Välkommen tillbaka!

I den här delen (3.1) av bloggserien går vi igenom grunduppsättningen och layout av gränssnittet för en administrationssida byggd med MVC i ASP.NET Core; från start till mål med all kod du behöver.

Eftersom inga anrop görs mot API:et eller Inloggningen (IdentityServer) vi byggde i föregående avsnitt går det bra att börja här och senare göra del 2.

Värt att tänka på om du hoppar över del 2 är att vi i det avsnittet går igenom vissa tekniska saker som kan behöva vara på plats i din utvecklingsmiljö inför arbetet med denna guide.

Innehåll i bloggserien:

* [del 1](http://www.agero.se/blogg/asp-dotnet-core-del1-vad-och-varf%C3%B6r) - en introduktion till .NET Core
* [del 2](http://www.agero.se/blogg/asp-dotnet-core-del2-web-api-implementation-och-autentisering-med-identity-och-identityserver)  - vi skapar ett Web Api med autentisering och enhetstester/integrationstester.
* **del 3** - en administrationssida i ASP.NET Core MVC (tag helpers, routing, autentisering)
  + **3.1 - (här och nu) Grunden och det visuella - utveckling av gränssnittet i MVC**
  + 3.2 - Kommunikation mellan MVC Controller och remote MVC Web API
  + 3.3 - xUnit Integrationstester och enhetstester mot remote Web API med autentisering
* del 4 - en klient i form av en mobilapp i Ionic.
* del 5 - vi refaktoriserar appen, använder Azure DocumentDb för att lagra konsultprofilerna och flyttar konfigurationen av IdentityServer till databasen.
* del 6 - vi driftsätter appen i Azure
* därefter kommer det att implementeras någon typ av administration för att hantera användare och roller, planen var att jag skulle använda mig av [IdentityManager](https://github.com/identitymanager) men det finns ingen färdig implementation för .NET Core Identity ännu så vi får se hur jag löser det, inget är bestämt än och har du tips får du gärna kontakta mig.
* i fortsättningen kommer vi när tid finns att göra en app i React och React Native för administration av innehåll.

*Tips! För att inte missa något är det bäst att du prenumererar på bloggen genom att skriva in din e-postadress här till höger.*

Administrationsgränssnitt

Vi bygger ett enkelt admin för hanteringen av konsultprofiler.

Projektkrav

Inför utvecklingen av administrationen kom det fram att beställaren även vill ha en bild till profilerna, samt en summeringstext, detta kommer alltså i framtiden att behöva införas i API:et vi byggde i föregående del (del2).

Varje profil ska därmed innehålla

* Förnamn
* Efternamn
* Summering
* Beskrivning
* Bild

När jag är inloggad som administratör

* Ska alla profiler listas på första sidan och profilerna ska ha en summering istället för hela beskrivningen så att profilerna blir lätta att överblicka.
* På första sidan ska jag kunna
  + klicka på en profil för att visa hela profilen
  + radera en profil
  + lägga till en ny profil
* På profilsidan (detaljvyn) ska jag kunna
  + radera profilen
  + aktivera redigeringsläge för profilen.
* På redigeringssidan ska jag kunna
  + radera profilen
  + ändra namn, beskrivning och bild

Avgränsning

Sedan tidigare har vi alltså krav på att profilen ska bestå av förnamn, efternamn och en beskrivning. Nu tillkommer alltså även en profilbild.

Vi börjar med att lägga till en "placeholder"-bild i administrationen och när vi senare refaktoriserar API:et för att hantera bilder uppdaterar vi administrationen till att hantera även redigering av bilderna.

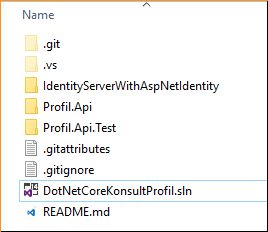
Tester

Som stöd i utvecklingsprocessen för det här projektet använder jag mig självklart av tester, både enhetstester och integrationstester med xUnit.

För att göra guiden mer fokuserad på byggandet av funktionaliteten har jag valt att bryta ut testimplementationerna till en separat guide som du kan se här *[…Kommande…]*

Implementation

Projektkatalogen ser ut så här innan vi går vidare med implementationen

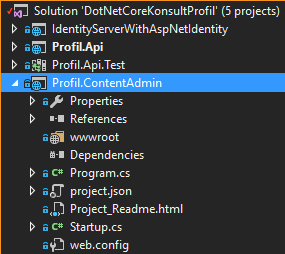


Skapa ny branch i Git

Skapa en ny Branch från Master och döp den till "AddingContentAdmin"

Skapa grunden för administrationssidorna

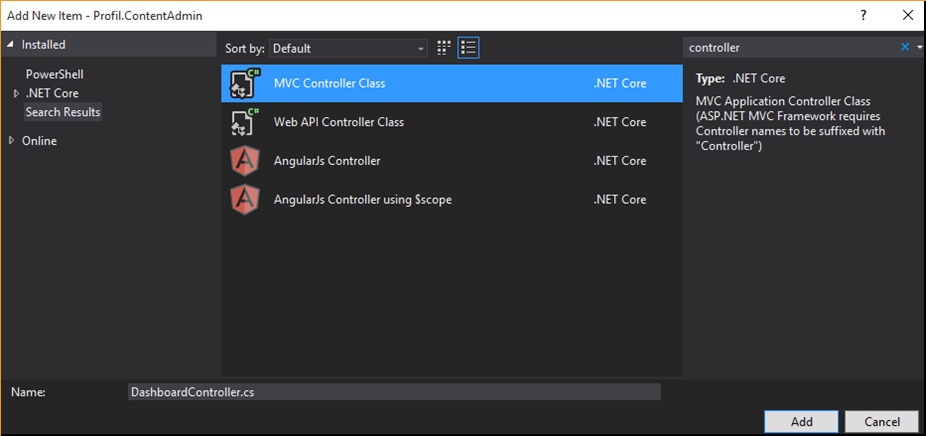
Skapa ett nytt projekt av typen ASP.NET Core Web Application med namnet "Profil.ContentAdmin" och välj "Empty" template.



Vi bestämmer oss för att första sidan ska vara en översiktsvy där alla profiler listas.

