

# Webudvikling med Visual Studio og C#

Carsten Breum



Carsten Breum

---

# Web-udvikling

## – med Visual Studio 2005 og C#

---

---

Web-udvikling – med Visual Studio 2005 og C#  
© 2007 Carsten Breum og Ventus Publishing ApS  
ISBN 978-87-7061-154-1

# Indholdsfortegnelse

1.	Forord	6
2.	Den første site (Hello World)	7
3.	Web Forms	13
3.1	En forklaring på Hello World	13
3.2	En input og output form. Simpel registrering	19
3.3	En custom validerings kontrol	37
3.4	AdRotator kontrollen	41
3.5	Custom Controls	45
3.5.1	En simpel Custom Control	45
3.5.2	En Composite Custom Control	48
4.	Cookies	53
5.	En mini shop	58
5.1	Databasen Mini Shop	58
5.2	Oprettelse af databasen	59
5.3	Tabellerne i mini shoppen oprettes	59



 **MTHøjgaard**

**BEDRE  
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



5.4	Views oprettes i Mini Shop databasen	60
5.5	NET Mini Shoppen oprettes	61
5.6	ADO.NET. DataSet	64
5.7	SqlDataSource og GridView	64
5.8	GridView kontrollen konfigureres	71
5.9	Session variabel. Shopping Cart/Indkøbsvogn	78
5.10	En Web user control. Indkøbsvognen	80
5.11	Orderen modtages	88
6.	Exception handling	97
7.	Debugging	103
8.	Web services	110
8.1	En simpel web service	110
9.	Afslutning	121
10.	Index	122
11.	Noter	124



### Ses vi til DSE-Aalborg?

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**



# 1. Forord

Denne bog henvender sig til alle der skal igang med web udvikling med Visual Studio 2005. Bogen vil gradvist indføre dig i hvordan du med Visual Studio .NET kan opbygge en website ved hjælp af de mange kontroller der er i toolboxen. I bogen benyttes C# som programmeringssprog, idet det er som skræddersyet til .NET. Eksemplerne i bogen benytter SQL Server 2005 Developer Edition. Du kan hente en gratis version kaldet *SQL Server 2005 Express Edition* via Microsofts hjemmeside (<http://www.microsoft.com/sql/editions/express/default.msp>) som også kan benyttes. Den gratis version er begrænset i størrelse, men er attraktiv for det lille firma eller for firmaer, der har behov for at distribuere en database sammen med deres webapplikation. Eksemplerne vil dog være relativt simple og Access eller andre databaser som MySQL kunne også benyttes.

Bogen vil ikke være en lærebog i C# og det forudsættes at læseren har nogen kendskab til udvikling, men selv en uerfaren skal jo starte et sted og måske er de mange eksempler et godt udgangspunkt til efter denne bog at kaste sig over en egentlig C# bog.

I bogen benyttes for det meste de engelske fagudtryk for at undgå at gøre vold på det danske sprog. Jeg mener ikke at man for enhver pris skal oversætte alle ord til sit eget sprog og specielt inden for programmeringssprog og teknologi synes jeg man skal lade være og istedet bruge de originale engelske udtryk. Nogle ord som f.eks. control er mange steder oversat til kontrol ligesom andre oplagte ord er oversat.

Fokus vil være på at gennemgå basale kontroller til opbygning af en website. Ting som Master Pages, Membership og Role Management, Themes og Skins og Web Parts er alle relevante, men jeg har ikke fundet plads til at gennemgå dem i denne bog og vil istedet henvise til anden litteratur, som f.eks. Professional ASP.NET 2.0 fra forlaget Wrox.

## 2. Den første site (Hello World)

I dette kapitel vil jeg gennemgå hvordan du opretter en website med Visual Studio 2005. Istedet for at komme med en masse teori og forklaringer om C#, Web forms etc. synes jeg at vi kaster os ud i et praktisk eksempel med det samme og gemmer de mange forklaringer til senere.

Som udgangspunkt skal du installere følgende på din computer:

1. Windows XP Professional eller Windows Vista
2. (Internet Information Server )
3. Visual Studio 2005

Efter installation af Windows XP har du mulighed for at addere ekstra windows komponenter og her kan du vælge at installere Internet Information Server. Under udvikling af simple eksempler kan den web server der følger med Visual Studio benyttes, men så snart der begynder at komme flere afhængigheder til web applikationen er det nødvendigt at udvikle direkte på IIS'en. Når det er på plads installerer du Visual Studio 2005. Til eksemplerne i bogen er Visual Studio 2005 Team Edition benyttet. Du kan også benytte den gratis version kaldet Visual Studio Express eller blot Visual Studio Professional.

Pyh, efter en rum tid og den obligatoriske reboot er vi nu klar til at lave vores første web site.



**CISO Conference**  
Produced by **Inspired**

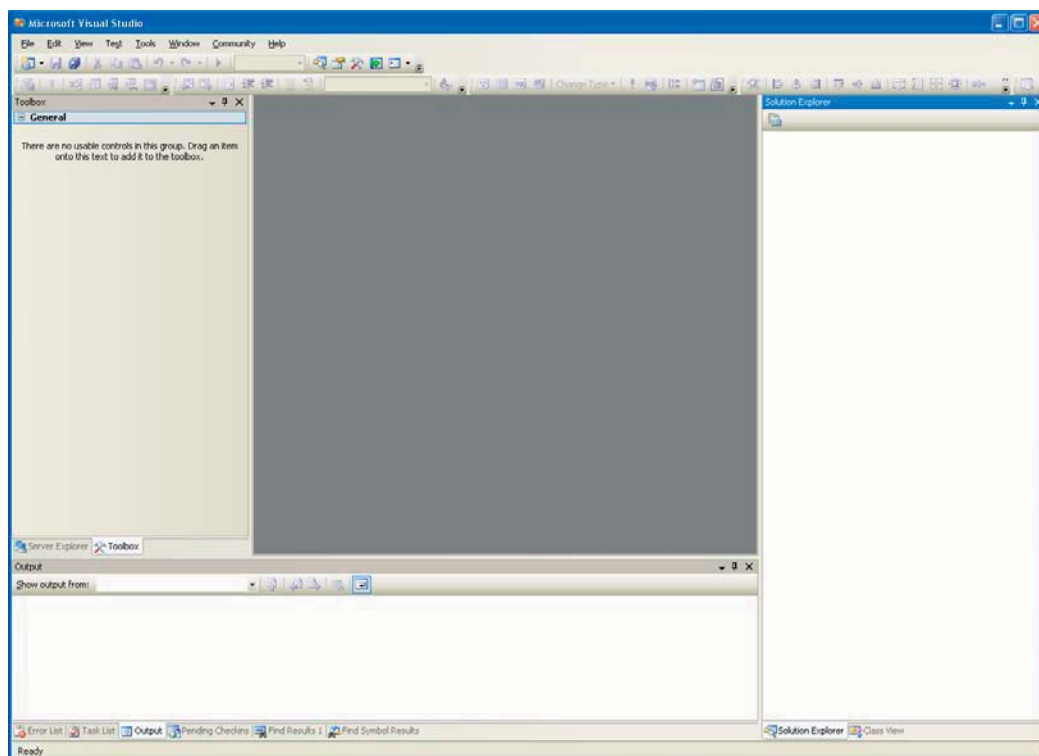
**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**



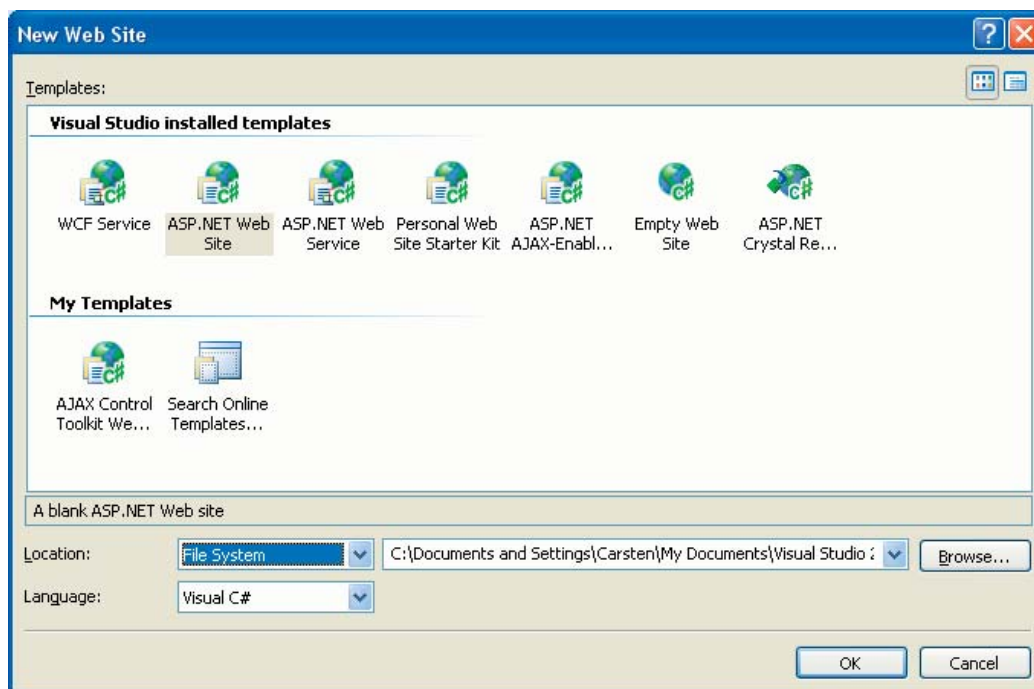
Start nu Visual Studio 2005 og vælg Visual C# Development Settings. Luk start siden. Du skulle have et miljø der ser ca. således ud:



