Exercise 11: MVC – Inventory system

This exercise is fairly easy to follow, but take your time whit each step and make sure that you understand. In this application we won't put all that much weight on effectivity, optimization or database structure but instead focus completely on MVC and how it works.

Application

Before we start we will need a project to work in.
Create a new MVC5 project by going to: *File* ->
New -> *Project* -> *Installed* -> *Visual C#* -> *Web* -> ASP.NET Web Application -> enter a name for the project -> choose MVC (no authentication)

The Model:

The first thing you will do is to create a model for the application, it will be fairly simple and represent a product in the inventory.
1. Create a class in the Models folder and call it "Product"
2. Now create the following *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

Once we have our model we have to create a data-access by means of a database context.
1. Create the folder *DataAccessLayer* through *SolutionExplorer* in *Visual Studio*.
2. Create the *StorageContext* class in the folder.
3. Add "*using System.Data.Entity;* " in *StorageContext.cs* (If you don't have that using, right click '*References*' in the *solution explorer*. You can also check that *Entity Framework* is installed by checking *Nuget Package Manager*)
4. Make sure that the *StorageContext* class inherent "*DbContext* "
5. Create a public *constructor* for the *StorageContext* that calls the *base constructor* with the input string "*DefaultConnection"*
6. Lastly create the following property "*public DbSet<Models.Product> Products {get; set;}*" in the "*StorageContext*" class

Generate the Database with Entity Framework

Now it is high time to generate the database!
1. Open the package-manager-console (PMC) through Tools -> Nuget/Library package manager -> Package Manager Console
2. Now write "Enable-Migrations" into the PMC
3. Go to the newly generated Migrations.Configuration-class
4. In the seed method we will add some basic data that we want to have in the database from the start. We do this by mimicking the code that exists in the pre-generated comment, but using our own C*lass* and *DbSet* (don't forget to add your model as a *using statement*)
5. When we are satisfied with the seed-method we will use the "*Add-Migration [Namn]* " command where [Name] is a descriptive name for the migration. A common practice is to call the first migration *Init*
6. After that we directly use the *Update-Database* command to update and create our database.

Create our controller

Now we have almost all the back-end we need, we only need to add the part that actually handles the communication between the *front-end* and *back-end*, namely a controller.

1. Right click the *Controllers-folder*
2. Choose *Add-Controller*
3. Choose “*MVC5 Controller with views using Entity Framework* ”
4. Choose your *Product* class as M*odel Class*
5. Choose your *StorageContext* as *Context class*
6. The *Controller* should have gotten the name *ProductsController* .

Add navigation

Add navigation in the header for Products

Views

To create your own *views*:

1. Go to your *Controller* in the code
2. Add the ActionResult “*public ActionResult Electronics(){return View();}*” after “*public ActionResult Index (){...}”*
3. Right click *Electronics* and choose AddView
4. In the wizard you will get a number of different chooses, in this instance we need a List.
5. Enter a name for the view and click add. A .cshtml file will be added to the views-folder corresponding to the controller.
6. Write a LINQ query and send it to the view as a model for example:

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

7. Create a view that only shows products that are out of stock.
8. Create your own View that would be practical in this context.