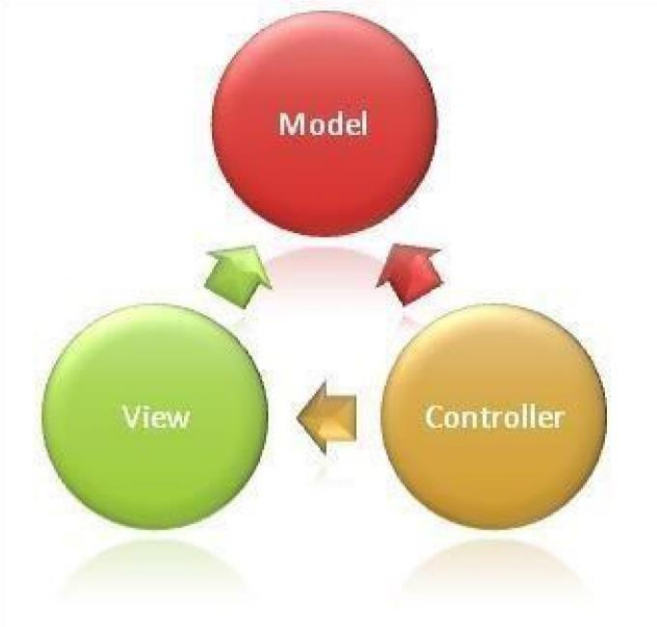
Av **Erik ”Zkall” Svensson**
@esdev.se

ASP.NET – MVC Model View Controller C#



3 objekt

Model - Skapar modeler som med hjälp av LINQ automatiskt skappar databastabeller. Model kan kallas både bakifrån koden i controllern och i viewen.

View - Där vi slänger in all html, det är här som man skriver all kod som syns. Kan kalla models med medskickade parametrar och skicka med modeler i post.

Controller – Controllern är där vi skriver all C# kod, det är här logiken sker. Kan komma åt models härifrån och man kan skicka in dom i en view som parameter.

Model

En model deklareras i en class. En klass är som en mall där man lägger in mallar för objekt.

```
[Table("UserProfile")]
3 references
public class UserProfile
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    0 references
    public int UserId { get; set; }
    2 references
    public string UserName { get; set; }
}

2 references
```

Så ser en model ut för en class. Key deklarerar att det är index (vilket som egentligen inte behövs då det genereras automatiskt. Fokusera på userid och username. Vi behöver även lägga till [Table(”tabellnamnIDatabasen”)] så att den automatiskt generar ett table där. Ett tips här är att skriva propp och trycka tabb. Då skrivs hela get set uts automatiskt.

Dbcontext

```
5 references
public class UsersContext : DbContext
{
    2 references
    public UsersContext()
        : base("DefaultConnection")
    {
    }

    2 references
    public DbSet<UserProfile> UserProfiles { get; set; }
}
```

När vi är klara med alla modeller så ska vi lägga till en dbcontext. Utan db contexten så får vi ingen databas skapning och våran bind till modelerna funkar inte, dom blir då bara tomma modeler.

Skriv som ovan och deklarera en DbSet<classnamn> klassnamnen { get; set; }

Gör detta en gång för varje class du har och glöm inte bort defaultconnection parametern. Mer om databasar i en senare artikel.

Viewmodel

Viewmodel är vad vi kallar modeler som inte skrivs i databasen och endast används i en view för att kunna ha flera olika modeller i samma. Det deklarerars precis som models fast du skippar Table[”namn”]. Annars än det funkar det precis som vanligt.

Controller

```
namespace MvcApplication1.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public ActionResult Index()
        {
            ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

            return View();
        }

        0 references
        public ActionResult About()
        {
            ViewBag.Message = "Your app description page.";

            return View();
        }

        0 references
        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
```

En controller ser ut på följande sätt.

Ett actionresult är en länk till en view från controllern. Om du döper den till index så kommer den att kommas åt genom controllername/actionresult i detta fall Home/Index För att komma åt models i controllern skriver vi som så här.

```
UsersContext db = new UsersContext();
```

Du kan skriva den direct under controller classen, då slipper du deklarera den flera gånger. Ända gången du behöver deklarera om den är om du ska göra dubbla loopar... men det är något vi inte går in på än.

```
UserProfile _user = db.UserProfiles.FirstOrDefault(x => x.UserId == 0);
```

Skriver du för att deklarera och ge värde till en user variabel. (detta är det som är inbyggt när du kör internet application templatet.

X => X.userid kallas för LINQ och är en av flera sätt som man kan kommunicera med en databas med.

Det finns även SQL-commandon och Dataset. Dom är gårdagens teknologi och du bör fokusera på att lära dig LINQ. Linq är enkla uttryck som förvandlas till sql utan att du behöver skriva det själv. Jag dubblade min arbetshastighet när jag kom in i LINQ och lärde mig det ordentligt.

Du kan ta ut en singel variabel med ett uttryck, det som den ovan säger är kolla i userprofiles efter data som har attributen userid satt till 0. Du deklarerear då en var variabeln eller en class variabel.

