REST. Ny funktion/service og Søgning/Filter samt Paging i din REST API

Opgave 1: Styring af Router (URI) i din REST API

Opgave 1.1: Generel URI til din REST API

Du skal i denne opgave undersøge den generelle URI (route) til din XXXXs**Controller** (fra simple Rest Service).

Når du har åbnet din XXXXsController, så vil der før klasse være en annotation
'`[Route("api/[controller]")]`' – Forklar hvad den gør?
Og hvorfor din URI ser ud som den gør (se bl.a. i hjælpesiden/swagger når du kører programmet)?

Hvordan kan du ændre din generelle URI/Route? Forklar de to måder.
Prøv de to måder.

Opgave 1.2: Opdel metode og URI(Route)

Du skal nu splitte metode fra URI (route) i de eksisterende 5 Services (funktioner) i din REST API.

Fx ser find element ud fra et id således ud '`[HttpGet("{id}")]`'

Du skal dele den op i to annotations én til metoden og en til URI'en (routen), hvorved den vil således ud:

```
[HttpGet]
[Route("{id}")]
```

Gør det samme for Put og Delete.

Prøv om din REST API stadig virker.

Opgave 1.3: Opret en ny metode/funktion til dit REST API

Du har hidtil lavet 5 services/funktioner til dit REST API.

Du skal nu tilføje en funktion/service til dit REST API.
Det kan for eksempel være at finde alle Actors født efter 1995 (eller du finder på noget andet)

Du skal definere HTTP-metode, samt URI for din nye Funktion/Service, samt implementere den i ActorsControlleren samt i din bagved liggende ActorRepository (fra dit library). Samt passende Status koder.

Opgave 1.4: Prøv din REST-service med den nye service/funktion

Kør din REST-service og prøv de forskellige metoder igennem den brugergrænseflade (swagger), der popper op.

Opgave 2: Mere generel Søgning/filtrering i din REST API

Opgave 2.1: Søgning / filtrering med 'query' i din REST API

Du skal nu implementere, at din REST API kan understøtte søgninger med queries (?). Fx:

- …/api/Actors/search?BirthBefore=2000
- …/api/Actors/search?BirthAfter=1990
- …/api/Actors/search?BirthAfter=1980&BirthBefore=2001

For at kunne lave dette skal du lave tre ting:

1. record (fx: FilterXXX) til at have søge-værdierne i.
2. Refakturer din XXXXRepository klasse (+ interface) , så den understøtter filtering
3. Refakturer din ActorsController klasse, så den indeholder den ny funktion/service i dit REST API
4.

Opgave 2.1.1 FilterXXX-Klasse

Du skal lave en record FilterXXXX i din "model" mappe. Denne record skal bare have properties til de værdier der ønskes søgning/filtrering på
fx: fra eksemplet ovenover to properties 'BirthBefore' og 'BirthAfter'

HUSK at lave properties nullable, dvs. de har et '?'.

F.eks. public record FilterDto (int? BirthAfter, …);

Opgave 2.1.2 Refakturer din XXXXRepository klasse

Du skal nu understøtte i (dit interface og) repository-klasse, at der en metode søg (search), der returnere en liste af data (objekter).

Du kan, fordi du har lavet properties nullable, tjekke om de er null, dvs de er ***ikke*** sat i søge-strengen.
Kun properties, der er sat, skal benyttes i søgningen.

Hvad sker der, hvis ingen af properties er sat? (design spørgsmål)

Måske du skulle lave en tilhørende UnitTest ☺

Opgave 2.1.3 Refakturer din ActorsController klasse

Du skal definere og implementere en service/funktion i dit REST API, der understøtter en søgning.

- Hvilken type http metode er det?
- Hvilken URI(Route) definere du?
- Hvilke statuskoder skal den returnere?

Når du skal 'hente' søge informationen skal du nu benytte din filterXXX klasse. Dvs. i din controller metodes parameter liste skal du have ([FromQuery] FilterXXXX filter).

FX:

```
public  IActionResult Search([FromQuery] FilterItem filter)
```

Opgave 2.2: Prøv din REST-service med den nye søge-funktion

Kør din REST-service og prøv de forskellige metoder igennem den brugergrænseflade (swagger), der popper op.

Ekstra A: Paging i din REST API (denne opgave kan springes over – men prøv ☺)

I stedet for at få alle data / objekter på én gang, kan din REST API understøtte at give mindre mængder ad gangen.

Ifølge artiklen Oswago Universitet RESTful Service Best Practices kan dette gøres på to forskellige måder:

1. Gennem specificering i URI (fx: GET ...?offset=25&limit=25)
2. Gennem Header fields

Denne opgave omhandler den 2. måde (den 1. ligner lidt det du allerede har været igennem med query).

Opgave A.1 Refakturer din XXXXRepository

Du skal i din XXXXRepository refakturere dine GET-metoder, så de kan understøtte paginering, dvs. du overloader Get-metoderne, så de kan tage 2 ekstra parametre (lav-interval, høj-interval).
Du skal også refakturerer dit interface 'Ixxxx Repository'.

Opgave A.2 Refakturer din XXXXsController

Du skal i dine GET-metoder aflæse om der er et header-field 'Range', og så kalde den rette metode i manageren, samt sætte Content-Range som header-field svaret.
Baggrund : http://cs.oswego.edu/~alex/teaching/csc435/RESTful.pdf side 26

Du kan aflæse et header-field ved:

```
String rangeValue = Request.Headers["Range"];  // værdien er fx "0-9" – de første ti
                                               // eller 10-19 – fra 10 og de næste ti
```

Du skal desuden sætte værdier i svaret ved:

```
Response.Headers.Add("Content-Range", "<<værdien>>");
```

```
// hvor værdien er fx: ”0-9/45” dvs de første ti af 45
// eller 10-19/* dvs fra 10 og de næste ti ud af ukendt antal
```

Hint aflæsning af Range værdi:

```
String[] values = rangeValue.Split('-');
Int lowIx = int.Parse(values[0]);
Int highIx = int.Parse(values[1]);
```

Opgave A.3 Prøv dit REST API

Kør din REST service.
Prøv dit REST API gennem Postman, hvor du kan sætte header-fields.