SUPERHERO PROJEKT

TEC, H3, 2022

RESUME

Fullstack Applikation, dotnet 6 + Angular 14.
Repository Pattern og Unittests

Jack Baltzer

INDHOLDSFORTEGNELSE

SUPERHERO PROJECT

FORMÅL

Dette dokument følger undervisningen og indeholder kodeeksempler og screenshots.

Konceptet er at konstruere et **full-stack** projekt med en **API** og en **Clientside applikation**. API implementerer et **Repository Pattern**, og alle lagene verificieres via **Unittests**

Der benyttes **dotnet 6** og **Angular 14**, samt **git** til versionsstyrring

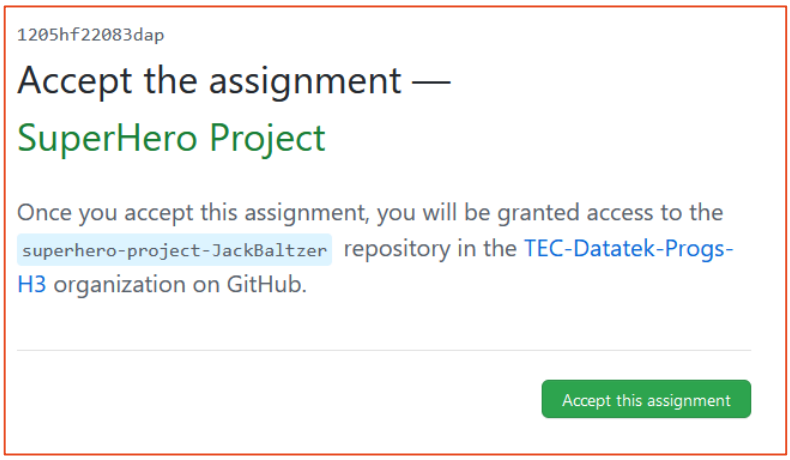
Programmer er **Visual Studio 2022** til API, og **Visual Studio Code** til clientside applikationen.

Sørg for at have installeret Visual Studio 2022 (community) før du går videre.
https://visualstudio.microsoft.com/vs/community/

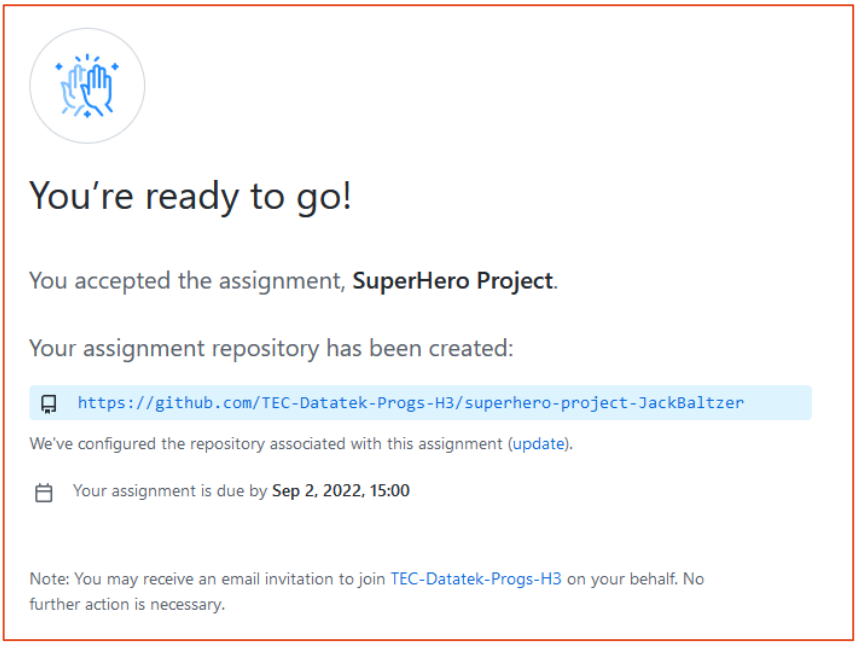
GITHUB ASSIGNMENT

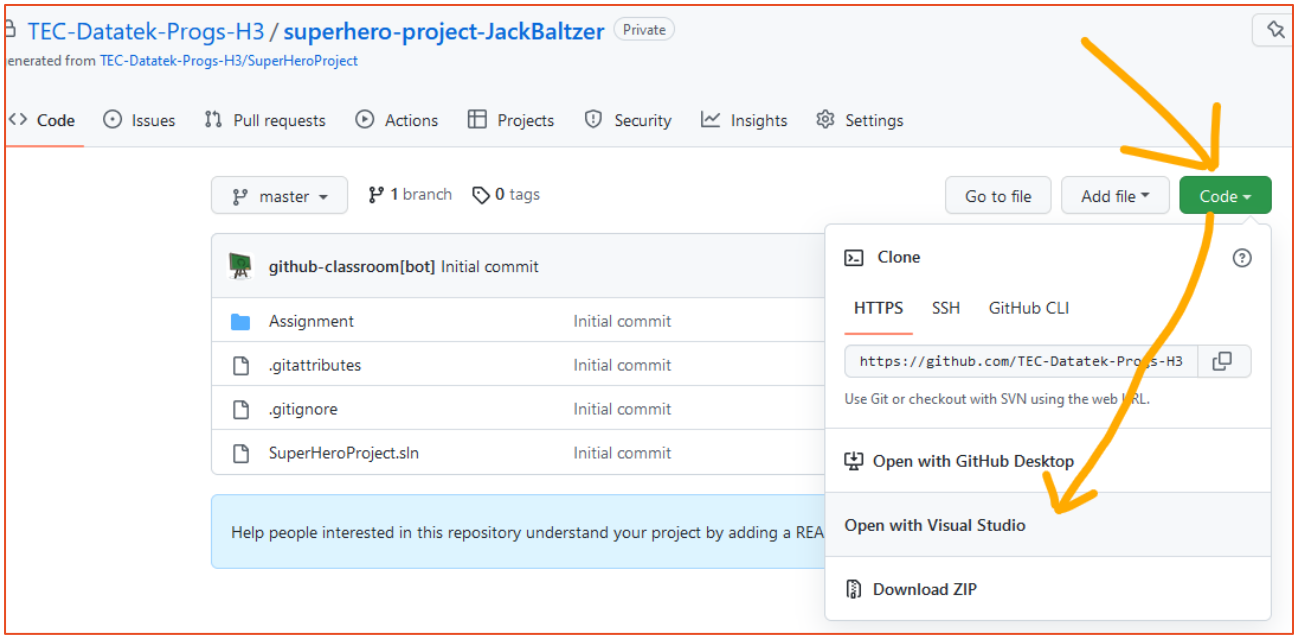
Via din browser, gå på https://classroom.github.com/a/Ob-FXKV9 og vælg dit navn i listen der kommer frem, dette binder din git-account sammen med navnet du har valg, så læreren kan finde ud af hvem der har afleveret.

Når du har valg navn, klik på den grønne knap "**Accept this assignment**" for at få adgang til opgaven.



Derefter går der et kort øjeblik, hvorefter du kan opdatere siden **[F5]** og se linket til dit nye github repository.

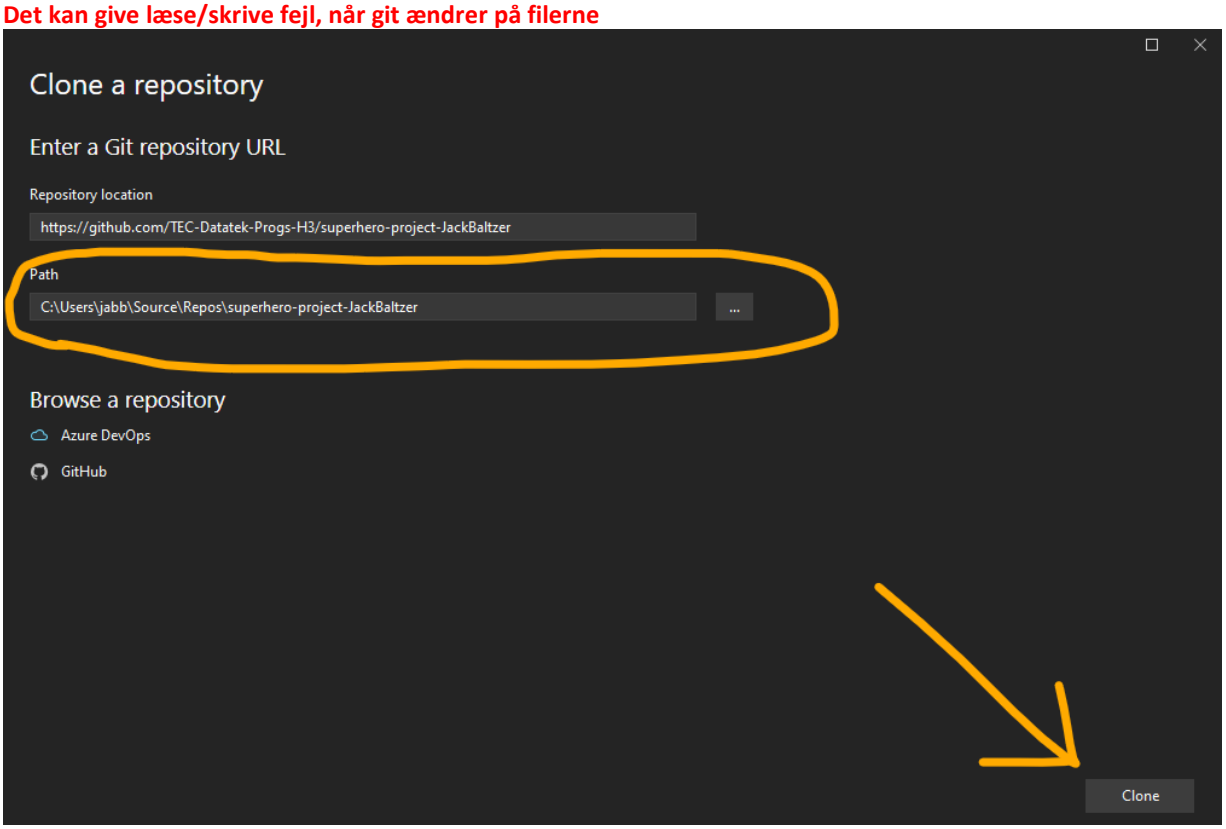




Det åbner Visual Studio, og samtidigt åbner dialogboksen for at clone dit repository. Sørg for at vælge en god placering til dit projekt lokalt, klik derefter på "**Clone**"

! VIGTIGT Lad være med benytte synkroniserede mapper som Dropbox eller OneDrive !

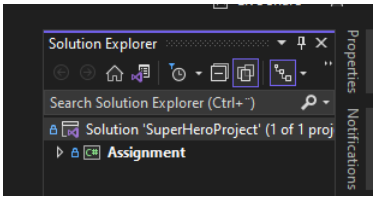
Det kan give læse/skrive fejl, når git ændrer på filerne

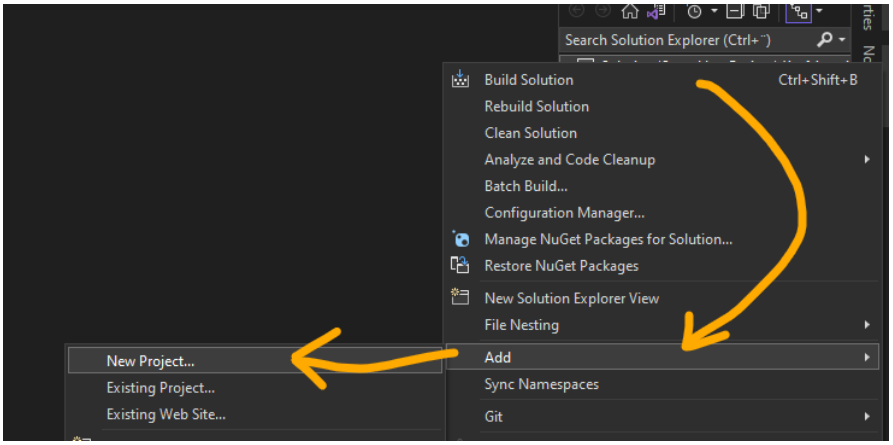


Nu har du en solution, hvor der ligger et projekt kaldet Assignment. Assignment indeholder en kopi af dette tekst dokument.

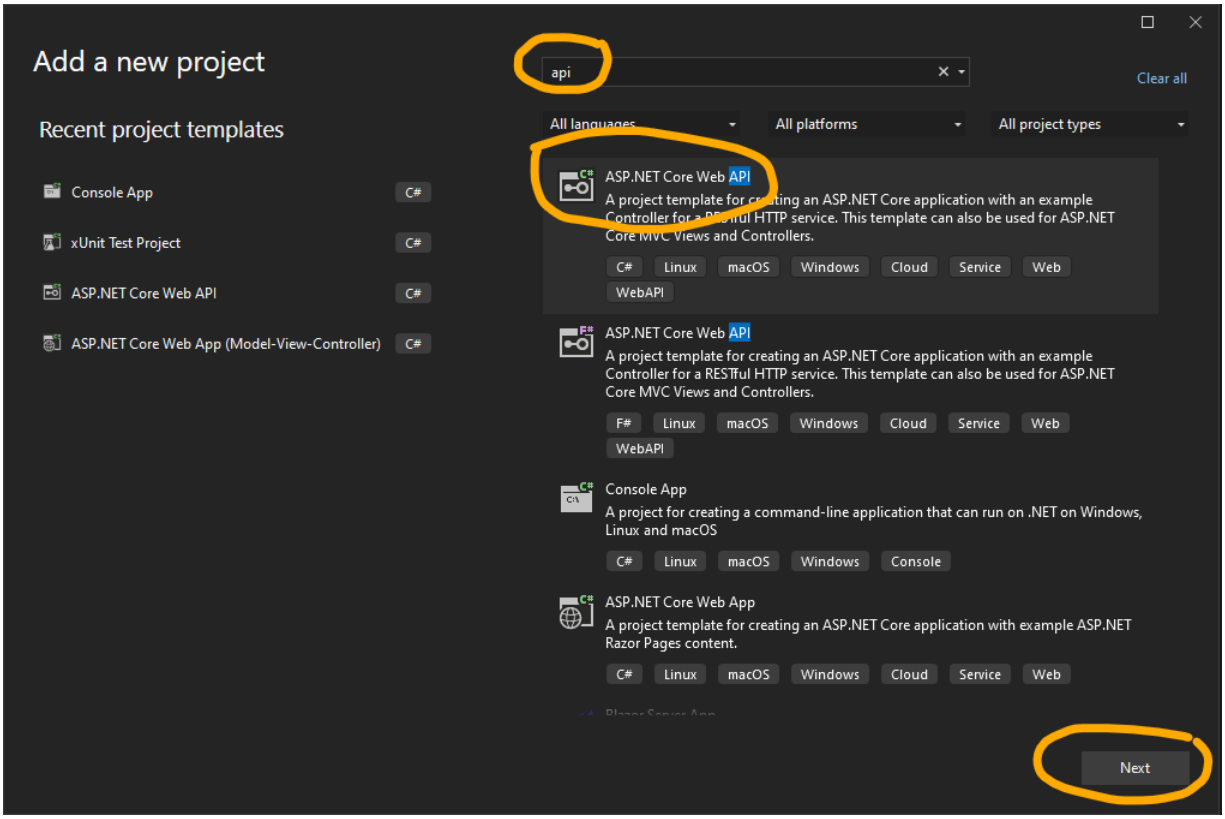
OPRET API PROJEKTET

API projektet skal placeres inde i SuperHero solution, dette klares ved at højreklikke på "**Solution SuperHeroProject**" i Solution Explorer

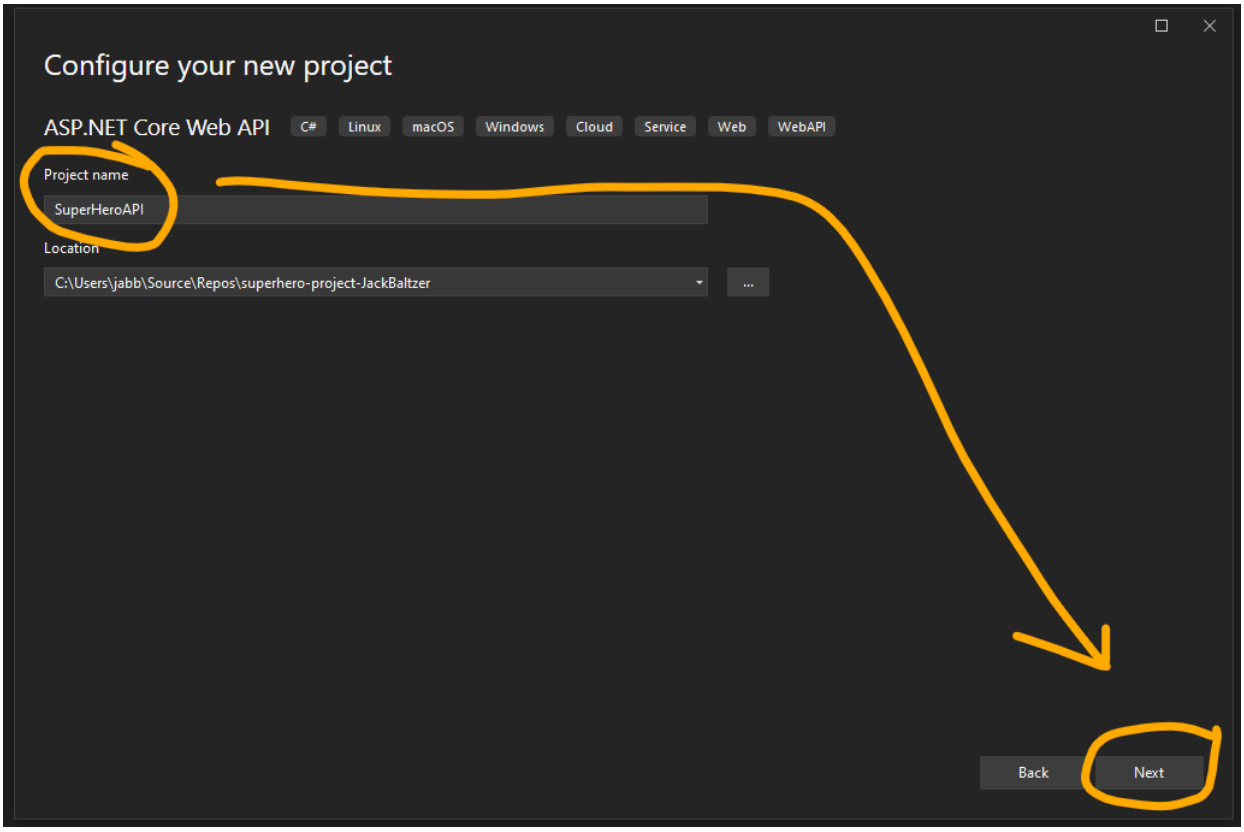




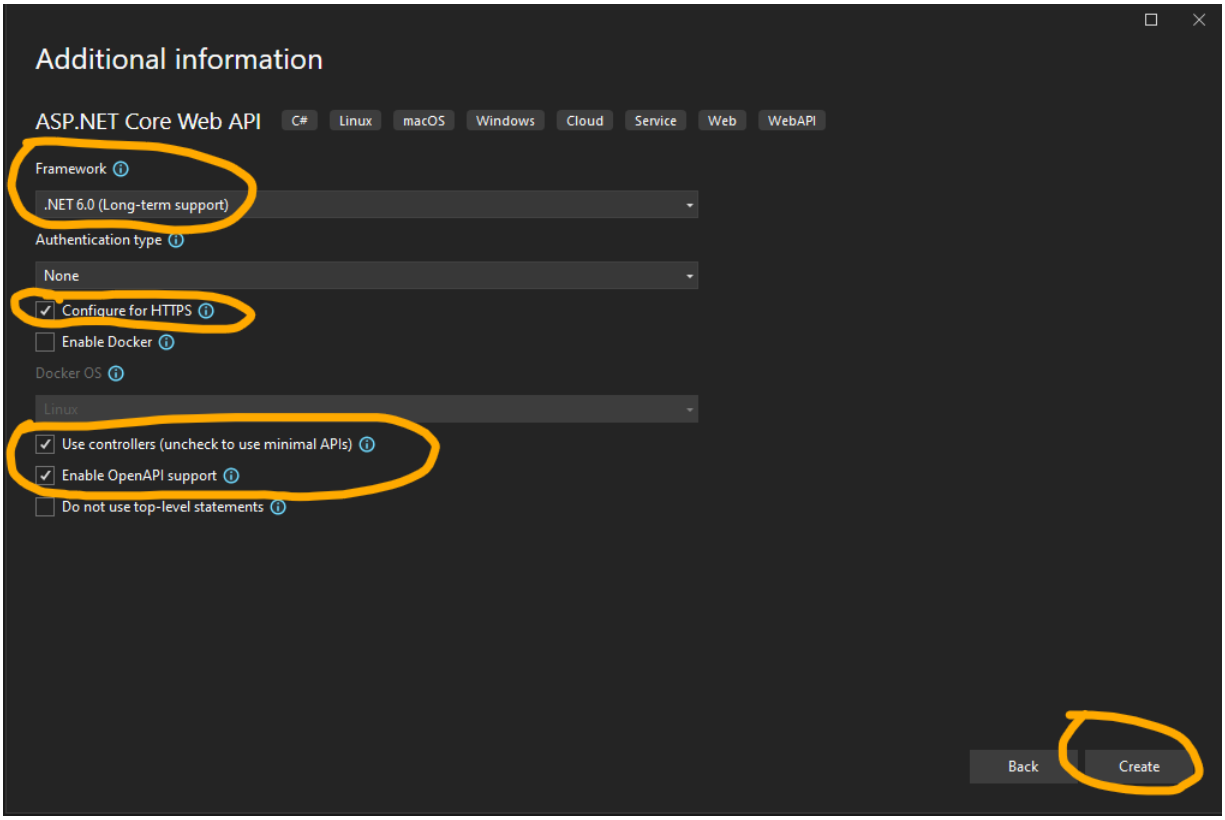
Søg efter "**api**" og sørg for at vælge "**C# ASP.NET Core Web API**" og klik derefter på "**Next**".

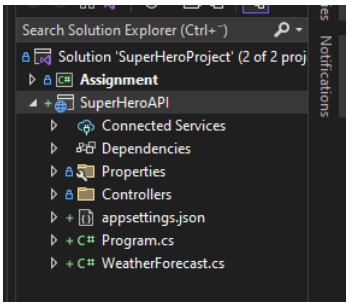


Navngiv projektet "**SuperHeroAPI**" og klik på "**Next**".

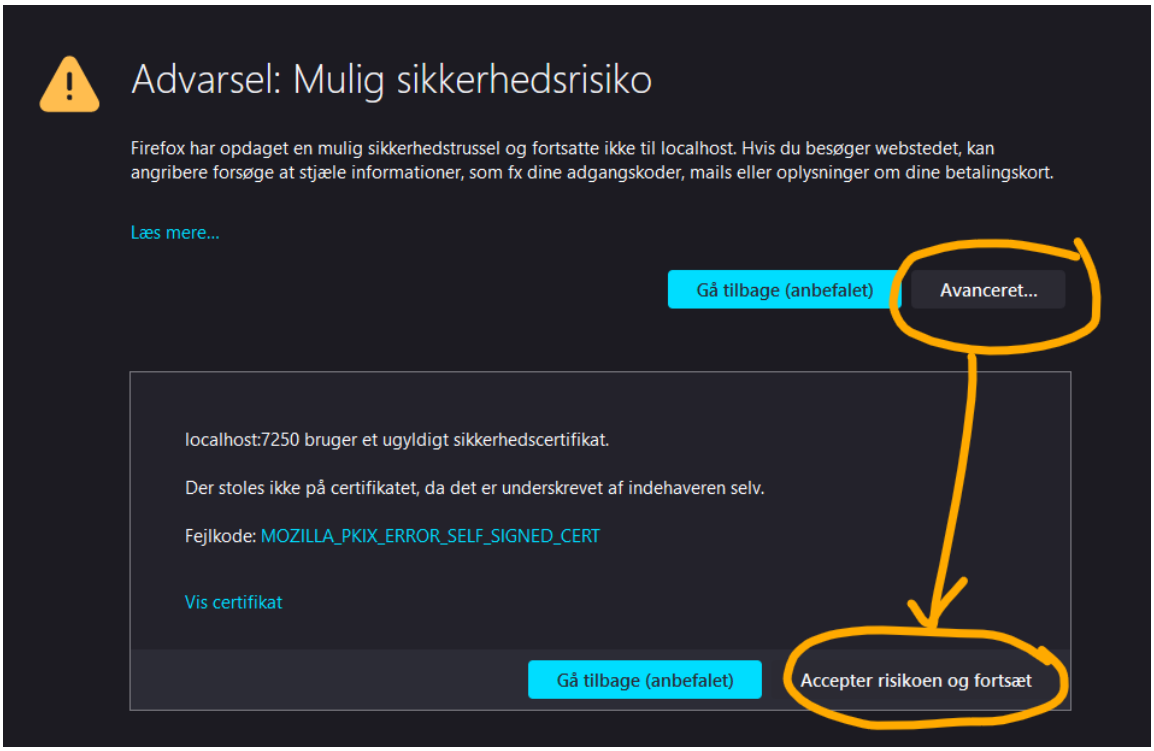


Sørg for at vælge "**.NET 6.0 (Long-term support)**" under framework, og sikre der er flueben ud for "**Configure for HTTPS**". Sørg også for at der er flueben ved "**Use Controllers**" og "**Enable OpenAPI support**" *(så vi får swagger opsat)*.