Vi ska lägga till en controller för vyn och vi kallar den "Dashboard". Enligt konventioner placeras controllern i en katalog med namnet "Controllers" och klassnamnet slutar med ”Controller”.

Skapa katalogen ”Controllers” och däri filen ”DashboardController.cs”



Controllern ser ut så här när den är skapad från mallen "MVC Controller Class"

public class DashboardController : Controller

{

// GET: //

public IActionResult Index()

{

var model = new ConsultantProfilesViewModel();

model = GetDummyProfiles();

return View(model);

}

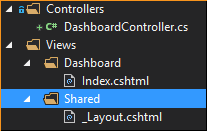
}

Med controllern på plats kan vi skapa vyerna för denna och enligt konventionerna ligger alla vyer i katalogen "Views" i en underkatalog med samma namn som controllern ("Dashboard"). Vyerna i sin tur är namngivna efter motsvarande action ("Index").

Skapa den första vyn Index.cshtml i katalogstrukturen enligt ovan.

Eftersom vi kommer att ha en meny i toppen av alla sidor lägger vi också till en layoutsida, vilken index kommer att renderas inuti.

\_Layout.cshtml ligger enligt konventionerna i katalogen "Shared" direkt under "Views" Filstrukturen ska nu se ut så här



\_Layout.cshtml ser ut så här om den är skapad med mallen "MVC View Layout Page" i Visual Studio.

<!DOCTYPE html>  
   
<html>  
<head>  
 <meta name="viewport" content="width=device-width" />  
 <title>@ViewBag.Title</title>  
</head>  
<body>  
 <div>  
 @RenderBody()  
 </div>  
</body>  
</html>

@RenderBody() renderar sidor som är beroende av layouten, i vårt fall Index.

Uppdatera Index.cshtml enligt nedan

@{  
 ViewBag.Title = "Hantera konsultprofiler";  
 Layout = "\_Layout";  
}  
<h2>Här ska vi lista alla konsultprofiler</h2>

Nu har vi en controller och vyer på plats för att testköra projektet visuellt. Det saknas bara lite konfiguration i Startup.cs

Eftersom vi använder MVC behöver vi lägga till nugetpaket för detta i "project.json". Samtidigt passar vi på att uppdatera .NET Core (alla paket) till 1.1

Uppdatera project.json till att se ut så här.

{  
 "dependencies": {  
 "Microsoft.NETCore.App": {  
 "version": "1.1.0",  
 "type": "platform"  
 },  
 "Microsoft.AspNetCore.Diagnostics": "1.1.0",  
 "Microsoft.AspNetCore.Server.IISIntegration": "1.1.0",  
 "Microsoft.AspNetCore.Server.Kestrel": "1.1.0",  
 "Microsoft.Extensions.Logging.Console": "1.1.0",  
   
 **"Microsoft.AspNetCore.Mvc": "1.1.0"**  
 },  
   
 "tools": {  
 "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.1.0-preview4-final"  
 },  
   
 "frameworks": {  
 "netcoreapp1.1": {  
 "imports": [  
 "dotnet5.6",  
 "portable-net45+win8"  
 ]  
 }  
 },  
…

*Tips!   
Du kan i den här guiden kopiera mina textexempel i varje steg för att slippa skriva koden själv, men i de fall du gör nya saker eller något du behöver träna på rekommenderar jag att du faktiskt skriver koden för hand och använder mina steg som referenser. Du gör så klart som du vill och vad som passar dig bäst! =)*

Uppdatera Startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

loggerFactory.AddConsole();

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseMvc(ConfigureRoutes);

}

private void ConfigureRoutes(IRouteBuilder routeBuilder)

{

routeBuilder.MapRoute("Default", "{controller=Dashboard}/{action=Index}/{id?}");

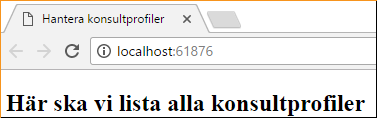
}

*Här lägger vi till en default route som pekar på dashboard controller.*

Nu är vi redo för att testköra projektet.

*Tips!  
När du utvecklar i ASP.NET Core kan du göra det till en vana att starta applikationen utan debugging ("Start without debugging, Ctrl+F5), detta gör att du kan ladda om webbläsaren när du sparat kodändringarna och då får (nästan) direkt feedback utan att behöva bygga om projektet.*

Starta projektet, du ska nu få följande resultat.



Ok, då har vi en fungerande grundkonfiguration för MVC och ett minimalt startprojekt att bygga vidare på.

Gör en commit i ditt Git =)

Nästa steg är att förbereda gränssnittet med CSS och ikoner

Förbered UI för att lista profiler

Eftersom vi vill ha en någorlunda snygg vy gör vi det enkelt för oss och drar in Bootstrap samt ikoner från material design i projektet innan vi börjar "rita".

Både Bootstrap och material design icons finns som npm-paket, vi börja med att lägga till npm som en dependency.

*Npm kan du lägga till antingen via "add item" och template "npm Configuration file" eller genom att skapa en fil med namnet "package.json" i projektroten*

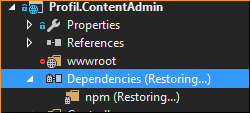
Uppdatera package.json med projektnamn och lägg till bootstrap så att den ser ut så här

{  
 "version": "1.0.0",  
 "name": "profil\_contentadmin",  
 "private": true,  
 "devDependencies": {  
 },  
 "dependencies": {  
 "bootstrap": "3.3.7"  
 //,  
 // "material-design-icons": "3.0.1"  
 }  
}

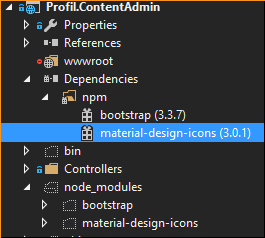
Om du vill kan du lägga till "material-design-icons" som en dependency men det tar flera minuter att ladda ner paketet, så om du inte vill vänta på det just nu kan du nöja dig med att hämta via ikoner via CDN, vi visar strax hur.

