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.

Applikationen

Innan vi börjar måste vi ha något att arbeta i. Skapa ett nytt MVC5 projekt genom *File -> New -> Project -> Installed -> Visual C# -> Web* -> Döp projektet till "Storage"-> OK -> Välj *MVC*

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.

1. Skapa en klass i mappen *Models* som ni döper till "*Product*"
2. Skapa nu följande *properties*:
    a. *int Id*
    b. *string Name*
    c. *int Price*
    d. *string Category*
    e. *string Shelf*
    f. *int Count*
    g. *string Description*

Data access, Context

Efter vi har vår modell måste vi skapa en data-access via en databaskontext.

1. Skapa mappen *DataAccessLayer* via *SolutionExplorer* i *Visual Studio*.
2. Skapa klassen *StorageContext* i den nya mappen.
3. I *StorageContext.cs* lägg till "*using System.Data.Entity;*" (Om denna saknas så får du manuellt lägga till den i projektet genom att högerklicka *'References'* i er *solution explorer.* Om detta händer kan det även vara värt att kontrollera att ni har *Entity Framework* installerat i *Nuget Package Manager*)
4. Gör sedan så *StorageContext* att klassen ärver från "*DbContext*"
5. Gör en publik *konstruktor* för *StorageContext* som anropar *baskonstruktorn* med inputsträngen "*DefaultConnection*"
6. Skapa sedan propertyn "public DbSet<Models.Product> Products { get; set; }"

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 "Enable-Migrations"
3. Gå in i den genererade Migrations.Configuration-klassen
4. I seed metoden skall vi fylla på med lite grund-data som vi vill ska finnas i databasen redan från början. Gör detta genom att ta inspiration från den kod som finns i de genererade kommentarerna fast applicera det på era egna *Klasser* och *DbSet* (Glöm inte att lägga till er modell som *using statement*).
5. När vi är nöjda med seed-metoden skall vi använda oss av kommandot "*Add-Migration Namn*" där namn är ett beskrivande namn. Förslagsvis använder ni *Initial* vid en första *migration*.
6. Därefter kör vi direkt *Update-Database* kommandot för att uppdatera och skapa vår *databas*.

Skapa våra kontroller

Nu när vi har allt bakomliggande vi behöver så bör vi, innan vi skapar *front-end*, ha ett sätt att kommunicera mellan *back-end* och *front-end*, alltså en *kontroller*.

1. Högerklicka på *Controllers-mappen*
2. Välj *Add-Controller*
3. Välj "*MVC5 Controller with views using Entity Framework*"
4. Välj er *Product* klass som *modellklass*
5. Välj er *StorageContext* som *Contextklass*
6. *Controllern* bör ha fått namnet *ProductsController*.

Lägg till navigation

Lägg till navigation i *headern* för *Products*.

Views

För att skapa egna *views* går ni till väga på följande sätt:

1. Gå till er *Controller* i koden.
2. På valfri plats, efter “public ActionResult Index (){...}” lägger ni till en egen ActionResult. “public ActionResult Electronics() { return View();} “
3. Högerclicka på Test och välj AddView
4. I wizzarden får ni en mängd val, välj det som passar, i detta fallet List.
5. Skriv in namnet på viewn och klicka på add. En .cshtml fil kommer genereras i views-mappen för controllern.
6. Skriv in en LINQ sats och skicka med till vyn som modell, exempelvis

```
var model = db.Products.Where(i => i.Category == “Electronics”).ToList();
return View(model);
```

7. Skapa skapa en vy som endast visar varor som är slut i lagret.
8. Hitta på en egen vy som skulle vara praktisk i sammanhanget.