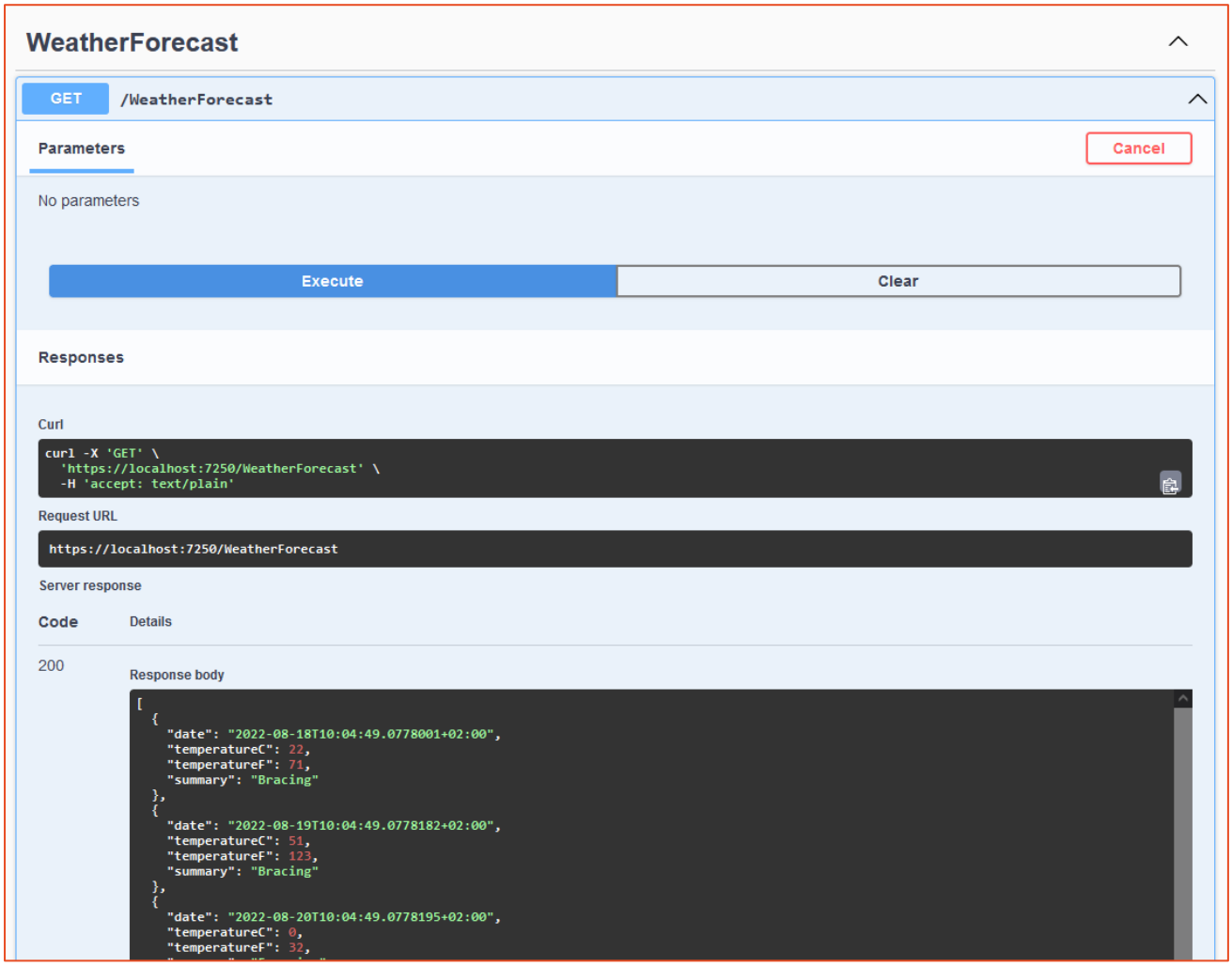




Højreklik på "**SuperHeroAPI**" og vælg "**Set as Startup Project**", dererfter tryk **[CTRL]** + **[F5]** for at køre projektet



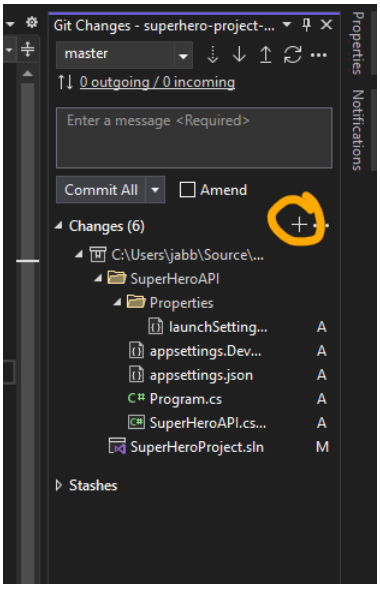
Tryk på den blå linje "**[GET] /WeatherForecast**" og klik på "**Try it out**" og derefter på "**Execute**". Så burde man se det resultat der kommer fra API'en.



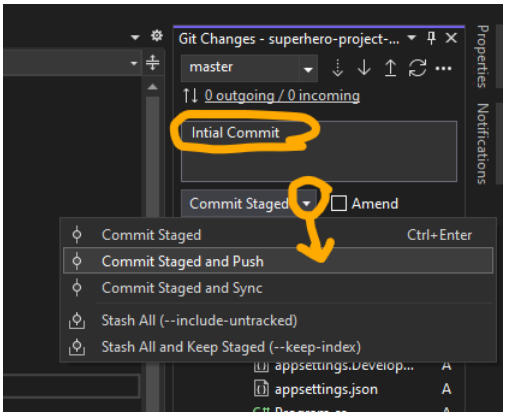
Slet filerne "**WeatherForecast.cs**" og "**WeatherForecastController.cs**" da de ikke skal bruges til noget fremover.

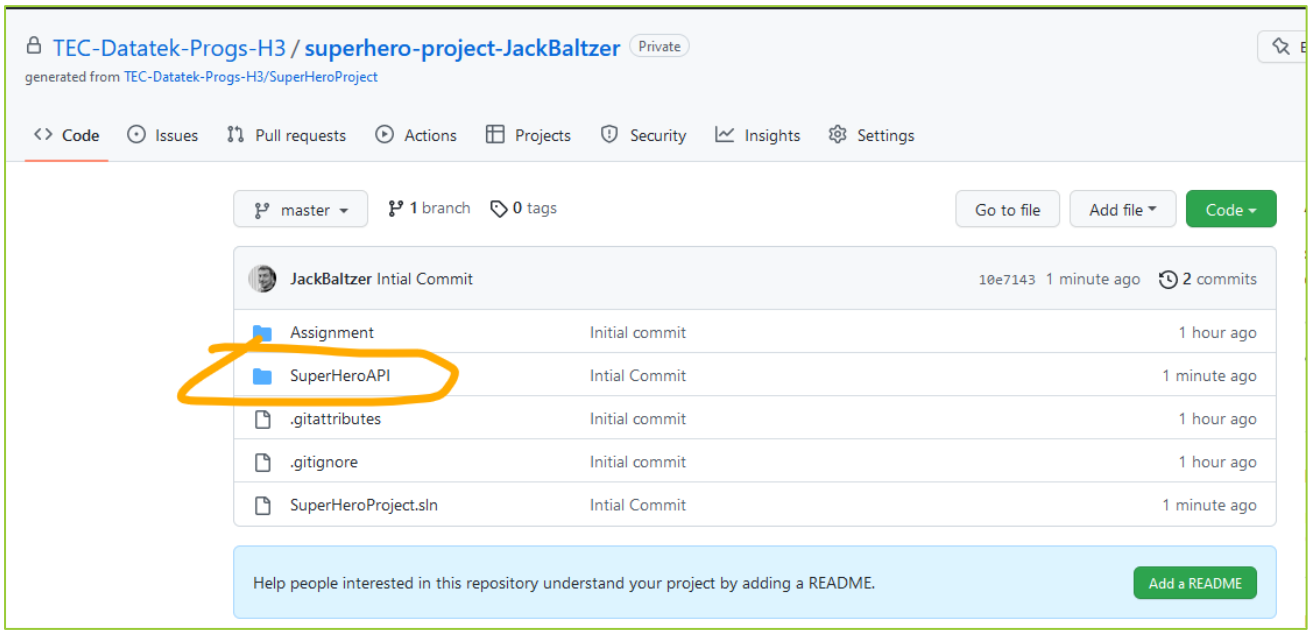
GIT

Find "**Git Changes**" vinduet under "**View**" menuen, og klik på plusset ud for changes.



Derefter skriv en besked i tekstfeltet og klik på den lille pil ud for "**Commit All**" og vælg "**Commit Staged and Push**".



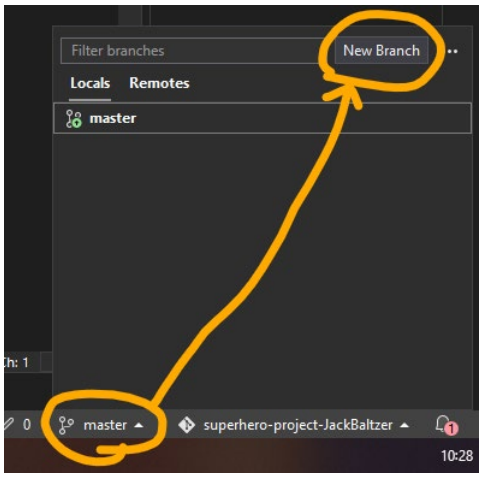


Git Branch

Det vil være et rigtig godt tidspunkt at bryde git-repository op så den version af koden der ligger i ”**Master**” altid er en velfungernede udgave af vores kode.

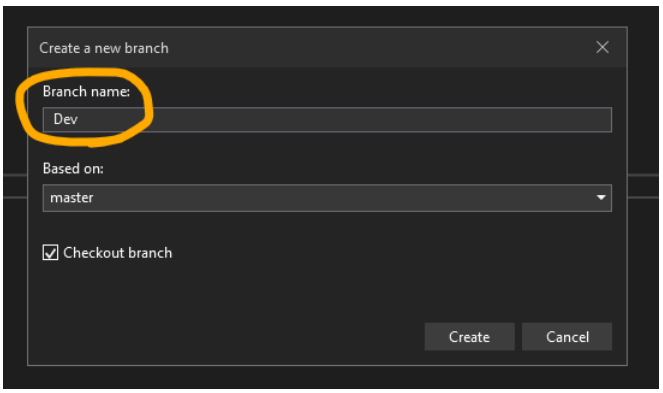
Til det benyttes en ”**Branch**”.

I Visual Studio nederest i højre hjørne, klikkes på ”**master**” og i vinduet der kommer op, klikkes på ”**New Branch**”



I vinduet der kommer frem, angives navnet på den Branch vi vil oprette. Jeg kalder min ”**Dev**” da det er meningen det er min udviklings branch.

Klik Create, og læg mærke til der hvor der før stod ”**master**” står der nu ”**Dev**”… så er du klar.

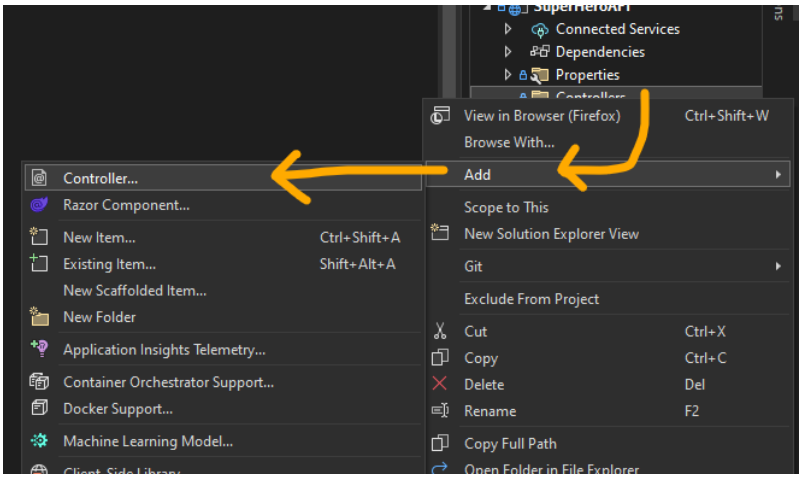


KOD ALDRIG DIREKTE PÅ MASTER!!!!

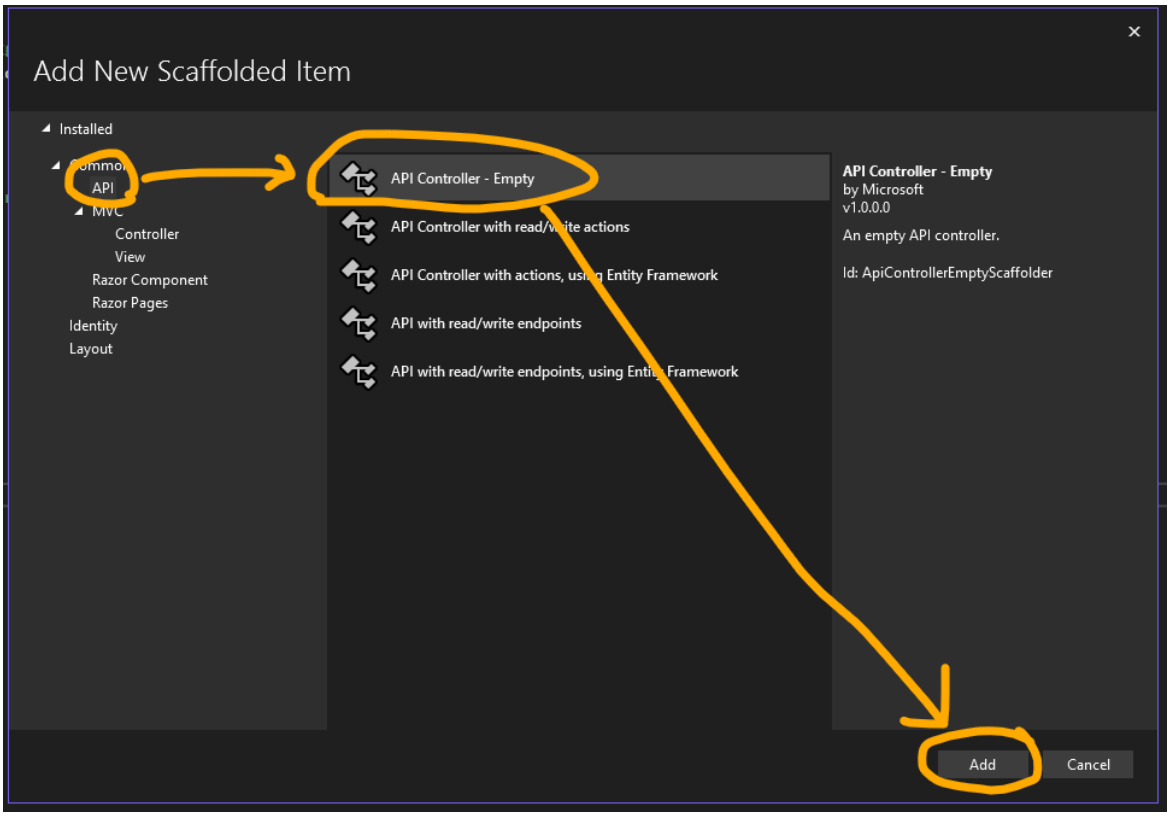
SUPERHERO CONTROLLER

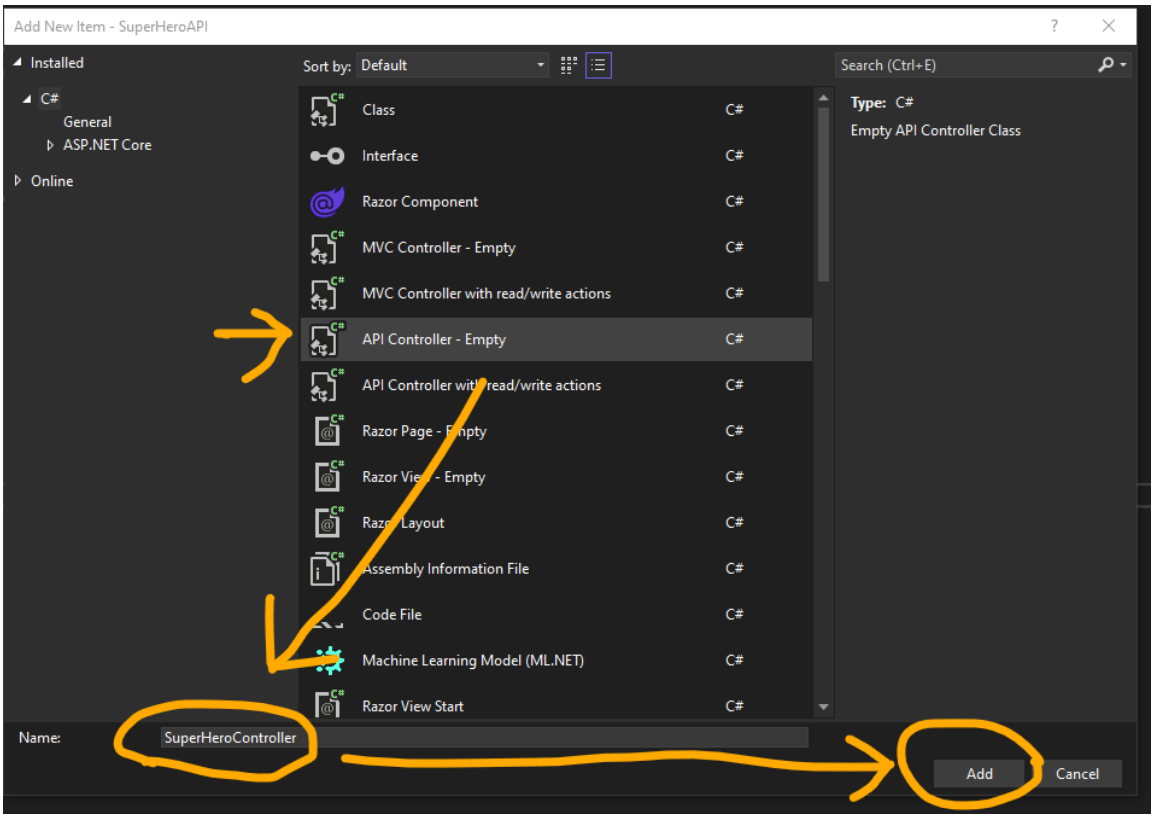
API'en har det formål at den skal lad os kunne udføre fuld "**CRUD**" på superhelte. Det betyder vi skal kunne hente alle helte, hente en helt, oprette en ny, redigere en eksisterende helt, og slette en eksisterende helt.

Indgangen i API'en er altid en Controller. Da den modtager forespørgsler og svarer tilbage med resultaterne.



Sørg for at vælge "**API**" og "**API Controller – Empty**", klik på "**Add**".





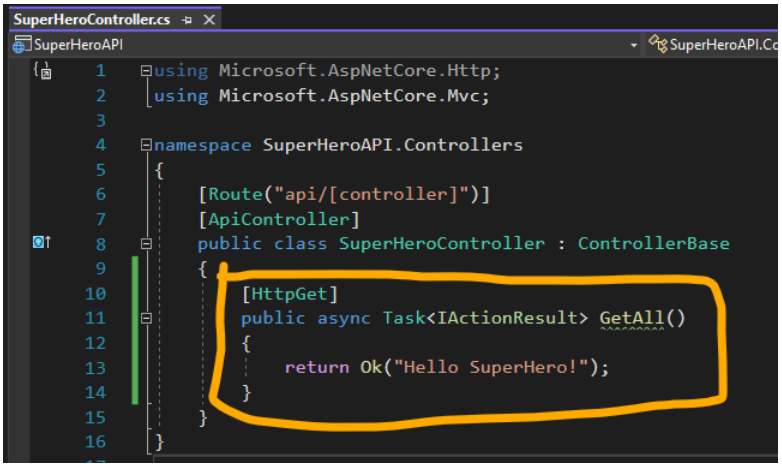
Så har vi en helt tom ny controller til vores helte.

GetAll()

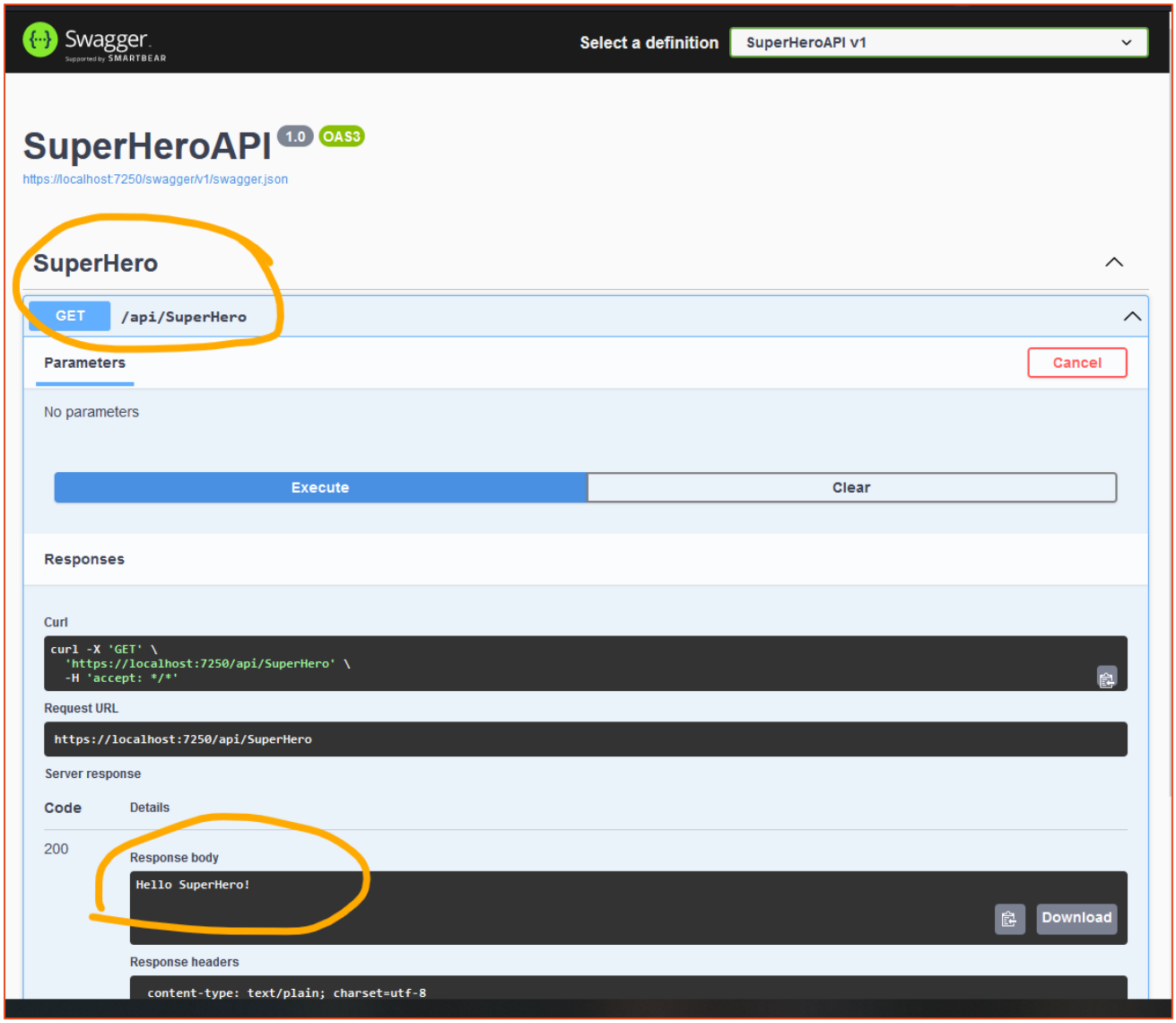
For at oprette et endpoint der kan returnere alle superhelte i vores system, skal vi oprette en metode i controlleren.

Metoden skal opsættes så den kan tilgås via et "**HttpGet**" kald, og den returnerer et "**IActionResult**".

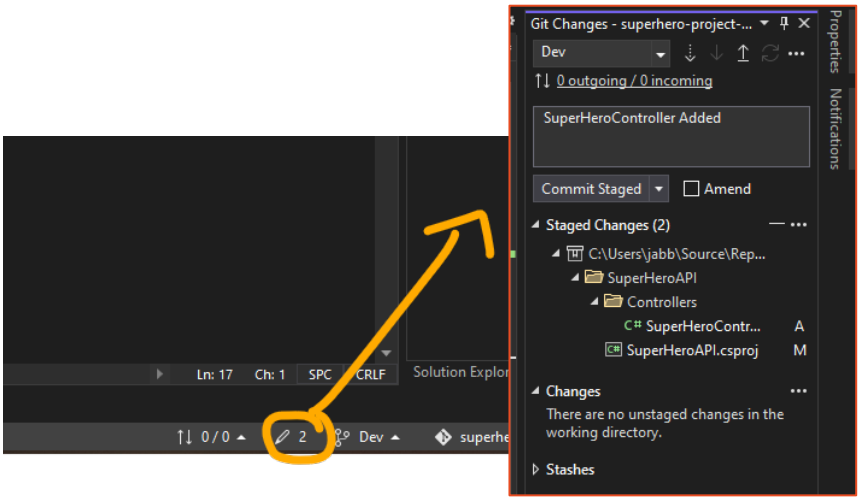
Da det er en "**WebAPI**", giver det mening at sørge for metoden er "**async**", så vil applikationen være "**multi-threaded**" fra starten af, og vil opleves mere responsiv efterhånden som programmet vokser.



Gem og kør applikationen **[CTRL]** + **[F5]** og se "**Swagger**" automatisk har fundet den nye Controller *(og fjernet WeatherForecast)*. Execute "**[GET] /api/SuperHero**" og se resultatet "**Hello SuperHero!**"



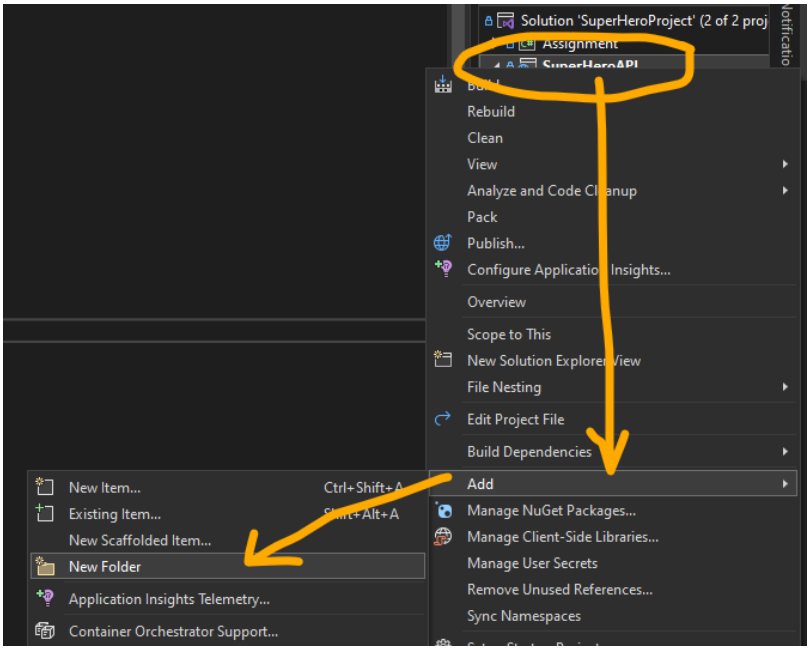
Gem alt, og commit ændringerne til din Dev branch!