*Självklart kan du även köra bootstrap via CDN och skippa installationen av npm helt, men följer du stegen lär du dig förhoppningsvis något nytt =).*

När filen sparas kommer VS att hämta ner beroenden (material design icons tar som sagt några minuter)



och det ser ut så här i projektstrukturen när det är klart.



I nuläget kör vi alla filer från deras befintliga kataloger och servar inget från wwwroot (vilket annars är det vanliga efter deploy).

För att komma åt statiska filer från disk (CSS, bilder…) måste man konfigurera detta i Startup.Configure, via en middleware/metod som heter UseStaticFiles i modulen "Microsoft.AspNetCore.StaticFiles"

Lägg till "Microsoft.AspNetCore.StaticFiles": "1.1.0" i project.json som en dependency.

Anropar man UseStaticFiles() utan parameter är det från "wwwroot" filerna läses. Vi vill läsa från katalogen "node\_modules" så att vi kommer åt bootstrap (och ikonerna); detta konfigurerar vi så här I Startup.Configure (före UseMvc)

...  
 app.UseStaticFiles(new StaticFileOptions {   
 RequestPath = "/node\_modules",   
 //lyssnar endast på ”path” som börjar med "/node\_modules"  
 FileProvider = new PhysicalFileProvider(Path.Combine(env.ContentRootPath, "node\_modules"))  
 });  
  
 *app.UseMvc(ConfigureRoutes);*   
 ...

Det här fungerar utmärkt, men för att göra koden lite trevligare bryter vi ut detta i en extension så att vi kan skriva "app.UseNodeModules" Samtidigt passar vi på att göra en extension även för katalogen "content" där vi kommer att lägga bilder och egna CSS-filer.

Skapa en ny katalog "Middleware" och däri en ny klass "ApplicationBuilderExtension.cs"

Uppdatera enligt nedan (och läs igenom för att förstå vad som händer).

using System.IO;  
using Microsoft.Extensions.FileProviders;  
   
//det är enligt konvention att lägga IApplicationBuilder extensions i detta namespace (ms aspnetcore builder)  
namespace Microsoft.AspNetCore.Builder  
{  
     public static class ApplicationBuilderExtension   
 {  
 public static IApplicationBuilder UseNodeModules(this IApplicationBuilder app, string projectRoot)  
 {  
 var path = Path.Combine(projectRoot, "node\_modules");  
 var fileProvider = new PhysicalFileProvider(path);  
   
 var options = new StaticFileOptions  
 {  
 RequestPath = "/node\_modules",   
 //lyssnar endast på ”path” som börjar med "/node\_modules"  
 FileProvider = fileProvider  
 };  
   
 app.UseStaticFiles(options);  
   
 return app;  
 }  
   
 public static IApplicationBuilder UseContentFiles(this IApplicationBuilder app, string projectRoot)  
 {  
 var path = Path.Combine(projectRoot, "content");  
 var fileProvider = new PhysicalFileProvider(path);  
   
 var options = new StaticFileOptions  
 {  
 RequestPath = "/content", lyssnar endast på ”path” som börjar med "content"  
 FileProvider = fileProvider  
 };  
   
 app.UseStaticFiles(options);  
 return app;  
 }  
 }  
}

Och uppdatera Startup.Config enligt nedan

...   
 app.UseNodeModules(env.ContentRootPath);  
 app.UseContentFiles(env.ContentRootPath);   
   
 *app.UseMvc(ConfigureRoutes);*  
...

I \_Layout.cshtml kan vi nu lägga till sökvägarna till bootstrap och ikoner enligt följande (välj CDN eller lokalt). Observera att VS (och Resharper) i det här läget inte ser sökvägen som normalt, därför får du ingen intellisense.

<title>@ViewBag.Title</title>  
   
 <link href="/node\_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />  
 @\*<link href="/node\_modules/material-design-icons/iconfont/material-icons.css" rel="stylesheet" />\*@  
 <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

Nu har vi grunden klar och kan lägga till HTML med styling. Nästa steg är att lägga till en överliggande meny.

UI - Lägg till meny

Skapa en CSS-fil med katalogstruktur enligt "content/css/layout.css"

Och lägg till följande CSS

.navigation {

list-style: none;

margin: 0;

background: deepskyblue;

display: flex;

flex-flow: row wrap;

justify-content: flex-start;

}

.navigation a {

text-decoration: none;

display: block;

padding: 1em;

color: white;

}

.navigation a:hover {

background: rgb(0, 165, 220);

}

.rightnav {

margin-left: auto;

padding-right: 5rem;

}

/\*Här använder vi media query för att styra menyn till en kolumn istället för rad när browsern blir smalare än 600px\*/

@media all and (max-width: 600px) {

.navigation {

flex-flow: column wrap;

padding: 0;

}

.navigation a {

text-align: center;

padding: 10px;

border-top: 1px solid #ffffff;

border-top: 1px solid rgba(255,255,255,0.3);

border-bottom: 1px solid #000000;

border-bottom: 1px solid rgba(0,0,0,0.1);

}

.navigation li:last-of-type a {

border-bottom: none;

}

.rightnav {

all: unset;

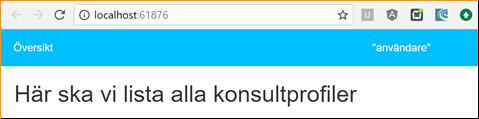
}

}

Uppdatera \_Layout.cshtml enligt nedan för att aktivera denna CSS

...  
 <link href="/content/css/layout.css" rel="stylesheet" />  
</head>  
<body>  
 <nav class="navigation">  
 <div>  
 <a href="/">Översikt</a>  
 </div>  
 <div class="rightnav">  
 @\*todo: här kommer vi senare att lägga till användarnamn och utloggning\*@  
 <a href="/">"användare"</a>  
 </div>  
 </nav>  
   
 <div class="container">  
 @RenderBody()  
 </div>  
</body>  
</html>

Ladda om webbläsaren/projektet oh verifiera att du har följande resultat.



Med menyn på plats är nästa steg att lista profilerna

UI - Lista konsultprofiler

Nu ska vi uppdatera vyerna och controllern för att visa upp konsultprofilerna. Eftersom det här är en översiktsvy med listning av samtliga profiler kommer vi inte att visa all information, endast bild, namn och en summering.