Figur 1. Visual Studio - Start

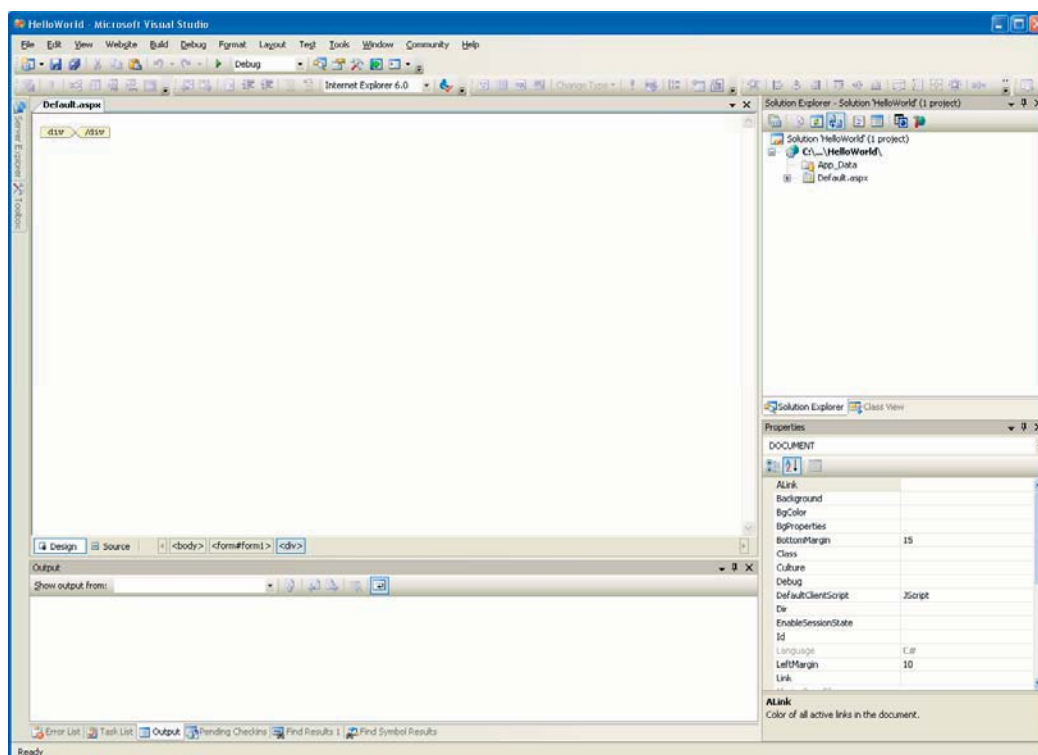


Vælg File\New\Web Site. Følgende dialog dukker op:



Figur 2. New Web Site dialog.

Bemærk at du ikke nødvendigvis har de samme projekt templates installeret som jeg, men template kaldet ASP.NET Web Site er den vi er på jagt efter her. Vælg den og vælg hvor du vil have projektet liggende via enten tekst feltet eller browse knappen. Kald folderen "HelloWorld" som navnet på din første .NET Web site. Nu er projektet oprettet og du har nu et projekt med en fil kaldet Default.aspx. Det er den fil vi nu skal ind og rette i. Dit miljø ser nu ca. således ud:

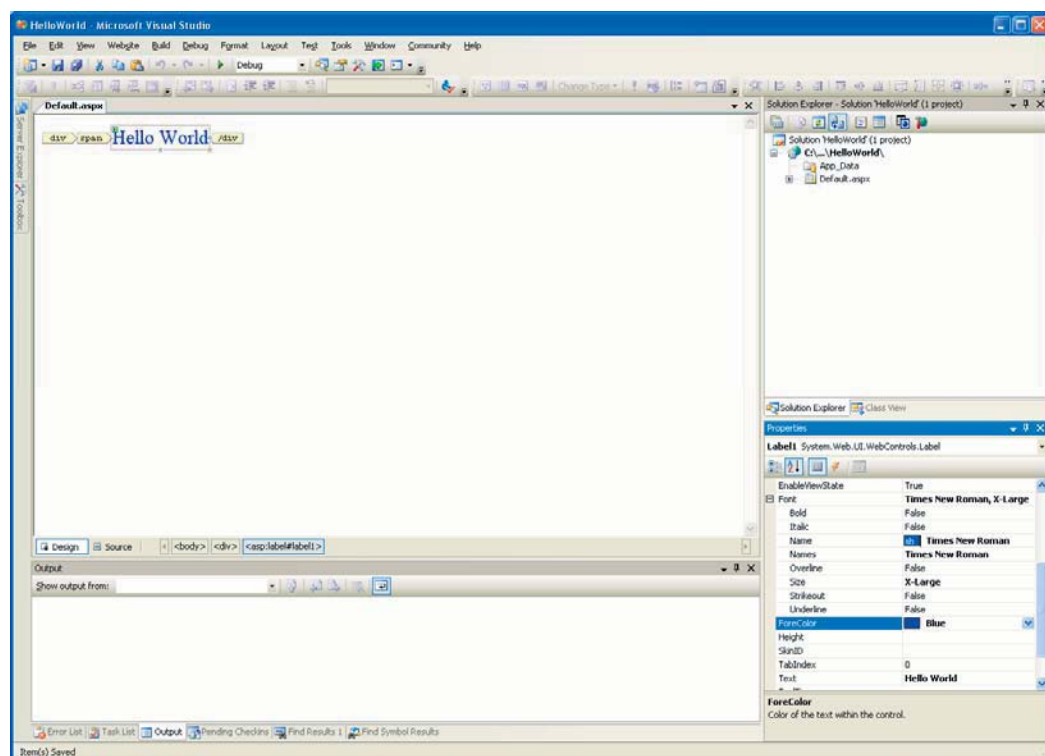


Figur 3. Web formen til Hello World.

Den side du kigger på er en Web form. En Web form er det sted du anbringer de kontroller du ønsker at slutbrugeren skal benytte til at kommunikerer med din web site. Mere om det senere. Nu vil blot have gang i Hello World.

Som Default skulle toolboxen være tilgængelig i venstre side af Visual Studio 2005 og kontrollerne skulle være synlige. Hvis ikke bringes toolboxen frem under menuen View, item Toolbox. Vælg en Label kontrol med venstre musetast, der holdes ned mens du "trækker" kontrollen over på Web formen. Her har du faktisk allerede det første eksempel på hvor let det er at addere kontroller til din web site. Det er ren drag and drop. Lad os nu få ændret teksten fra Label til Hello World og lad os ændre farven og fonten, samt størrelsen på fonten. Highligt lablen ved at vælge den med et enkelt klik med musen. I nederste højre hjørne har du nu adgang til properties på din label kontrol. En property er egenskaber ved kontrollen så som størrelse, tekst der skal vises, placering, farve, osv. Find den property der hedder "Text" og ændre teksten fra "Label" til "Hello World". Find også property'en "Font" og ændre "Name" property'en til "Times new roman". Under "Size" for fonten vælges "X-large". På selve label kontrollen vælges "ForeColor" til eksempelvis en blå farve.

Du skulle nu have følgende skærmbillede:



Figur 4. Web formen med Hello World label.

An advertisement for Bookboon. The background shows a man in a white shirt resting his head on his hands, looking tired or stressed, with a laptop in front of him. The ad text is overlaid on the image.

**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

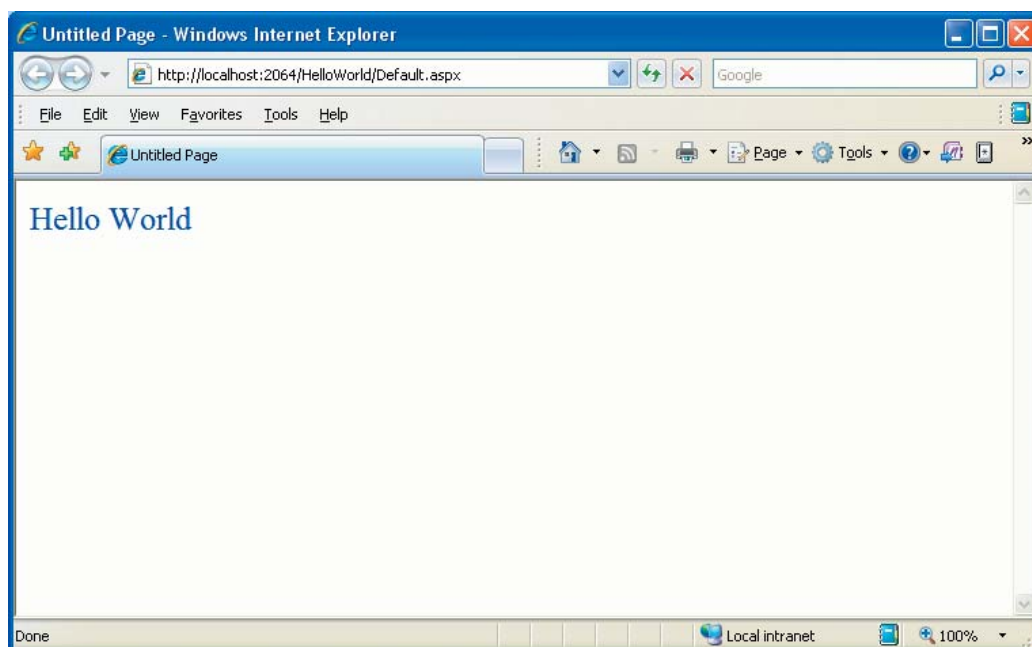
2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**

Inden vi kan se resultatet i en Web browser skal projektet først compileres. I Build menuen vælges Build Solution el. blot Ctrl+Shift+B. Visual Studio 2005 skulle gerne compilerer uden problemer og give dig følgende output i bunden af skærmen:

```
----- Build started: Project: C:\...\HelloWorld\, Configuration: Debug .NET -----  
Validating Web Site  
Building directory '/HelloWorld/'.  
  
Validation Complete  
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
```

Der var igen fejl eller warnings. Vi er nu klar til at se resultatet af vores arbejde i en web browser. Der er nu flere muligheder, men lad os starte internet explorere direkte fra miljøet og samtidig sætte en debug af vores Web Form igang. Dette gøres ved i Debug menuen at vælge Start eller blot trykke på F5. Følgende skulle gerne vise sig i internet explorere:



Figur 5. Hello World vist i Internet Explorer.

Nu er vi igang og i de næste kapitler gennemgår vi de forskellige kontroller gennem praktiske eksempler.

## 3. Web Forms

### 3.1 En forklaring på Hello World

Hvad var det der skete i kapitel 0? Vi skrev ikke så meget som en linie kode selv, men alligevel lykkedes det at producere en webside der skrev Hello World. Nu vil mange nok hævde at det er rimelig banalt at få en web server til at frembringe en side der skriver Hello World uden at skrive code og ja, det er da rigtig nok, men kernen her er at vi har produceret siden ved hjælp af Web Forms samt en Label kontrol alt sammen i C# (udtales C sharp).