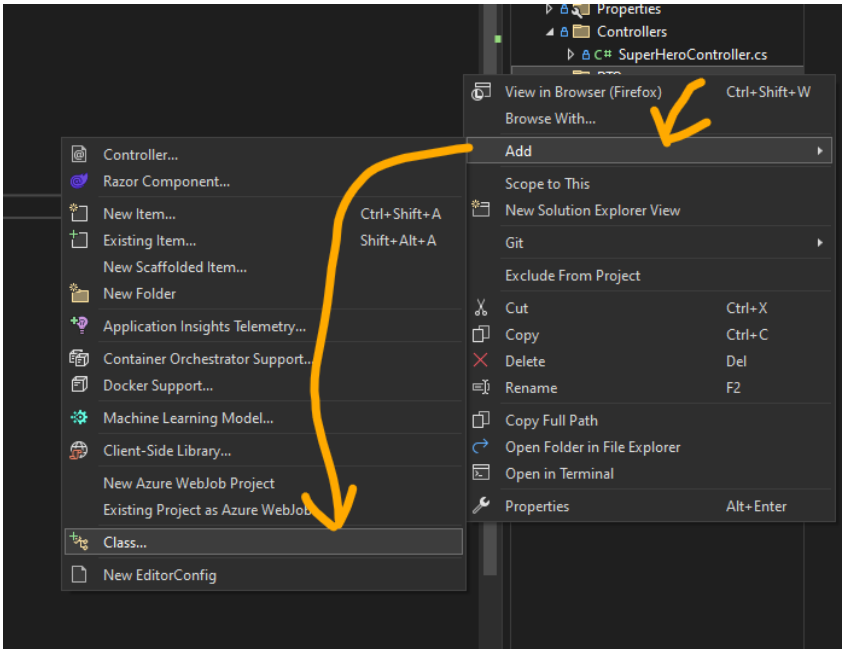
DTO'S – DATA TRANSFER OBJECTS

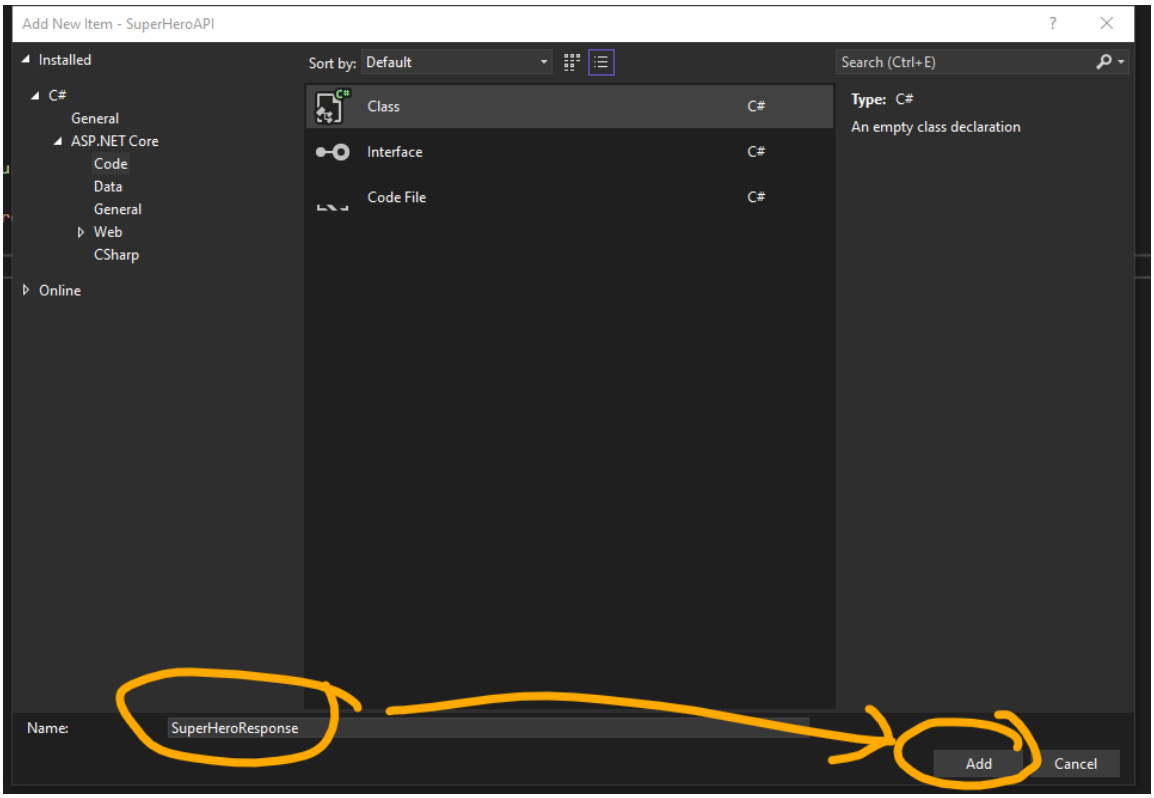
Vi vil gerne kunne repræsentere en SuperHero i vores kode, det vil gøre vores arbejde lettere, hvis vi ved hvordan en SuperHero er struktureret. Så kan vi validere input og sikre output har alle de ønskede værdier.

Højreklik på "**SuperHeroAPI**" projektet, vælg "**Add**" og derefter "**New Folder**" og kald den "**DTOs**".

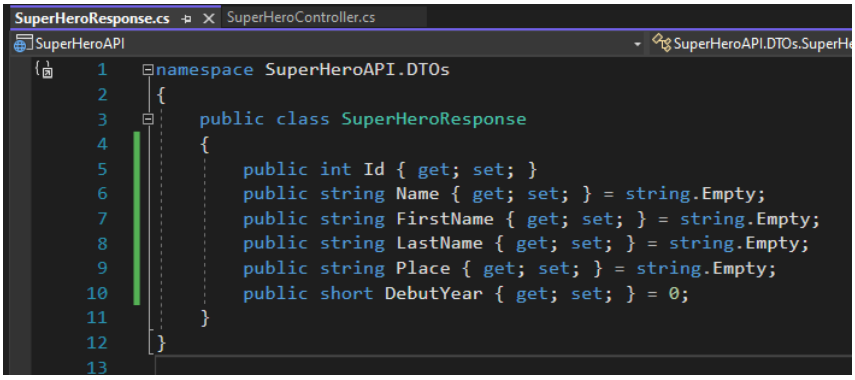


Højreklik på den nye "**DTOs**" mappe, vælg "**Add**" og derefter "**Class**".





Opret alle de følgende egenskaber i "**SuperHeroResponse**" klassen



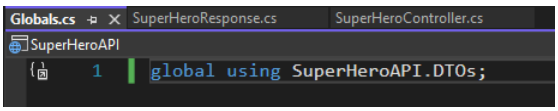
Nu kan vi oprette en liste af superhelte som kan returneres via "**GetAll()**" i controlleren.

GLOBAL USINGS

I **C# 10** kan vi benytte "**global usings**", så vi kan definere et sted med vores "**usings**", frem for at skulle pege på resourcerne mange steder i flere filer.

Højreklik på "**SuperHeroAPI**" projektet, vælg "**Add**" og derefter "**New Item**". Opret en "**CodeFile**" kaldet "**Globals.cs**".

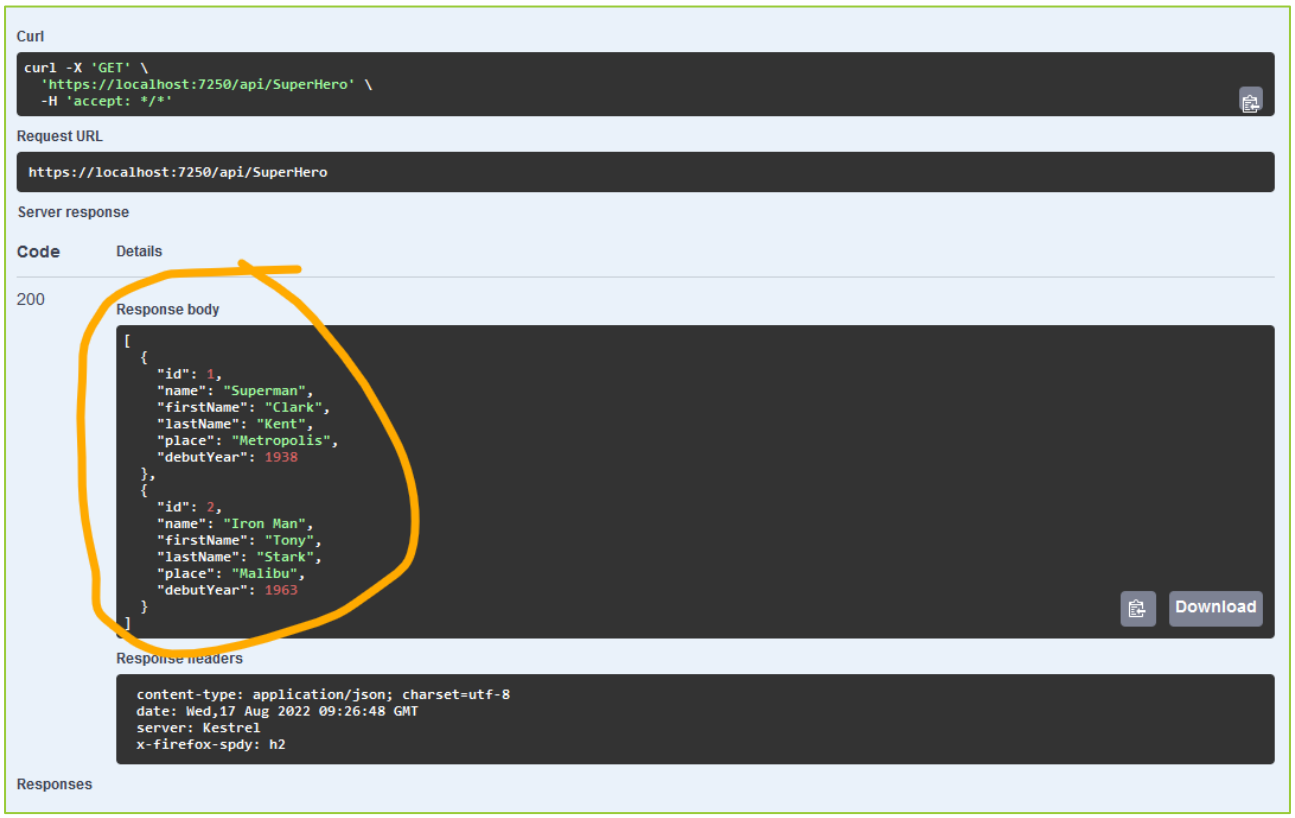
I den tomme fil tilføjes denne linje "**global using SuperHeroAPI.DTOs;**" det er alt hvad filen skal indeholde lige nu.



GetAll – DTO returnering

Tilpas Controller metoden ”**GetAll()**”, så der oprettes en liste af SuperHeroResponses, læg mærke til at klassen ”**SuperHeroResponse**” er kendt nu pga global using.

```
using Microsoft.AspNetCore.Mvc;

namespace SuperHeroAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class SuperHeroController : ControllerBase
    {
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            List<SuperHeroResponse> superHeroes = new();

            superHeroes.Add(new SuperHeroResponse
            {
                Id = 1,
                Name = "Superman",
                FirstName = "Clark",
                LastName = "Kent",
                Place = "Metropolis",
                DebutYear = 1938
            });
            superHeroes.Add(new SuperHeroResponse
            {
                Id = 2,
                Name = "Iron Man",
                FirstName = "Tony",
                LastName = "Stark",
                Place = "Malibu",
                DebutYear = 1963
            });

            return Ok(superHeroes);
        }
    }
}
```



UNITTESTS

Hvorfor teste?

Det er rigtig smart, at vi kan holde styr på at al funktionalitet i vores kode, fungerer hele tiden. Dvs når vi ændrer på noget i koden, så skal vi helst ikke ødelægge den oprindelige funktionalitet. Og det tager generelt alt for lang tid at teste samtlige funktioner manuelt, hver gang noget kode er tilføjet.

"**GetAll()**" skal altid returnere en liste af "**SuperHeroResponses**".

Dog kan det være at systemet ikke indeholder nogle superhelte endnu, så er det en tom liste der skal komme ud af programmet. Og hvad nu hvis database kaldet fejler, eller noget helt andet er gået galt, så skal vores kode kunne håndtere det.

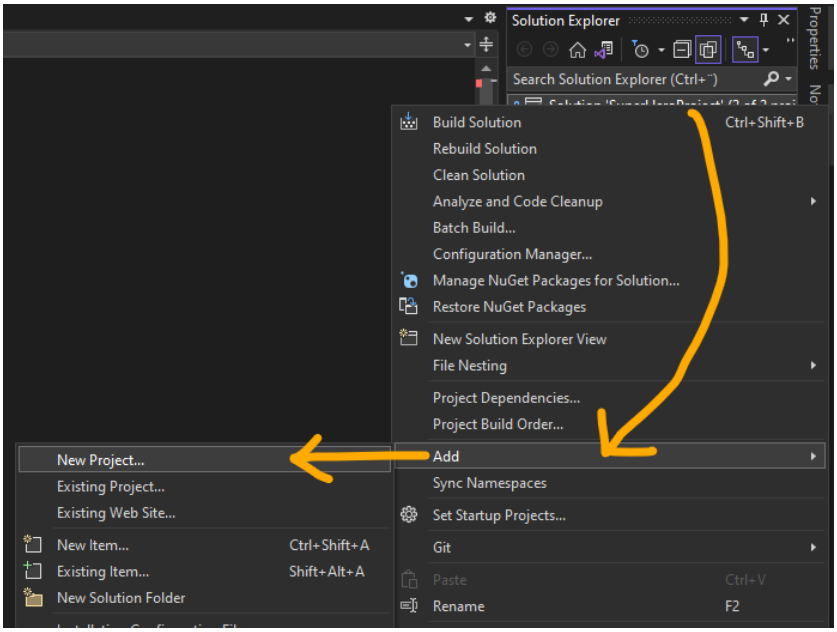
Og vi skal kunne "**bevise**" at vi har håndteret det! Her benyttes et koncept kaldet "**UnitTesting**".

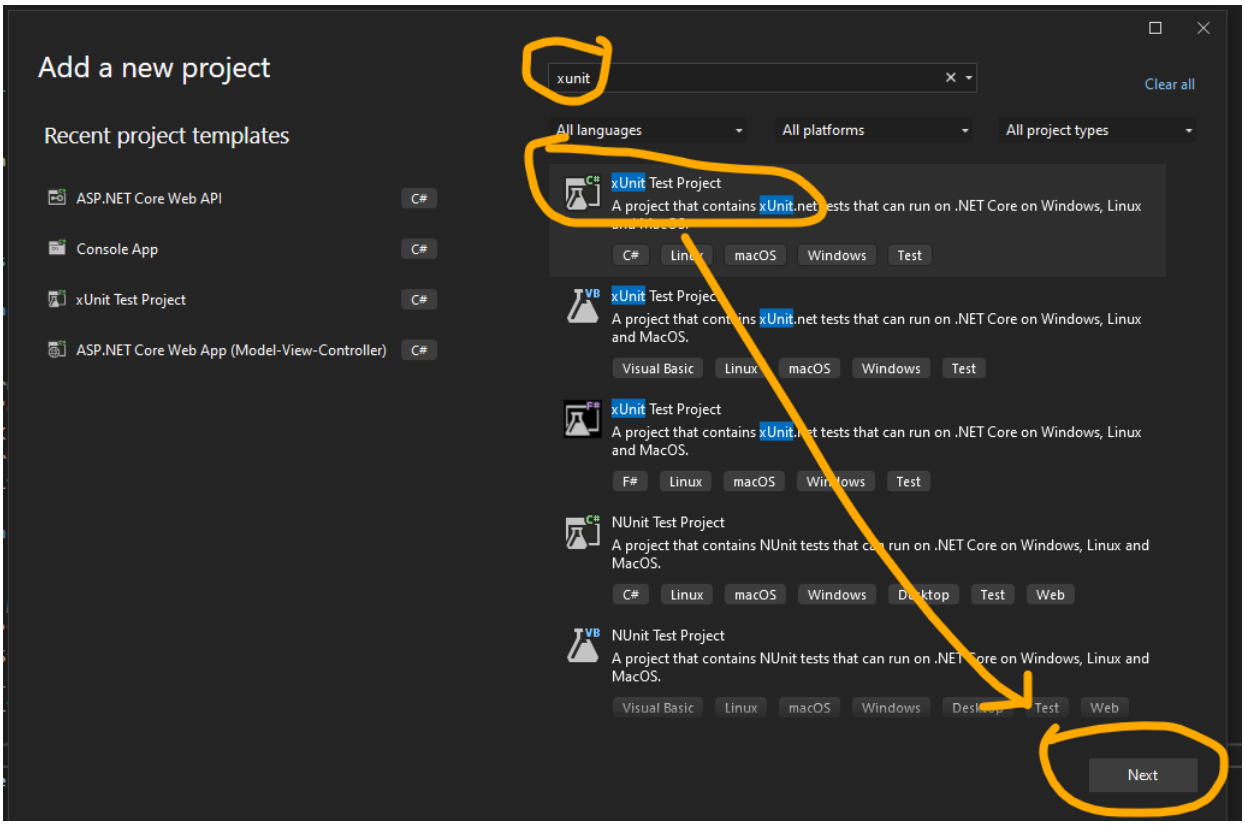
Det er i bund og grund at vi skriver lidt kode, som kører den ønskede kodefunktionalitet, og derefter ser vi via kode på resultatet og kan få indsigt i hvor koden har eventuelle fejl og mangler. Vi kan derved udføre mange tests igen og igen, samt hurtigt få et overblik over hvor koden fejler.

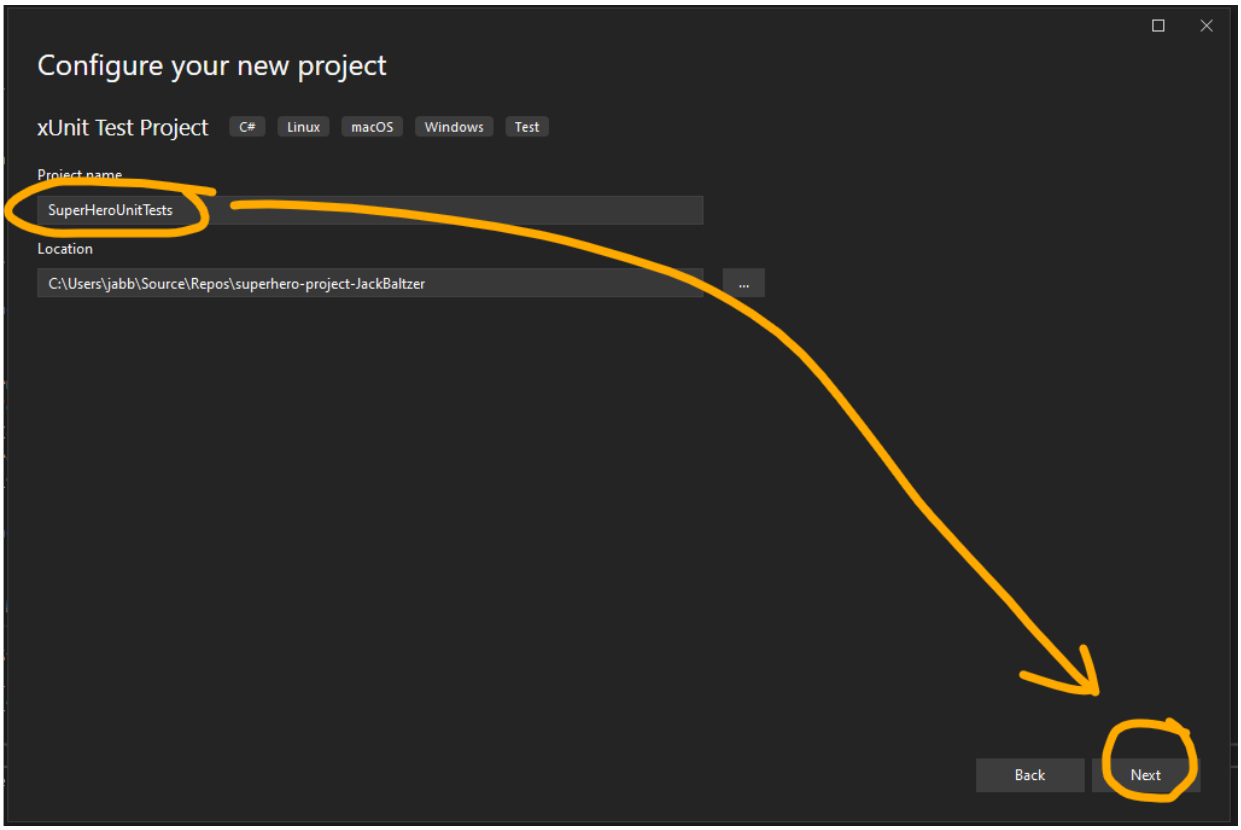
I vores projekt vil vi benytte "**Xunit**", og senere bygger vi "**Moq**" ovenpå *(mere om det når det bliver aktuelt)*.

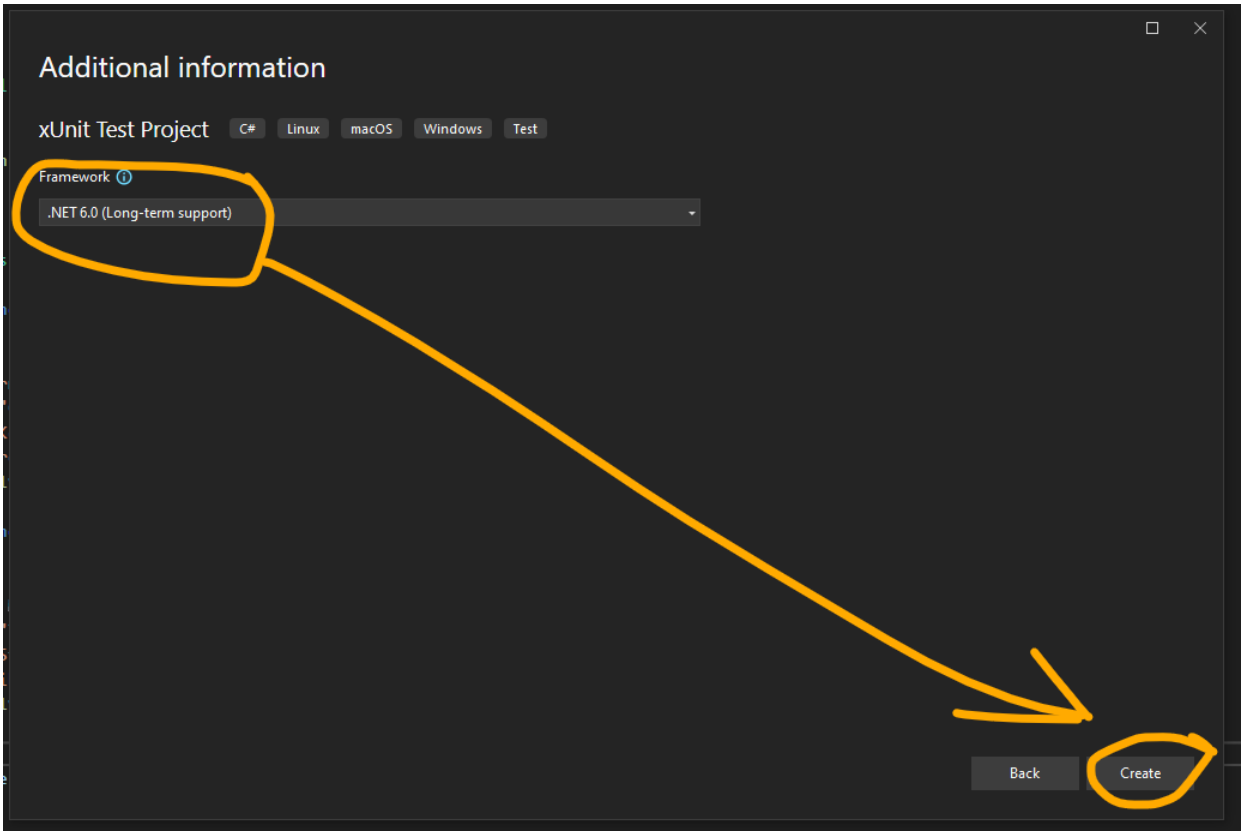
UnitTest Projektet

Opret et nyt projekt i din Solution, lige som da du oprettede SuperHeroAPI projektet.

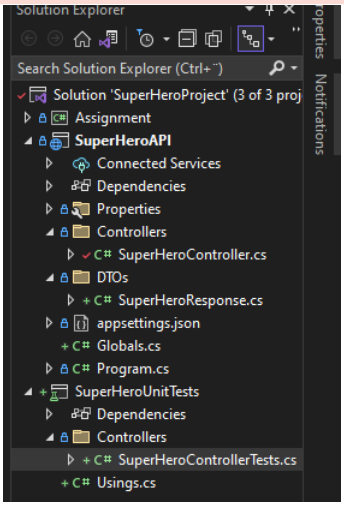

Solution Explorer
Search Solution Explorer (Ctrl+;)
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Analyze and Code Cleanup
Batch Build...
Configuration Manager...
Manage NuGet Packages for Solution...
Restore NuGet Packages
New Solution Explorer View
File Nesting
Project Dependencies...
Project Build Order...
New Project...
Existing Project...
Existing Web Site...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Solution Folder
Add
Sync Namespaces
Set Startup Projects...
Git
Paste Ctrl+V
Rename F2


Add a new project
xunit
Clear all
Recent project templates
All languages
All platforms
All project types
ASP.NET Core Web API C#
Console App C#
xUnit Test Project C#
ASP.NET Core Web App (Model-View-Controller) C#
xUnit Test Project
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux
C# macOS Windows Test
xUnit Test Project
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Linux macOS Windows Test
xUnit Test Project
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and MacOS.
F# Linux macOS Test
NUnit Test Project
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Test Web
NUnit Test Project
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Linux macOS Windows Test Web
Next


Configure your new project
xUnit Test Project
C#
Linux
macOS
Windows
Test
Project name
SuperHeroUnitTests
Location
C:\Users\jabb\Source\Repos\superhero-project-JackBaltzer
...
Back
Next


Additional information
xUnit Test Project
C#
Linux
macOS
Windows
Test
Framework
.NET 6.0 (Long-term support)
Back
Create

UnitTest Projekt Strukturen



Vi kommer til at teste en del forskellige filer i dette testprojekt, derfor kan det være en rigtig god ide at starte med en god fil-struktur.

Slet den "**UnitTest.cs**" fil der kom med fra starten, sørg for at lade "**Usings.cs**" filen være. Omdøb den eventuelt til "**Globals.cs**" hvis du foretrækker det.

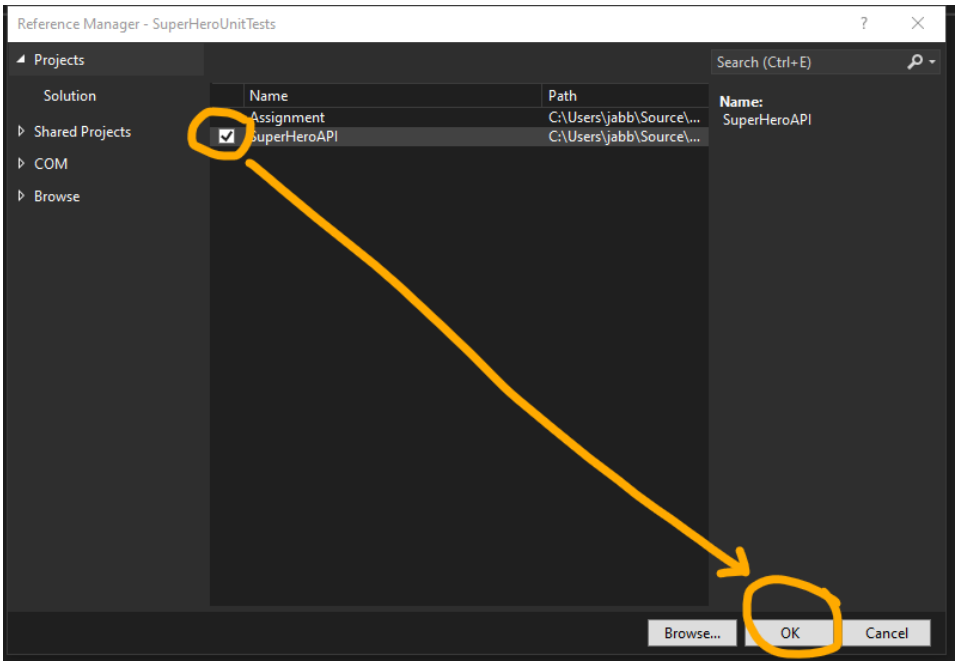
Opret en mappe kaldet "**Controllers**" i "**SuperHeroUnitTests**" projektet.

