Övning 11: MVC - Lagersystem

Denna uppgift är relativt enkelt att följa, men ta god tid vid varje steg och se till att ni förstår! I denna applikation skall vi inte lägga stor vikt på effektivitet, optimering eller databasstruktur utan helt fokusera på MVC och hur det fungerar.

Jag känner inte att vi behöver titta på den här övningen då den är steg för steg.
Vi går även igenom övningen nu på fredag.
Men givetvis svarar vi på frågor eller förtydligar om något är oklart!

Applikationen

Innan vi börjar måste vi ha något att arbeta i. Skapa ett nytt MVC.Core projekt genom *File -> New -> Project -> Installed -> Visual C# ->*.NET.Core ->ASP.NET Core Web Application.
Döp projektet till "Storage"-> OK -> Välj *Web Application (Model View Controller)*

Modellen:

Det första ni skall göra är en modell för applikationen, den kommer vara relativt enkel och spegla en vara eller ett föremål i lagret.

Skapa en klass i mappen *Models* som ni döper till "*Product*"
Skapa nu följande auto *properties*:

int Id
string Name
int Price
DateTime Orderdate
string Category
string Shelf
int Count
string Description

Skapa våra kontroller

Nu när vi har vår modell Product klar så bör vi, innan vi skapar vårt user interface (front-end), ha ett sätt att kommunicera mellan *back-end* och *front-end*, alltså en *kontroller*.

1. Bygg projektet med control-b
2. Högerklicka på *Controllers-mappen*
3. Välj *Add, New Scaffolded item*
4. Välj "*MVC Controller with views using Entity Framework*"
5. Välj er *Product* klass som *modellklass*
6. Välj att skapa ett nytt Context genom att trycka på +
7. *Controllern* bör ha fått namnet *ProductsController*.
8. Tryck Ok

Visual studio genererar en hel del kod automatiskt åt oss. Gå in i den koden som har skapats för att se vad som är gjort. Det gör att ni får en bättre förståelse för hur allt hänger ihop!

Klassen StorageContext har skapats. Det är en klass som använder sig av EntityFramework (ORM mappar mellan databasen och vår C# kod).
Den har ett property av typen DbSet<Product> Product som vi använder oss av för att hämta data från databasen (Titta på den genererade koden i ProductsController).

Konfigurationen har den fått i Program.cs (Startup.cs när ni tittar på E-L).

```
services.AddDbContext<StorageContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("StorageContext")));
```

I Program.cs har StorageContext registrerats som en service. Med andra ord lagts till i vår dependecy injector container så vi har åtkomst till den i resten av applikationen (Titta på konstruktorn i ProductsController).

Det är i options som vi gör våran konfiguration. Här bestämmer vi att vi ska använda oss av en Sql databas.
Som argument skickas connection strängen med som vi hämtar från våran appsettings.json där den har lagts till.
Slutligen skickas konfigurationen som ett argument till konstruktorn i StorageContext.

För de som vill ha mer på fötterna går Scott går igenom hela processen steg för steg i början av sin E-L som ni har som komplement i E-L listan för mvc. Var medvetna här att han kör en äldre version av Core. Men själva principen är den samma. I verkligheten jobbar man inte heller alltid men det senaste så egentligen är der bara bra att se på olika implementationer.

Generera Databasen med Entity Framework

Nu är det hög tid att generera databasen!

1. Öppna package-manager-console (PMC) via Tools -> Nuget/Library package manager -> Package Manager Console
2. Skriv därefter in "*Add-Migration Namn*" där namn är ett beskrivande namn. Förslagsvis använder ni *Init* vid en första *migration.*
   En ny mapp i projektet kommer nu ha skapats: Migrations. Där i kommer även ett script genererats.
3. Därefter kör vi direkt *Update-Database* kommandot för att uppdatera och skapa vår *databas.* Först här körs scriptet och databasen skapas.