Hvis du har kendskab til Visual Basic eller C# kender du i forvejen til Forms. En Web Form og en Form er i princippet det samme. I en C# applikation opererer man med Forms og i en ASP.NET C# applikation opererer man med Web Forms. En Form er en container hvorpå du placerer en eller flere kontroller. En Web Form er altid gemt i en fil der ender på .aspx. Web Formen er den grafiske repræsentation over for brugeren mens selve program koden ligger gemt i en fil der ender på .aspx.cs. Det her er jo rigtig smart idet der nu er en klar adskildelse mellem det grafiske brugerinterface og selve program koden. I Hello World placerede vi en label kontrol på vores web form og Visual Studio 2005 generede den nødvendige program kode. Lad os se engang hvad der skete. Tryk på Source så Default.aspx bliver vist i code view istedet for design view.



**MTHøjgaard**

**BEDRE  
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Font-Names="Times New Roman" Font-
Size="X-Large"
                ForeColor="Blue" Text="Hello World"></asp:Label></div>
        </form>
    </body>
</html>
```

Code behind filen ser således ud:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Ovenstående viser flere ting så lad os tage dem en af gangen. Label1 er vores label kontrol. Du kan finde den ovenfor navngivet som ID="Label1". I code behind filen kan vi se at der er oprettet en partial klasse kaldet \_Default. Her har vi adgang til kontrollerne fra aspx filen og kan således referere vores label som Label1, hvilket vi dog ikke har gjort, men som vi skal se eksempler på senere. Aspx filen og codebehind filen udgør tilsammen Web formen. Partial betyder blot at koden for en klasse kan ligge i flere filer. Oven for har Visual Studio 2005 altså erklæret vores label kontrol som en <asp:Label ...> Hmm, hvad er nu det for noget? Forklaringen skal findes i den udstrakte brug af namespaces som benyttes i C#. Vores label kontrol lever i namespace kaldet



`System.Web.UI.WebControls`. Et namespace er et fredet område om man vil, hvor variabel navne, klasse navn osv kan leve i harmoni med hverandre sålænge de er forskellige inden for det givne namespace. Det er jo rigtig smart for på den måde kan vi have flere klasser med de samme navne (hvor smart det end måtte være) blot de er erklærede i forskellige namespaces. Det vi ser ovenfor er at vores label kontrol befinder sig i et namespace kaldet *WebControls*. *WebControls* befinder sig i et namespace kaldet *UI* og *UI* befinder sig i et namespace kaldet *Web* og endelig befinder *Web* sig i et namespace kaldet *System*. Vi refererer altså til kontroller (klasser) og namespaces med et punktum ”.”. *System* er roden til det hele. Herfra vokser alle klasser og alle namespaces op og forgrener sig akkurat som grenene fra stammen på et træ.

Vi kan også se af ovenstående kode at der er erklæret en klasse kaldet *\_Default*. Den nedarver fra *System.Web.UI.Page* hvor vi nu allerede ved at *System* og *Web* og *UI* alle er namespaces. *Page* må så være en klasse idet man ikke kan nedarve et namespace. Den er også nøglen til alle Web Forms, idet de nedarver fra denne klasse.

Det er nu vigtigt at skelne mellem server afviklet kode og client afviklet kode. **Alle Web Forms afvikles på serveren. Punktum!** Det er spild af tid at forsøge på alle mulige fixfakserier for at få dem afviklet hos clienten, altså i web browseren. Glem det. Det kan du ikke. Hvis man vil lave noget i browseren (clienten) gør man det typisk i form af Java scripts<sup>ii</sup>.

Da vi i web browseren loadede vores Web Form afviklede serveren koden og generede en HTML side som blev leveret til browseren. Derfor kan alle browsere i princippet benyttes til at vise ASP.NET applikationer.

Den HTML generede side kan du finde ved i internet explorer at højre klikke på den viste side og i popup menuen at vælge *View Source*. Hvis du har den danske version af IE hedder det sikkert *Vis kode*, men det vil jeg overlade til læseren at verificere. Jeg er ikke særlig vild med de danske versioner af operativ systemet samt IE og lign. Source coden ser således ud:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>Untitled Page</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJODExMDE5NzY5ZGSOS9HTby9XXjai05XrUTJb9lrmGg==" />
</div>
<div>
  <span id="Label1" style="color:Blue;font-family:Times New Roman;font-size:X-Large;">Hello
World</span></div>
</form>
</body>
</html>
```



Hvis man kender til HTML er ovenstående kode ikke særlig kompliceret. Jeg giver dog en forklaring alligevel. Jeg forudsætter at du har basalt kendskab til HTML i min forklaring. *Head* her kan sidens titel angives og da vi ikke har skrevet noget vi siden optræde med navnet "Untitled Page". *body* sektionen starter med en *form*. Den afspejler faktisk vores web form, idet det nu er her vi finder vores label. I dette tilfælde er det unødvendigt med en form, men det er altså ikke optimeret bort i Visual Studio 2005 compileren. I formen er der en hidden (usynlig) kontrol, der har fået navnet "\_VIEWSTATE" med en besynderlig værdi "

/wEPDwUJODExMDE5NzY5ZGSOS9HTby9XXjai05XrUTJb9lrmGg==". Det er faktisk vældig smart for denne *tag* benytter serveren til at holde styr på hvordan kontrollen sidst var vist hos brugeren. Senere vil vi se på indholdet valgt i en combobox hvor denne tag giver langt mere mening end for en label. Under alle omstændigheder skal du ikke længere selv holde styr på hvad det er brugeren sidst havde valgt. Resten er lige ud af landevejen HTML med en *span* til at sætte en given *style* for vores label svarende til hvad vi nu valgte inde i Visual Studio 2005.



**Ses vi til DSE-Aalborg?**

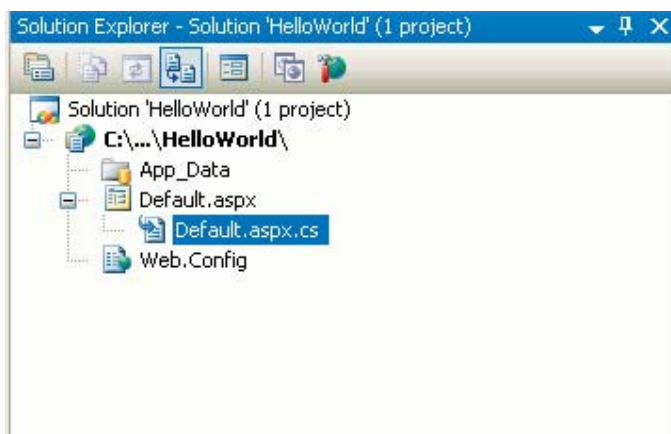
Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**



Lad os så lige kigge på de øvrige filer som Visual Studio 2005 har genereret til os. Filerne findes ude til højre øvest i Solution Explorer. Det drejer sig om følgende filer.



Figur 6. Solution Explorer.

Web.Config filen blev oprettet da du valgte at debugge applicationen. I denne fil defineres typisk hvordan web applikationen skal opføre sig og hvilke eksterne resourcer den har adgang til.

Det var det! Du har nu skrevet og forstået din første web applikation.

Inden vi forlader dette afsnit vil jeg dog lige nævne et par andre filer det kan være nyttigt at kende til. De er dog ikke med som default i det her projekt. Det skal nævnes at der ikke er en csproj fil tilknyttet denne type web projekt. Man kan kalde typen for amatør projekt el. lign. Med Service pack 1 af Visual Studio genindførte Microsoft muligheden for at have Web projekter i csproj filer efter pres fra brugerne.

Hvis vi havde oprettet projektet via File\New\Project ville vi også have fået en AssemblyInfo fil. AssemblyInfo.cs indeholder en beskrivelse af det *assembly manifest* som vores web applikation består af. En assembly manifest beskriver hvordan de enkelte komponenter, så som billeder og forskellige dll'er relaterer til hinanden og er at finde i enten en dll eller en exe fil. I Windows kataloget findes et katalog kaldet assembly hvor du kan se hvilke assemblys, der er installeret på din maskine.

Global Application Class (Global.asax) er stort set som den gammel kendte Global.asa som du måske kender fra din tid med ASP (Active Server Pages) programmering. Filen ser således ud:

```
<%@ Application Language="C#" %>
```

```
<script runat="server">
```

```
    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
    }

    void Application_End(object sender, EventArgs e)
    {
        // Code that runs on application shutdown
    }

    protected void Application_BeginRequest(object sender, EventArgs e)
    {
    }

    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs
    }

    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
    }

    void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends.
        // Note: The Session_End event is raised only when the sessionstate mode
        // is set to InProc in the Web.config file. If session mode is set to
        StateServer
        // or SQLServer, the event is not raised.
    }

</script>
```

Ligesom i den gamle Global.asa er der i Global.asax fil de gammelkendte events som man kan abonnere på.

Application\_Start eventet afsendes når serveren startes.

Session\_Start afsendes når en ny bruger connecter til din site. Her har du så mulighed for at sætte et eller andet op som du nu synes kunne være praktisk.

Application\_BeginRequest afsendes hver gang brugeren laver en ny forespørgsel til serveren. Det kunne f.eks. være i forbindelse med at der trykkes på en submit button.

Og til hver af de tre ovenstående events er der et event som afslutter dem.

Måske har du ikke forstået meningen med alt hvad der er skrevet og fortalt i dette afsnit, men fortvivl ikke for meget bliver gentaget og når du er færdig med bogen og du har fået det mere erfaring og forståelse kan du bare vende tilbage til dette kapitel og lige skimme de for dig vigtigste detaljer.