Först måste controllern returnera profilerna till indexvyn och för att hantera en lista av profiler smidigt behöver vi följande

* En vymodell för listan av profiler
* En vymodell för profilen
* En partiell vy (MVC HTML.Partial) vilken är en template för hur en profil ska visas.

Vi börjar med vymodellerna, skapa en ny katalog "ViewModels" och där i lägger du följande två klasser

ConsultantProfilesViewModel.cs

public class ConsultantProfilesViewModel

{

public IEnumerable ConsultantProfiles { get; set; }

//detta gör att vi inte får "null reference exception" när vi loopar över profilerna i vyn

public ConsultantProfilesViewModel()

{

ConsultantProfiles = new List();

}

}

ConsultantProfileItemViewModel.cs

public class ConsultantProfileItemViewModel

{

public Guid Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Summary{ get; set; }

}

10.png

Nu kan vi uppdatera Index.cshtml till att använda profilernas vymodell, uppdatera filen enligt nedan

**@using Profil.ContentAdmin.ViewModels**  
**@model ConsultantProfilesViewModel**  
<h2>Hantera dina konsultprofiler</h2>  
   
<section class="list-header-container">  
 <h3>Lista av profiler</h3>  
</section>  
   
**<section class="list-container">**  
 **@foreach (var profileItem in Model.ConsultantProfiles)**  
 **{**  
 **@Html.Partial("\_ProfileItem", profileItem)**  
 **}**  
**</section>**

Inuti foreach-loopen vill vi generera listan av profiler och vi bryter ut den koden till en egen fil, nämligen "\_ProfileItem.cshtml",

Skapa \_ProfileItem.cshtml i samma katalog som Index och uppdatera filen enligt nedan. (Eftersom jag förbakat koden går vi händelserna lite i förväg och lägger till navigeringen samt raderaknappen på en gång)

...  
@using Profil.ContentAdmin.ViewModels  
@model ConsultantProfileItemViewModel  
   
<section>  
 <section>  
  
 <a asp-action="Details" asp-route-id="@Model.Id">  
 <img src="/content/images/profile.jpg">  
 </a>  
   
 <a asp-action="Details" asp-route-id="@Model.Id">  
 <p>"@Model.Summary"</p>  
 </a>  
   
 <a asp-action="Delete" asp-route-id="@Model.Id"><i class="medium material-icons">delete\_forever</i></a>  
 </section>  
   
 <section>  
 <a asp-action="Details" asp-route-id="@Model.Id">  
 @Model.FirstName @Model.LastName  
 </a>  
 </section>  
</section>

Här arbetar vi med Tag Helpers vilket är det nya sättet i MVC och ersätter HTML Helpers (vilka du fortfarande kan använda).

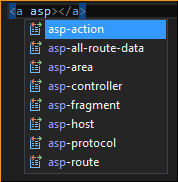
Fördelen med Tag Helpers är att de passar in i HTML-strukturen bättre och är därmed lättare att styla; dessutom får du intellisense i VS.

*Läs mer om Tag Helpers här*[*https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro*](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro)

Med "asp-action" säger du vilken action i controllern som ska anropas, i det här fallet "Details" och eftersom vi inte anger "asp-controller" anropas den vi är i.

"asp-route-*id*" bygger på "asp-route" och du lägger helt enkelt till vilket parameternamn du vill efter "route-", i vårt fall är det "id" vi vill skicka in i "Details" action.

*Det finns några fler "AnchorTagHelpers" som du får utforska själv.*



Ladda om browsern/projektet nu.

Som du ser får vi en NullReferenceException eftersom vi controllern inte returnerar en "ConsultantProfilesViewModel".

Uppdatera DashboardController.cs så att Index returnerar en "ConsultantProfilesViewModel"

...

public IActionResult Index()

{

var model = new ConsultantProfilesViewModel();

return View(model);

}

...

Ladda om sidan igen, nu ska du få ett resultat som ser ut så här.



För att ha något att jobba med när vi fixar utseendet lägger vi in dummydata temporärt i controllern.

Uppdatera controllern enligt följande

public IActionResult Index()

{

var model = new ConsultantProfilesViewModel();

model = GetDummyProfiles();

return View(model);

}

private ConsultantProfilesViewModel GetDummyProfiles()

{

return new ConsultantProfilesViewModel

{

ConsultantProfiles = new List

{

new ConsultantProfileItemViewModel

{

FirstName = "Jimi", LastName = "Friis",

Summary = "Gillar bra kod, affärsutveckling och golf",

Id = new Guid("00000000-0000-0000-0000-000000000001")

},

new ConsultantProfileItemViewModel

{

FirstName = "Pontus", LastName = "Nagy",

Summary = "Är vass på React och klär sig medvetet",

Id = new Guid("00000000-0000-0000-0000-000000000002")

},

new ConsultantProfileItemViewModel

{

FirstName = "Mikael", LastName = "Sundström",

Summary = "Glänser inom teknik och bakning",

Id = new Guid("00000000-0000-0000-0000-000000000003")

},

}

};

}

Ladda om sidan igen och du ska se en lista med de tre profilerna, inte så snyggt kanske men nu har vi ett resultat att modellera med i stylingen.



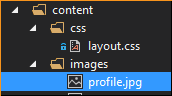
Härligt! Vi har en lista av profiler, nästa steg är att snygga till listelementen.

UI - profiler i visitkortsliknande format

Målet med den här listan är att göra varje profil till ett kort liknande ett visitkort.

Börja med att spara den här bilden som "/content/images/profile.jpg", det blir vår "placeholder".





För att justera innehållet använder jag mig av Flexbox och jag utvecklar med Chrome, använder du andra webbläsare kan stödet för Flexbox variera.