Opret en ny "**Class**" i "**Controllers**" mappen, kaldet "**SuperHeroControllerTests.cs**".

Ret "**internal class**" til "**public class**" så klassen SuperHeroControllerTests ikke længere er internal.

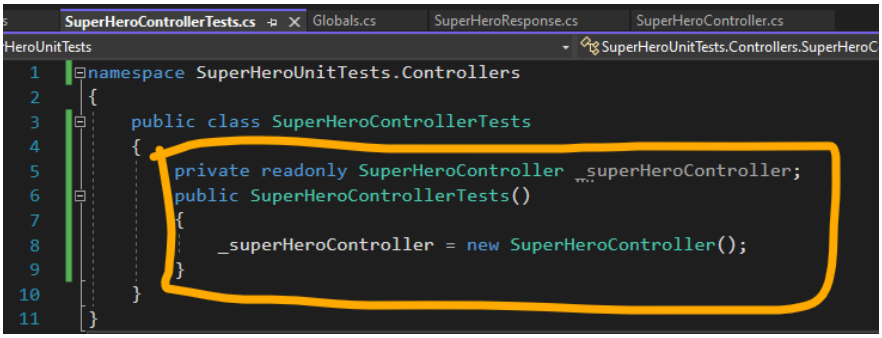
Filstrukturen burde nu se ud som her til højre:

Project Reference



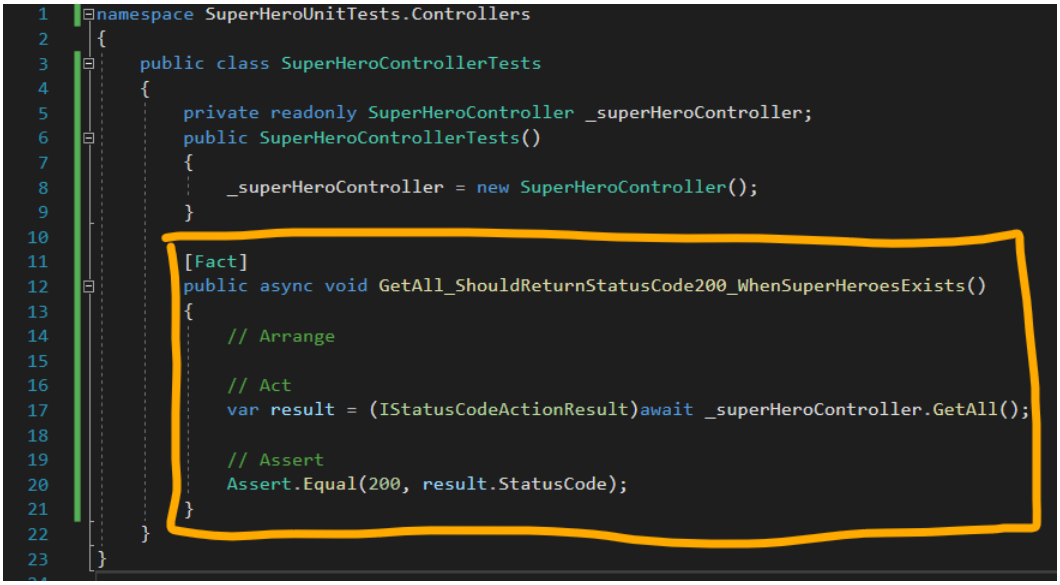
Xunit SuperHeroControllerTests Constructor

Da vi ønsker at udføre tests på "**SuperHeroContoller**" metoderne, skal vi oprette en instans af den i vores constructor. Hvis programmet ikke kan finde ud af hvad en "**SuperHeroController**" er, så mangler der en global using i "**Usings.cs**"



Efter constructoren kan oprette instanser af "**SuperHeroControlleren**", så kan vi skrive den første test.

En test "**annoteres**" med **[Fact]** hvis det er en metode der køres en gang pr test. Læg mærke til det forholdsvis lange metode navn "**GetAll_ShouldReturnStatusCode200_WhenSuperHeroesExists()**". Det er med vilje navnet er så langt, da navnet i sig selv beskriver hvad testens formål er… vi kommer til at have mange tests, derfor er navngivning ret vigtig, for overblikkets skyld.



De 3 x A

Ved Xunit arbejder man ud fra et princip om at man først arrangerer testen, derefter kører man det der skal testes, og så ser man på resultat og kommer med nogle krav som skal være opfyldt.

Det kaldes de 3 A'er, som står for "**Arrange**", "**Act**" og "**Assert**".

I vores første test, har vi ikke behov for at arrangere noget endnu, da vi stadig arbejder med 100% statiske data…vi kommer til at refactor denne test når service laget introduceres.

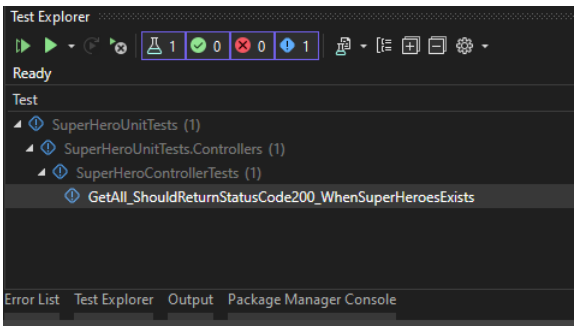
I Act området, kalder vi GetAll() og trækker resultatet ud som et "**IStatusCodeActionResult**" , da det i denne test udelukkende er statuskoden vi er interesseret i.

Under Assert området, kigger på om "**result.StatusCode**" er det samme som "**200**".

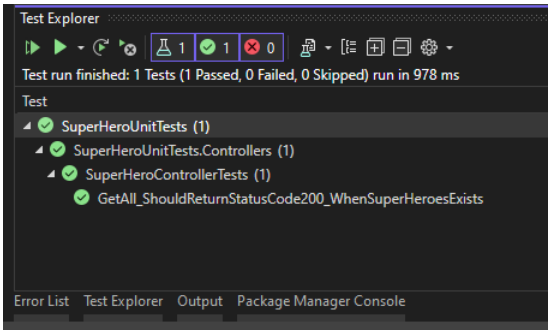
Assert.Equal sørger for at kaste en exception, hvis noget ikke stemmer, og vi behøver ikke skrive betingelser eller anden logik.

Kør Testen

Find Test Explorer under View, og fold alle punkterne ud. Alle metoder med **[Fact]** eller **[Theory]** vil optræde her i oversigten. Blå hvis de er nye ikke kørte tests, grønne hvis de har bestået, og røde hvis de har fejlet.



Kør ved at klikke på "**dobbelt pilen**", så udføres samtlige kendte tests. Der er kun en lige nu, og den burde bestå!



Gem og commit til Github!

SUPERHERO SERVICE

Vi har en Controller som kan returnere en liste af SuperHeroes, og en test som kan køres når som helst, men koden er statisk og fungerer kun som demonstration lige nu. Lad os trække listen af SuperHeroes ud i en Service, så kan vi abstrahere data væk fra controlleren, og bryde systemet op i mindre dele *(units)* som efterfølgende kan tests individuelt.

SuperHeroService – Interface

Da vi både vil arbejde med servicen i Controlleren, og vi vil udføre tests på servicen igennem Xunit, så giver det god mening at arbejde ud fra et interface, frem for konkrete objekter. ASP.NET Core Web Api har alt indbygget til at understøtte "**Dependency Injection**", så fundamentet er lagt for os.

Opret en mappe kaldet "**Services**" i roden af "**SuperHeroAPI**" projektet, lige som "**DTOs**" mappen.

Tilføje en ny class kaldet "**SuperHeroService.cs**" i mappen "**Services**"

```
namespace SuperHeroAPI.Services
{
    public interface ISuperHeroService
    {
        Task<List<SuperHeroResponse>> GetAll();
    }

    public class SuperHeroService : ISuperHeroService
    {
        public async Task<List<SuperHeroResponse>> GetAll()
        {
            List<SuperHeroResponse> superHeroes = new();

            superHeroes.Add(new SuperHeroResponse
            {
                Id = 1,
                Name = "Superman",
                FirstName = "Clark",
                LastName = "Kent",
                Place = "Metropolis",
                DebutYear = 1938
            });
            superHeroes.Add(new SuperHeroResponse
            {
                Id = 2,
                Name = "Iron Man",
                FirstName = "Tony",
                LastName = "Stark",
                Place = "Malibu",
                DebutYear = 1963
            });

            return superHeroes;
        }
    }
}
```

DEPENDENCY INJECTION

```
    [ApiController]
    public class SuperHeroController : ControllerBase
    {
        private readonly ISuperHeroService _superHeroService;

        public SuperHeroController(ISuperHeroService superHeroService)
        {
            _superHeroService = superHeroService;
        }
```

Derudover skal vi sætte programmet op, så den ved hvordan ISuperHeroService skal serveres.

Tilpas "**SuperHeroAPI/Program.cs**":

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ISuperHeroService, SuperHeroService>();


builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

```


SUPERHEROCONTROLLER.GETALL()

Nu er vi klar til at skrive den endelige "**GetAll()**" metode i vores "**SuperHeroController**", fordi vi lader Servicen bestemme hvilken data der skal returneres.

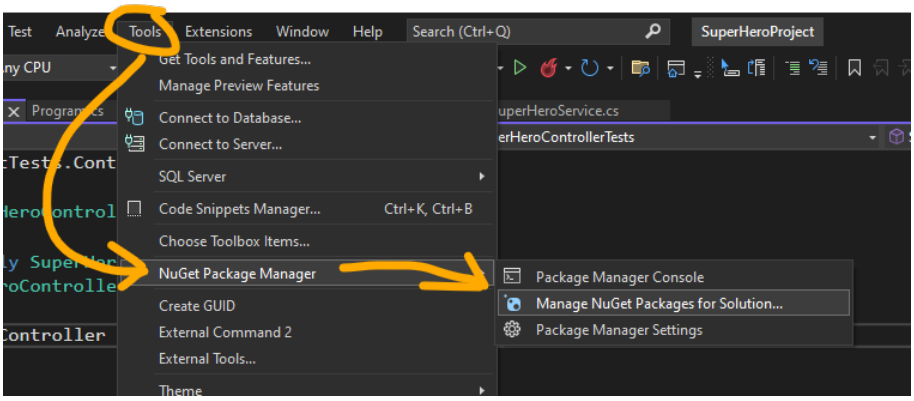
```
17          [HttpGet]
18          public async Task<IActionResult> GetAll()
19          {
20              try
21              {
22                  List<SuperHeroResponse> superHeroes = await _superHeroService.GetAll();
23
24                  if(superHeroes == null)
25                  {
26                      return Problem("Nothing was returned from service, this is unexpected");
27                  }
28
29                  if(superHeroes.Count() == 0)
30                  {
31                      return NoContent();
32                  }
33
34                  return Ok(superHeroes);
35              }
36              catch (Exception ex)
37              {
38                  return Problem(ex.Message);
39              }
40          }
```

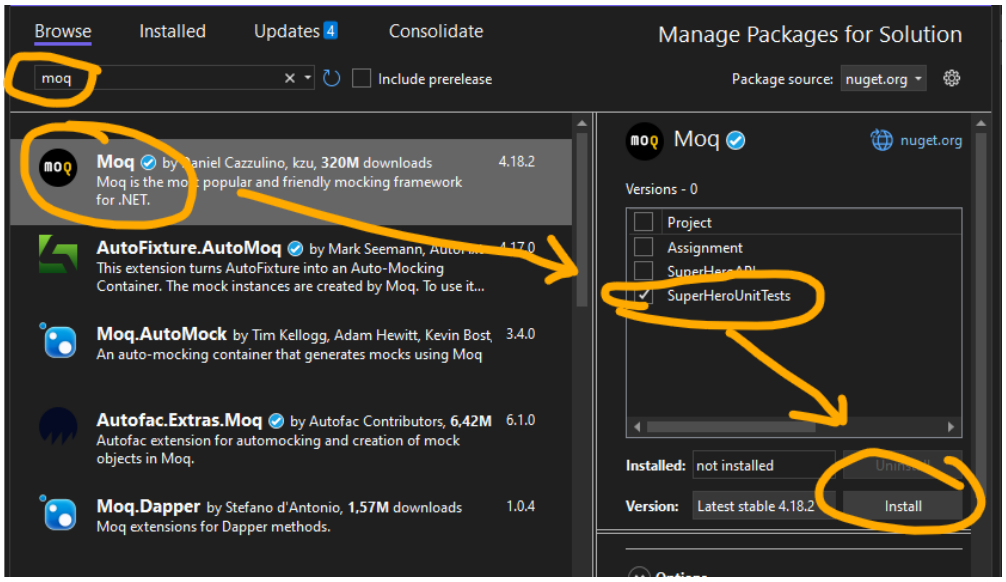
Her tager vi hånd om alle de scenarier vi bør håndtere. Der returneres "**200**" med "**Ok()**" når der er data, "**204**" med "**NoContent()**" hvis der ikke er nogle superhelte, og "**500**" med "**Problem()**" hvis der sker fejl.

"**try/catch**" blokken er en god ide for at sikre at alle fejl giver status "**500**" og at vi kan håndtere der er sket en fejl.

MOCK

Nu skal vi opdatere Unit-testen, for vores Controller er blevet afhængig af "**ISuperHeroService**", og det kan vores test ikke håndtere lige nu.






```
3   public class SuperHeroControllerTests
4   {
5       private readonly SuperHeroController _superHeroController;
6       private Mock<ISuperHeroService> _superHeroServiceMock = new();
7
8       public SuperHeroControllerTests()
9       {
10          _superHeroController = new SuperHeroController(_superHeroServiceMock.Object);
11      }
12
```

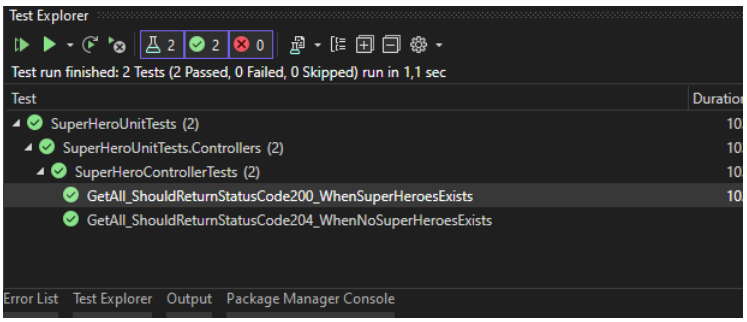
```
13      [Fact]
14      public async void GetAll_ShouldReturnStatusCode200_WhenSuperHeroesExists()
15      {
16          // Arrange
17          List<SuperHeroResponse> superHeroes = new();
18
19          superHeroes.Add(new SuperHeroResponse
20          {
21              Id = 1,
22              Name = "Superman",
23              FirstName = "Clark",
24              LastName = "Kent",
25              Place = "Metropolis",
26              DebutYear = 1938
27          });
28
29          _superHeroServiceMock
30              .Setup(x => x.GetAll())
31              .ReturnsAsync(superHeroes);
32
33          // Act
34          var result = (IStatusCodeActionResult)await _superHeroController.GetAll();
35
36          // Assert
37          Assert.Equal(200, result.StatusCode);
38      }
```

Vi opretter en liste af "**SuperHeroResponses**", fordi vi sætter servicen til at returnere den liste. De to metoder "**Setup()**" og "**ReturnsAsync()**", er der hvor vi bestemmer hvordan servicen skal fungere i den pågældende test.

GetAll() NoContent() Testen

Kopier hele metoden "**GetAll_ShouldReturnStatusCode200_WhenSuperHeroesExists()**" og indsæt den igen lige neden under, samt omdøb kopien til "**GetAll_ShouldReturnStatusCode204_WhenNoSuperHeroesExists()**".

```
40          [Fact]
41          public async void GetAll_ShouldReturnStatusCode204_WhenNoSuperHeroesExists()
42          {
43              // Arrange
44              List<SuperHeroResponse> superHeroes = new();
45
46              _superHeroServiceMock
47                  .Setup(x => x.GetAll())
48                  .ReturnsAsync(superHeroes);
49
50              // Act
51              var result = (IStatusCodeActionResult)await _superHeroController.GetAll();
52
53              // Assert
54              Assert.Equ l(204,  esult.StatusCode);
55          }
```



Problem() når der kommer null

Kopier "**GetAll_ShouldReturnStatusCode204_WhenNoSuperHeroesExists()**" og indsæt den som før nederst, omdøb den til "**GetAll_ShouldReturnStatusCode500_WhenServiceReturnsNull()**".

Tilpas "**.ReturnsAsync()",** så der står "**() => null**" som argument. Det er en anonym metode der blot returnere null, frem for et "**null argument**" til "**ReturnsAsync()**".

Husk at ændre "**Assert.Equal()**" så det er tallet "**500**" der forventes

```
57          [Fact]
58          public async void GetAll_ShouldReturnStatusCode500_WhenServiceReturnsNull()
59          {
60              // Arrange
61              _superHeroServiceMock
62                  .Setup(x => x.GetAll())
63                  .ReturnsAsync(() => null);
64
65              // Act
66              var result = (IStatusCodeActionResult)await _superHeroController.GetAll();
67
68              // Assert
69              Assert.Equal(500, result.StatusCode);
70          }
```


Problem() når der sker fejl

Kopier "**GetAll_ShouldReturnStatusCode500_WhenServiceReturnsNull()**" og indsæt den som før nederst, omdøb den til "**GetAll_ShouldReturnStatusCode500_WhenExceptionIsRaised()**".

Tilpas "**.ReturnsAsync()**", så der står "**() => throw new Exception("This is an exception")**" som argument. Husk at "**Assert.Equal()**" stadig skal kigge efter "**500**".

```
72          [Fact]
73          public async void GetAll_ShouldReturnStatusCode500_WhenExceptionIsRaised()
74          {
75              // Arrange
76              _superHeroServiceMock
77                   .Setup(x => x.GetAll())
78                   .ReturnsAsync(() => throw new Exception("This is an Exception"));
79
80              // Act
81              var result = (IStatusCodeActionResult)await _superHeroController.GetAll();
82
83              // Assert
84              Assert.Equal(500, result.StatusCode);
85          }
```

Når de 4 tests består, gemmes alt, og der commites til Github!

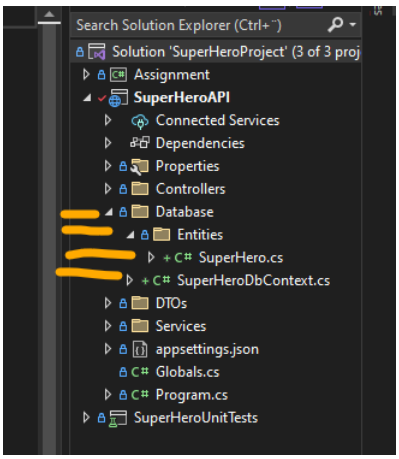
SUPERHERO REPOSITORY

Nu er vi klar til at introducere en Database, så vores data ikke blot skal være statiske lister.

Her vil vi stadig have muligheden for at teste om "**Repository**" laget gør det vi forventer, så der skal også oprettes tests til dette, og igen er det smart med et interface.

Vi kommer til at benytte EntityFramework ud fra et "**CodeFirst**" princip.

Via NuGet Packet Manager, installer "**Microsoft.EntityFrameworkCore**" *(6.x.x)*, "**Microsoft.EntityFrameworkCore.Tools**" *(6.x.x)*, "**Microsoft.EntityFrameworkCore.Design**" *(6.x.x)* og "**Microsoft.EntityFrameworkCore.SqlServer**" *(6.x.x)*



Opret en mappe i roden af "**SuperHeroAPI**" projektet, kald mappen "**Database**". Derefter opret en mappe kaldet "**Entities**" inde i den nye "**Database**" mappe.

Opret en class kaldet "**SuperHero.cs**" i "**Entities**" mappen, og opret en class kaldet "**SuperHeroDbContext.cs**" i "**Database**" mappen.

SuperHero Entity

Opsæt "/Database/Entites/SuperHero.cs" med de egenskaber en superhelt skal bestå af.

Sørg for at pege på de nødvendige namespaces. Du kan placere dem i din Globals.cs, eller her i filen.

[Key] fortæller "EntityFramwork" at "Id" egenskaben skal være den "primære nøgle". Gør det let for dig selv og kald den for "Id" frem for AuthorId eller andre variationer på "Id". Det er desværre et tradeoff vi nå acceptere når det er EntityFramework der arbejdes med.

[Column()] benyttes til at fortælle hvilke "SQL datatyper" hver egenskab skal oprettes med.

```cs
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SuperHeroAPI.Database.Entities
{
    public class SuperHero
    {
        [Key]
        public int Id { get; set; }

        [Column(TypeName = "nvarchar(32)")]
        public string Name { get; set; } = string.Empty;

        [Column(TypeName = "nvarchar(32)")]
        public string FirstName { get; set; } = string.Empty;

        [Column(TypeName = "nvarchar(32)")]
        public string LastName { get; set; } = string.Empty;

        [Column(TypeName = "nvarchar(32)")]
        public string Place { get; set; } = string.Empty;

        [Column(TypeName = "smallint")]
        public short DebutYear { get; set; } = 0;
    }
}
```

```cs
namespace SuperHeroAPI.Database
{
    public class SuperHeroDbContext : DbContext
    {
        public SuperHeroDbContext(DbContextOptions<SuperHeroDbContext> options) : base(options) { }

        public DbSet<SuperHero> SuperHero { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<SuperHero>().HasData(
                new SuperHero
                {
                    Id = 1,
                    Name = "Superman",
                    FirstName = "Clark",
                    LastName = "Kent",
                    Place = "Metropolis",
                    DebutYear = 1938
                },
                new SuperHero
                {
                    Id = 2,
                    Name = "Iron Man",
                    FirstName = "Tony",
                    LastName = "Stark",
                    Place = "Malibu",
                    DebutYear = 1963
                });
        }
    }
}
```

CONNECTIONSTRING

I dette projekt vælger jeg at benytte en "MSSQL LocalDb", for simplicitetens skyld.

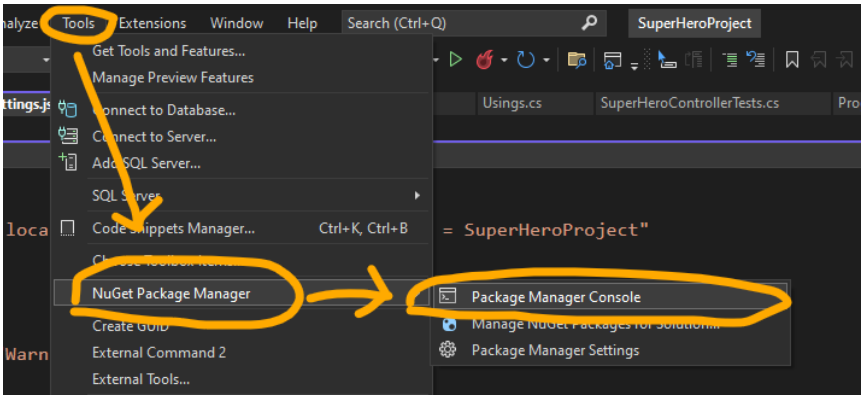
Vi skal have en "ConnectionString" til den database vi ønsker at arbejde med. Denne tilføjes i filen "appsettings.json" i roden af SuperHeroAPI projektet.

```
{
  "ConnectionStrings": {
    "Default": "Data Source = (localdb)\\MSSQLLocalDB; Initial Catalog = SuperHeroProject"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

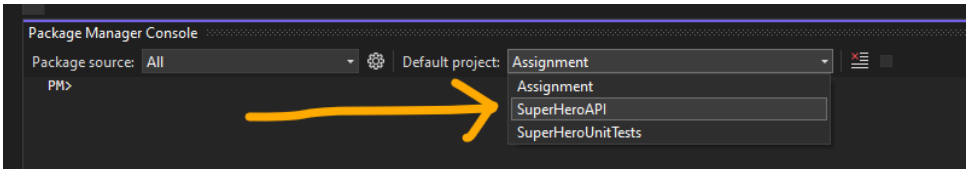
Åben "**Program.cs**" og tilføj forbindelsen til databasen:

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddTransient<ISuperHeroService, SuperHeroService>();

builder.Services.AddDbContext<SuperHeroDbContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("Default"));
});

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

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

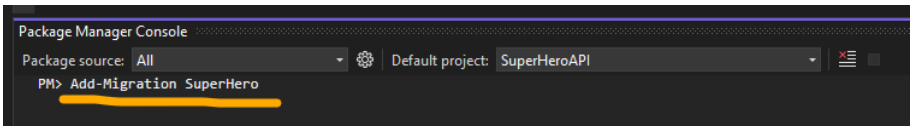
ADD-MIGRATION

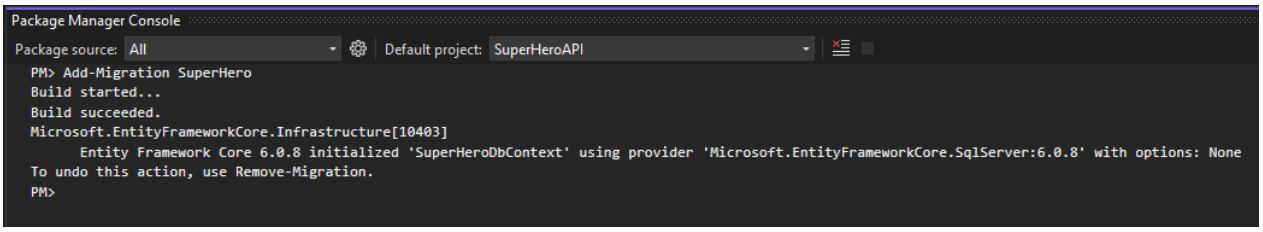


Her er det meget vigtigt at vælge "**SuperHeroAPI**" som "**Default Project**", ellers fejler det:

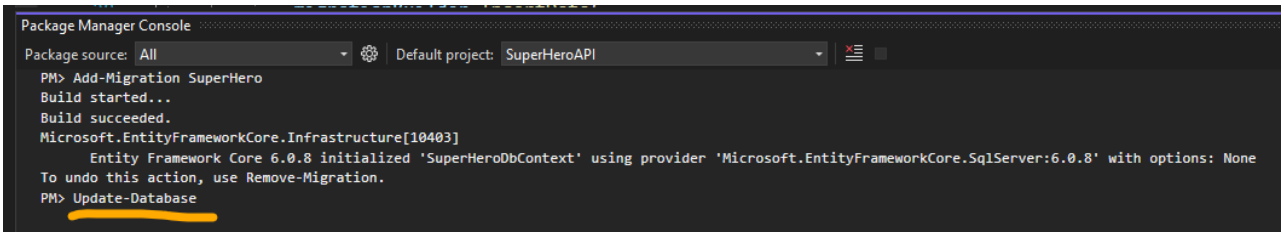


Derefter udføres kommandoen "**Add-Migration SuperHero**"

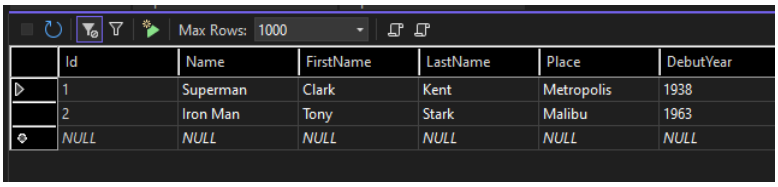


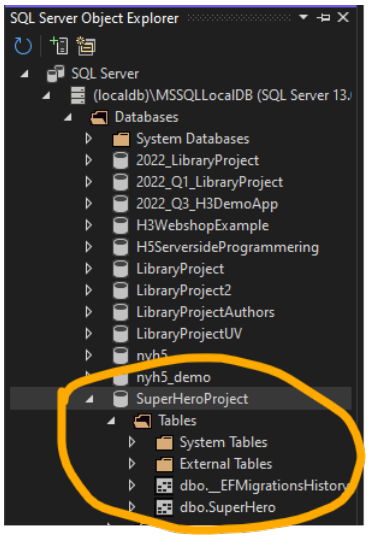


Derefter køres kommandoen "**Update-Database**" det er her selve databasen ændres, og der skulle gerne køres en masse SQL i konsollen.