**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

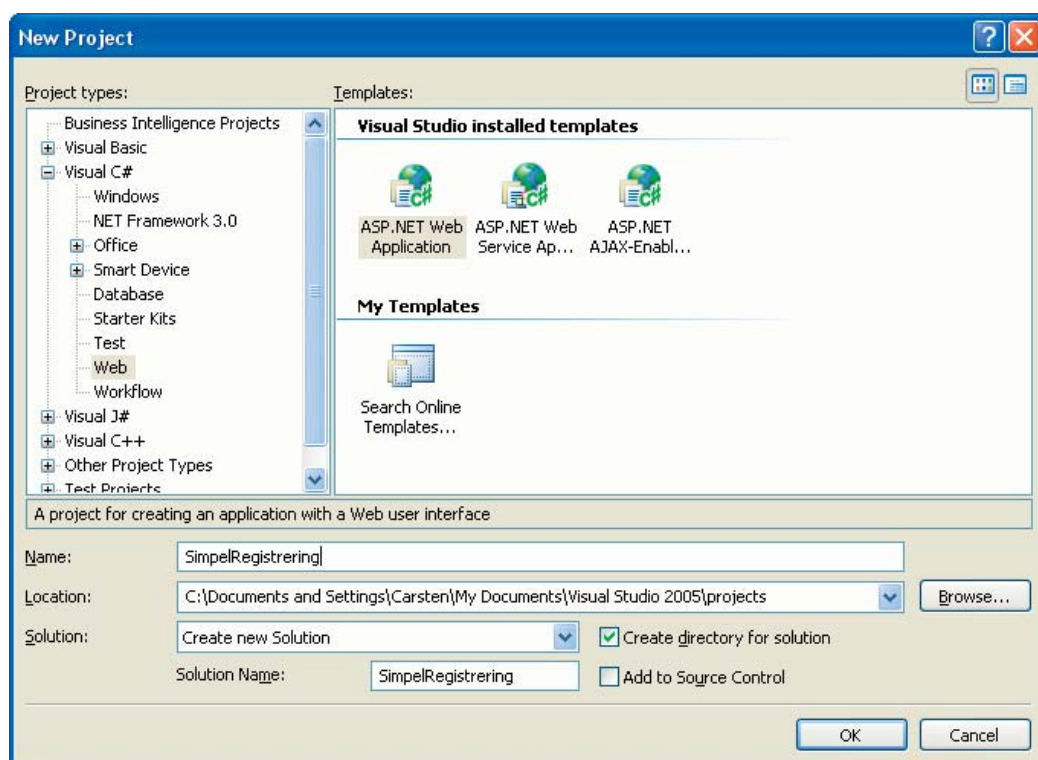
**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

## 3.2 En input og output form. Simple registrering

For at Web forms rigtig kommer til udfoldelse må vi have bygget noget mere på en blot label kontroller. I dette kapitel laver vi en simpel registrerings form (dog uden at skele til grafisk design) hvor der er mulighed for at indtaste navn og adresse, samt foretage nogle valg via comboboxe, radio buttons samt checkboxe. Siden kunne f.eks. bruges i forbindelse med registrering på en website der gerne vil vide lidt mere om dig. Vi gemmer oplysningerne i en fil og generere en svar side til brugeren.

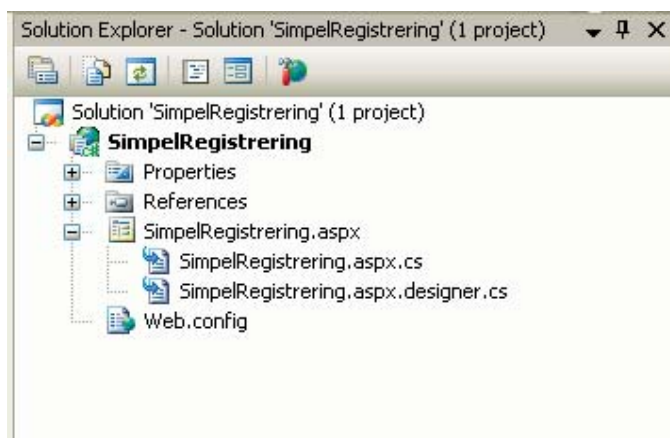
I Visual Studio 2005 opretter vi et nyt C# ASP.NET projekt kaldet *SimpelRegistrering* ved at gå til *File/New/Project*. Du skulle nu have følgende dialog:



Figur 7. Simple Registrerings new project dialog.

Vi er nu klar til at addere kontroller til vores web form, men først vil vi lige omdøbe filnavnet på vores webform fra *Default.aspx* til *SimpelRegistrering.aspx*. Du omdøber let filnavnet ved med *Solution Explorer* at vælge filen og højre klikke og fremkalde en popup menu. I den vælges så *Rename* og filen omdøbes. *Solution Explorer* ser nu således ud:





Figur 8. Solution Explorer med det omdøbte filnavn.

Vi vil nu opbygge input siden ved at benytte *Toolbox*'en.

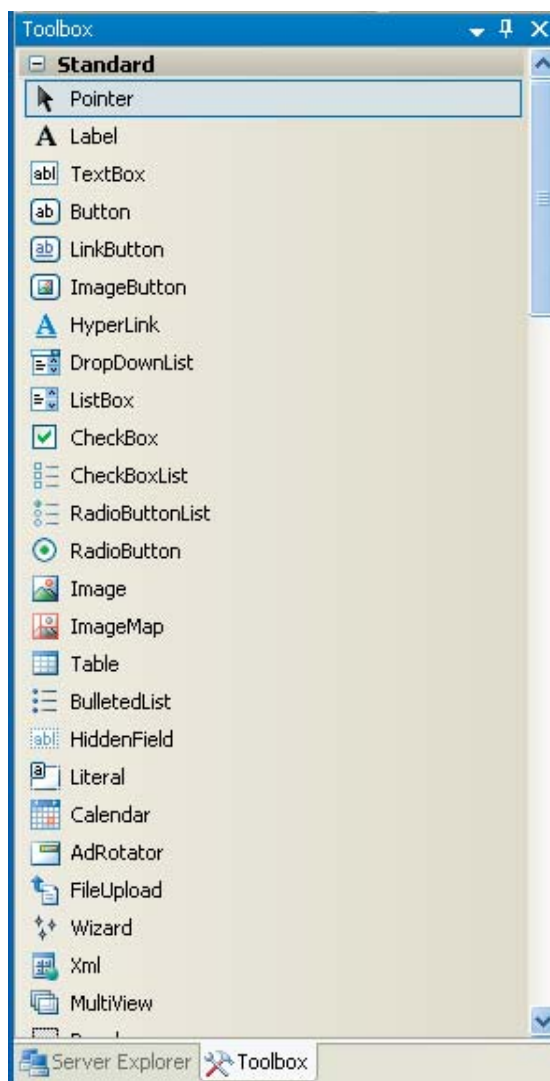


**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium** is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

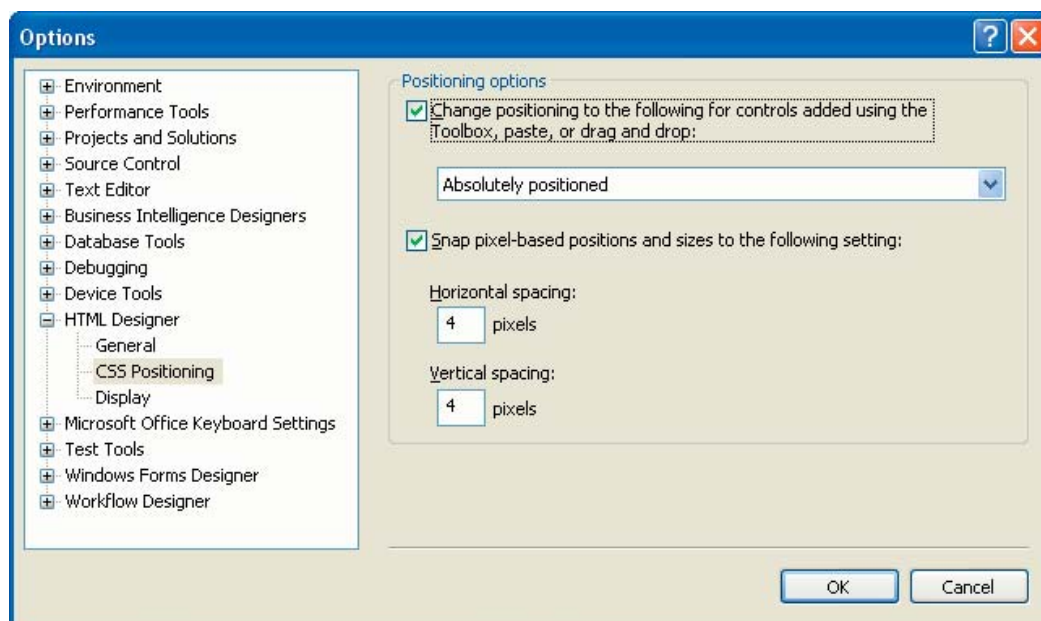
**bookboon.com**



Figur 9. Toolbox. Kontroller til webforms. Bemærk at ikke alle er vist.

Vi trækker nu de kontroller fra *toolbox*'en vi har brug for over på vores Web Form, men inden da vil vi lige sætte Visual Studio til at køre med absolut position for vores kontroller. Det gøres via *Layout/Position/Auto-Position Options* du får nu følgende dialog:





Figur 10. Opsætning til absolut position.



**MTHøjgaard**

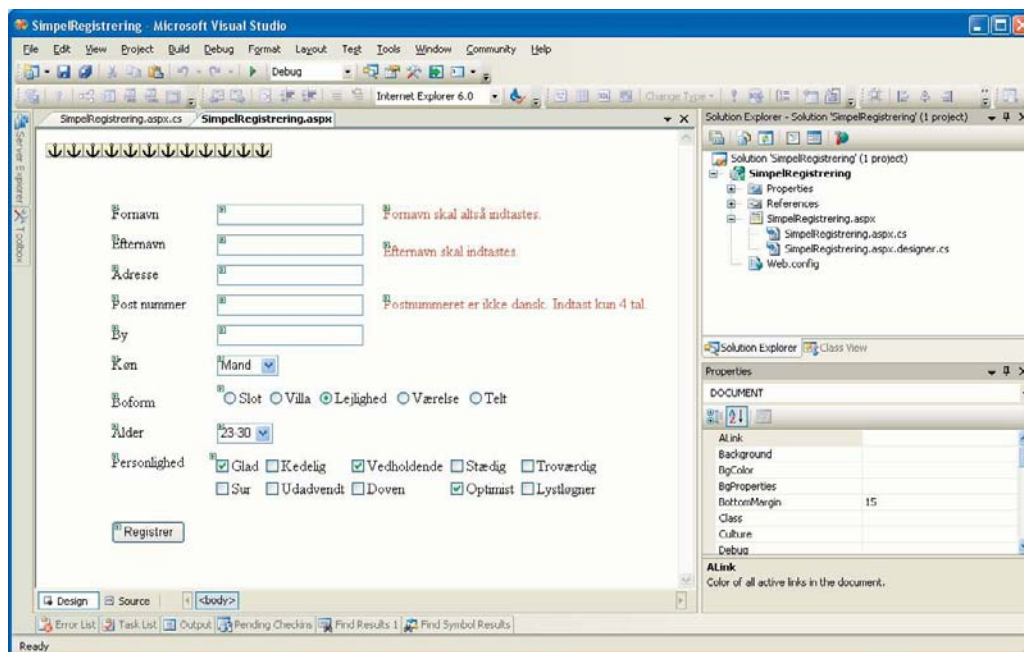
## BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)

Til det her demo formål er det praktisk med absolut position fordi kontrollerne kan flyttes rundt med musen til de er hvor vi gerne vil have dem. I praksis ser man ofte tables benyttet istedet til placering af kontroller.