*Tips!  
En bra introduktion till Flexbox*[*https://css-tricks.com/snippets/css/a-guide-to-flexbox/*](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) *Rolig och interaktiv tutorial i Flexbox*[*http://flexboxfroggy.com/*](http://flexboxfroggy.com/)

Vi börjar med att sätta CSS-klasser i \_ProfileItem.cshtml

Uppdatera enligt följande

<section class="list-item">  
 <section class="item-image-and-summary">  
   
 <a class="item-image anchor-unset" asp-action="Details" asp-route-id="@Model.Id">  
 <img src="/content/images/profile.jpg">  
 </a>  
   
 <a class="item-summary anchor-unset" asp-action="Details" asp-route-id="@Model.Id">  
 <p>"@Model.Summary"</p>  
 </a>  
   
 <a asp-action="Delete" asp-route-id="@Model.Id"><i class="medium material-icons">delete\_forever</i></a>  
 </section>  
   
 <section class="item-name-section">  
 <a class="item-name anchor-unset" asp-action="Details" asp-route-id="@Model.Id">  
 @Model.FirstName @Model.LastName  
 </a>  
 </section>  
</section>

Skapa filen "/content/css/profileitem.css"

Lägg till länken till filen i \_Layout.cshtml

...   
 <link href="/content/css/layout.css" rel="stylesheet" />  
 <link href="/content/css/profileitem.css" rel="stylesheet" />

...

Som av en händelse råkar jag ha en hel styling färdig som passar just för våra profilkort =)

Uppdatera profileitem.css enligt följande

/\* https://css-tricks.com/snippets/css/a-guide-to-flexbox/ \*/

/\* http://flexboxfroggy.com/ \*/

/\*tänk på att display:flex inte ärvs utan måste sättas för varje "container" som ska ha flex items\*/

.list-container {

display: flex;

flex-flow: row wrap;

}

/\*override anchor styles för items\*/

.anchor-unset {

text-decoration: none !important;

color: #000000;

}

.list-item {

flex: 1 16%;

flex-flow: column nowrap;

border-radius: 4px;

margin-top: 1rem;

margin-right: 1rem;

box-shadow: 2px 2px 10px #888888;

min-width: 25em;

max-width: 25em;

display: flex;

}

.list-item a {

/\*låt inte text skifta färg i länkarna när man hovrar\*/

color: inherit;

}

.item-image-and-summary {

display: flex;

flex-flow: row nowrap;

}

.item-image img {

max-width: 12rem;

}

.item-summary {

padding-top: 2em; /\*se till att texten inte hamnar i höjd med raderaknappen\*/

/\*begränsa textens yta\*/

max-width: 20em;

max-height: 9em;

overflow: hidden;

text-align: center;

font-style: italic;

display: flex;

align-items: flex-end; /\*tryck ner texten mot botten\*/

}

..item-name-section {

background-color: #00bfff;

background-color: rgba(0, 191, 255, .1);

border-radius: 2px;

padding-top: 0.2em;

display: flex;

flex-flow: row nowrap;

}

.item-name {

padding-left: 3rem;

width: 100%;

}

.item-buttons {

margin-left: auto;

justify-content: space-between;

flex-flow: row nowrap;

display: flex;

}

.item-buttons a {

margin-left: auto;

}

Ladda om sidan, du ska nu ha resultatet enligt bilden nedan.

* Om du minskar respektive ökar bredden på fönstret flyttas profilkorten så att så många som möjligt visas i bredd, korten behåller även sin storlek.



Fint! - Ja om man gillar blått - jag gillar blått! =)

Nu är vi klara med förstasidan, nästa steg blir att lägga till detaljvyn där man ser hela konsultprofilen.

Details

Börja med att lägga till "Views/Dashboard/Details.cshtml" Uppdatera vyn enligt följande

@using Profil.ContentAdmin.ViewModels  
@model ConsultantProfileViewModel  
   
@{  
 ViewBag.Title = "Konsultprofil för " + @Model.FirstName;  
 Layout = "\_Layout";  
}  
   
<article class="detail-and-edit-page">  
   
 <section class="detail-and-edit-header-section">  
 <h3 class="detail-and-edit-name-header">@Model.FirstName @Model.LastName</h3>  
   
 <div class="detail-and-edit-photo">  
 <img src="/content/images/profile.jpg">  
 </div>  
   
 <div class="detail-and-edit-header-buttons">  
 <a asp-action="Edit" asp-route-id="@Model.Id"><i class="medium material-icons">mode\_edit</i></a>  
 <a asp-action="Delete" asp-route-id="@Model.Id"><i class="medium material-icons">delete\_forever</i></a>  
 </div>  
 </section>  
   
 <section class="detail-description-section">  
 <p>@Model.Description</p>  
 </section>  
</article>

*Får du varningar i din editor för att ConsultantProfileViewModel inte hittas beror det på att den inte finns än.*

Lägg till "ViewModels/ConsultantProfileViewModel.cs"

Uppdatera klassen enligt följande

public class ConsultantProfileViewModel

{

public Guid Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Description { get; set; }

}

För att vi ska kunna navigera från profillistan till detaljvyn behöver vi implementera en "Details" action i controllern.

Uppdatera DashboardController enligt följande

public IActionResult Details(Guid id)

{

var model = new ConsultantProfileViewModel();

return View(model);

}

Nu borde vi kunna navigera till "Details".

Ladda om sidan och navigera genom att klicka på ett kort.

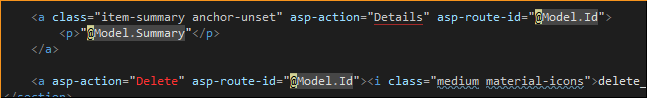
Som du märker händer ingenting när du klickar trots att länken är där, pekaren blir heller inte den vanliga pekhanden vilket den borde bli.

Att det inte fungerar beror helt enkelt på den lilla detaljen att man måste lägga till en referens till Mvc.TagHelpers i .cshtml-filer där man använder TagHelpers.

Uppdatera \_ProfileItem.cshtml med referensen

@using Profil.ContentAdmin.ViewModels  
@model ConsultantProfileItemViewModel  
**@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers**...

*Tips!  
Om du likt mig använder Resharper får du lite extra hjälp som visning av vilka actions i "asp-action" som faktiskt finns och vilka som inte finns. Så här ser det ut när "Details" action finns men inte "Delete".*



*Det kan vara bra att ha i bakhuvudet att när Resharper inte reagerar på dina asp-actions saknar du troligen referensen @addTagHelper*

Ladda om listan igen och prova att klicka på ett kort för att verifiera att länkar och controller fungerar.