4. Öppna -> Server object explorer och öppna upp er databas högerklicka på tabell namnen och välj show data.
5. Nu går det bra att fylla på med lite grunddata. Det går alldeles utmärkt att fylla på med några produkter direkt i tabellerna från Server object explorer.
   Ni skriver in lite data direkt in i tabellen. (Högerklicka på product => view data)

Titta igenom koden som har genererats och se vad den faktiskt gör. Gå igenom processen att skapa en ny produkt från att användaren trycker på Create knappen i index vyn tills det är klart. Titta både på vyerna och i controllern och följ hela kedjan. Viktigt att ni följer flödet och ser vad som händer! Lite som vi gjorde gemensamt igår.

Lägg till navigation

1. Lägg till navigation i *headern* för att komma direkt till *Products/Index*.
2. Gå in i Startup.cs och ändra routingen vi vill inte längre att Home ska vara default Kontroller.
3. Lägg till data annotations för att skapa restriktioner för vad som kan matas in. Tex min max värden (Range), titta gärna vad det finns för olika attribut att välja på.
4. Låt Category vara obligatoriskt med [Required] attributet.
5. Lägg även till [DataType(DataType.Date)]] för att snygga till orderdate.
6. Eftersom vi har förändrat hur databasen ser ut med våra nya regler måste vi lägga till en ny migration med Add-Migration “NamnPåMigrationen”.
7. Sen Update-Database för att köra scriptet som genererades i föregående punkt.

ViewModel

Vy modeller använder vi oss för att definiera vad för information (data) vi behöver i den aktuella vyn. Vi är kanske inte intresserade av att ha en exakt representation av det vi lagrat i databasen i vyn.
Utan vill visa upp annan data eller del av datan vi har lagrat.

Vi skapar upp en ny vy modell:

1. Skapa en klass i mappen *Models* som ni döper till “*ProductViewModel*” med följande *properties*:
   string Name
   int Price
   int Count
   int InventoryValue

2. Skapa ett nytt ActionResult i Products kontrollern
3. Skapa upp en ny instans av en lEnumerable<ProductViewModel>()
4. Hämta alla produkter från databasen och mappa den data vi ska använda till vår nya ProductViewModel

5. Summera det totala värdet för varje enskild produkt och tilldela InventoryValue resultat.
6. Skapa en ny vy.
    a. Gå till er *Controller* i koden.
    b. Högerklicka i metoden och välj AddView
    c. I wizarden får ni en mängd val, välj det som passar, i detta fallet List.
    d. Se till att **INTE** ange något i Data context class:
    Vy modellen har inget att göra med databasen! Det är bara en modell för vyn.
    e. Skriv in namnet på vyn och klicka på add. En .cshtml fil kommer genereras i views-mappen för controllern.
    f. Titta på koden som genererats vyn är nu hårt typad

Filtrera

Nu vill vi kunna filtrera produkterna efter kategori.

1. Skapa ett ny metod i kontrollern som tar en sträng som parameter med namnet category.
2. Filtrera produkterna med hjälp av värdet category.
3. Skicka tillbaka resultatet till vyn.
4. Skapa ett sökformulär på sidan med tillhörande knapp. För att kunna söka på produkter. Använd er av Form Tag Helpers.
    Glöm inte att specificera vad parametern har för namn! (vad model bindern ska försöka binda på)

Passa gärna på att experimentera och lägg till ytterligare funktionalitet ni skulle tycka kunna vara användbar!

Extra

Lite överkurs för de som har tid! Skapa en dropdown meny med de kategorier som finns i databasen. Visa upp den bredvid sökfältet.

I sökfältet vill vi nu söka på produktnamn istället. När vi har valt en kategori så filtreras resultatet både på vald kategori och produktnamn.

Om vi ej valt någon kategori enbart på produktnamn. Och givetvis enbart på vald kategori om vi ej skrivit in något i sökfältet.

Tips! Skapa en IEnumerable<SelectListItem> samt utöka modellen med de värden som vyn behöver.

Lycka Till!!!!