Træk nu kontroller ind til du får nedenstående:



Figur 11. Simpel registrerings Web Form.

På Figur 11 er vist hvordan registrerings formen ser ud. Registrerings formen er bygget op af standard Web Form kontroller. Til venstre er der udelukkende brugt labels. Til navn og adresse er der brugt *TextBox* kontroller. Til angivelse af køn og alder er der brugt *DropDownList* (også kendt som combo box) kontroller. Til angivelse af boform er der brugt en *RadioButtonList* kontrol. For at få den til at "ligge ned" er *RepeatDirection* propertyen sat til *Horizontal*. Til personlighed er der brugt en *CheckBoxList* kontrol, der indeholder 5 kolonner. Dette er gjort ved at sætte *RepeatColumns* til 5. Endelig er der en *Button* kontrol til at sende hele molivitten afsted til serveren.

Det er de grundlæggende kontroller som i mange tilfælde dækker hvad der er behov for i forbindelse med en registrering. Indholdet må vi lade ligge idet eksemplet netop kun tjener som eksempel, men principperne er gode nok.

På Figur 11 er der også ialt tre special kontroller. Vi kan se at der er en kontrol der siger "Fornavn skal altså indtastes." og en der siger "Efternavn skal indtastes.". Disse to kontroller har til formål at forhindre brugeren i at sende en registreringsform afsted uden at fornavn og efternavn er angivet. *RequiredFieldValidator* som kontrollen hedder bindes til en bestemt kontrol og verificere så at feltet ikke er blankt. Endelig er der kontrollen der siger "Postnummeret er ikke dansk. Indtast kun 4 tal." som er en *RegularExpressionValidator* kontrol. Regulære udtryk kan i denne kontrol benyttes til at sikre at input er i overensstemmelse med det som det regulære udtryk foreskriver. Det er en meget fleksibel validerings kontrol, som du vil benytte til at matche at input er på en helt bestemt form. Hvis

du ikke skulle kende regulære udtryk kan du finde mere om emnet i MSDN eller i hjælpen til Visual Studio 2005, men lad os her se hvordan det regulære udtryk til validering af postnummeret ser ud.

Det anvendte regulære udtryk ser således ud:

`(DK-)?\d{4}`

`(DK-)` beskriver at "DK-" er det første som kan forekomme i et postnummer. Parentesen har her til formål at samle "DK-" som én gruppe og er altså ikke en del af det der matches. `?` beskriver 0 eller en match af det som står umiddelbart foran spørgsmåls tegnet.. Dvs at "DK-" kan undværes og det er jo helt i overensstemmelse med den måde vi normalt angiver et dansk postnummr på. `\d` beskriver et tal. Dette udtryk er måske bedre kendte på formen `[0-9]`. `{4}` beskriver at det som står umiddelbart foran skal gentages præcis fire gange. På ren dansk betyder det at der skal forekomme fire tal i området 0000-9999. Det er jo ikke alle gyldige postnumre, men til brug for dette eksempel er valideringen fin nok. Eksempler på gyldige postnumre der matches af ovenstående regulære udtryk er: DK-3600, DK-5200, 6000 og 8050.

Hvis brugeren altså indtaster noget som de tre validerings kontroller ikke kan godkende må han prøve igen. Koden for valideringskontrollerne afvikles som javascript i browseren hos brugeren og netværket undgår belastende trafik af data som ikke overholde de mest simple validerings kriterier. Mere kompleks validering foregår typisk på serveren så som tjek for om navnet allerede findes i en database. På serveren er det god skik at lave den simple validering alligevel, for nogen kunne måske have onde hensigter og forsøge at finde huller i serverkoden.

I dette eksempel gemmes informationen som indtastes i en fil på harddisken. I en "rigtig" applikation ville man benytte en database og hvordan man gør det skal vi se senere.

Når vores bruger trykker på registrer samler vi det event (hændelse) op i vores Web Form. Web Forms er i event drevne forstået på den måde at der opstår en eller anden ændring på siden som afføder et event som samles op i vores kode. Vi vil i dette tilfælde gerne samle det event op der skyldes at brugeren trykker på vores registrerings button. Dette sker via en *delegate* funktion. En delegate funktion er blot en callback funktion. I C# er metoden med callback funktioner udvidet så den er typesikker og objektorienteret. Den letteste måde at få auto genereret koden på er ved at dobbelt klikke på *Registrer* knappen, men hvis du vil kan du selvfølgelig skrive koden selv. Koden der umiddelbart er interessant ser således ud:

```
private void Registrer_Click(object sender, System.EventArgs e)
{
}
}
```

Derudover genereres der kode til at dirigere klikket hen til funktionen *Registrer\_Click*. *System.EventHandler* er her deklareringen af delegate funktionen *Registrer\_Click*. Koden ligger i aspx filen og ser således ud:

```
<asp:button id="Registrer" style="Z-INDEX: 119; LEFT: 80px; POSITION: absolute;
TOP: 416px" runat="server" ToolTip="Når du klikker her sender du informationen af
sted." Text="Registrer" OnClick="Registrer_Click"></asp:button>
```

Umiddelbart behøver du ikke at spekulere så meget på det netop fordi Visual Studio 2005 hjælper med at skrive den trivielle kode for dig. Du skal koncentrere dig om at skrive hvad der skal ske efter at brugeren har sendt sine informationer til din server!

Jeg har valgt at identificere de enkelte kontroller (dog ikke *label* kontrollerne) med det samme navn som lablen der står umiddelbart til venstre for kontrollen (se evt. Figur 11. Sempel registrerings Web Form.). Koden der bliver afviklet når brugeren trykker på *Registrer* knappen ser således ud:



**Ses vi til DSE-Aalborg?**

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**



```

private void Registrer_Click(object sender, System.EventArgs e)
{
1      string fileName = Path.Combine(Request.PhysicalApplicationPath,
    "RegInfo.txt");

2      StreamWriter regInfo = new StreamWriter(fileName, false);

3      regInfo.WriteLine("Fornavn: " + Fornavn.Text);
4      regInfo.WriteLine("Efternavn: " + Efternavn.Text);
5      regInfo.WriteLine("Adresse: " + Adresse.Text);
6      regInfo.WriteLine("Post nummer: " + PostNummer.Text);
7      regInfo.WriteLine("By: " + By.Text);
8      regInfo.WriteLine("Køn: " + Køn.SelectedItem.Text);
9      regInfo.WriteLine("Boform: " + Boform.SelectedItem.Text);
10     regInfo.WriteLine("Alder: " + Alder.SelectedItem.Text);

11     string personlighedsText = "";
12     foreach (ListItem item in Personligheds.Items)
13     {
14         if(item.Selected == true)
15         {
16             personlighedsText += item.Text + " ";
17         }
18     }
19     regInfo.WriteLine("Personligheds: " + personlighedsText);
20
21     regInfo.Close();
22
23     Response.Redirect("RegistreringsBekræftigelse.aspx?Fornavn=" +
Fornavn.Text);
}

```

I SmpelRegistrering.aspx.cs er der desuden adderet følgende kode, der fortæller kompilatoren at vi vil i kontakt med System.IO namespace. Man behøver ikke angive brugen af et namespace med kommandoen *using*, idet man blot kan henvise direkte til namespace ved at skrive navnet fuldt ud.

```
using System.IO;
```

I ovenstående kode benyttes f.eks. StreamWriter klassen der befinder sig i System.IO namespace. Hvis ”using System.IO” udelades skulle jeg skrive System.IO.StreamWriter istedet for blot StreamWriter. Fordelen ved at udelade *using* er at det er synligt hvor en klasse kommer fra, men til gengæld skal man skrive mere code og det kan også gå ud over læsbarheden. Hvordan man vælger at gøre er dog et spørgsmål om smag og code stil.



Lad os gennemgå koden linie for linie:

**Linie 1:**

```
string fileName = Path.Combine(Request.PhysicalApplicationPath, "RegInfo.txt");
```

Formålet med ovenstående linie er oprette et filnavn incl. path til filen "RegInfo.txt", som vi vil oprette og gemme vores oplysninger fra brugeren i. Vores Web Form nedarver fra *System.Web.UI.Page*, der har en property kaldet *Request*. *Request* kender du muligvis fra ASP og har samme funktion. *Request* giver dig bla. mulighed for at tilgå informationer som kommer fra brugeren. I *RegistreringsBekræftigelse.aspx.cs* har jeg et eksempel på brugen af *Request* i forbindelse med tilgang til information sendt af brugeren. I ovenstående linie benyttes *Request* dog til at få den fysiske path til der på serveren hvor vores web form ligger. *PhysicalApplicationPath* returnerer path'en så jeg behøver blot at addere filnavnet til path'en for at nå mit oprindelig mål. *Path.Combine* gør netop det. Når koden afvikles kommer *fileName* til at indeholde "C:\\Documents and Settings\\Carsten\\My Documents\\Visual Studio 2005\\Projects\\SimpelRegistrering\\SimpelRegistrering\\RegInfo.txt". (Hos dig vil stien selvfølgelig matche hvor du har dit projekt liggende)

**Linie 2:**

```
StreamWriter regInfo = new StreamWriter(fileName, false);
```

*StreamWriter* klassen findes i *System.IO* og benyttes til at skrive information til en Stream. En stream er ikke nødvendigvis en fil på harddisken, men kunne ligeså godt være en et sted i hukommelsen eller på en anden maskine der tilgås via DCOM eller via internettet. En stream er dog at betragte som en sekventiel følge af data ligesom strømmen af vand i en flod. Ovenstående sætning opretter et *StreamWriter* object kaldet *regInfo*. Objektet knyttes til en filen på harddisken og parametren *false* angiver at vi ønsker at overskrive filen hvis den findes i forvejen.