Du bör få följande resultat



Perfekt! Nu kan vi navigera, men det finns en sak jag vill göra innan vi går vidare och det är att bryta ut referenserna från html-sidorna så att vi slipper tänka på det i kommande vyer.

ViewImports & ViewStart

I MVC finns det en standardsida för att importera referenser, nämligen "\_ViewImports".  
Kod som ska köras innan vyer laddas kan läggas i standardsidan "\_ViewStart".

*Både ViewStart och ViewImports kan förekomma i flera exemplar på olika nivå i katalogträdet för att specificera mer granulära inställningar på djupare nivå, en fil på rotnivå trumfas av en fil längre ut/ner i trädet.*

*Här kan du läsa mer om MVC views och layout*[*https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout*](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout)

Lägg till "Views/\_ViewImports.cshtml".

Uppdatera enligt följande

@using Profil.ContentAdmin.ViewModels  
   
@\*@addTagHelper T, Microsoft.AspNetCore.Mvc.TagHelpers - T to load specific type, \* to load all tag helpers from assembly\*@  
@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

Lägg till "Views/\_ViewStart.cshtml".

Uppdatera enligt följande (görs automatiskt om du använder VS template)

@{  
 Layout = "\_Layout";  
}

Ta bort alla dessa referenser från Index.cshtml och Details.cshtml

Ladda om browsern och verifiera att sidorna ser ut som tidigare.

Styling

För att få fylla ut vyn med data så att den blir produktionsliknande innan vi modellerar utseendet uppdaterar vi controllern enligt nedan

...

public IActionResult Details(Guid id)

{

var model = new ConsultantProfileViewModel

{

FirstName = "Jimi",

LastName = "Friis",

Id = new Guid("00000000-0000-0000-0000-000000000001"),

Description = $"Affärsintresserad Prestigelös Agil Utvecklare med fokus på Rätt. Helikopterperspektiv – Har lätt för att överblicka och förstå hur processer, system och arbetsflöden hänger ihop. Professionell – Tar ansvar och initiativ, frågar och vill veta varför innan vi gör. Resultatinriktad – Drivs av att leverera bra lösningar på riktiga och viktiga problem, Rätt sak i Rätt tid ger Rätt kundnytta. Lagspelare – Trivs bäst i team och kan om det behövs ta ledande roller. ”Det optimala utvecklingsteamet enligt mig är tvärfunktionellt och agilt med mandat till förändringar och snabb tillgång till affärens områdesexperter” Som person är Jimi professionell, ödmjuk, serviceinriktad, social, ärlig, glad och lyhörd, men för den delen inte blyg i att göra sina åsikter hörda. Arbetsmetoder som TDD, BDD och DDD ligger Jimi varmt om hjärtat . Högskolestudier inom både företagsekonomi och datavetenskap, en IT-bakgrund inom serverdrift och nätverk samt erfarenhet från diverse olika yrkesområden utanför IT-världen och ett organisatoriskt sinne underlättar förståelsen för och anpassningen till olika typer av verksamheter."

};

return View(model);

}

...

När du laddar om sidan ska du ha följande resultat.



Vi har nu ett innehåll att jobba med och som den gode tv-kocken har jag av en händelse en bit färdig CSS i ugnen redo att förtäras. =)

Skapa först filen "content/css/details-and-edit-view.css"

Lägg sedan till referensen till filen i \_Layout.cshtml

...  
<link href="/content/css/profileitem.css" rel="stylesheet" />  
**<link href="/content/css/details-and-edit-view.css" rel="stylesheet" />**  
...

Uppdatera slutligen CSS-filen enligt nedan

/\*detail and edit view common\*/

.detail-and-edit-page {

max-width: 70rem;

}

.detail-and-edit-header-section {

display: flex;

flex-flow: column-reverse nowrap;

}

.detail-and-edit-header-buttons {

display: flex;

flex-flow: row nowrap;

padding-top: 0.5rem;

margin-left: auto;

}

.detail-and-edit-header-buttons a {

padding-left: 1rem;

}

.detail-and-edit-name-header {

margin-top: auto;

}

.detail-and-edit-photo {

margin-top: -3rem;

padding-bottom: 0.4em;

align-self: center;

}

/\*detail view only \*/

.detail-description-section {

padding-right: 1rem;

}

Om du laddar om sidan ska det se ut så här, lite snyggare enligt mig! Fast profilbilden är inte min, jag får byta ut den i framtiden ;-)



Jag känner mig nöjd med detaljvyn i det här läget och redo för att gå vidare, nästa steg är redigeringsvyn.

Edit

Eftersom redigeringsvyn blir väldigt lik detaljvyn kan vi börja med att kopiera allt i från detaljvyn och justera de små skillnaderna.

Skapa filen "Views/Dashboard/Edit.cshml"

Uppdatera Edit enligt nedan

@model ConsultantProfileViewModel  
   
@{  
 ViewBag.Title = "Redigera Konsultprofil för " + @Model.FirstName;  
}  
   
<section class="detail-and-edit-page">  
 <section class="detail-and-edit-header-section">  
   
 <h3 class="detail-and-edit-name-header">@Model.FirstName @Model.LastName</h3>  
   
 <div class="detail-and-edit-photo">  
 <img src="/content/images/profile.jpg">  
 </div>  
 <div class="detail-and-edit-header-buttons">  
 <a asp-action="Details" asp-route-id="@Model.Id"><i class="medium material-icons">pageview</i></a>  
 <a asp-action="Delete" asp-route-id="@Model.Id"><i class="medium material-icons">delete\_forever</i></a>  
 </div>  
 </section>  
   
   
 <section class="detail-description-section">  
 <p>@Model.Description</p>  
 </section>  
</section>

Lägg till en Edit-action i controllern; vi kopierar Detail-action rakt av eftersom vi behöver samma presentationsdata just nu.

public IActionResult Edit(Guid id)