Højreklik på "**dbo.SuperHero**" og vælg "**View Data**", så burde du se de to default helte er automatisk indsat.





SUPERHEROREPOSITORY OPSÆTNING

Opret en ny mappe i "**SuperHeroAPI**" kaldet "**Repositories**" og opret en class kaldet "**SuperHeroRepository.cs**" i den.

Lige som ved "**SuperHeroService**", har vi brug for et interface, så vi kan injecte efter behov, og vi kan benytte "**moq**" i testfasen.

```
1  namespace SuperHeroAPI.Repositories
2  {
3      public interface ISuperHeroRepository
4      {
5          Task<List<SuperHero>> GetAll();
6      }
```


Derefter implementeres interfacet i klassen "**SuperHeroRepository**", som også får "**SuperHeroDbContext**" serveret i en constructor.

```
8   public class SuperHeroRepository : ISuperHeroRepository
9   {
10      private readonly SuperHeroDbContext _context;
11
12      public SuperHeroRepository(SuperHeroDbContext context)
13      {
14          _context = context;
15      }
16
17      public async Task<List<SuperHero>> GetAll()
18      {
19          return await _context.SuperHero.ToListAsync();
20      }
21  }
```

FORBIND REPOSITORY OG SERVICE

I "**Program.cs**" skal vi huske at tilføje "**ISuperHeroRepository**" så Dependency Injection kan servere den korrekte klasse.

```
1   var builder = WebApplication.CreateBuilder(args);
2
3   // Add services to the container.
4   builder.Services.AddTransient<ISuperHeroService, SuperHeroService>();
5   builder.Services.AddTransient<ISuperHeroRepository, SuperHeroRepository>();
6
7   builder.Services.AddDbContext<SuperHeroDbContext>(options =>
8   {
9       options.UseSqlServer(builder.Configuration.GetConnectionString("Default"));
10  });
```

```
8   public class SuperHeroService : ISuperHeroService
9   {
10      private readonly ISuperHeroRepository _superHeroRepository;
11
12      public SuperHeroService(ISuperHeroRepository superHeroRepository)
13      {
14          _superHeroRepository = superHeroRepository;
15      }
16
```

Samt vi er nødt til at omskrive "**SuperHeroService.GetAll()**", så den benytter "**_superHeroRepository.GetAll()**" til at hente alle helte. Den nye implementering ser sådan her ud:

```
17      public async Task<List<SuperHeroResponse>> GetAll()
18      {
19          List<SuperHero> superHeroes = await _superHeroRepository.GetAll();
20
21          if (superHeroes != null)
22          {
23              return superHeroes.Select(superHero => new SuperHeroResponse
24              {
25                  Id = superHero.Id,
26                  FirstName = superHero.FirstName,
27                  LastName = superHero.LastName,
28                  Name = superHero.Name,
29                  DebutYear = superHero.DebutYear,
30                  Place = superHero.Place
31              }).ToList();
32          }
33
34          return null;
35      }
```

Og nu kan vi i swagger teste at der hentes de superhelte ud, som vi har liggende i databasen.

SUPERHEROREPOSITORY TESTS

For at teste "**SuperHeroRepository**" som et enkelt unit, er vi nødt til at abstrahere databasen ud, så det ikke er den faktiske database der arbejdes på, da det hurtigt bliver noget rod. I stedet benytter vi en "**InMemoryDatabase**" som fungerer præcis som den normale database, dog er den meget hurtigere og den bliver smidt ud hver gang en test er kørt.

Installer NuGet pakken "**Microsoft.EntityFrameworkCore.InMemory**" *(6.x.x)* på "**SuperHeroUnitTests**" projektet.

Derefter oprettes en mappe i "**SuperHeroUnitTests**" Projektet, kald mappen "**Repositories**" og opret en ny class kaldet "**SuperHeroRepositoryTests.cs**" i den nye mappe.Skriv følgende kode i "**SuperHeroRepositoryTests.cs**" husk at tilføje nødvendige usings i "**Usings.cs**"

```
1  namespace SuperHeroUnitTests.Repositories
2  {
3      public class SuperHeroRepositoryTests
4      {
5          private readonly DbContextOptions<SuperHeroDbContext> _options;
6          private readonly SuperHeroDbContext _context;
7          private readonly SuperHeroRepository _superHeroRepository;
8
9          public SuperHeroRepositoryTests()
10         {
11             _options = new DbContextOptionsBuilder<SuperHeroDbContext>()
12                 .UseInMemoryDatabase(databaseName: "SuperHeroRepository")
13                 .Options;
14
15             _context = new(_options);
16
17             _superHeroRepository = new(_context);
18         }
19     }
20 }
```

Med den opsætning, kan vi udføre alle de tests vi ønsker, uden at vi piller ved den faktiske database. Værdien der sættes på "**databaseName**" er vigtig… sørg for den er **unik**, dvs når du skal oprette andre repository test klasser (f.eks. teams!) så skal databasen i den klasse hedde noget andet end "**SuperHeroRepository**".

SuperHeroRepository Test GetAll()

Den første test vi sætter op, er testen der demonstrer vi får en liste af SuperHeroes ud, når der er data i databasen.

Da vi arbejder med asynkronitet, kan vi risikere en anden test har indsat elementer i databasen, og derfor sørger vi for at starte alle tests med at slette "**InMemoryDatabasen**". De to SuperHero Add's er kollapset i dette screenshot, se næste for detaljerne.

```
20         [Fact]
21         public async void GetAll_ShouldReturnListOfSuperHeroes_WhenSuperHeroesExists()
22         {
23             // Arrange
24             await _context.Database.EnsureDeletedAsync();
25
26             _context.SuperHero.Add(new()...);
35
36             _context.SuperHero.Add(new()...);
45
46             await _context.SaveChangesAsync();
47
48             // Act
49             var result = await _superHeroRepository.GetAll();
50
51             // Assert
52             Assert.NotNull(result);
53             Assert.IsType<List<SuperHero>>(result);
54             Assert.Equal(2, result.Count);
55         }
```

collapsed!

Her er de to SuperHeroes foldet ud. Det er de samme data som står i "**SuperHeroDbContext**" under "**OnModelCreating**"

```
26          _context.SuperHero.Add(new()
27          {
28              Id = 1,
29              Name = "Superman",
30              FirstName = "Clark",
31              LastName = "Kent",
32              Place = "Metropolis",
33              DebutYear = 1938
34          });
35
36          _context.SuperHero.Add(new()
37          {
38              Id = 2,
39              Name = "Iron Man",
40              FirstName = "Tony",
41              LastName = "Stark",
42              Place = "Malibu",
43              DebutYear = 1963
44          });
```

Kør testen, og se at alt meget gerne burde bestå uden udfordringer.

GetAll() returnerer tom liste

Systemet bør også kunne bekræfte at "**SuperHeroRepository.GetAll()**" kan returnere en tom liste, hvis databasen er tom.

```
58          [Fact]
59          public async void GetAll_ShouldReturnEmptyListOfSuperHeroes_WhenNoSuperHeroesExists()
60          {
61              // Arrange
62              await _context.Database.EnsureDeletedAsync();
63
64              // Act
65              var result = await _superHeroRepository.GetAll();
66
67              // Assert
68              Assert.NotNull(result);
69              Assert.IsType<List<SuperHero>>(result);
70              Assert.Empty(result);
71          }
```

SUPERHEROREPOSITORY DEN FULDE CRUD

"**ISuperHeroRepository**" mangler at beskrive hvordan resten af "**CRUD**" metoderne ser ud, så dem skal vi tilføje nu:

```
3   public interface ISuperHeroRepository
4   {
5       Task<List<SuperHero>> GetAll();
6       Task<SuperHero> GetById(int superHeroId);
7       Task<SuperHero> Create(SuperHero newSuperHero);
8       Task<SuperHero> Update(int superHeroId, SuperHero updateSuperHero);
9       Task<SuperHero> Delete(int superHeroId);
10  }
```

Nu vil "**SuperHeroRepository**" brokke sig, da klassen ikke implementerer de nye metoder endnu, det klares med et klik på den lille pære, eller **[CTRL]** + **[.]** og vælg "**Implement Interface**".

```
12  public class SuperHeroRepository : ISuperHeroRepository
13  {
14      private readonly SuperHeroDbConte
15
16      public SuperHeroRepository(SuperH
17      {
18          _context = context;
19      }
20
21      public async Task<List<SuperHero>> GetAll()
22      {
```

interface SuperHeroAPI.Repositories.ISuperHeroRepository

CS0535: 'SuperHeroRepository' does not implement interface member 'ISuperHeroRepository.GetById(int)'

CS0535: 'SuperHeroRepository' does not implement interface member 'ISuperHeroRepository.Create(SuperHero)'

CS0535: 'SuperHeroRepository' does not implement interface member 'ISuperHeroRepository.Update(int, SuperHero)'

CS0535: 'SuperHeroRepository' does not implement interface member 'ISuperHeroRepository.Delete(int)'

Show potential fixes (Alt+Enter or Ctrl+.)

Så oprettes alle de nødvendige metoder, dog uden den faktiske kode… Her kommer de 5 CRUD metoder i deres fulde ordlyd:

```
21          public async Task<List<SuperHero>> GetAll()
22          {
23              return await _context.SuperHero.ToListAsync();
24          }
```

```
26          public async Task<SuperHero> GetById(int superHeroId)
27          {
28              return await _context.SuperHero.FirstOrDefaultAsync(s => s.Id == superHeroId);
29          }
```

```
31          public async Task<SuperHero> Create(SuperHero newSuperHero)
32          {
33              _context.SuperHero.Add(newSuperHero);
34              await _context.SaveChangesAsync();
35              return newSuperHero;
36          }
```

```
38          public async Task<SuperHero> Update(int superHeroId, SuperHero updateSuperHero)
39          {
40              SuperHero superHero = await GetById(superHeroId);
41              if (superHero != null)
42              {
43                  superHero.Name = updateSuperHero.Name;
44                  superHero.FirstName = updateSuperHero.FirstName;
45                  superHero.LastName = updateSuperHero.LastName;
46                  superHero.Place = updateSuperHero.Place;
47                  superHero.DebutYear = updateSuperHero.DebutYear;
48
49                  await _context.SaveChangesAsync();
50              }
51              return superHero;
52          }
```

```
54          public async Task<SuperHero> Delete(int superHeroId)
55          {
56              SuperHero superHero = await GetById(superHeroId);
57              if (superHero != null)
58              {
59                  _context.SuperHero.Remove(superHero);
60                  await _context.SaveChangesAsync();
61              }
62              return superHero;
63          }
```


ALLE SUPERHEROREPOSITORYTESTS

Den komplette samling

GetAll_ShouldReturnListOfSuperHeroes_WhenSuperHeroesExists()

```
20      [Fact]
21      public async void GetAll_ShouldReturnListOfSuperHeroes_WhenSuperHeroesExists()
22      {
23          // Arrange
24          await _context.Database.EnsureDeletedAsync();
25
26          _context.SuperHero.Add(new()...);
35
36          _context.SuperHero.Add(new()...);
45
46          await _context.SaveChangesAsync();
47
48          // Act
49          var result = await _superHeroRepository.GetAll();
50
51          // Assert
52          Assert.NotNull(result);
53          Assert.IsType<List<SuperHero>>(result);
54          Assert.Equal(2, result.Count);
55      }
```

collapsed!

```
26          _context.SuperHero.Add(new()
27          {
28              Id = 1,
29              Name = "Superman",
30              FirstName = "Clark",
31              LastName = "Kent",
32              Place = "Metropolis",
33              DebutYear = 1938
34          });
35
36          _context.SuperHero.Add(new()
37          {
38              Id = 2,
39              Name = "Iron Man",
40              FirstName = "Tony",
41              LastName = "Stark",
42              Place = "Malibu",
43              DebutYear = 1963
44          });
```

```
58      [Fact]
59      public async void GetAll_ShouldReturnEmptyListOfSuperHeroes_WhenNoSuperHeroesExists()
60      {
61          // Arrange
62          await _context.Database.EnsureDeletedAsync();
63
64          // Act
65          var result = await _superHeroRepository.GetAll();
66
67          // Assert
68          Assert.NotNull(result);
69          Assert.IsType<List<SuperHero>>(result);
70          Assert.Empty(result);
71      }
```

GetById_ShouldReturnSuperHero_WhenSuperHeroExists()

```
73          [Fact]
74          public async void GetById_ShouldReturnSuperHero_WhenSuperHeroExists()
75          {
76              // Arrange
77              await _context.Database.EnsureDeletedAsync();
78
79              int superHeroId = 1;
80
81              _context.SuperHero.Add(new()
82              {
83                  Id = superHeroId,
84                  Name = "Superman",
85                  FirstName = "Clark",
86                  LastName = "Kent",
87                  Place = "Metropolis",
88                  DebutYear = 1938
89              });
90
91              await _context.SaveChangesAsync();
92
93              // Act
94              var result = await _superHeroRepository.GetById(superHeroId);
95
96              // Assert
97              Assert.NotNull(result);
98              Assert.IsType<SuperHero>(result);
99              Assert.Equal(superHeroId, result.Id);
100         }
```

```
102         [Fact]
103         public async void GetById_ShouldReturnNull_WhenSuperHeroDoesNotExist()
104         {
105             // Arrange
106             await _context.Database.EnsureDeletedAsync();
107
108             // Act
109             var result = await _superHeroRepository.GetById(1);
110
111             // Assert
112             Assert.Null(result);
113         }
```

```
115         [Fact]
116         public async void Create_ShouldAddNewIdToSuperHero_WhenSavingToDatabase()
117         {
118             // Arrange
119             await _context.Database.EnsureDeletedAsync();
120
121             int expectedNewId = 1;
122
123             SuperHero superHero = new()
124             {
125                 Name = "Superman",
126                 FirstName = "Clark",
127                 LastName = "Kent",
128                 Place = "Metropolis",
129                 DebutYear = 1938
130             };
131
132             // Act
133             var result = await _superHeroRepository.Create(superHero);
134
135             // Assert
136             Assert.NotNull(result);
137             Assert.IsType<SuperHero>(result);
138             Assert.Equal(expectedNewId, result.Id);
139         }
```


```
141         [Fact]
142         public async void Create_ShouldFailToAddNewSuperHero_WhenSuperHeroIdAlreadyExists()
143         {
144             // Arrange
145             await _context.Database.EnsureDeletedAsync();
146 
147             SuperHero superHero = new()
148             {
149                 Id = 1,
150                 Name = "Superman",
151                 FirstName = "Clark",
152                 LastName = "Kent",
153                 Place = "Metropolis",
154                 DebutYear = 1938
155             };
156 
157             await _superHeroRepository.Create(superHero);
158 
159             // Act
160             async Task action() => await _superHeroRepository.Create(superHero);
161 
162             // Assert
163             var ex = await Assert.ThrowsAsync<ArgumentException>(action);
164             Assert.Contains("An item with the same key has already been added", ex.Message);
165         }
```

```
167         [Fact]
168         public async void Update_ShouldChangeValuesOnSuperHero_WhenSuperHeroExists()
169         {
170             // Arrange
171             await _context.Database.EnsureDeletedAsync();
172             int superHeroId = 1;
173             SuperHero newSuperHero = new()
174             {
175                 Id = superHeroId,
176                 Name = "Superman",
177                 FirstName = "Clark",
178                 LastName = "Kent",
179                 Place = "Metropolis",
180                 DebutYear = 1938
181             };
182             _context.SuperHero.Add(newSuperHero);
183             await _context.SaveChangesAsync();
184             SuperHero updateSuperHero = new()
185             {
186                 Id = superHeroId,
187                 Name = "new Superman",
188                 FirstName = "new Clark",
189                 LastName = "new Kent",
190                 Place = "new Metropolis",
191                 DebutYear = 1999
192             };
193 
194             // Act
195             var result = await _superHeroRepository.Update(superHeroId, updateSuperHero);
196 
197             // Assert
198             Assert.NotNull(result);
199             Assert.IsType<SuperHero>(result);
200             Assert.Equal(superHeroId, result.Id);
201             Assert.Equal(updateSuperHero.Name, result.Name);
202             Assert.Equal(updateSuperHero.FirstName, result.FirstName);
203             Assert.Equal(updateSuperHero.LastName, result.LastName);
204             Assert.Equal(updateSuperHero.Place, result.Place);
205             Assert.Equal(updateSuperHero.DebutYear, result.DebutYear);
206         }
```


```
208         [Fact]
209         public async void Update_ShouldReturnNull_WhenSuperHeroDoesNotExist()
210         {
211             // Arrange
212             await _context.Database.EnsureDeletedAsync();
213 
214             int superHeroId = 1;
215 
216             SuperHero updateSuperHero = new()
217             {
218                 Id = superHeroId,
219                 Name = "Superman",
220                 FirstName = "Clark",
221                 LastName = "Kent",
222                 Place = "Metropolis",
223                 DebutYear = 1938
224             };
225 
226             // Act
227             var result = await _superHeroRepository.Update(superHeroId, updateSuperHero);
228 
229             // Assert
230             Assert.Null(result);
231         }
```

```
234         [Fact]
235         public async void Delete_ShouldReturnDeletedSuperHero_WhenSuperHeroIsDeleted()
236         {
237             // Arrange
238             await _context.Database.EnsureDeletedAsync();
239 
240             int superHeroId = 1;
241             SuperHero newSuperHero = new()
242             {
243                 Id = superHeroId,
244                 Name = "new Superman",
245                 FirstName = "new Clark",
246                 LastName = "new Kent",
247                 Place = "new Metropolis",
248                 DebutYear = 1999
249             };
250 
251             _context.SuperHero.Add(newSuperHero);
252             await _context.SaveChangesAsync();
253 
254             // Act
255             var result = await _superHeroRepository.Delete(superHeroId);
256             var superHero = await _superHeroRepository.GetById(superHeroId);
257 
258             // Assert
259             Assert.NotNull(result);
260             Assert.IsType<SuperHero>(result);
261             Assert.Equal(superHeroId, result.Id);
262 
263             Assert.Null(superHero);
264         }
```

Delete_ShouldReturnNull_WhenSuperHeroDoesNotExist()

```
266         [Fact]
267         public async void Delete_ShouldReturnNull_WhenSuperHeroDoesNotExist()
268         {
269             // Arrange
270             await _context.Database.EnsureDeletedAsync();
271
272             // Act
273             var result = await _superHeroRepository.Delete(1);
274
275             // Assert
276             Assert.Null(result);
277         }
```

Kør alle tests, og de burde bestå uden udfordringer!

SUPERHEROSERVICE DEN FULDE CRUD

Flere DTO's

Opret en ny class i DTO's mappen, kald den for "**SuperHeroRequest.cs**" med følgende indhold:

```
1   using System.ComponentModel.DataAnnotations;
2
3   namespace SuperHeroAPI.DTOs
4   {
5       public class SuperHeroRequest
6       {
7           [Required]
8           [StringLength(32, ErrorMessage = "Name must not contain more than 32 chars")]
9           public string Name { get; set; } = string.Empty;
10
11          [Required]
12          [StringLength(32, ErrorMessage = "FirstName must not contain more than 32 chars")]
13          public string FirstName { get; set; } = string.Empty;
14
15          [Required]
16          [StringLength(32, ErrorMessage = "LastName must not contain more than 32 chars")]
17          public string LastName { get; set; } = string.Empty;
18
19          [Required]
20          [StringLength(32, ErrorMessage = "Place must not contain more than 32 chars")]
21          public string Place { get; set; } = string.Empty;
22
23          [Required]
24          [Range(1900, 2500, ErrorMessage = "DebutYear must be between 1900 and 2500")]
25          public short DebutYear { get; set; } = 1900;
26      }
27  }
```

Denne DTO indeholder nogle "**annotationer**" som beskriver hvordan en valid "**SuperHeroRequest**" ser ud, og .NET vil automatisk generere fejlbeskeder hvis et modtaget element ikke opfylder kriterierne.

ISuperHeroService alle metoderne

Udvid "**ISuperHeroService**" så den indeholder alle CRUD metoderne:

```
3       public interface ISuperHeroService
4       {
5           Task<List<SuperHeroResponse>> GetAll();
6           Task<SuperHeroResponse> GetById(int superHeroId);
7           Task<SuperHeroResponse> Create(SuperHeroRequest newSuperHero);
8           Task<SuperHeroResponse> Update(int superHeroId, SuperHeroRequest updateSuperHero);
9           Task<SuperHeroResponse> Delete(int superHeroId);
10      }
```

Husk at implementer alle metoderne, benyt gerne auto implementation.

AutoMapping mellem DTO og Entity

```
84          private static SuperHero MapSuperHeroRequestToSuperHero(SuperHeroRequest superHero)
85          {
86              return new SuperHero
87              {
88                  Name = superHero.Name,
89                  FirstName = superHero.FirstName,
90                  LastName = superHero.LastName,
91                  Place = superHero.Place,
92                  DebutYear = superHero.DebutYear
93              };
94          }
```

Den anden modtager et "**SuperHero**" objekt, og returnerer et "**SuperHeroResponse**" objekt

```
96          private static SuperHeroResponse MapSuperHeroToSuperHeroResponse(SuperHero superHero)
97          {
98              return new SuperHeroResponse
99              {
100                 Id = superHero.Id,
101                 Name = superHero.Name,
102                 FirstName = superHero.FirstName,
103                 LastName = superHero.LastName,
104                 Place = superHero.Place,
105                 DebutYear = superHero.DebutYear
106             };
107         }
```

```
21          public async Task<List<SuperHeroResponse>> GetAll()
22          {
23              List<SuperHero> superHeroes = await _superHeroRepository.GetAll();
24
25              if (superHeroes != null)
26              {
27                  return superHeroes.Select(superHero => MapSuperHeroToSuperHeroResponse(superHero)).ToList();
28              }
29
30              return null;
31          }
```

```
33          public async Task<SuperHeroResponse> GetById(int superHeroId)
34          {
35              SuperHero superHero = await _superHeroRepository.GetById(superHeroId);
36
37              if (superHero != null)
38              {
39                  return MapSuperHeroToSuperHeroResponse(superHero);
40              }
41
42              return null;
43          }
```

SuperHeroService.Create(SuperHeroRequest superHeroRequest)

```
45          public async Task<SuperHeroResponse> Create(SuperHeroRequest superHeroRequest)
46          {
47              SuperHero superHero = MapSuperHeroRequestToSuperHero(superHeroRequest);
48
49              SuperHero insertedSuperHero = await _superHeroRepository.Create(superHero);
50
51              if (insertedSuperHero != null)
52              {
53                  return MapSuperHeroToSuperHeroResponse(insertedSuperHero);
54              }
55
56              return null;
57          }
```

```
59          public async Task<SuperHeroResponse> Update(int superHeroId, SuperHeroRequest superHeroRequest)
60          {
61              SuperHero superHero = MapSuperHeroRequestToSuperHero(superHeroRequest);
62
63              SuperHero updatedSuperHero = await _superHeroRepository.Update(superHeroId, superHero);
64
65              if (updatedSuperHero != null)
66              {
67                  return MapSuperHeroToSuperHeroResponse(updatedSuperHero);
68              }
69
70              return null;
71          }
```

```
73          public async Task<SuperHeroResponse> Delete(int superHeroId)
74          {
75              SuperHero deletedSuperHero = await _superHeroRepository.Delete(superHeroId);
76
77              if (deletedSuperHero != null)
78              {
79                  return MapSuperHeroToSuperHeroResponse(deletedSuperHero);
80              }
81
82              return null;
83          }
```

SUPERHEROSERVICE TESTS

Opret og konfiguerer SuperHeroServiceTests klassen

Lige som med Repository tests, så opretter vi en mappe kaldet ”**Services**” i ”**SuperHeroUnitTests**” projektet, og i den nye mappe, oprettes en ny klasse kaldet ”**SuperHeroServiceTests.cs**”

```
public class SuperHeroServiceTests
{
    private readonly SuperHeroService _superHeroService;
    private readonly Mock<ISuperHeroRepository> _superHeroRepositoryMock = new();

    public SuperHeroServiceTests()
    {
        _superHeroService = new(_superHeroRepositoryMock.Object);
    }
```


SuperHeroService alle CRUD tests

```
13          [Fact]
14          public async void GetAll_ShouldReturnListOfSuperHeroResponses_WhenSuperHerosExists()
15          {
16              // Arrange
17              List<SuperHero> superHeros = new();
18
19              superHeros.Add(new()
20              {
21                  Id = 1,
22                  Name = "Superman",
23                  FirstName = "Clark",
24                  LastName = "Kent",
25                  Place = "Metropolis",
26                  DebutYear = 1938
27              });
28
29              superHeros.Add(new()
30              {
31                  Id = 2,
32                  Name = "Iron Man",
33                  FirstName = "Tony",
34                  LastName = "Stark",
35                  Place = "Malibu",
36                  DebutYear = 1963
37              });
38
39              _superHeroRepositoryMock
40                  .Setup(x => x.GetAll())
41                  .ReturnsAsync(superHeros);
42
43              // Act
44              var result = await _superHeroService.GetAll();
45
46              // Assert
47              Assert.NotNull(result);
48              Assert.Equal(2, result.Count);
49              Assert.IsType<List<SuperHeroResponse>>(result);
50          }
```

```
52          [Fact]
53          public async void GetAll_ShouldReturnEmptyListOfSuperHeroResponses_WhenNoSuperHerosExists()
54          {
55              // Arrange
56              List<SuperHero> superHeros = new();
57
58              _superHeroRepositoryMock
59                  .Setup(x => x.GetAll())
60                  .ReturnsAsync(superHeros);
61
62              // Act
63              var result = await _superHeroService.GetAll();
64
65              // Assert
66              Assert.NotNull(result);
67              Assert.Empty(result);
68              Assert.IsType<List<SuperHeroResponse>>(result);
69          }
```