**Linie 3-7:**

I disse linier skriver vi via *WriteLine* metoden linier til vores *StreamWriter* objekt. Data kommer fra *Text* propertyen på de *TextBox* kontroller, der hører til navn og adresse informationer.

**Linie 8-10:**

Igen skriver vi til vores *StreamWriter* objekt via *WriteLine* metoden og data hentes som før via en *Text* property på kontrollerne. Dog skal man lige via en *SelectedItem* property. I de nævnte kontroller kan kun én item være valg af gangen derfor behøver vi blot at tilgå *SelectedItem* propertyen. Læg mærke til at selvom vi har både en *RadioButtonList* og to *DropDownList* kontroller er måden man tilgår text på præcis den samme.

**Linie 11-19:**

De valgte items fra vores *CheckBoxList* findes frem via en *foreach* loop.

```
foreach (ListItem item in Personlighed.Items)
```

De items brugeren har valgt ud for hans Personlighed findes ved at løbe gennem alle items i *CheckBoxList* kontrollen og se på propertyen *Selected* for at afgøre om den enkelte item er valgt eller ej. Hver item i kontrollen er af typen

*ListItem*. Hvis en item er valgt gemmer vi dens *Text* property i strengen *personlighedsText* og adskiller hver valgt item text med et mellemrum. Efter loopet skriver vi som tidligere teksten med *WriteLine* metoden til vores *StreamWriter* objekt.

#### Linie 21:

*StreamWriter* objektet lukkes med *Close* metoden. Dette sikre at data gemmes i streamen og i vores tilfælde at de bliver skrevet til filen på harddisken.



**CISO Conference**  
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

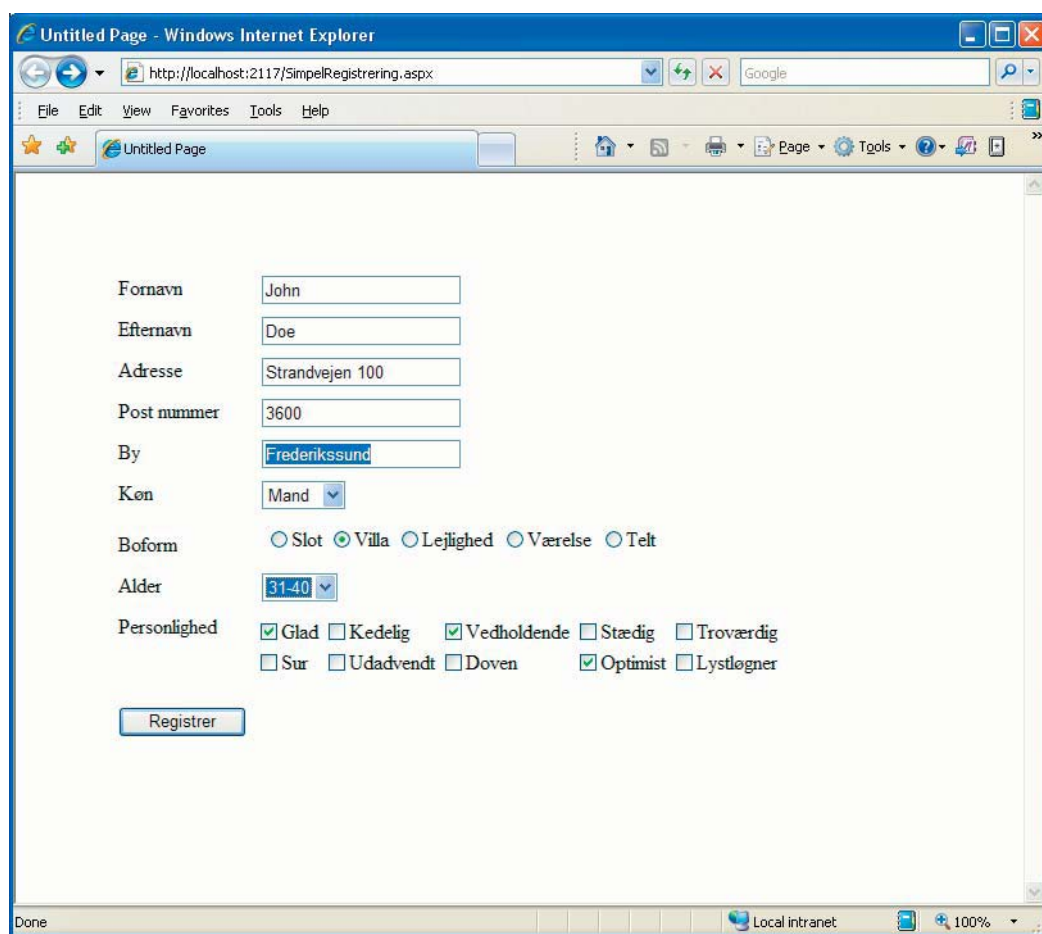


**Linie 23:**

```
Response.Redirect("RegistreringsBekræftigelse.aspx?Fornavn=" + Fornavn.Text);
```

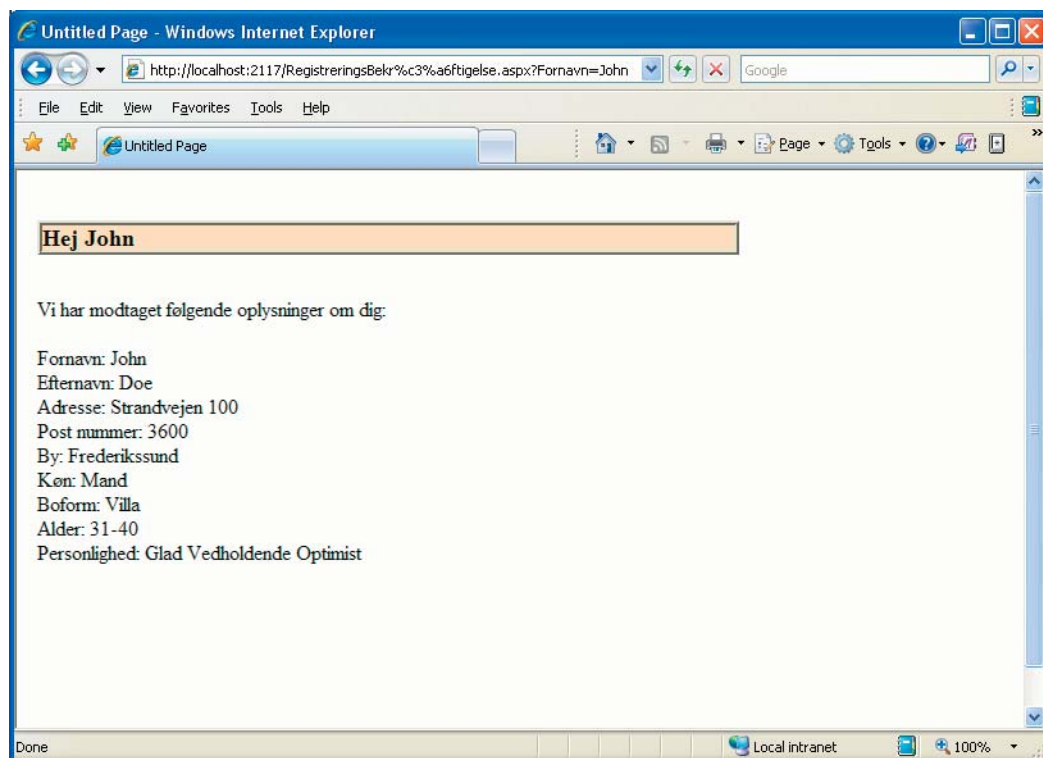
Efter at vi har gemt data lader vi en anden Web Form præsentere de gemte data for brugeren. *Response* propertyen er ligesom *Request* propertyen en del af vores Web Form og kender du til ASP kender du også til *Response* objektet. *Response* objektet giver bla. mulighed for at skrive direkte til den side som skal sendes tilbage til brugeren og i det dette eksempel at bede serveren om at load en anden side end den nuværende med *Redirect* metoden. Læg mærke til at der overføres data med "*RegistreringsBekræftigelse.aspx?Fornavn*"= + *Fornavn.Text*. ? angiver at der nu kommer en parameter kaldet *Fornavn* og vi tildeler parametren værdien der er i vores *Fornavn* kontrol.

Når koden afvikles ser det således ud:

The image is a screenshot of a Windows Internet Explorer browser window. The title bar reads "Untitled Page - Windows Internet Explorer". The address bar shows the URL "http://localhost:2117/SimpelRegistrering.aspx". The browser's menu bar includes "File", "Edit", "View", "Favorites", "Tools", and "Help". The main content area displays a registration form with the following fields and controls: "Fornavn" (text box with "John"), "Efternavn" (text box with "Doe"), "Adresse" (text box with "Strandvejen 100"), "Post nummer" (text box with "3600"), "By" (text box with "Frederikssund"), "Køn" (dropdown menu with "Mand" selected), "Boform" (radio buttons for "Slot", "Villa" (selected), "Lejlighed", "Værelse", "Telt"), "Alder" (dropdown menu with "31-40" selected), and "Personlighed" (checkboxes for "Glad", "Kedelig", "Vedholdende" (checked), "Stædig", "Troværdig", "Sur", "Udadvendt", "Doven", "Optimist", "Lystløgner"). At the bottom of the form is a "Registrer" button. The status bar at the bottom shows "Done", "Local intranet", and a zoom level of "100%".