{

var model = new ConsultantProfileViewModel

{

FirstName = "Jimi",

LastName = "Friis",

Id = new Guid("00000000-0000-0000-0000-000000000001"),

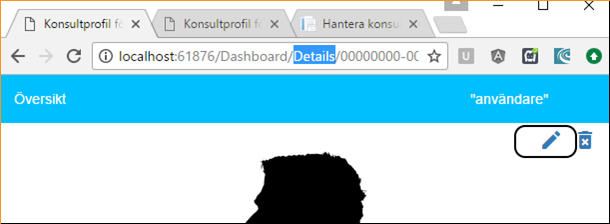
Description = $"Affärsintresserad Prestigelös Agil Utvecklare med fokus på Rätt. Helikopterperspektiv – Har lätt för att överblicka och förstå hur processer, system och arbetsflöden hänger ihop. Professionell – Tar ansvar och initiativ, frågar och vill veta varför innan vi gör. Resultatinriktad – Drivs av att leverera bra lösningar på riktiga och viktiga problem, Rätt sak i Rätt tid ger Rätt kundnytta. Lagspelare – Trivs bäst i team och kan om det behövs ta ledande roller. ”Det optimala utvecklingsteamet enligt mig är tvärfunktionellt och agilt med mandat till förändringar och snabb tillgång till affärens områdesexperter” Som person är Jimi professionell, ödmjuk, serviceinriktad, social, ärlig, glad och lyhörd, men för den delen inte blyg i att göra sina åsikter hörda. Arbetsmetoder som TDD, BDD och DDD ligger Jimi varmt om hjärtat . Högskolestudier inom både företagsekonomi och datavetenskap, en IT-bakgrund inom serverdrift och nätverk samt erfarenhet från diverse olika yrkesområden utanför IT-världen och ett organisatoriskt sinne underlättar förståelsen för och anpassningen till olika typer av verksamheter."

};

return View(model);

}

Spara filerna, ladda om Details



och navigera till redigeringen.



Navigeringen är på plats.

Det vi behöver göra nu är att möjliggöra redigering; det gör vi genom att lägga till och ändra fälttyper, samt en spara-knapp.

Uppdatera Edit.cshtml genom att ersätta den sista sektionen (Description) enligt följande

...   
 *@\*<section class="detail-description-section">*  
 *<p>@Model.Description</p>*  
 *</section>\*@*  
   
 <section class="edit-section">  
 <form method="post" asp-antiforgery="true" class="">  
 <fieldset>  
 <section>  
 <div>  
 <label asp-for="FirstName"></label>  
 <input asp-for="FirstName" required class="border-style input-field-border-style" />  
 <span asp-validation-for="FirstName"></span>  
 </div>  
 <div>  
 <label asp-for="LastName"></label>  
 <input asp-for="LastName" required class="border-style input-field-border-style" />  
 <span asp-validation-for="LastName"></span>  
 </div>  
 </section>  
   
 <section class="edit-description-section">  
 <div class="edit-description-label-and-save">  
 <label class="edit-description-label" asp-for="Description"></label>  
 <a class="edit-save-button-top" href="javascript:document.getElementsByTagName('form').item(0).submit()">  
 <i class="material-icons">save</i>  
 </a>  
 </div>  
 <textarea class="border-style textarea-border-style" cols="50" rows="10" name="description">@Model.Description</textarea>  
 <span asp-validation-for="Description"></span>  
 <a class="edit-save-button-bottom" href="javascript:document.getElementsByTagName('form').item(0).submit()">  
 <i class="material-icons">save</i>  
 </a>  
 </section>  
 </fieldset>  
 </form>  
 </section>  
...

Uppdatera CSS-filen "details-and-edit-view.css" enligt följande

...

/\*edit view only \*/

.border-style {

border-radius: 5px;

border-color: deepskyblue;

border-color: rgba(0, 191, 255, 0.2);

border-width: 0rem;

background-color: lightgray;

}

.input-field-border-style {

padding-left: 1rem;

background-color: rgba(173, 216, 230, 0.2); /\*lightblue\*/

}

.textarea-border-style {

border-radius: 5px;

border-color: deepskyblue;

border-color: rgba(0, 191, 255, 0.2);

border-width: 0rem;

background-color: lightgray;

background-color: rgba(173, 216, 230, 0.1); /\*lightblue\*/

}

/\*\*edit fields\*/

.edit-section {

display: flex;

flex-flow: column nowrap;

}

.edit-description-section {

padding-top: 1rem;

padding-right: 1.5rem;

display: flex;

flex-flow: column nowrap;

}

.edit-description-label-and-save {

flex-flow: row nowrap; /\*det här är default och kan/bör utelämnas\*/

align-items: center;

/\*justify-content: center;\*/

display: flex;

}

.edit-description-label {

align-self: center;

}

.edit-save-button-top {

/\*padding-left: 1.5rem;\*/

margin-left: auto;

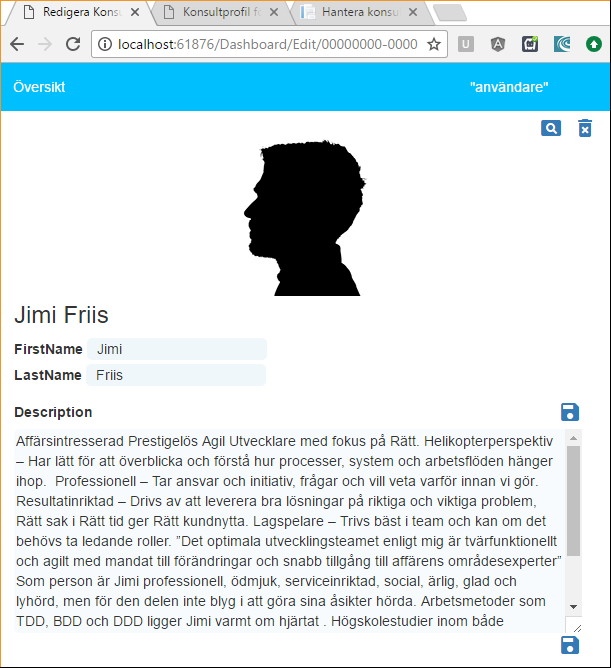
}

.edit-save-button-bottom {

margin-left: auto;

}

Spara filerna och ladda om vyn, du bör nu ha följande resultat



*Jag har här valt att både visa namnet i rubrikformat och att ha separata redigeringsfält, något jag troligen kommer att justera framöver för att endast visa namnen i redigeringsfälten.*

En avgränsning jag gjort här är att inte lägga till javascript för att visa användaren när något är ändrat och när det är sparat, eller validering av input med HTML5. Både validering och återkoppling till användaren tillkommer framöver och är något du själv kan lägga till.