GetById_ShouldReturnSuperHeroResponse_WhenSuperHeroExists()

```
71          [Fact]
72          public async void GetById_ShouldReturnSuperHeroResponse_WhenSuperHeroExists()
73          {
74              // Arrange
75              int superHeroId = 1;
76
77              SuperHero superHero = new()
78              {
79                  Id = superHeroId,
80                  Name = "Superman",
81                  FirstName = "Clark",
82                  LastName = "Kent",
83                  Place = "Metropolis",
84                  DebutYear = 1938
85              };
86
87              _superHeroRepositoryMock
88                  .Setup(x => x.GetById(It.IsAny<int>()))
89                  .ReturnsAsync(superHero);
90
91              // Act
92              var result = await _superHeroService.GetById(superHeroId);
93
94              // Assert
95              Assert.NotNull(result);
96              Assert.IsType<SuperHeroResponse>(result);
97              Assert.Equal(superHero.Id, result.Id);
98              Assert.Equal(superHero.FirstName, result.FirstName);
99              Assert.Equal(superHero.LastName, result.LastName);
100             Assert.Equal(superHero.Name, result.Name);
101             Assert.Equal(superHero.DebutYear, result.DebutYear);
102             Assert.Equal(superHero.Place, result.Place);
103         }
```

```
105         [Fact]
106         public async void GetById_ShouldReturnNull_WhenSuperHeroDpesNotExists()
107         {
108             // Arrange
109             int superHeroId = 1;
110
111             _superHeroRepositoryMock
112                 .Setup(x => x.GetById(It.IsAny<int>()))
113                 .ReturnsAsync(() => null);
114
115             // Act
116             var result = await _superHeroService.GetById(superHeroId);
117
118             // Assert
119             Assert.Null(result);
120         }
```

Create_ShouldReturnSuperHeroResponse_WhenCreateIsSuccess()

```
122         [Fact]
123         public async void Create_ShouldReturnSuperHeroResponse_WhenCreateIsSuccess()
124         {
125             // Arrange
126             SuperHeroRequest newSuperHero = new()
127             {
128                 Name = "Superman",
129                 FirstName = "Clark",
130                 LastName = "Kent",
131                 Place = "Metropolis",
132                 DebutYear = 1938
133             };
134             int superHeroId = 1;
135             SuperHero superHero = new()
136             {
137                 Id = superHeroId,
138                 Name = "Superman",
139                 FirstName = "Clark",
140                 LastName = "Kent",
141                 Place = "Metropolis",
142                 DebutYear = 1938
143             };
144
145             _superHeroRepositoryMock
146                 .Setup(x => x.Create(It.IsAny<SuperHero>()))
147                 .ReturnsAsync(superHero);
148
149             // Act
150             var result = await _superHeroService.Create(newSuperHero);
151
152             // Assert
153             Assert.NotNull(result);
154             Assert.IsType<SuperHeroResponse>(result);
155             Assert.Equal(superHero.Id, result.Id);
156             Assert.Equal(superHero.Name, result.Name);
157             Assert.Equal(superHero.FirstName, result.FirstName);
158             Assert.Equal(superHero.LastName, result.LastName);
159             Assert.Equal(superHero.Place, result.Place);
160             Assert.Equal(superHero.DebutYear, result.DebutYear);
161         }
```

```
163         [Fact]
164         public async void Create_ShouldReturnNull_WhenRepositoryReturnsNull()
165         {
166             // Arrange
167             SuperHeroRequest newSuperHero = new()
168             {
169                 FirstName = "Clark",
170                 LastName = "Kent",
171                 Name = "Superman",
172                 DebutYear = 1938,
173                 Place = "Metropolis"
174             };
175
176             _superHeroRepositoryMock
177                 .Setup(x => x.Create(It.IsAny<SuperHero>()))
178                 .ReturnsAsync(() => null);
179
180             // Act
181             var result = await _superHeroService.Create(newSuperHero);
182
183             // Assert
184             Assert.Null(result);
185         }
```

Update_ShouldReturnSuperHeroResponse_WhenUpdateIsSuccess()

```
187         [Fact]
188         public async void Update_ShouldReturnSuperHeroResponse_WhenUpdateIsSuccess()
189         {
190             // NOTICE, we do not test if anything actually changed on the DB,
191             // we only test that the returned values match the submitted values
192             // Arrange
193             SuperHeroRequest superHeroRequest = new()
194             {
195                 Name = "Superman",
196                 FirstName = "Clark",
197                 LastName = "Kent",
198                 Place = "Metropolis",
199                 DebutYear = 1938
200             };
201             int superHeroId = 1;
202             SuperHero superHero = new()
203             {
204                 Id = superHeroId,
205                 Name = "Superman",
206                 FirstName = "Clark",
207                 LastName = "Kent",
208                 Place = "Metropolis",
209                 DebutYear = 1938
210             };
211             _superHeroRepositoryMock
212                 .Setup(x => x.Update(It.IsAny<int>(), It.IsAny<SuperHero>()))
213                 .ReturnsAsync(superHero);
214
215             // Act
216             var result = await _superHeroService.Update(superHeroId, superHeroRequest);
217
218             // Assert
219             Assert.NotNull(result);
220             Assert.IsType<SuperHeroResponse>(result);
221             Assert.Equal(superHero.Id, result.Id);
222             Assert.Equal(superHeroRequest.Name, result.Name);
223             Assert.Equal(superHeroRequest.FirstName, result.FirstName);
224             Assert.Equal(superHeroRequest.LastName, result.LastName);
225             Assert.Equal(superHeroRequest.Place, result.Place);
226             Assert.Equal(superHeroRequest.DebutYear, result.DebutYear);
227         }
```

```
229         [Fact]
230         public async void Update_ShouldReturnNull_WhenSuperHeroDoesNotExists()
231         {
232             // Arrange
233             SuperHeroRequest superHeroRequest = new()
234             {
235                 Name = "Superman",
236                 FirstName = "Clark",
237                 LastName = "Kent",
238                 Place = "Metropolis",
239                 DebutYear = 1938
240             };
241
242             int superHeroId = 1;
243
244             _superHeroRepositoryMock
245                 .Setup(x => x.Update(It.IsAny<int>(), It.IsAny<SuperHero>()))
246                 .ReturnsAsync(() => null);
247
248             // Act
249             var result = await _superHeroService.Update(superHeroId, superHeroRequest);
250
251             // Assert
252             Assert.Null(result);
253         }
```

Delete_ShouldReturnSuperHeroResponse_WhenDeleteIsSuccess()

```
255         [Fact]
256         public async void Delete_ShouldReturnSuperHeroResponse_WhenDeleteIsSuccess()
257         {
258             // Arrange
259             int superHeroId = 1;
260 
261             SuperHero superHero = new()
262             {
263                 Id = superHeroId,
264                 Name = "Superman",
265                 FirstName = "Clark",
266                 LastName = "Kent",
267                 Place = "Metropolis",
268                 DebutYear = 1938
269             };
270 
271             _superHeroRepositoryMock
272                 .Setup(x => x.Delete(It.IsAny<int>()))
273                 .ReturnsAsync(superHero);
274 
275             // Act
276             var result = await _superHeroService.Delete(superHeroId);
277 
278             // Assert
279             Assert.NotNull(result);
280             Assert.IsType<SuperHeroResponse>(result);
281             Assert.Equal(superHero.Id, result.Id);
282         }
283 
```

```
284         [Fact]
285         public async void Delete_ShouldReturnNull_WhenSuperHeroDoesNotExists()
286         {
287             // Arrange
288             int superHeroId = 1;
289 
290             _superHeroRepositoryMock
291                 .Setup(x => x.Delete(It.IsAny<int>()))
292                 .ReturnsAsync(() => null);
293 
294             // Act
295             var result = await _superHeroService.Delete(superHeroId);
296 
297             // Assert
298             Assert.Null(result);
299         }
```

Kør alle tests, og se de alle består!

DEN FULDE SUPERHEROCONTROLLER CRUD

Alle CRUD Metoderne

SuperHeroController.GetAll()

```
17          [HttpGet]
18          public async Task<IActionResult> GetAll()
19          {
20              try
21              {
22                  List<SuperHeroResponse> superHeroes = await _superHeroService.GetAll();
23
24                  if (superHeroes == null)
25                  {
26                      return Problem("Nothing was returned from service, this is unexpected");
27                  }
28
29                  if (superHeroes.Count() == 0)
30                  {
31                      return NoContent();
32                  }
33
34                  return Ok(superHeroes);
35              }
36              catch (Exception ex)
37              {
38                  return Problem(ex.Message);
39              }
40          }
```

```
42          [HttpGet]
43          [Route("{superHeroId}")]
44          public async Task<IActionResult> GetById(int superHeroId)
45          {
46              try
47              {
48                  SuperHeroResponse superHeroResponse = await _superHeroService.GetById(superHeroId);
49
50                  if (superHeroResponse == null)
51                  {
52                      return NotFound();
53                  }
54
55                  return Ok(superHeroResponse);
56              }
57              catch (Exception ex)
58              {
59                  return Problem(ex.Message);
60              }
61          }
```

```
63          [HttpPost]
64          public async Task<IActionResult> Create(SuperHeroRequest superHeroRequest)
65          {
66              try
67              {
68                  SuperHeroResponse superHeroResponse = await _superHeroService.Create(superHeroRequest);
69
70                  if (superHeroResponse == null)
71                  {
72                      return Problem("SuperHero was not created, something failed...");
73                  }
74
75                  return Ok(superHeroResponse);
76              }
77              catch (Exception ex)
78              {
79                  return Problem(ex.Message);
80              }
81          }
```


```
83          [HttpPut]
84          [Route("{superHeroId}")]
85          public async Task<IActionResult> Update([FromRoute] int superHeroId, SuperHeroRequest superHeroRequest)
86          {
87              try
88              {
89                  SuperHeroResponse superHeroResponse = await _superHeroService.Update(superHeroId, superHeroRequest);
90
91                  if (superHeroResponse == null)
92                  {
93                      return NotFound();
94                  }
95
96                  return Ok(superHeroResponse);
97              }
98              catch (Exception ex)
99              {
100                 return Problem(ex.Message);
101             }
102         }
```

```
104         [HttpDelete]
105         [Route("{superHeroId}")]
106         public async Task<IActionResult> Delete([FromRoute] int superHeroId)
107         {
108             try
109             {
110                 SuperHeroResponse superHeroResponse = await _superHeroService.Delete(superHeroId);
111
112                 if (superHeroResponse == null)
113                 {
114                     return NotFound();
115                 }
116
117                 return Ok(superHeroResponse);
118             }
119             catch (Exception ex)
120             {
121                 return Problem(ex.Message);
122             }
123         }
```

ALLE SUPERHEROCONTROLLER TESTS

GetAll()

Vi har allerede de 4 tests fra tidligere, dem er der ikke ændret på.

GetById_ShouldReturnStatusCode200_WhenDataExists()

```
87          [Fact]
88          public async void GetById_ShouldReturnStatusCode200_WhenDataExists()
89          {
90              // Arrange
91              int superHeroId = 1;
92
93              SuperHeroResponse SuperHero = new()
94              {
95                  Id = superHeroId,
96                  FirstName = "Clark",
97                  LastName = "Kent",
98                  Name = "Superman",
99                  DebutYear = 1938,
100                 Place = "Metropolis"
101             };
102
103             _superHeroServiceMock
104                 .Setup(x => x.GetById(It.IsAny<int>()))
105                 .ReturnsAsync(SuperHero);
106
107             // Act
108             var result = (IStatusCodeActionResult)await _superHeroController.GetById(superHeroId);
109
110             // Assert
111             Assert.Equal(200, result.StatusCode);
112         }
```

```
114         [Fact]
115         public async void GetById_ShouldReturnStatusCode404_WhenSuperHeroDoesNotExists()
116         {
117             // Arrange
118             int superHeroId = 1;
119
120             _superHeroServiceMock
121                 .Setup(x => x.GetById(It.IsAny<int>()))
122                 .ReturnsAsync(() => null);
123
124             // Act
125             var result = (IStatusCodeActionResult)await _superHeroController.GetById(superHeroId);
126
127             // Assert
128             Assert.Equal(404, result.StatusCode);
129         }
```

```
131         [Fact]
132         public async void GetById_ShouldReturnStatusCode500_WhenExceptionIsRaised()
133         {
134             // Arrange
135             _superHeroServiceMock
136                 .Setup(x => x.GetById(It.IsAny<int>()))
137                 .ReturnsAsync(() => throw new Exception("This is an exception"));
138
139             // Act
140             var result = (IStatusCodeActionResult)await _superHeroController.GetById(1);
141
142             // Assert
143             Assert.Equal(500, result.StatusCode);
144         }
```

Create_ShouldReturnStatusCode200_WhenSuperHeroIsSuccessfullyCreated()

```
146         [Fact]
147         public async void Create_ShouldReturnStatusCode200_WhenSuperHeroIsSuccessfullyCreated()
148         {
149             // Arrange
150             SuperHeroRequest newSuperHero = new()
151             {
152                 FirstName = "Clark",
153                 LastName = "Kent",
154                 Name = "Superman",
155                 DebutYear = 1938,
156                 Place = "Metropolis"
157             };
158 
159             int superHeroId = 1;
160 
161             SuperHeroResponse superHeroResponse = new()
162             {
163                 Id = superHeroId,
164                 FirstName = "Clark",
165                 LastName = "Kent",
166                 Name = "Superman",
167                 DebutYear = 1938,
168                 Place = "Metropolis"
169             };
170 
171             _superHeroServiceMock
172                 .Setup(x => x.Create(It.IsAny<SuperHeroRequest>()))
173                 .ReturnsAsync(superHeroResponse);
174 
175             // Act
176             var result = (IStatusCodeActionResult)await _superHeroController.Create(newSuperHero);
177 
178             // Assert
179             Assert.Equal(200, result.StatusCode);
180         }
```

```
182         [Fact]
183         public async void Create_ShouldReturnStatusCode500_WhenExceptionIsRaised()
184         {
185             // Arrange
186             SuperHeroRequest newSuperHero = new()
187             {
188                 FirstName = "Clark",
189                 LastName = "Kent",
190                 Name = "Superman",
191                 DebutYear = 1938,
192                 Place = "Metropolis"
193             };
194 
195             _superHeroServiceMock
196                 .Setup(x => x.Create(It.IsAny<SuperHeroRequest>()))
197                 .ReturnsAsync(() => throw new Exception("This is an exception"));
198 
199             // Act
200             var result = (IStatusCodeActionResult)await _superHeroController.Create(newSuperHero);
201 
202             // Assert
203             Assert.Equal(500, result.StatusCode);
204         }
```

Update_ShouldReturnStatusCode200_WhenSuperHeroIsSuccessfullyUpdated()

```
206         [Fact]
207         public async void Update_ShouldReturnStatusCode200_WhenSuperHeroIsSuccessfullyUpdated()
208         {
209             // Arrange
210             SuperHeroRequest updateSuperHero = new()
211             {
212                 FirstName = "Clark",
213                 LastName = "Kent",
214                 Name = "Superman",
215                 DebutYear = 1938,
216                 Place = "Metropolis"
217             };
218
219             int superHeroId = 1;
220
221             SuperHeroResponse superHeroResponse = new()
222             {
223                 Id = superHeroId,
224                 FirstName = "Clark",
225                 LastName = "Kent",
226                 Name = "Superman",
227                 DebutYear = 1938,
228                 Place = "Metropolis"
229             };
230
231             _superHeroServiceMock
232                 .Setup(x => x.Update(It.IsAny<int>(), It.IsAny<SuperHeroRequest>()))
233                 .ReturnsAsync(superHeroResponse);
234
235             // Act
236             var result = (IStatusCodeActionResult)await _superHeroController.Update(superHeroId, updateSuperHero);
237
238             // Assert
239             Assert.Equal(200, result.StatusCode);
240         }
```

```
242         [Fact]
243         public async void Update_ShouldReturnStatusCode404_WhenSuperHeroDoesNotExists()
244         {
245             // Arrange
246             SuperHeroRequest updateSuperHero = new()
247             {
248                 FirstName = "Clark",
249                 LastName = "Kent",
250                 Name = "Superman",
251                 DebutYear = 1938,
252                 Place = "Metropolis"
253             };
254
255             int superHeroId = 1;
256
257             _superHeroServiceMock
258                 .Setup(x => x.Update(It.IsAny<int>(), It.IsAny<SuperHeroRequest>()))
259                 .ReturnsAsync(() => null);
260
261             // Act
262             var result = (IStatusCodeActionResult)await _superHeroController.Update(superHeroId, updateSuperHero);
263
264             // Assert
265             Assert.Equal(404, result.StatusCode);
266         }
```

Update_ShouldReturnStatusCode500_WhenExceptionIsRaised()

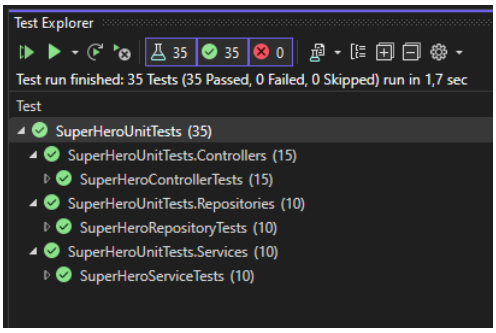
```
268         [Fact]
269         public async void Update_ShouldReturnStatusCode500_WhenExceptionIsRaised()
270         {
271             // Arrange
272             SuperHeroRequest updateSuperHero = new()
273             {
274                 FirstName = "Clark",
275                 LastName = "Kent",
276                 Name = "Superman",
277                 DebutYear = 1938,
278                 Place = "Metropolis"
279             };
280 
281             int superHeroId = 1;
282 
283             _superHeroServiceMock
284                 .Setup(x => x.Update(It.IsAny<int>(), It.IsAny<SuperHeroRequest>()))
285                 .ReturnsAsync(() => throw new Exception("This is an exception"));
286 
287             // Act
288             var result = (IStatusCodeActionResult)await _superHeroController.Update(superHeroId, updateSuperHero);
289 
290             // Assert
291             Assert.Equal(500, result.StatusCode);
292         }
```

```
294         [Fact]
295         public async void Delete_ShouldReturnStatusCode200_WhenSuperHeroIsDeleted()
296         {
297             // Arrange
298             int superHeroId = 1;
299 
300             SuperHeroResponse superHeroResponse = new()
301             {
302                 Id = superHeroId,
303                 FirstName = "Clark",
304                 LastName = "Kent",
305                 Name = "Superman",
306                 DebutYear = 1938,
307                 Place = "Metropolis"
308             };
309 
310             _superHeroServiceMock
311                 .Setup(x => x.Delete(It.IsAny<int>()))
312                 .ReturnsAsync(superHeroResponse);
313 
314             // Act
315             var result = (IStatusCodeActionResult)await _superHeroController.Delete(superHeroId);
316 
317             // Assert
318             Assert.Equal(200, result.StatusCode);
319         }
```

```
321         [Fact]
322         public async void Delete_ShouldReturnStatusCode404_WhenSuperHeroDoesNotExists()
323         {
324             // Arrange
325             int superHeroId = 1;
326             _superHeroServiceMock
327                 .Setup(x => x.Delete(It.IsAny<int>()))
328                 .ReturnsAsync(() => null);
329 
330             // Act
331             var result = (IStatusCodeActionResult)await _superHeroController.Delete(superHeroId);
332 
333             // Assert
334             Assert.Equal(404, result.StatusCode);
335         }
```


```
337         [Fact]
338         public async void Delete_ShouldReturnStatusCode500_WhenExceptionIsRaised()
339         {
340             // Arrange
341             int superHeroId = 1;
342
343             _superHeroServiceMock
344                 .Setup(x => x.Delete(It.IsAny<int>()))
345                 .ReturnsAsync(() => throw new Exception("This is an exception"));
346
347             // Act
348             var result = (IStatusCodeActionResult)await _superHeroController.Delete(superHeroId);
349
350             // Assert
351             Assert.Equal(500, result.StatusCode);
352         }
```

GENNEMFØR ALLE TESTS

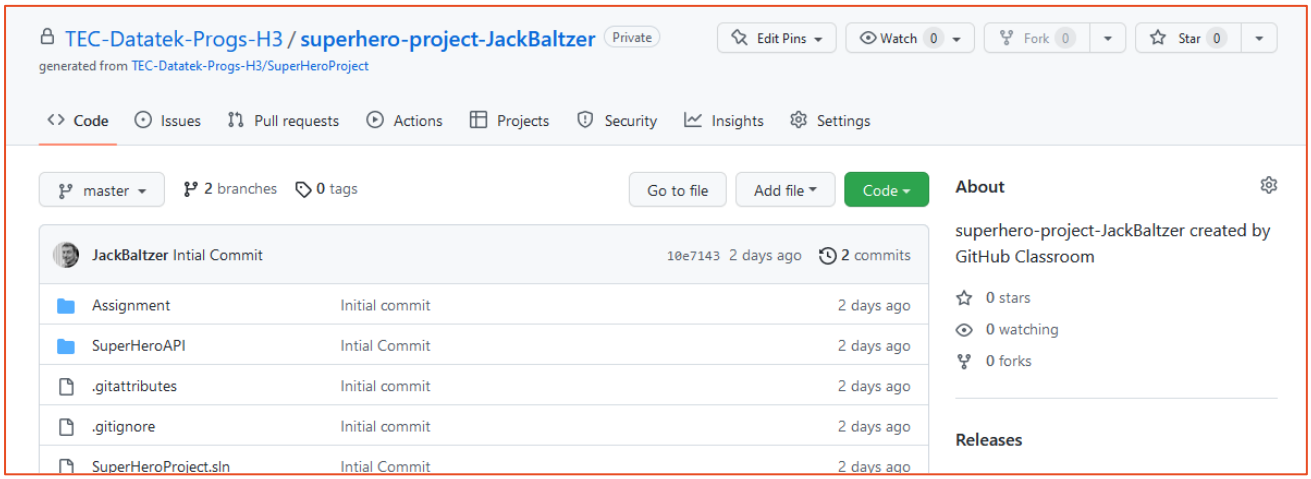


Gem alt og commit til Github!

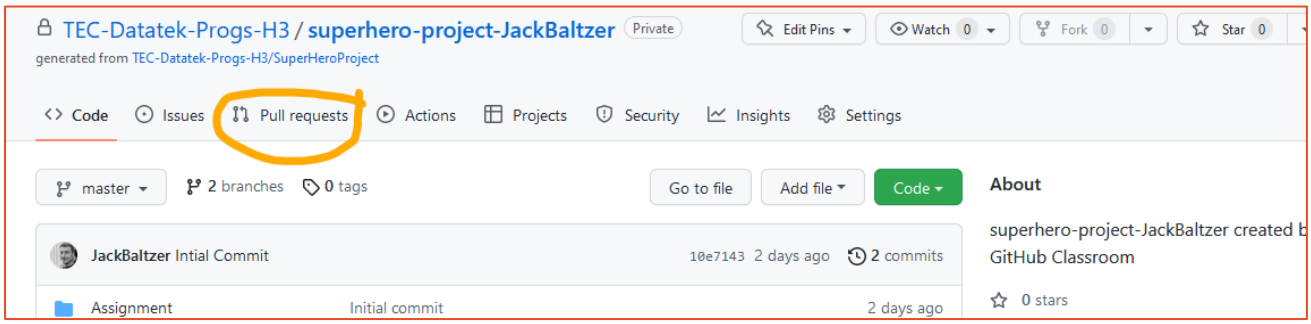
PULL REQUESTS

Da målet med vores API er opnået nu, kan vi flette "**dev branch**" ind i vores "**master branch**".

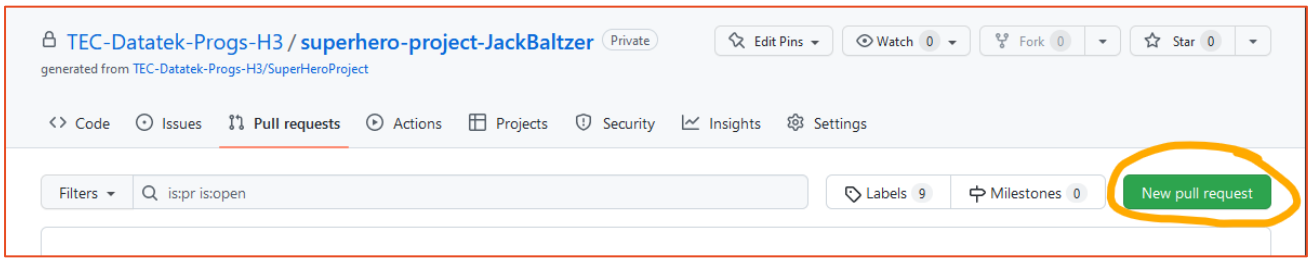
Her viser jeg hvordan man laver en "**pull-request**" igennem github hjemmesiden. En pull-request er en metode til at samle to branches oven i en af dem, hvor man kan gennemgå alle de konflikter der kan opstå når to versioner af en fil skal samles. Det lyder nok mere kompliceret end det faktisk er, så lad os se et eksempel. Her er masterbrancen på github, den er relativt tom, prøv at åbne SuperHeroAPI og se der kun ligger en Program.cs og lidt konfigations filer.



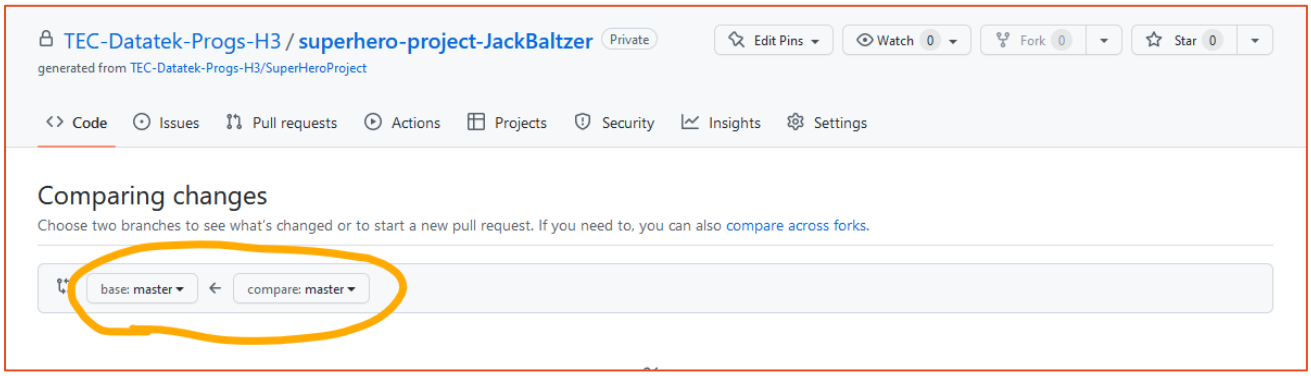
Klik på "**Pull requests**" for at komme i gang med den.