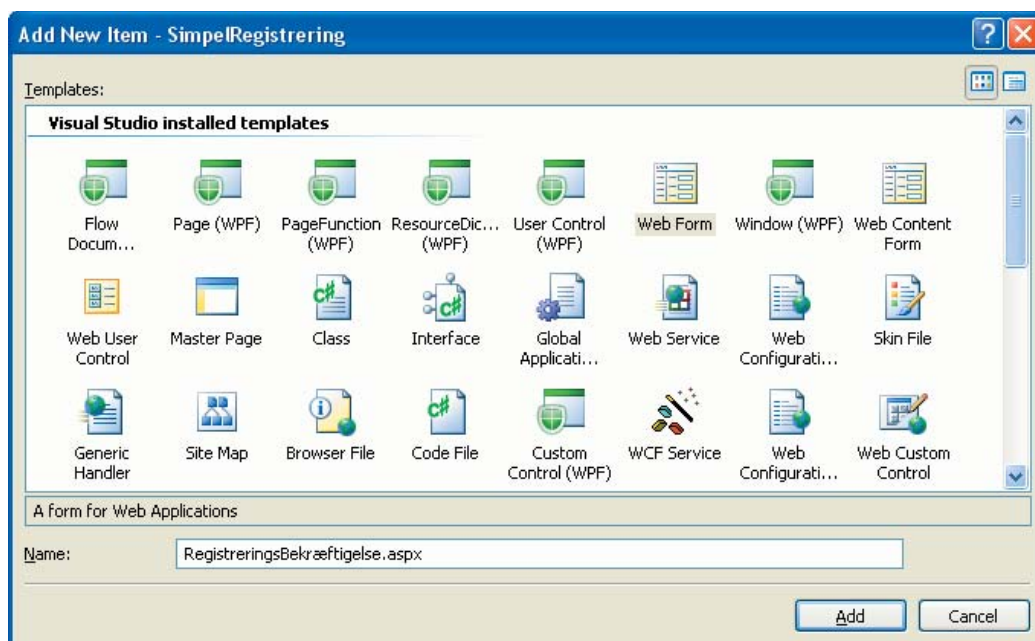
Figur 12. Simpel registrering i Internet Explorer.

Og svar siden ser således ud:



Figur 13. Svar siden til brugeren.

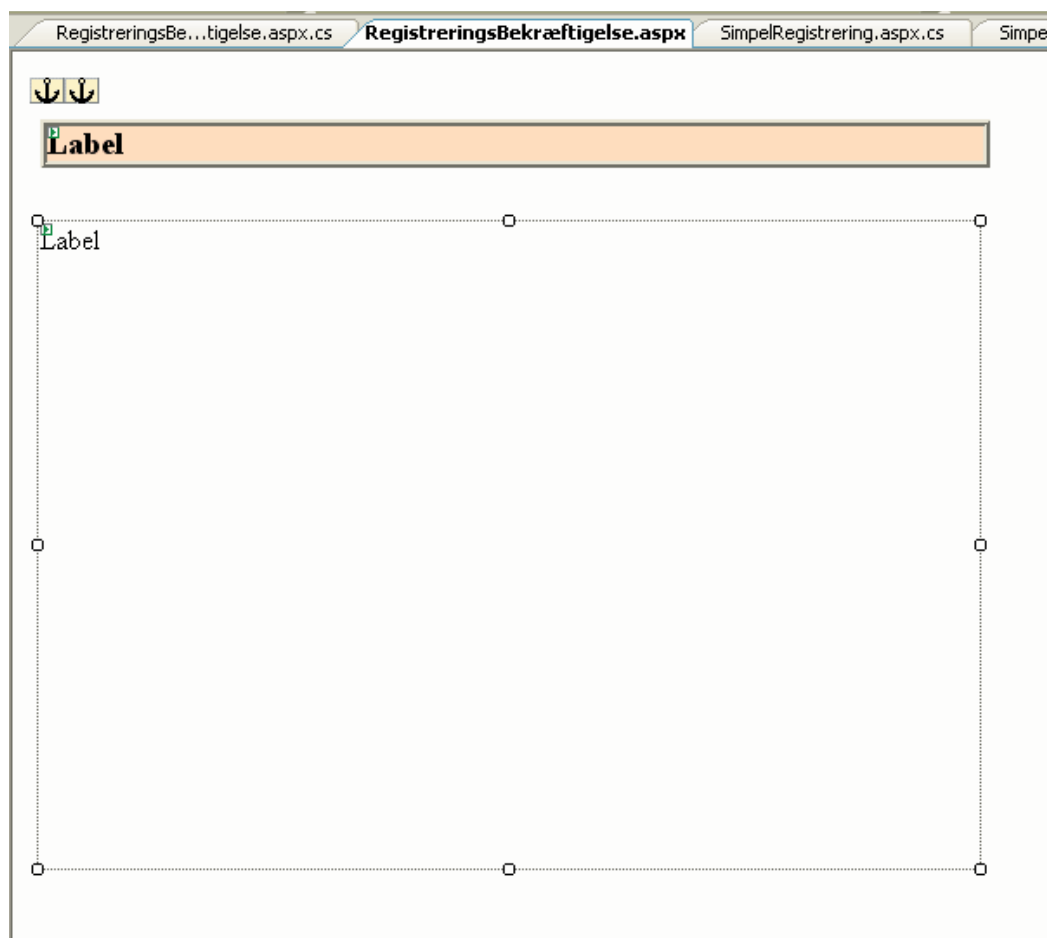
Svar *RegistreringsBekræftigelse.aspx* er adderet til projektet ved i menuen *Project* at vælge *Add New Item* og så klikke på Web Form templaten. Man kunne også i *Solution Explorer* (se Figur 8) vælge projekt navnet *SimpelRegistrering* og højre klikke på det og derfra vælge *Add->New Item*. Man får så følgende dialog:



Figur 14. RegistreringsBekræftigelse Web Form oprettelse.

Man vælger *Web Form* typen og indtaster det ønskede filnavn og vælger *Add*. I *Solution Explorer* er det nye filnavn nu fremkommet og der er kommet en Web Form som man kan indsætte kontroller i. Jeg har indsat to label kontroller. Det ser således ud:

The advertisement features a background image of a man in a white shirt sitting at a desk, resting his head on his hand with a look of stress or exhaustion, while a laptop is open in front of him. On the left, there is a small thumbnail of the eBook cover for 'Your Boss: Sorted!'. The text on the ad reads: 'Max's next Bookboon eBook Your Boss: Sorted! By Patrick Forsyth - 55 pages'. At the bottom, a dark blue banner contains the text 'Unlock your life. Bookboon Premium is your key.' followed by '2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.' and the 'bookboon.com' logo. A green speech bubble with a hand cursor icon points to the ad, containing the text 'Click on the ad to read more'.



Figur 15. RegistreringsBekræftigelse Web Formen.

Læg mærke til at den øverste label har fået en anden baggrunds farve og anden font størrelse end default. Desuden er *BorderStyle* propertyen sat til *Ridge*. Prøv selv at lege med de forskellige properties og find ud af hvordan og hvor let du kan ændre udseendet på dine kontroller. Koden der læser RegInfo.txt fra harddisken findes i *Page\_Load* metoden på *RegistreringsBekræftigelse* Web Formen. Load eventet sendes til Web Formen umiddelbart inden den sendes afsted fra serveren til brugeren og giver dig mulighed for f.eks. at fylde noget meningsfuldt i kontrollerne iden de sendes til brugeren.

Koden i `Page_Load` ser således ud:

```
private void Page_Load(object sender, System.EventArgs e)
{
1     Overskrift.Text = "Hej " + Request["Fornavn"];

2     string fileName = Request.PhysicalApplicationPath + "RegInfo.txt";
3     StreamReader regInfo = new StreamReader(fileName, false);
4     string personlighedsText = "Vi har modtaget følgende oplysninger om
dig:<br><br>";

5     string line = regInfo.ReadLine();
6     while (line != null)
7     {
8         personlighedsText += line + "<br>";
9         line = regInfo.ReadLine();
10    }
11    RegInfoText.Text = personlighedsText;
}
```

#### Linie 1:

Lablen kaldet *Overskrift* sættes til at vise teksten "Hej " plus den tekst der sendes med i parametren kaldet *Fornavn*. Det er den tekst som vi sendte med i linie 23 på *SimpelRegistrerings* formen. Teksten findes i *Request* proprietien på vores Web Form og hentes frem via parametrens navn.

#### Linie 2:

Som i *SimpelRegistrerings* Web Formen bruger vi *Request* objektet til at finde path'en til vores *RegInfo.txt* fil.

#### Linie 3:

Da vi skal læse *RegInfo.txt* filen benyttes *StreamReader* klassen som kan læse det vi skrev med *StreamWriter*. Vi opretter objektet *regInfo* til formålet. Husk iøvrigt på at en *klasse* er en model af noget, mens af et *objekt* er en forekomst (instans) af *klassen*.

#### Linie 4:

Vi opretter en streng variabel kaldet *personlighedsText* til at gemme oplysningerne fra filen i og tildeler den en start værdi.

**Linie 5-10:**

RegInfo.txt filen læses linie for linie.

**Linie 11:**

*RegInfoText* lablen tildeles hvad vi læste fra filen.

Herefter loades siden og sendes afsted til brugeren.



