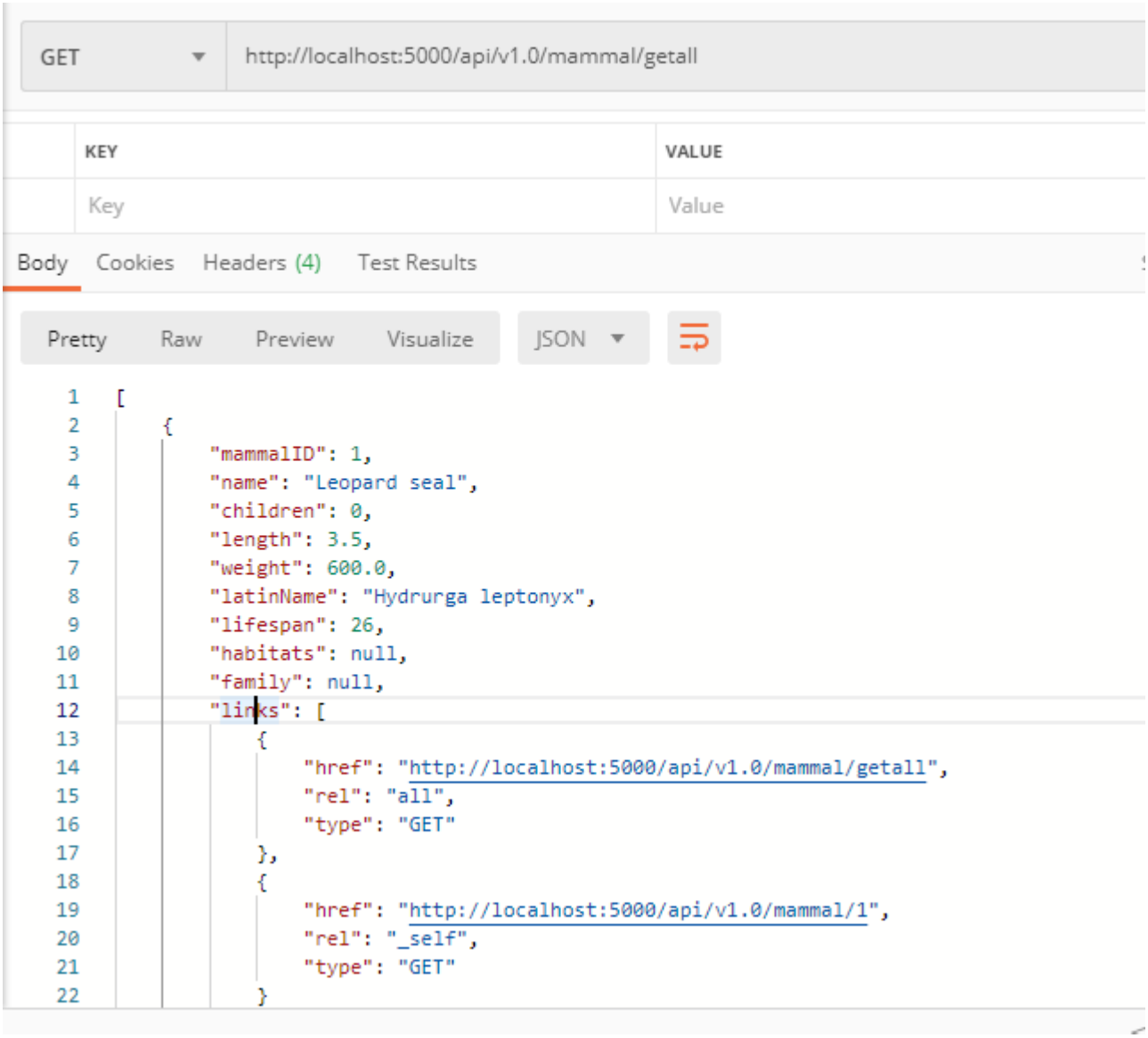
Sedan hämtar den sökvägen från det itemet i den metoden den letar efter och relationen och skapar en länk av det.