Og derefter klik på den grønne knap "**New pull request**"

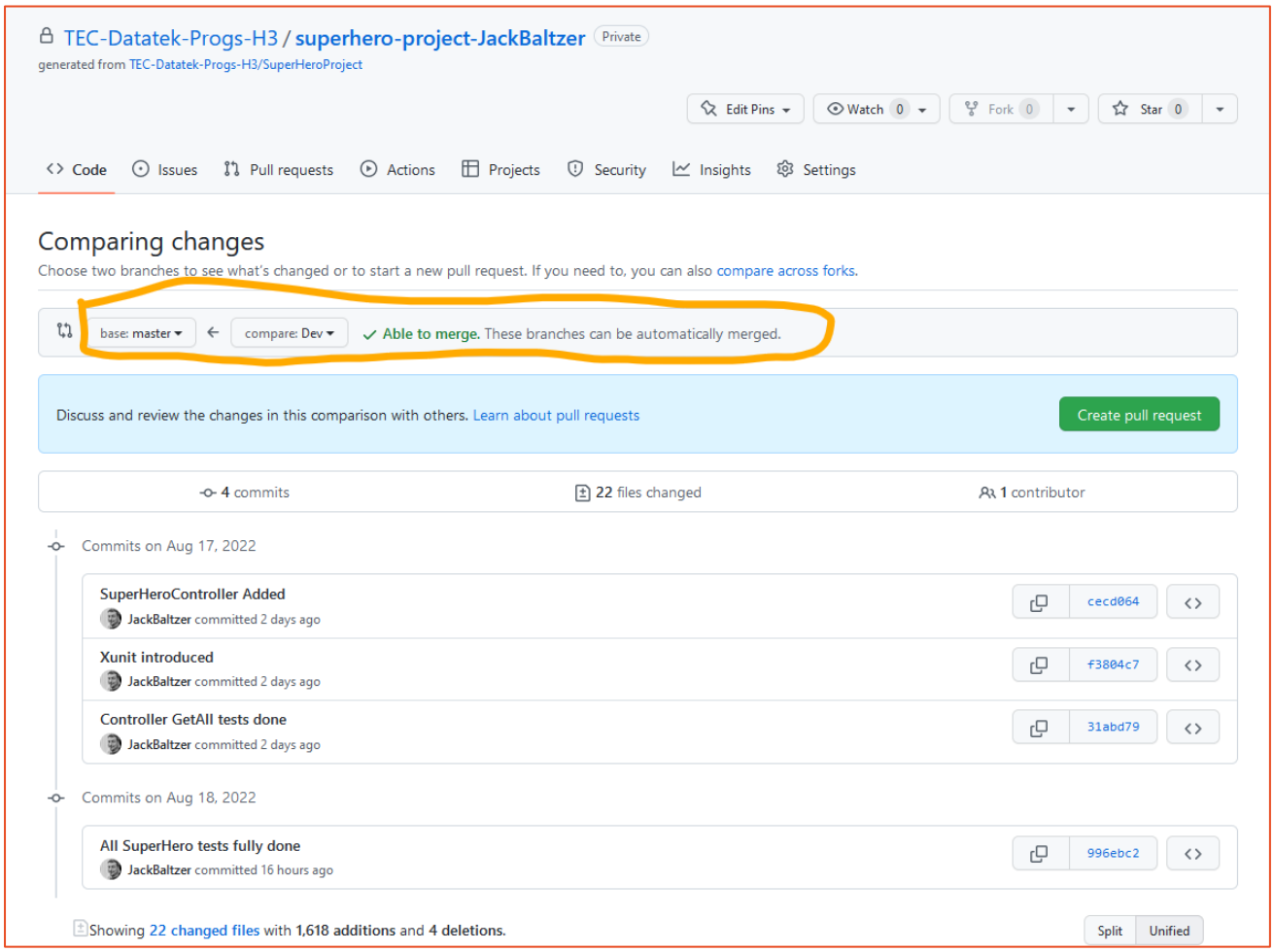


Sørg for at tilpasse de to bokse, så der står "**base:master**" <- "**compare: Dev**"

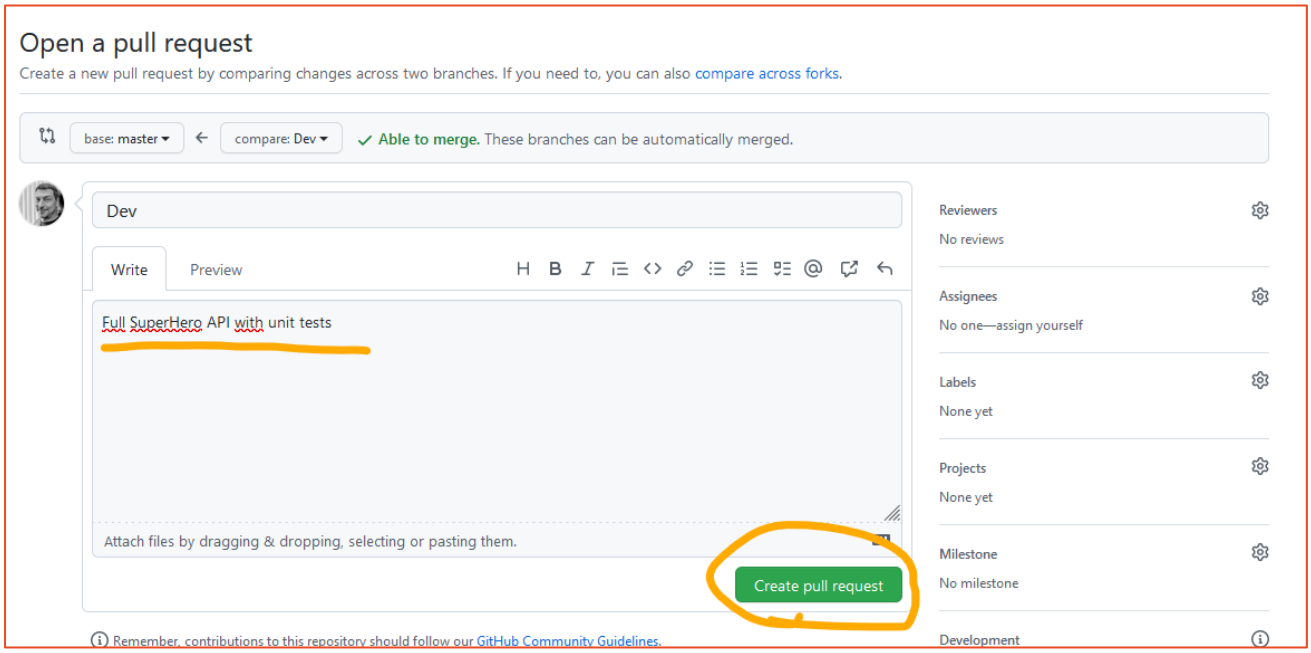


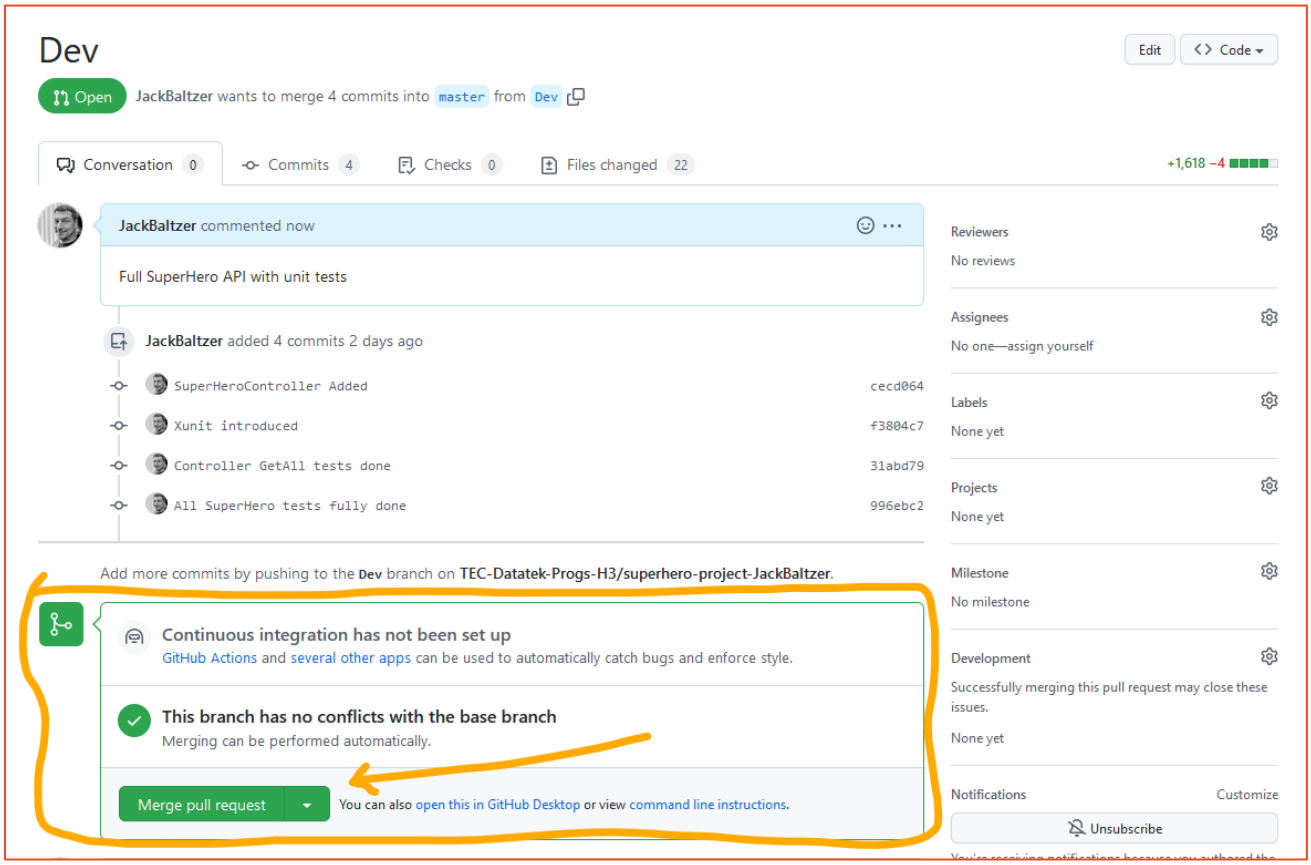
Når du ændrer til "**compare:Dev**" burde der dukke en masse op på skærmen, og i dette tilfælde burde der ikke være nogle konflikter, da vi stort set kun har oprettet nye filer, og kun tilføjet enkelte linjer til eksisterende filer.

Klik på den grønne "**Create pull request**".

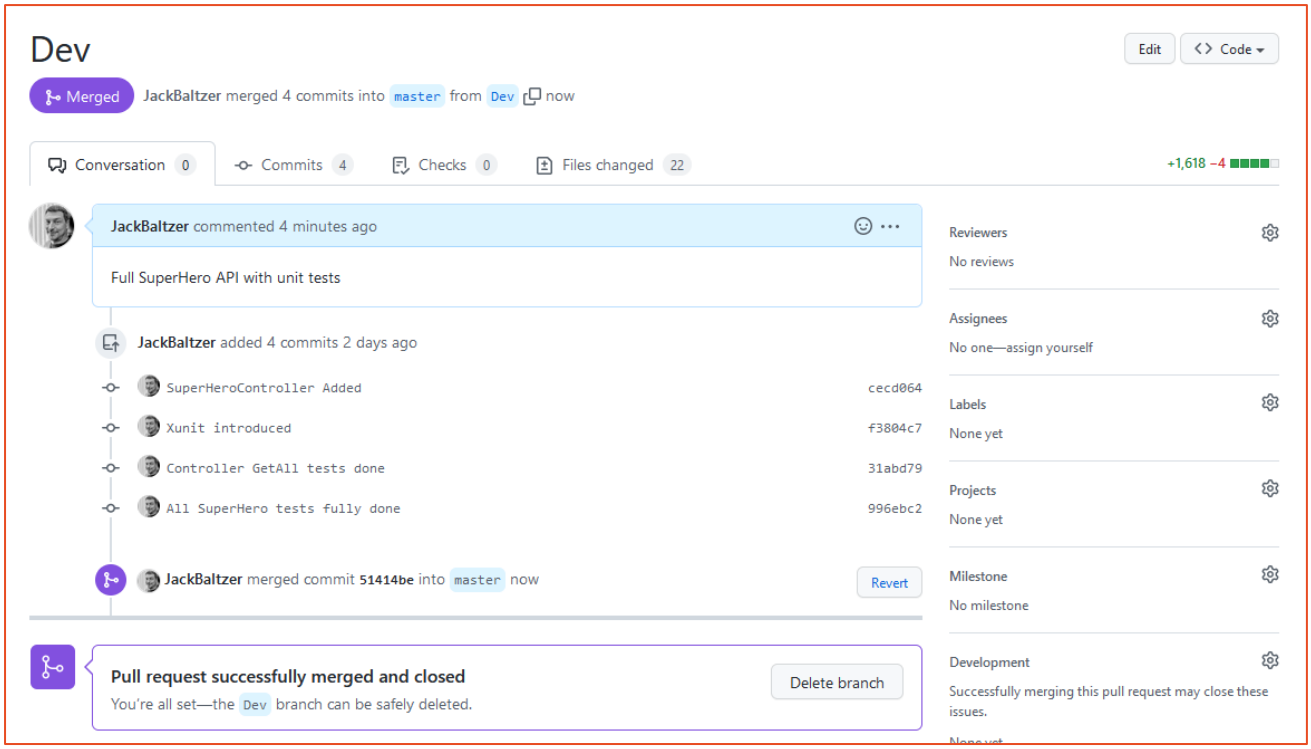


Når man fletter brances sammen, skal man også skrive en "**commit besked**" lige som når man udfører et normalt "**commit**", her bør skrive hvad det er der flettes ind i branchen, så man fremover har et bedre overblik.





Bekræft "**merge**" og afvent at github bliver færdig med at flette filerne og mapperne sammen, derefter får man en bekræftelse på at pull requesten er gennemført. Her kan man så vælge at slette "**Dev branch**" hvis der ikke skal udvikles mere, eller man kan bare lukke browservinduet og så er man færdig med sin pull request. Vi skal arbejde videre med vores Dev, så **lad være med at slette den** her.



CLIENTSIDE APPLIKATIONEN

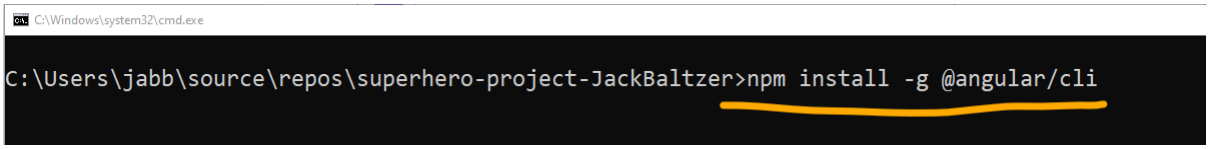
NodeJS, Angular og Visual Studio Code

Her skal vi lige sikre os du har de nødvendige programmer installeret på din computer.

Det er vigtigt at have https://nodejs.org/en/ LTS version 16.17.0 - Angular kører på NodeJS.

Når Node er installeret, så kan vi installere Angular CLI som vi kommer til at benytte til at generere clientside applikationen.

Åben en Command Prompt, og naviger ind i roden af "**SuperHeroProjekt**" mappen *(der hvor .sln filen ligger)*. Kør kommandoen "**npm install -g @angular/cli**"



Når angular er installeret, så kør kommandoen "**ng version**" og bekræft det er **Angular version 14.1.3** *(eller derover),* og **Node version 16.17.0** *(eller derover)*.

```
C:\Windows\system32\cmd.exe

C:\Users\jabb\source\repos\superhero-project-JackBaltzer>ng version

     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 14.1.3
Node: 16.17.0
Package Manager: npm 8.15.0
OS: win32 x64

Angular:
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1401.3 (cli-only)
@angular-devkit/core         14.1.3 (cli-only)
@angular-devkit/schematics   14.1.3 (cli-only)
@schematics/angular          14.1.3 (cli-only)


C:\Users\jabb\source\repos\superhero-project-JackBaltzer>
```

Opret SuperHeroClient projektet

DET ER MEGET VIGTIG AT DU HAR VALGT "DEV BRANCH" SOM DEN AKTIVE BRANCH I VISUAL STUDIO, DA VI KOMMER TIL AT OPRETTE EN MASSE KODE DER SKAL BEARBEJDES FØR DET ER FUNKTIONELT!

```
C:\Users\jabb\source\repos\superhero-project-JackBaltzer>ng new SuperHeroClient
? Would you like to add Angular routing? (y/N)
```

Derefter spørger Angular hvilken stylesheet teknologi du vil benytte, her vælger jeg klassisk "**CSS**" ved at trykke **[enter]**.

```
C:\Users\jabb\source\repos\superhero-project-JackBaltzer>ng new SuperHeroClient
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

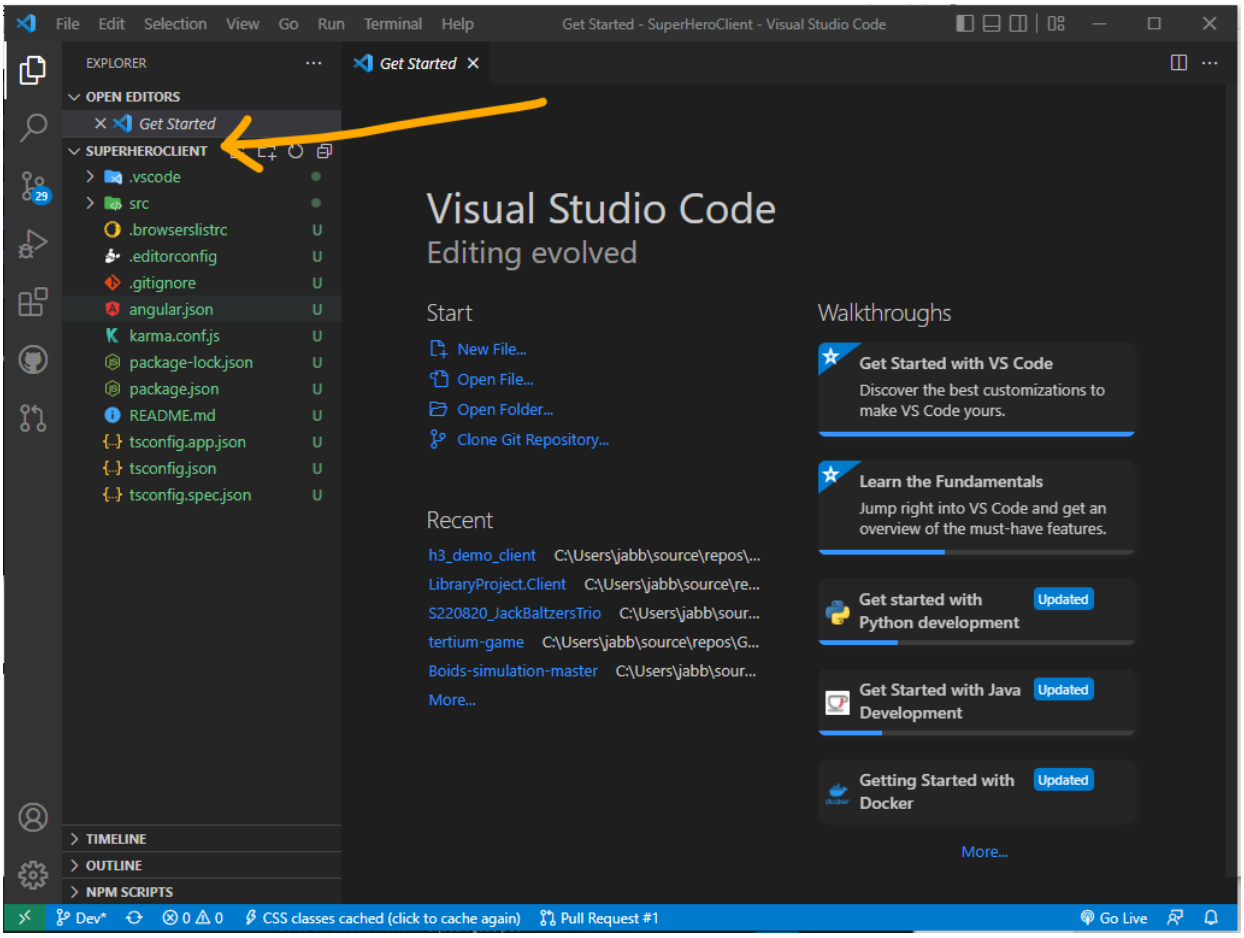
Derefter går angular i gang med at oprette mapper og filer til grund-applikationen. Det kan godt tage lidt tid, der er en del filer og moduler der skal oprettes og hentes.

```
CREATE SuperHeroClient/src/app/app.module.ts (393 bytes)
CREATE SuperHeroClient/src/app/app.component.html (23115 bytes)
CREATE SuperHeroClient/src/app/app.component.spec.ts (1100 bytes)
CREATE SuperHeroClient/src/app/app.component.ts (219 bytes)
CREATE SuperHeroClient/src/app/app.component.css (0 bytes)
√ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.

C:\Users\jabb\source\repos\superhero-project-JackBaltzer>
```

Skriv "**cd SuperHeroClient**" for at navigere ind i den nye mappe vi har fået af Angular.

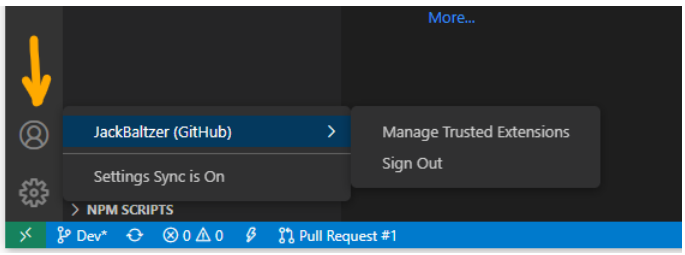
```
CREATE SuperHeroClient/src/app/app.component.css (0 bytes)
√ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.

C:\Users\jabb\source\repos\superhero-project-JackBaltzer>cd SuperHeroClient

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>
```

```
    Directory is already under version control. Skipping initialization of git.

C:\Users\jabb\source\repos\superhero-project-JackBaltzer>cd SuperHeroClient

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>code .
```



GIT og VSCODE

Sørg for at du også er logget på Git igennem VSCode. Det gør arbejdet lidt lettere, da VSCode også kan arbejde direkte med Git.



Kør Angular applikationen

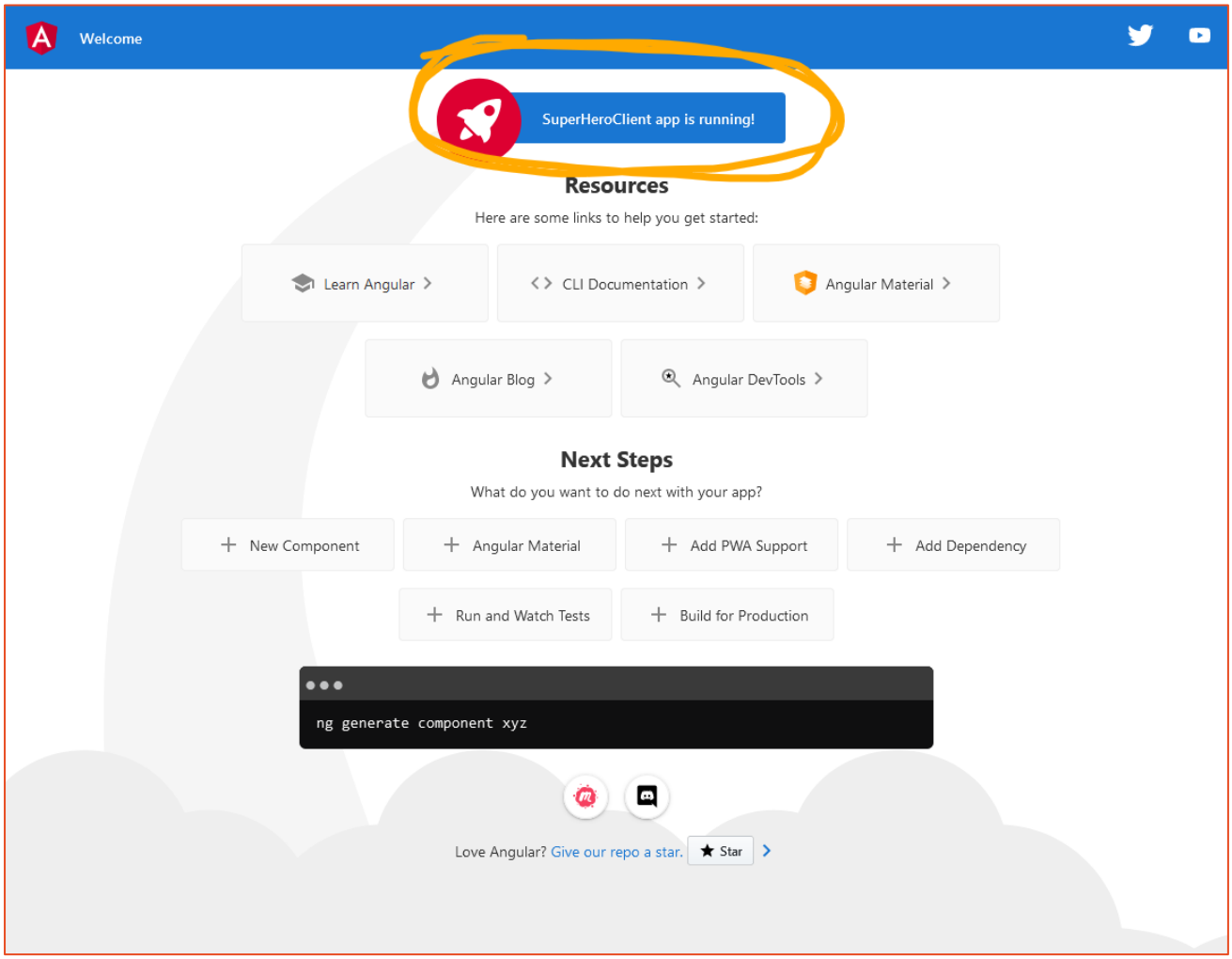
Inden vi går i gang med vores superhelte, er det en god ide at tjekke om alt fungerer som det skal.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. Alle rettigheder forbeholdes.

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>npm start
```

Det tager lidt tid første gang, men derefter sker ændringerne heldigvis relativt hurtigt.

```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. Alle rettigheder forbeholdes.

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>npm start

> super-hero-client@0.0.0 start
> ng serve

✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.04 MB |
polyfills.js          | polyfills     | 315.35 kB |
styles.css, styles.js | styles        | 207.38 kB |
main.js               | main          |  50.19 kB |
runtime.js            | runtime       |   6.53 kB |

                      | Initial Total |   2.61 MB

Build at: 2022-08-19T07:11:57.594Z - Hash: e8c2b16cdc7d1aa6 - Time: 17080ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


✓ Compiled successfully.
```

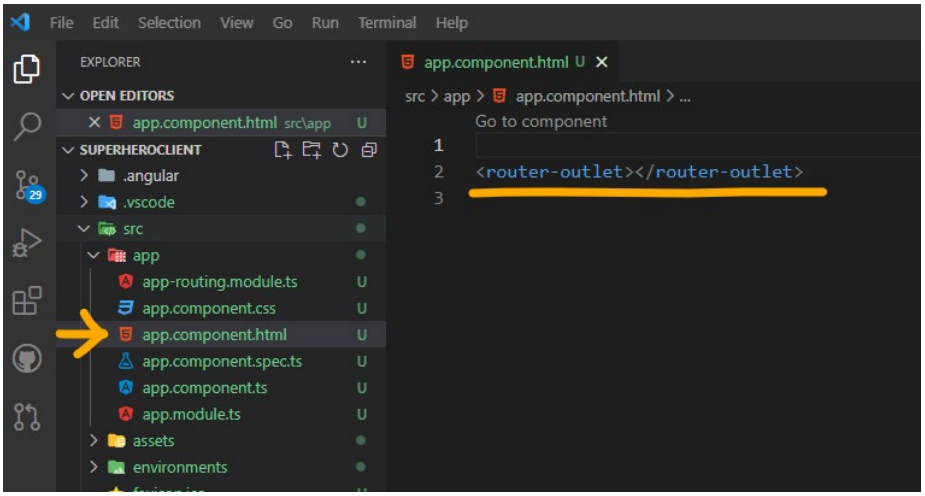

Hold [CTRL] ned og klik med musen på "http://localhost:4200/" og se din angular app kører



Ryd forsiden

Vi er ikke interesseret i at have standard Angular forsiden, vi vil i stedet have vores egen der kan vise superheltene.

