WCF del 1: MVC

Denna del går ut på att programmera den första delen av denna uppgift, ”Telefonboken”. Det är alltså det projekt ni börjar på här som ni ska fortsätta med på WCF del 2.

Denna del kan verka något lång men det är utförliga beskrivningar och inte lika mycket arbete. Tänk även på att det är arbete som ni slipper göra i del 2

Er uppgift är att skapa en Webapplikation i MVC som ska fungera som en enkel Telefonbok. Telefonboken ska stödja följande funktionalitet:

- Lägg till kontakt
- Editera en kontakt
- Ta bort en kontakt
- Sök efter kontakter på, förnamn, efternamn, telefonnummer.

För det behöver ni:

1. Definiera en Modell för objektet ”Kontakt” som innehåller information om en kontakt.
2. Använda er av en fördefinierad statisk klass ”Lagring” som ska fungera som telefonbokens lagring *. Denna klass finns under Models i det skalprojekt som ni kommer att använda er av och denna innehåller metoderna HämtaAllaKontakter, LäggTillKontakt, TaBortKontakt och ÄndraKontakt.
3. Skapa en Controller för att styra och delegera flödet i applikationen.
4. Samt skapa ett antal Views för GUI:et. Här kommer ni att behöva ytterligare bekanta er med och öva era praktiska kunskaper med ”Razor” som är en motor som är till för att förenkla syntaxen för att skapa webb-applikationens sidor(Views). Mer om detta sen.

* I den kommande WCF del 2 kommer denna lagringsklass bytas ut mot en extern källa genom användningen av en så kallad WCFl Service men det behöver ni inte tänka på i denna del.

Uppgift 1 – Hämta och öppna skalprojektet.

Öppna skalprojektet "Telefonbok" som finns med i denna mapp.

Var god läs dokumentet ”Introduktion_MVC_WCFServices” som finns att tillgå i samma mapp som detta dokument innan ni ger er på del 2.

I detta skalprojekt har jag även inkluderat en färdig klass ”Lagring” och påbörjat klassen Kontakt som ni i denna del ska färdigställa. Dessa finns under mappen Models.

Uppgift 2 – Skapa en modell för Kontakt.

Som du har sett så finns redan klassen Kontakt under mappen Models men för tillfället innehåller den endast egenskapen Id som är till för att ge varje kontakt ett unikt ID-nummer. För att göra telefonboken meningsfull så vill vi nu även att en Kontakt ska innehålla egenskaperna: Förnamn, Efternamn och Telefonnummer. Definiera dessa egenskaper med get-set funktionalitet.

Kodexempel:
public string Förnamn { get; set; }

Observera att denna klass endast behöver dessa egenskaper och inget annat. En konstruktor behövs inte eftersom default konstruktorn funkar fint.

Uppgift 3 – Lagringsutrymmet.

Nu när du har definierat klassen Kontakt så är det dags att titta närmare på det lagringsutrymme som Kontakt-objekten ska lagras i.

I skalprojektet så finns det en färdig klass ”Lagring” (som ligger i mappen Models) som kommer att sköta lagring och hantering av kontakter. Denna klass har deklarerats som statisk och publik så att man kan använda sig av den utan att behöva instansiera något objekt. Denna implementation innebär att de kontakter som ni sparar i telefonboken endast sparas i minnet och kommer därför att försvinna när ni stänger programmet. Därför ska ni i del 2 (inte nu) skapa en Service som sparar ner dessa på XMLl filer men just nu nöjer vi oss med denna implementation.

Kodexempel på statisk klass:

```
public static class Lagring
{
}
```

Öppna nu Lagringsklassen och läs kommentarerna så att ni förstår hur den fungerar. Denna klass använder sig alltså av en List som internt lagringsutrymme som i detta fall har fått namnet ”internLagring”. Den innehåller även en statisk variabel ”id” som är till för att ge Kontakt-objekten unika ID-nummer när de läggs till i internLagring. Ni kan se i metoden ”LäggTillKontakt” hur denna variabel används som en räknare som ger Kontaktobjekten unika ID-nummer.