Länken/länkarna läggs till i resp. DTO som skall hämtas och visas av queriet:

```
2 references
internal MammalDTO HateoasMainLinks(MammalDTO mammal)
{
    MammalDTO mammalDto = mammal;

    mammalDto.Links.Add(UrlLink("all", "GetAll", null));
    mammalDto.Links.Add(UrlLink("_self", "GetMammalAsync", new { id = mammalDto.MammalID }));

    return mammalDto;
}
```


GET
http://localhost:5000/api/v1.0/mammal/getall
KEY
VALUE
Key
Value
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"mammalID": 1,
"name": "Leopard seal",
"children": 0,
"length": 3.5,
"weight": 600.0,
"latinName": "Hydrurga leptonyx",
"lifespan": 26,
"habitats": null,
"family": null,
"links": [
{
"href": "http://localhost:5000/api/v1.0/mammal/getall",
"rel": "all",
"type": "GET"
},
{
"href": "http://localhost:5000/api/v1.0/mammal/1",
"rel": "_self",
"type": "GET"
}

Skriven av Micael Wollter
2020-05-22

Implementation av HATEOAS-links till era API-controllerklasser:

Nedan följer hur man implementerar HateoasControllerBase.cs i era riktiga Controllers.

I er riktiga Controller måste följande namespace vara med. Exempel tas från MammalController.cs

```
1   using AutoMapper;
2   using MammalAPI.DTO;
3   using MammalAPI.Models;
4   using MammalAPI.Services;
5   using Microsoft.AspNetCore.Http;
6   using Microsoft.AspNetCore.Mvc;
7   using Microsoft.AspNetCore.Mvc.Infrastructure;
8   using System;
9   using System.Collections.Generic;
10  using System.Threading.Tasks;
11  using System.Linq;
12
```

Klassen/controllern måste **ärva**, och **Konstruktorn** måste göras om till följande (pro-tip: zooma in på pdf:en):

```
[ApiController]
[Route("api/v1.0/[controller]")]
1 reference
public class MammalController : HateoasMammalControllerBase
{
    private readonly IMammalRepository _repository;
    private readonly IMapper _mapper;

    0 references
    public MammalController(IMammalRepository repository, IMapper mapper, IActionDescriptorCollectionProvider actionDescriptorCollectionProvider) : base(actionDescriptorCollectionProvider)
    {
        _repository = repository;
        _mapper = mapper;
    }
```

Implementation på metoder:

Implementationen tar i åtanke Automapper, men ser lite annorlunda ut beroende om man hämtar flera objekt samtidigt ***(getall)*** eller endast ett objekt ***(getid).***

Hämta flera (getall):

Gör om var mappedResult = _mapper.Map<MammalDTO[]>(results); till följande:

```
[HttpGet("GetAll", Name ="GetAll")]
0 references
public async Task<IActionResult> Get()
{
    try
    {
        var results = await _repository.GetAllMammals();
        IEnumerable<MammalDTO> mappedResult = _mapper.Map<MammalDTO[]>(results);
        IEnumerable<MammalDTO> mammalsresult = mappedResult.Select(m => HateoasMainLinks(m));

        return Ok(mammalsresult);
    }
```

Hämta en (getid):

Vid hämtning av endast en behövs bara return OK göras om till följande:

```
[HttpGet("{id:int}", Name = "GetMammalAsync")]
0 references
public async Task<IActionResult> GetMammalById(int id)
{
    try
    {
        var result = await _repository.GetMammalById(id);
        var mappedResult = _mapper.Map<MammalDTO>(result);

        return Ok(HateoasMainLinks(mappedResult));
    }
```

Skriven av Micael Wollter
2020-05-22

Slutord

Det tog galet lång tid att hitta en vettig tutorial att fungera, och modifiera koden för att få den att fungera till API:t

Men med den här guiden tror jag det kommer bli hyfsat enkelt för er att vidareimplementera APIT samt lägga till länkar till övriga CRUD metoder. Ni har även chans att försöka vidareutveckla det så det kanske kan bli såsom StarwarsAPIt med egna länkar.

I mångt om mycket bygger alla tutorials jag har kollat på, på samma kodbas. Dvs de tre .cs filerna jag gått igenom, där samtliga använder Name = ”” - attributet i httpget.

I övrigt har de väldigt olika implementationer i de riktiga controllerklasserna, samt att de kan ha mer komplicerade, utökade baskodslösningar.

När man väl börjar förstå koden, så finns det otroliga potential att utforska. Kanske hinner man det inte i denna kursen, men på egen hand kan man lätt skapa ett eget projekt och bara kopiera koden och börja utforska ☺