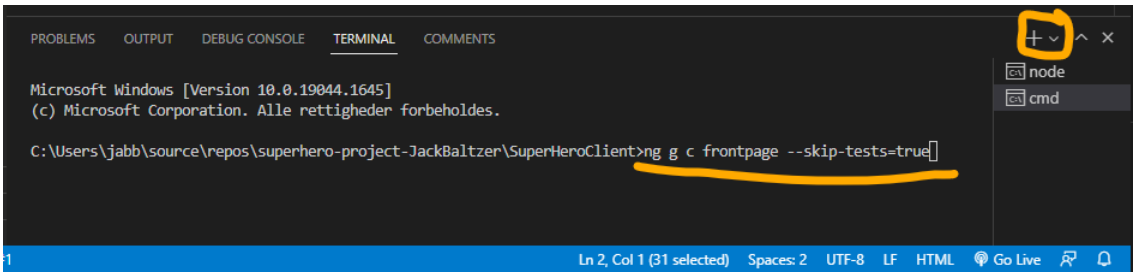
Find filen "**src/app/app.component.html**" og fjern alt undtagen linjen "**<router-outlet></router-outlet>**" gem og se browseren nu er en tom hvid side.



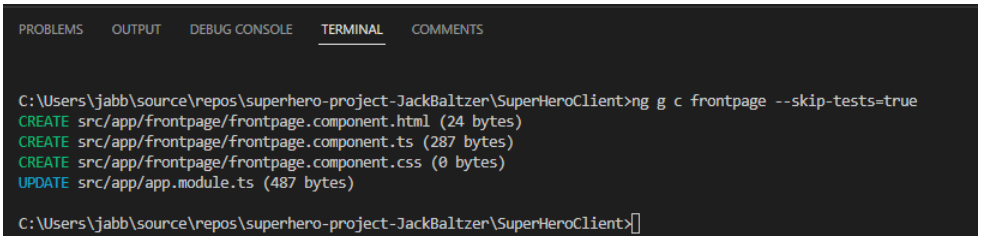
ANGULAR KOMPONENTER

Vi vil gerne have et nyt forside komponent, som kan indlæses når vi besøger hjemmesiden.

Til det opretter vi et nyt komponent, det er den tilgang angular bygger på, at man opretter komponenter til det meste.

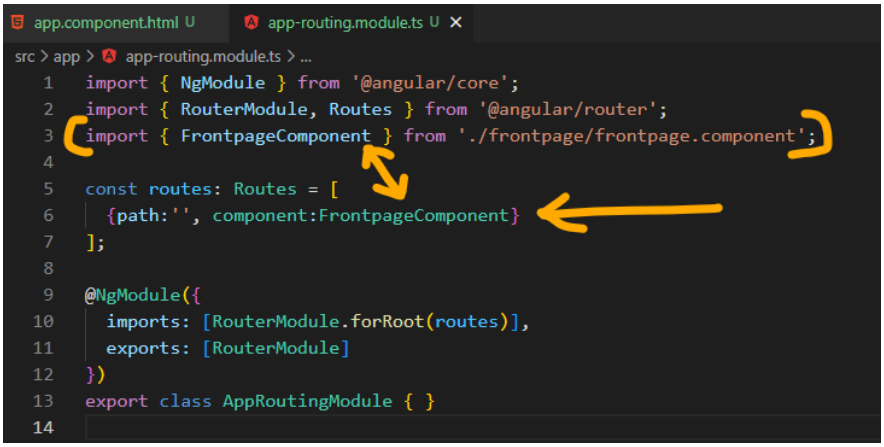


Så vil Angular oprette nogle nye filer, og opdatere "**hovedmodulet**" i applikationen så den kender til det nye komponent. Et komponent består af noget "**css-styling**", noget "**html-markup**", og en "**typescript-kode**" fil.

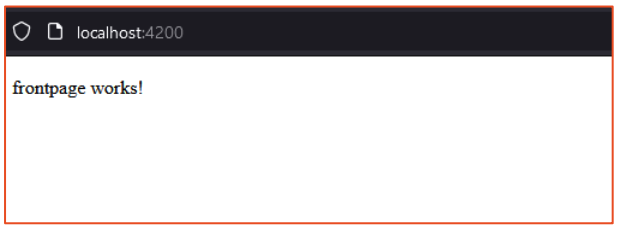


Routing

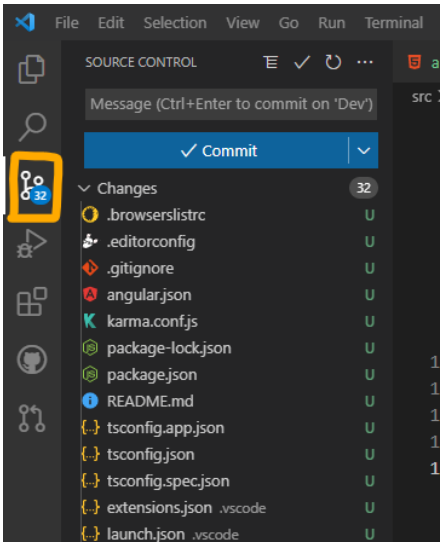
For at oprette en route i angular, skal vi have fat i "**src/app/app-routing.module.ts**" filen. Her tilføjes et element i routes-arrayet. Når du skriver "**FrontPageComponent**", burde VSCode intellisense komme frem og selv indsætte det nødvendige "**import statement**". Det er meget lettere at lade VSCode håndtere det, frem for selv at holde styr på mappestrukturen…



Gem og se browseren meget gerne skulle være en hvid side med teksten "**frontpage works!**"

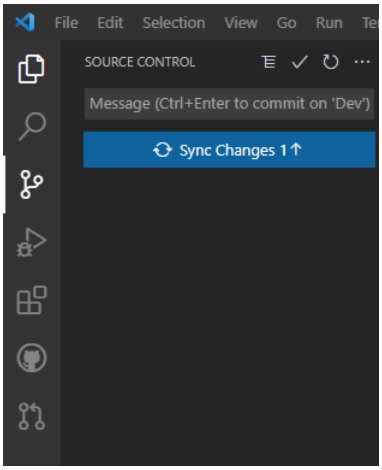


Det betyder der er hul igennem til vores forside modul. Nu vil det være et rigtig godt tidspunkt at tænke på at commit til github.



Men det er meget meget vigitgt at sikre mappen "**node_modules**" ***IKKE*** er med i filer der committes. Der burde være ca 32 filer når man klikker på "**Source Control**" i venstre side af VSCode

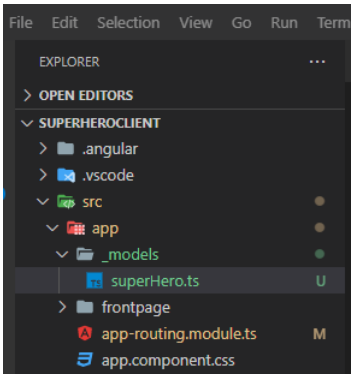
HVIS DIN VISER NODE_MODULES, SÅ SKAL DU STOPPE HER OG FÅ HJÆLP!



ANGULAR OG API FORBINDELSE

Models i Angular

Vi har brug for at kunne repræsentere en SuperHero i vores Angular applikation, præcis som vores "**DTO**" og "**Entitet**" i API'en.



Her opretter vi filerne og mapperne manuelt. Opret en mappe kaldet "**_models**" i "**src/app/**" og i den mappe oprettes en fil kaldet "**superHero.ts**"

_models er blot for at holde denne mappe øverst i mappe strukturen, det er mere overskueligt når applikationen vokser.

Modeller i angular er baseret på interfaces, som er typestærke. Dvs vi skal beskrive hvilke datatyper hver egenskab har.

Læg mærke til store og små bogstaver. Vi skriver det præcis som det ser ud fra API'en

Server response

Code	Details
200	Response body

```
{
  "id": 1,
  "name": "Superman",
  "firstName": "Clark",
  "lastName": "Kent",
  "place": "Metropolis",
  "debutYear": 1938
}
```

src > app > _models > superHero.ts > ...

```ts
export interface SuperHero {
  id: number;
  name: string;
  firstName: string;
  lastName: string;
  place: string;
  debutYear: number;
}
```

Nu kan vi arbejde med modellen på vores forside.

Tilpas "**src/app/frontpage/frontpage.component.ts**", start på linjen med "**superHeroes: SuperHero[] = []**"

Der burde VSCode automatisk indsætte model import linjen, når du skriver "**SuperHero**".

Derefter oprettes et par superhelte manuelt, så vi kan få noget ud på skærmen hurtigt.

src > app > frontpage > frontpage.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { SuperHero } from '../_models/superHero';

@Component({
  selector: 'app-frontpage',
  templateUrl: './frontpage.component.html',
  styleUrls: ['./frontpage.component.css']
})
export class FrontpageComponent implements OnInit {
  superHeroes: SuperHero[] = [];

  constructor() { }

  ngOnInit(): void {
    this.superHeroes.push({
      id: 1,
      name: "Superman",
      firstName: "Clark",
      lastName: "Kent",
      place: "Metropolis",
      debutYear: 1938
    });
    this.superHeroes.push({
      id: 2,
      name: "Iron Man",
      firstName: "Tony",
      lastName: "Stark",
      place: "Malibu",
      debutYear: 1963
    });
  }
}

```

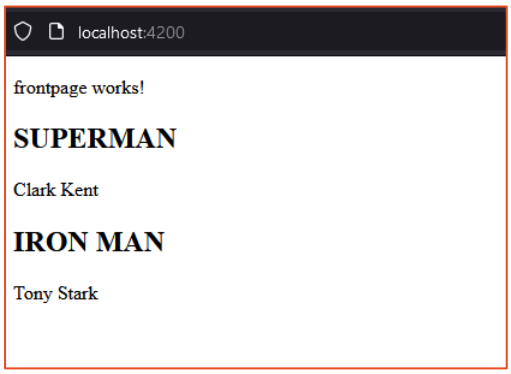
Derefter skal vi ændre "**src/app/frontpage/frontpage.component.html**", så den løber igennem de helte der er tilgængelige:

src > app > frontpage > frontpage.component.html > ...

Go to component

```html
<p>frontpage works!</p>

<div *ngFor="let superHero of superHeroes">
  <h2>{{ superHero.name | uppercase }}</h2>
  <p>{{ superHero.firstName }} {{ superHero.lastName }}</p>
</div>

```

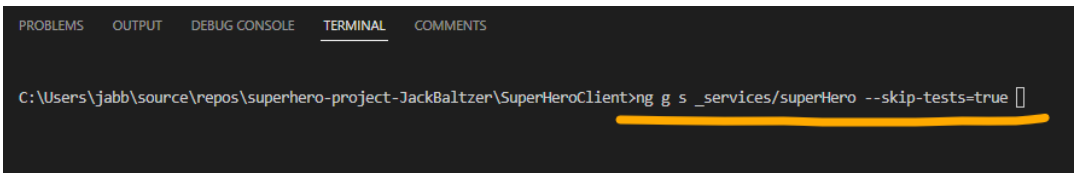

Resultatet i browseren burde være noget i denne stil. Læg mærke til transformationen af navnet til store bogstaver.



Angular Services

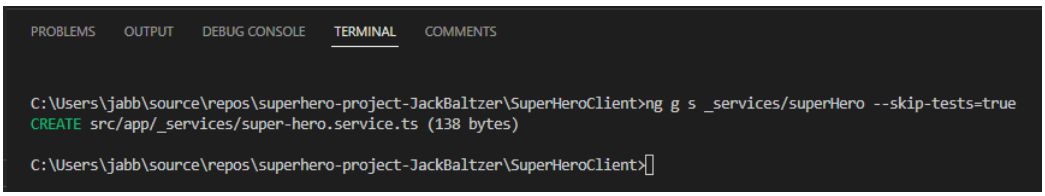
Nu ved vi der kan udskrives superhelte på forside, så vil det give mening at ringe op til vores API'en og hente heltene ud derigennem. Og her er det smart at oprette en service, så kan vi let skifte APIen ud, uden at skulle omkode klienten, i hvertfald kun med få ændringer hvis data strukturen ændres.

Services oprettes igennem en Angluar CLI kommando "**ng g s _services/superHero --skip-tests=true**"



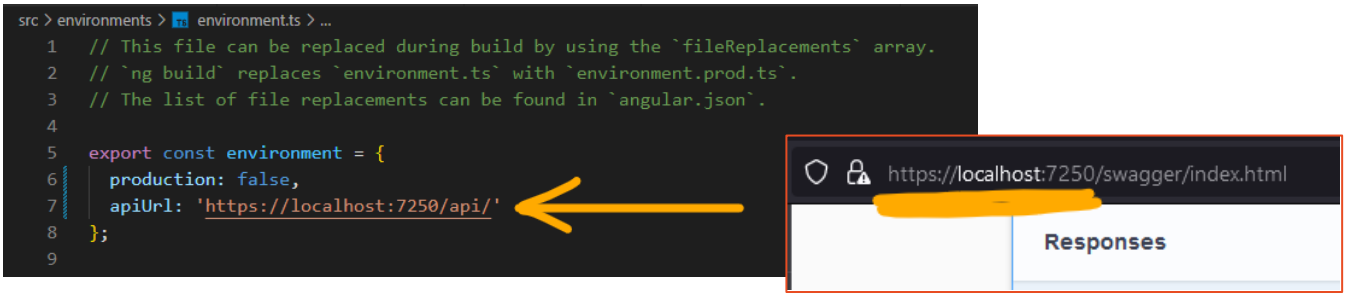
Her oprettes en ny mappe kaldet "**_services**" og i den oprettes en service-fil kaldet "**super-hero.service.ts**"

Angular har det lidt svært med camelcase, så den oversætter gerne til kebab-case i stedet.. ja, det kaldes "**kebab-case**"



Environment variabler

Under udviklingen af applikationen arbejder vi på en lokal adresse, men engang i fremtiden når applikationen skal på nettet vil adressen ændre sig. For at gøre det lidt smart, gemmer vi adressen i filen "**src/app/environments/environments.ts**", her tilføjes adressen til vores api.



Det er ikke swagger vi skal besøge igennem angular, derfor erstattes "**swagger**" med "**api**" som er den korrekt adresse.

Nu kan vi forbinde til API i servicen, ved at oprette en privat variabel kaldet "**apiUrl**", start med den, så tilføjer VSCode selv import! Husk at denne service er til "**superHero**", derfor skal det tilføjes til adressen.

src > app > _services > super-hero.service.ts > ...

```ts
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class SuperHeroService {
  private apiUrl = environment.apiUrl + 'superHero';

  constructor() { }
}
```

Angular HttpClient

src > app > app.module.ts > ...

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FrontpageComponent } from './frontpage/frontpage.component';

@NgModule({
  declarations: [
    AppComponent,
    FrontpageComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Derefter kan vi benytte "**Dependency Injection**", til at servere "**HttpClient**" modulet i vores service.

src > app > _services > super-hero.service.ts > ...

```ts
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class SuperHeroService {
  private apiUrl = environment.apiUrl + 'superHero';

  constructor(private http: HttpClient) { }
}
```

Angular service.getAll

Når vi arbejder med Angular og services, kan vi benytte et smart koncept kaldet "**Observables**". Det er strengt taget ikke nødvendigt i dette eksempel, men princippet er godt at kende, derfor benytter jeg det her.

"**Observable**" ligger i modulet "**rxjs**" som VSCode gerne skulle skrive automatisk.

src > app > _services > super-hero.service.ts > ...

```
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Observable } from 'rxjs';
4  import { environment } from 'src/environments/environment';
5  import { SuperHero } from '../_models/superHero';
6
7  @Injectable({
8    providedIn: 'root'
9  })
10 export class SuperHeroService {
11   private apiUrl = environment.apiUrl + 'superHero';
12
13   constructor(private http: HttpClient) { }
14
15   getAll(): Observable<SuperHero[]> {
16     return this.http.get<SuperHero[]>(this.apiUrl);
17   }
18 }
```

src > app > frontpage > frontpage.component.ts > ...

```
1  import { Component, OnInit } from '@angular/core';
2  import { SuperHero } from '../_models/superHero';
3  import { SuperHeroService } from '../_services/super-hero.service';
4
5  @Component({
6    selector: 'app-frontpage',
7    templateUrl: './frontpage.component.html',
8    styleUrls: ['./frontpage.component.css']
9  })
10 export class FrontpageComponent implements OnInit {
11   superHeroes: SuperHero[] = [];
12
13   constructor(private superHeroService: SuperHeroService) { }
14
15   ngOnInit(): void {
16     this.superHeroService.getAll().subscribe(x => this.superHeroes = x);
17   }
18 }
```

CORS

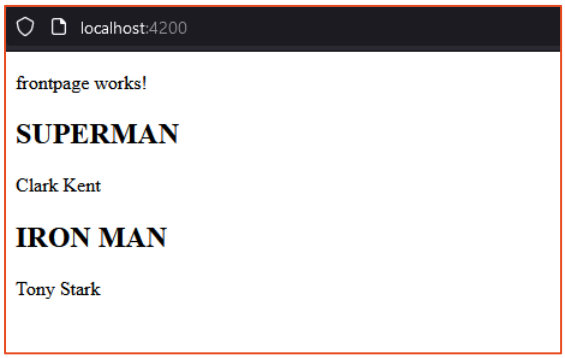
Det burde være nok, men fordi vi forsøger at tilgå API fra en anden URL end den API kører på, så forhinder API'en forespørgslen.

> Forespørgsel til fremmed websted blokeret: Politikken for samme oprindelse tillader ikke læsning af fjernressourcen *https://localhost:7250/api/superHero*. (Årsag: CORS-headeren 'Access-Control-Allow-Origin' findes ikke). Statuskode: 200. [Læs mere]

Det er standard opførsel, og forventet. Problemet kaldet "**Cross Origin Resource Sharing**" og kan konfigureres så vi tillader adgang fra andre kilder.

Åben "**Program.cs**" i "**SuperHeroAPI**" projektet, og tilføj den fremhævede linje.

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseCors(policy => policy.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

app.UseAuthorization();

app.MapControllers();

app.Run();
```



ANGULAR SUPERHERO ADMIN

Lad os oprette en mappe til adminpanelet, så det er pakket lidt væk fra almindelige komponenter, samtidigt med mappen, kan vi oprette komponentet til superHero administrationen.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>ng g c admin/superHero --skip-tests=true
CREATE src/app/admin/super-hero/super-hero.component.html (25 bytes)
CREATE src/app/admin/super-hero/super-hero.component.ts (290 bytes)
CREATE src/app/admin/super-hero/super-hero.component.css (0 bytes)
UPDATE src/app/app.module.ts (668 bytes)

C:\Users\jabb\source\repos\superhero-project-JackBaltzer\SuperHeroClient>
```


Og lad os tilføje en route til adminpanelet, så det kan indlæses i browseren.

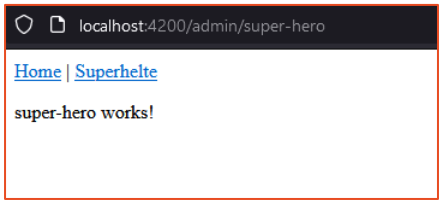
src > app > app-routing.module.ts > ...

```ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { SuperHeroComponent } from './admin/super-hero/super-hero.component';
import { FrontpageComponent } from './frontpage/frontpage.component';

const routes: Routes = [
  { path: '', component: FrontpageComponent },
  { path: 'admin/super-hero', component: SuperHeroComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

src > app > app.component.html > ...

```
Go to component
<nav>
  <a routerLink="/">Home</a> |
  <a routerLink="/admin/super-hero">Superhelte</a>
</nav>
<router-outlet></router-outlet>

```

Gem og se i browseren at der er et par links hvor man kan klikke frem og tilbage imellem forside og superhero admin.



SuperHero Service alle CRUD metoderne

```ts
getAll(): Observable<SuperHero[]> {
  return this.http.get<SuperHero[]>(this.apiUrl);
}
```

.getById(superHeroId: number)

```ts
getById(superHeroId: number): Observable<SuperHero> {
  return this.http.get<SuperHero>(`${this.apiUrl}/${superHeroId}`);
}
```

.create(superHero: SuperHero)

```ts
create(superHero: SuperHero): Observable<SuperHero> {
  return this.http.post<SuperHero>(this.apiUrl, superHero);
}
```


.update(superHero: SuperHero)

```
27  update(superHero: SuperHero): Observable<SuperHero> {
28    return this.http.put<SuperHero>(`${this.apiUrl}/${superHero.id}`, superHero);
29  }
```

.delete(superHeroId: number)

```
31  delete(superHeroId: number): Observable<SuperHero> {
32    return this.http.delete<SuperHero>(`${this.apiUrl}/${superHeroId}`);
33  }
```

Angular FormModule

For at kunne håndtere inputfelter og binding imellem forms og komponenter, kan vi benytte et angular modul kaldet "**FormsModule**" som ligger i "**@angular/forms**". Det skal tilføjes til "**imports-arrayet**" i "**app.module.ts**". Så får vi muligheden for at benytte **[(ngModel)]** til **to vejs binding** af data.

src > app > app.module.ts > ...

```
1   import { NgModule } from '@angular/core';
2   import { BrowserModule } from '@angular/platform-browser';
3   import { HttpClientModule } from '@angular/common/http';
4
5   import { AppRoutingModule } from './app-routing.module';
6   import { AppComponent } from './app.component';
7   import { FrontpageComponent } from './frontpage/frontpage.component';
8   import { SuperHeroComponent } from './admin/super-hero/super-hero.component';
9   import { FormsModule } from '@angular/forms';
10
11  @NgModule({
12    declarations: [
13      AppComponent,
14      FrontpageComponent,
15      SuperHeroComponent
16    ],
17    imports: [
18      BrowserModule,
19      AppRoutingModule,
20      HttpClientModule,
21      FormsModule
22    ],
23    providers: [],
24    bootstrap: [AppComponent]
25  })
26  export class AppModule { }
27
```

Angular SuperHero admin funktionerne

Vores SuperHeroAdmin panel, skal kunne udføre CRUD. Dvs vi skal hente alle SuperHeroes og vise dem i en liste. Så kan vi ud for hver helt enten rette eller slette. Når vi retter, bliver en formuler forudfyldt med heltens data, og hvis vi sletter forsvinder helten.

Deruover har vi formularen som kan benyttes til oprette en ny helt. Det giver rigtig god mening at genbruge den samme formular til både oprettelse og redigering.

Det letter vores arbejde hvis vi har nogle komponent globale variabler til at holde på alle SuperHeroes og på en enkelt SuperHero.

src > app > admin > super-hero > super-hero.component.ts > SuperHeroComponent > edit

```ts
import { Component, OnInit } from '@angular/core';
import { SuperHero } from 'src/app/_models/superHero';
import { SuperHeroService } from 'src/app/_services/super-hero.service';

@Component({
  selector: 'app-super-hero',
  templateUrl: './super-hero.component.html',
  styleUrls: ['./super-hero.component.css']
})
export class SuperHeroComponent implements OnInit {
  superHeroes: SuperHero[] = [];
  superHero: SuperHero = { id: 0, name: '', firstName: '', lastName: '', place: '', debutYear: 0 };

  constructor(private superHeroService: SuperHeroService) { }

  ngOnInit(): void {
    this.superHeroService.getAll().subscribe(x => this.superHeroes = x);
  }
```

```ts
  edit(superHero: SuperHero): void {
    this.superHero = superHero;
  }
```

```ts
  delete(superHero: SuperHero): void {
    if (confirm('Er du sikker på du vil slette?')) {
      this.superHeroService.delete(superHero.id)
        .subscribe(() => {
          this.superHeroes = this.superHeroes.filter(x => x.id != superHero.id);
        })
    }
  }
```

```ts
  cancel(): void {
    this.superHero = { id: 0, name: '', firstName: '', lastName: '', place: '', debutYear: 0 };
  }
```

SuperHeroComponent.save()

```
37  save(): void {
38    if (this.superHero.id == 0) {
39      this.superHeroService.create(this.superHero)
40        .subscribe({
41          next: (x) => {
42            this.superHeroes.push(x);
43            this.superHero = { id: 0, name: '', firstName: '', lastName: '', place: '', debutYear: 0 };
44          },
45          error: (err) => {
46            console.log(Object.values(err.error.errors).join(', '));
47          }
48        });
49    } else {
50      this.superHeroService.update(this.superHero)
51        .subscribe({
52          error: (err) => {
53            console.log(Object.values(err.error.errors).join(', '));
54          },
55          complete: () => {
56            this.superHero = { id: 0, name: '', firstName: '', lastName: '', place: '', debutYear: 0 };
57          }
58        });
59    }
60  }
```

Brug lidt tid på at læse og forstå hvad denne ”**save()**” metode gør!

SuperHero Admin HTML delene

Formularen til at oprette, og til at rette, benytter ”**[(ngModel)]**” til at to vejs binde værdierne mellem inputfeltet og den bagvedliggende typescript fil. Hvis værdien ændres i htmlformularen, så ændres den automatisk i koden også, og omvendt.

```
3   <div>
4     <label>Navn</label>
5     <input [(ngModel)]="superHero.name">
6   </div>
7   <div>
8     <label>Fornavn</label>
9     <input [(ngModel)]="superHero.firstName">
10  </div>
11  <div>
12    <label>Efternavn</label>
13    <input [(ngModel)]="superHero.lastName">
14  </div>
15  <div>
16    <label>Sted</label>
17    <input [(ngModel)]="superHero.place">
18  </div>
19  <div>
20    <label>Debut År</label>
21    <input [(ngModel)]="superHero.debutYear">
22  </div>
23  <button (click)="save()">Gem</button>
24  <button (click)="cancel()">Annuller</button>
25
```

Sammen med formularen er der to knapper. Den ene kalder ”**save()**” metoden, som enten gemmer en ny, eller gemmer ændringerne, alt efter hvilken kontext knappen aktiveres. Annuller knappen nulstiller superHero, så formularen er tom.

Listen af eksisterende SuperHeroes, udskrives i en tabel, hvor der ud for hver SuperHero er 2 knapper, en for ret og en til slet. De to knapper kalder metoder i typescript filen, og medsender den SuperHero der tilhører knappen.

```
<table>
  <tr>
    <th>actions</th>
    <th>id</th>
    <th>Navn</th>
    <th>Fulde navn</th>
  </tr>
  <tr *ngFor="let superHero of superHeroes">
    <td>
      <button (click)="edit(superHero)">Ret</button>
      <button (click)="delete(superHero)">Slet</button>
    </td>
    <td>{{ superHero.id }}</td>
    <td>{{ superHero.name }}</td>
    <td>{{ superHero.firstName}} {{ superHero.lastName}}</td>
  </tr>
</table>
```

Gem alt og test det igennem i browseren. Du burde være istand til at udføre den FULDE CRUD hele vejen igennem til databasen og tilbage igen igennem API'en fra Angular klienten.

COMMIT OG PUSH TIL GITHUB, OG MERGE IND I MASTER!