Ett antal metoder är implementerade för att stödja CRUD(Create, Read, Update, Delete) funktionalitet i klassen. Metodnamnen är självförklarande och stödjer denna funktionalitet:

- Lägg till kontakt
- Ta bort en kontakt
- Editera en kontakt
- Hämta alla kontakter

Nedan följer metodsignaturerna för dessa metoder:

public IEnumerable<Kontakt> HämtaAllaKontakter()

public void LäggTillKontakt(Kontakt kontakt)

public void TaBortKontakt(int id)

public void ÄndraKontakt(Kontakt kontakt)

Dessa metoder använder sig av List-objektets egna metoder samt LINQ.

Som ni märker så finns det ingen metod för att söka efter kontakter. Det är tänkt att ni ska implementera den metoden själva med LINQ och lämplig metodsignatur.

Uppgift 4 – Skapa GUI:et

Nu har ni definierat klassen Kontakt och bekantat er med lagringsutrymmet till Telefonboken och det är dags att börja med webb-applikationens sidor(Views) som utgör applikationens GUI.

Det enklaste sättet att göra detta på är att först skapa en Controller som ni kan kalla för KontaktController. Detta gör ni genom att högerklicka på mappen Controllers och trycka Add -> Controller. Välj Empty controller. När ni skapat en ny controller så måste ni även ändra applikationens startpunkt genom att öppna filen ”RouteConfig.cs” under mappen ”App_Start” i Solution Explorer. Där kommer ni att se en rad som ser ut såhär:

```
defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
```

Byt ut ”Home” mot ”Kontakt” för att peka applikationen till er nya KontaktController istället.

I denna controller kan ni sedan skapa en viss typ av metod med return-typen ActionResult. En sådan metod har alltså följande metodsignatur:

```
public ActionResult VyNamn ()
{
}
```

Denna metod anropas då man i Web-applikationen navigerar till URL:
WebAppensURL/ControllerNamn/VyNamn

Skriver man sådana metoder så kan man låta C# generera en view med samma namn som då automatiskt hamnar under Views/Kontakt. För att göra detta högerklickar du på metodsignaturen och trycker Add -> View. En dialogruta dyker upp som ger ett antal valmöjligheter.
Välj Create Strongly typed View och välj sedan er Kontakt-klass under ”Model class”.
När ni väljer Create Strongly typed View så betyder det att denna View förväntar sig att objekt av denna typ skickas med till sidan för att där kunna visas/modifieras.
När ni exempelvis gör sidan LaggTillKontakt så ska ni alltså skicka med ett tomt Kontakt-objekt till View:en som där kan modifieras med Förnamn, Efternamn och Telefonnummer och sedan skickas med formuläret för att spara i Lagring-klassen.

Ett tomt Kontakt-objekt skapas och skickas till en View såhär:

```
public ActionResult LaggTillKontakt()
{
        Lab3_Telefonbok_FörnamnEfternamn.Models.Kontakt kontakt = new
        Lab3_Telefonbok_FörnamnEfternamn.Models.Kontakt();
        return View(kontakt);
}
```

(För att slippa skriva ut hela referensen till Kontakt klassen som i exemplet ovan kan ni lägga till ett using statement i er Controller: using [ErtProjektnamn].Models;)

Antingen kan ni välja att göra en tom View eller använda så kallade Scaffold Templates. Detta bör ni läsa om i filen ”Introduktion_MVC_WCFServices” som finns på kurssidan innan ni går vidare. Om ni väljer Scaffold templates så genereras ett färdigt formulär, annars så kan ni göra detta manuellt. Om ni väljer Scaffold template så se till att ni faktiskt förstår vad det är för kod som genereras.