Du kan även ta ut dom i en list, detta gör du genom att skriva:

```
var result = db.UserProfiles.ToList();
```

För att skicka en model till en view så skriver du return View(variabel-model)

```
return View(result);
```

Om du högerklickar på return View() får du fram add view. Det måste du trycka på första gången för att initialisera viewen. Mer om det i:

Spaghettikod

Spaghettikod är det vi kallar oläslig kod när du har 10 000 rader med kod utan användande av funktioner eller regions. Region skriver du så här

```
#region apa
if (ModelState.IsValid)
{
    // Attempt to register the user
    try
    {
        WebSecurity.CreateUserAndAccount(model.UserName, model.Password);
        WebSecurity.Login(model.UserName, model.Password);
        return RedirectToAction("Index", "Home");
    }
    catch (MembershipCreateUserException e)
    {
        ModelState.AddModelError("", ErrorCodeToString(e.StatusCode));
    }
}

// If we got this far, something failed, redisplay form
return View(model);
#endregion
```

Function skriver du så här, givetvis kan den heta vad som helst, som t.ex. apa!

```
0 references
public void function()
{
    //kod
}
```

Spaghettiexempel:

```
public ActionResult sök()
{
    CheckIfLoggedIn(); //dubletter en sök i loggedin och en i home. Bättre struktur önskbart.
    if(användarnamns == "")
    {
        Response.Redirect("../home/index");
    }

    string query = "";

    try
    {
        if(Request.QueryString["query"] != null)
            query = Request.QueryString["query"];
    }
    catch
    {

    }

    if(query != "")
    {
        SqlCommand meinCommand = new SqlCommand();
        meinCommand.Connection = connection;
        meinCommand.CommandText = "SELECT * FROM maträtter";
        connection.Open();
        SqlDataReader meinReader = meinCommand.ExecuteReader();
        bool didwedoit = false;
        while(meinReader.Read())
        {
            if(meinReader["namn"].ToString().ToLower().Contains(query.ToLower()))
            {
                int kcal = 0;
                int pris = 0;
                int protein = 0;
                int kolhydrater = 0;
                int fett = 0;

                try
                {
                    kcal = Convert.ToInt32(Convert.ToDecimal(meinReader["kcal"].ToString().Replace(".", ",")));
                }
                catch { }
                try
                {
                    pris = Convert.ToInt32(Convert.ToDecimal(meinReader["pris"].ToString().Replace(".", ",")));
                }
                catch { }
                try
                {
                    protein = Convert.ToInt32(Convert.ToDecimal(meinReader["protein"].ToString().Replace(".", ","))
                }
                catch { }
                try
                {
                    kolhydrater = Convert.ToInt32(Convert.ToDecimal(meinReader["kolhydrater"].ToString().Replace(".
                }
                catch { }
                try
                {
                    fett = Convert.ToInt32(Convert.ToDecimal(meinReader["fett"].ToString().Replace(".", ",")));
                }
                catch { }
                ViewBag.resultat += "<a href='maträtter?id=" + meinReader["id"].ToString() + "'><div onclick='locat
                didwedoit = true;
            }
        }
        connection.Close();
        if(didwedoit == false)
        {
            ViewBag.resultat = "Inget resultat funnet för \""+query+"\"";
        }
    }

    return View();
```

View

Här har vi då källan för vad vi ser, i föregående kapitel skrev jag om att du kan skicka en model till en view. För att kunna komma åt den så måste du deklarera modelen längst upp i dokumentet.

```
@model MvcApplication1.Models.RegisterModel
```

Efter det skirver du Model.Xattribut när du ska kommma åt den, den kommer med intellsense (autotype förslag) föreslå dom som finns tillgängliga. Kort och gott allt i ett paket, det är därför MVC är så grymt mycket bättre en webforms och sql commandon.

Det kallas razer syntax, C-Sharphtml (.cshtml) och det fungerar precis som vanligt med html skrivandet. Skillnaden är att du kan blanda in c# kod i viewen och på det sättet göra saker som du bara kan i code behind i web forms.

När du skriver c# kod så deklarerar du det med @. Se exempel under.

```
@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary()

    <fieldset>
        <legend>Registration Form</legend>
        <ol>
            <li>
                @Html.LabelFor(m => m.UserName)
                @Html.TextBoxFor(m => m.UserName)
            </li>
            <li>
                @Html.LabelFor(m => m.Password)
                @Html.PasswordFor(m => m.Password)
            </li>
            <li>
                @Html.LabelFor(m => m.ConfirmPassword)
                @Html.PasswordFor(m => m.ConfirmPassword)
            </li>
        </ol>
        <input type="submit" value="Register" />
    </fieldset>
}
```

Så här ser en vanlig form ut i en view. Du kan skriva ut form elements på 2 olika sätt.

Sätt 1: Se ovan. @Html.Label**For**(x => x.username) (observera **for**, detta är i linq. X kan heta vad som helst, du kan döpa den till apa om du vill så länge det är samma namn på andra sidan av =>.

Sätt 2: @Html.Label(”namn”, (object)value). Detta sättet använder inte linq och bindar inte med modelen så det är inte lika optimalt att använda. När du skickar detta med post måste du skriva

Request[”namn”] för varje variabel i controllern och då får vi spaghettikod, medans du med en model bara kan skriva model.Xattribut