Så långt har vi en till synes fungerande vy för redigeringen men jag vill bryta isär vyn.

Anledningen till att jag delar upp vyn är dels för att urskilja vad som är den statiska sidan och vad som är redigeringsformuläret, dels för återanvändning av kod och dels för att visa hur man laddar in lösa delar i vyn med hjälp av "HTML.Partial".

Skapa filen "Views/Dashboard/\_EditFieldsPartial.cshtml".

Flytta över sektionen "<section class="edit-section"> från Edit.cshtml

\_EditFieldsPartial ska nu se ut så här

@model ConsultantProfileViewModel  
   
<section class="edit-section">  
...  
</section>

Uppdatera "Edit.cshtml" enligt följande för att ladda in den partiella vyn där "edit-section" tidigare fanns

...  
@Html.Partial("\_EditFieldsPartial", Model)  
...

*Här kan man så klart välja att ha kvar en sektionstagg i Edit runt "HTML.Parial", du gör vad som känns bäst för dig. Dock ska sektionstaggen med CSS-klassen "edit-section" in i den pariella klassen eftersom den styr innehållet; därmed blir den partiella vyn komplett i sig själv.*

Spara filerna och ladda om vyn för att verifiera att du får samma resultat som tidigare.

Vi är nu klara med de första vyerna för att visa och redigera profilen, nästa steg är vyn för att lägga till en ny profil.

Create

Vyn för att skapa en ny profil ska i stort sett se ut som redigeringsvyn; eftersom min filosofi är att "ren kod är bra kod" skapar vi en separat vy.

*Man skulle kunna använda samma vy här som för redigeringen (och även detaljvyn) för*[*att inte ”repetera kod”*](https://en.wikipedia.org/wiki/Don't_repeat_yourself)*, men då behöver man kladda ner action-metoderna i controllern med logik för att veta huruvida man är i redigeringsläge eller inte.*

*Därtill skulle det behövas logik i HTML-sidan för at visa rätt knappar och till det kommer behörighetskontrollen med olika behörigheter för redigering, visa detaljer samt skapa ny… Ja du förstår, det blir helt enkelt inte bra. =)*

Skapa filen "Views/Dashboard/Create.cshtml"

Uppdatera filen enligt nedan

@model ConsultantProfileViewModel  
   
@{  
 ViewBag.Title = "Skapa Konsultprofil";  
}  
   
<section class="detail-and-edit-page">  
 <section class="detail-and-edit-header-section">  
   
 @\*todo Lägga till funktionalitet för redigering av foto\*@  
 <div class="detail-and-edit-photo">  
 <img src="/content/images/profile.jpg">  
 </div>  
   
 <div class="detail-and-edit-header-buttons">  
 <a asp-action="Index"><i class="medium material-icons">cancel</i></a>   
 </div>  
 </section>  
   
 @Html.Partial("\_EditFieldsPartial", Model)  
</section>

Som du ser blir det väldigt lite kod i den här vyn eftersom vi återanvänder den partiella vyn från editeringssidan. När det gäller CSS-klasserna kan man tycka att namnen bör uppdateras men det är inget jag lägger min energi på nu.

Enligt avgränsningarna i början av artikeln kommer vi inte att möjliggöra redigering av profilbilden i nuläget.

Vi är klara med vyn, men för att kunna navigera till den behöver vi en knapp någonstans, lämpligen på förstasidan ovanför listan av profiler. Vi behöver även en action-metod i controllern.

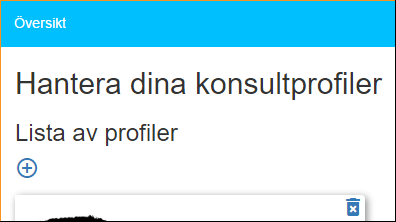
Uppdatera Index.cshtml enligt följande

...  
<h3>Lista av profiler</h3>  
 <a asp-action="Create"><i class="large material-icons">add\_circle\_outline</i></a>  
...

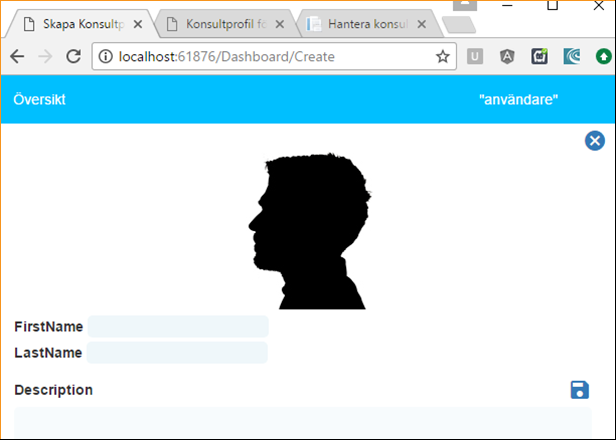
Uppdatera controllern enligt följande

public IActionResult Create()  
 {  
 return View(new ConsultantProfileViewModel());  
 }

Ladda om browsern och navigera till startsidan, "Översikt" i menyn, och du ska nu se knappen för att lägga till en ny profil.



Klicka på "lägg till"-knappen för att verifiera att vyn "Create" öppnas och ser ut så här.



Det var den sista vyn! Nu återstår bara en liten action för att radera profilerna.

Delete

Vi behöver ingen vy för att radera profiler men vi behöver implementera en action för detta i controllern.

Uppdatera controllern enligt följande

public IActionResult Delete(Guid id)

{

        //en lyckad delete skickar användaren till listan på startsidan

return RedirectToAction("Index");

}

Strålande!

Vi är klara med den visuella delen och kan gå vidare med implementationen av backend, dvs kommunikationen mellan vår DashboardController och API:et vi byggde i del 2.

Fortsättningen "del 3.2 - Kommunikation mellan MVC Controller och remote MVC Web API" hittar du här *[…inom kort när den finns tillgänglig, du har väl inte glömt att anmäla dig till prenumerationen här till höger?…]*