Det finns en färdig metod public ActionResult Index() som kommer utgöra applikationens startsida. Börja med att skapa en View för denna på samma sätt som beskrivet ovan och notera den nya filen under Views/Kontakt. Denna view ska visa upp alla kontakter i Telefonboken. I Controllern ska ni alltså skicka med en lista av alla Kontakter i Lagring till View:en som hämtas från Lagringsklassen.

Skapa sedan nya metoder och View:er för varje funktion som Telefonboken stödjer. Med undantag för TaBortKontakt, denna funktion behöver ingen egen View utan behöver endast en ActionResult metod i Controllern. Denna metod kan sedan anropas via en länk i Index View:en där man samtidigt skickar med id-numret för den kontakt som ska tas bort.

Metodsignaturer:

```
public ActionResult LaggTillKontakt()
public ActionResult AndraKontakt(int id)
public ActionResult SokKontakter()
public ActionResult TaBortKontakt(int id)
```

Obs! Om ni valde Scaffold Template så kom ihåg att välja en template som stämmer överrens med den funktion som View:en kommer att ha (Ex. LäggTillKontakt ska ha Create).

När ni har gjort detta så kan ni testa er applikation och lägga till diverse länkar på view:erna så att man kan navigera från Index till LaggTillKontakt och SokKontakt. Samt från de olika sidorna och tillbaks till Index. Se dokumentet på kurssidan för att läsa om hur ni gör detta och annat med Razor.

Länkar till AndraKontakt och TaBortKontakt ska ni lägga in på Index-sidan i varsin kolumn för varje rad som visar en kontakt.
På det sättet kan ni använda er av Kontakt-objektets id som respektive metod efterfrågar där ni sedan kan hämta eller ta bort Kontakt objektet.

Uppgift 5 – Implementera funktionaliteten mot Lagringsutrymmet.

För att få funktionaliteten att fungera med er statiska Lagringsklass så måste ni på något sätt kunna ta emot den information som skickas från vid submit på formulärerna i Vyerna när ni exempelvis lägger till en kontakt.

För att göra detta så gör ni en till metod i KontaktController för varje view där något skickas från formulär. Dessa metoder ska ha samma metodsignatur som den tidigare metoden som är kopplad till Viewen med skillnaden att [HttpPost] ska skrivas ovanför denna och innanför parenteserna så ska en variabel definieras med samma objekttyp som skickas från den specifika vyn (int för TaBortKontakt och Kontakt för resterande)

Denna variabel kommer innehålla det som skickades från formuläret i vyn och i kod-blocket ska den statiska Lagringsklassen som ni gjorde tidigare användas för att utföra den begärda åtgärden i Telefonbokens lagring.

Exempel:

```
[HttpPost]
public ActionResult LaggTillKontakt(Kontakt kontakt)
{

        Lagring.LäggTillKontakt(kontakt);
        return RedirectToAction(”ActionNamn”);

}
```

Ni byter ut ”ActionNamn” i exemplet ovan mot namnet på den Action-metod ni vill att man ska anropa efter att en kontakt lagts till.

För resterande funktioner: TaBortKontakt, EditeraKontakt, SokKontakter så använder ni på samma sätt de andra metoderna som ni skapade i Lagringsklassen.

I [HttpPost]-metoden för SokKontakter ska ni använda er av er sökmetod i Lagring-klassen där ni med hjälp av LINQ söker ut kontakter som matchar sökningen. Det finns olika sätt man skulle kunna utforma SokKontakter-Vyn men detta får ni själva bestämma hur ni vill göra. Det enklaste är nog dock att skapa ett formulär med endast en TextBox i vyn där användaren får skriva in valfri uppgift och sedan låta sökmetoden i Lagringsklassen söka bland alla kontakter med denna sträng på samtliga egenskaper (Förnamn, Efternamn, Telefonnummer) och returnera de kontakter som matchar strängen. Ni kan även välja om ni vill visa sökresultatet i en ny vy eller i samma vy (SokKontakter). Experimenterande uppmuntras!

Tips! För att göra sökfunktionen mer robust så tänk på att ni kan använda List-metoden Contains.

Lycka till!