 **MTHøjgaard**

**BEDRE  
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



Lad os slutte af med en komplet listning af begge Web Forms:

SimpelRegistrering.aspx:

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.IO;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

namespace SimpelRegistrering
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Registrer_Click(object sender, EventArgs e)
        {
            string fileName = Path.Combine(Request.PhysicalApplicationPath,
"RegInfo.txt");
            StreamWriter regInfo = new StreamWriter(fileName, false);

            regInfo.WriteLine("Fornavn: " + Fornavn.Text);
            regInfo.WriteLine("Efternavn: " + Efternavn.Text);
            regInfo.WriteLine("Adresse: " + Adresse.Text);
            regInfo.WriteLine("Post nummer: " + PostNummer.Text);
            regInfo.WriteLine("By: " + By.Text);
            regInfo.WriteLine("Køn: " + Køn.SelectedItem.Text);
            regInfo.WriteLine("Boform: " + Boform.SelectedItem.Text);
            regInfo.WriteLine("Alder: " + Alder.SelectedItem.Text);

            string personlighedsText = "";
            foreach (ListItem item in Personligheds.Items)
            {
                if (item.Selected == true)
                {
                    personlighedsText += item.Text + " ";
                }
            }
            regInfo.WriteLine("Personligheds: " + personlighedsText);
        }
    }
}
```



```

        regInfo.Flush();
        regInfo.Close();
        Response.Redirect("RegistreringsBekræftigelse.aspx?Fornavn=" +
Fornavn.Text);
    }
}

RegistreringsBekræftigelse.aspx:

using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.IO;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

namespace SimpelRegistrering
{
    public partial class RegistreringsBekræftigelse : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Overskrift.Text = "Hej " + Request["Fornavn"];

            string fileName = Request.PhysicalApplicationPath + "RegInfo.txt";
            StreamReader regInfo = new StreamReader(fileName, false);
            string personlighedsText = "Vi har modtaget følgende oplysninger om
dig:<br><br>";

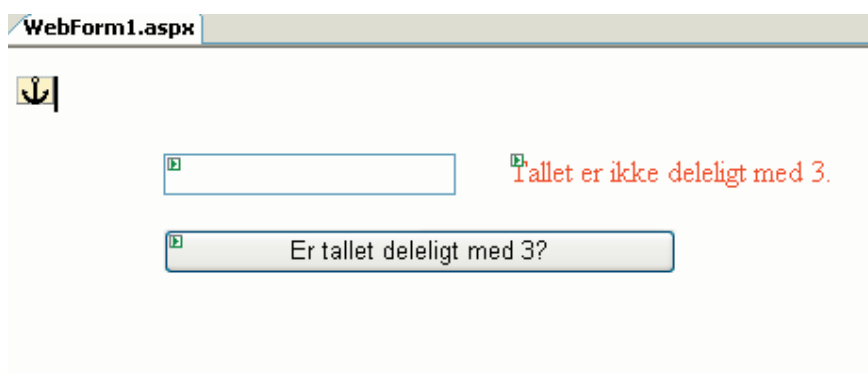
            string line = regInfo.ReadLine();
            while (line != null)
            {
                personlighedsText += line + "<br>";
                line = regInfo.ReadLine();
            }
            RegInfoText.Text = personlighedsText;
        }
    }
}

```

### 3.3 En custom validerings kontrol

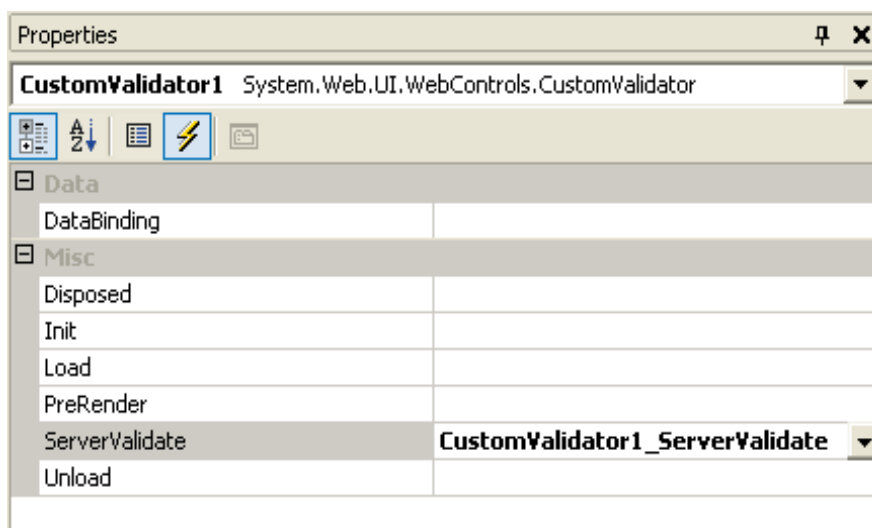
I det sidste eksempel så vi anvendelsen af validerings kontroller. Hvis ikke standard validerings kontrollerne opfylder ens behov har man også mulighed for at selv at skrive en validerings kontrol. En sådan kaldes for en *CustomValidator*. Lidt afhængig af hvor kompliceret valideringen er og hvilken type det er kan validering foregå enten hos clienten (dvs. i brugerens browser) eller på serveren. Generelt kan man sige at følsomme oplysninger altid skal valideres på serveren, dvs. sådan noget som passwords, mens ikke følsomme oplysninger som f.eks. om der er et minimum antal tegn i et password ligeså godt kan valideres hos clienten. Som eksempel vil vi skrive en validerings kontrol der kan afgøre om et tal er deleligt med 3.

Som set tidligere oprettes et ASP.NET projekt og i toolboxen vælges en *CustomValidator* kontrol. På Figur 16 ses kontrollen med teksten ”Tallet er ikke deleligt med 3”.



Figur 16. Custom validator kontrol

I property vinduet vælges et default metode navn til ServerValidate eventet. Det event affyres når der skal foretages validering af den kontrol der er bundet sammen med validerings kontrollen. Det gøres iøvrigt via ControlToValidate propertyen.



Figur 17. CustomValiate events



Ses vi til DSE-Aalborg?

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**

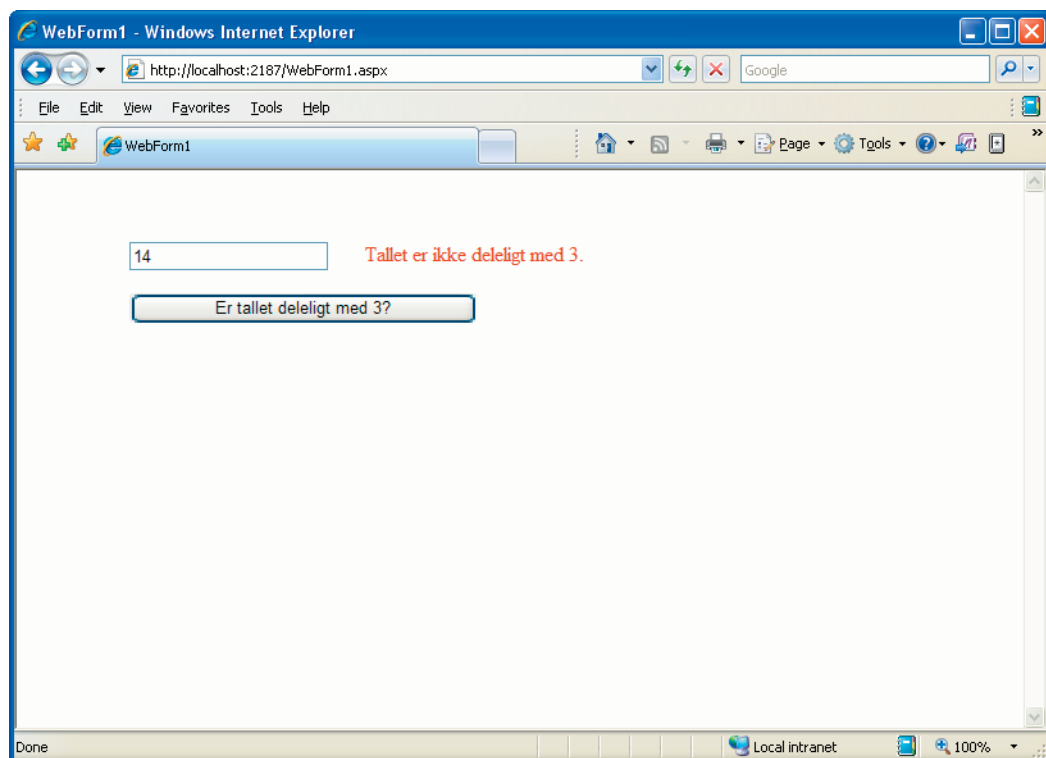


Koden til validerings kontrollen ser således ud:

```
private void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
    long number = long.Parse(args.Value);

    if(number % 3 == 0)
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
```

Som parameter til validerings metoden er et objekt af typen `ServerValidateEventArgs`. Objektet har to vigtige properties nemlig *Value* og *IsValid*. *Value* indeholder det der skal valideres. I dette tilfælde et tal (hvis nogen indtaster en streng fejler ovenstående kode. Fejlhåndtering diskuteres senere). Tallet tjekkes med modulo funktionen `%`. Hvis modulo returnerer 0 er tallet deleligt med 3 ellers ikke. Hvis vi ønsker at validering gik godt sættes propertyen *IsValid* til true ellers sættes den til false. En kørsel ser således ud i browseren:

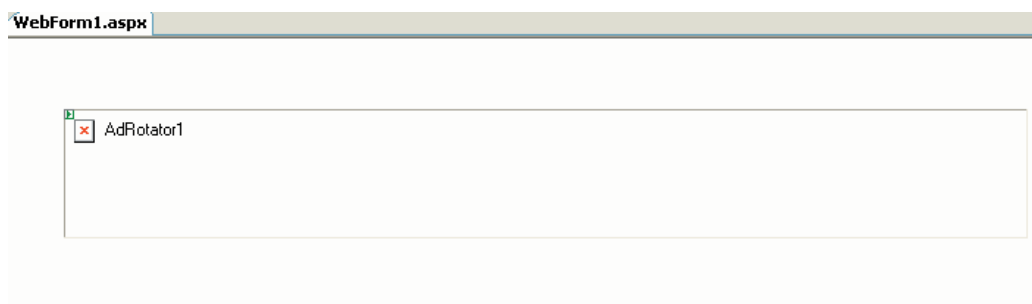


Figur 18. Eksempel på custom validering

## 3.4 AdRotator kontrollen

Når du surfer rundt på nettet har du helt sikkert lagt mærke til at banner reklamerne skifter når du surfer ind på den samme side flere gange. Mange websider er stadig lavet med "gammeldags" ASP teknologi (PHP eller Frontpage extensions), der også har en AdRotator til rådighed. ASP.NET AdRotator er dog XML baseret og gennemgås her i form af et eksempel. Målet for dette eksempel er at oprette en side indholdene en AdRotator kontrol med 3 reklamer, hvor reklame 1 vises med en sandsynlighed på 60%, reklame 2 med en sandsynlighed på 30% og reklame 3 med en sandsynlighed på 10%.

Der oprettes et ASP.NET projekt som vi har set tidligere. Jeg har kaldt eksemplet for AdRotatorEx. I toolboxen under Web Forms findes AdRotatoren der med drag and drop placeres på Web Formen.



Figur 19. AdRotator på web form.

AdRotator kontrollen skal kende til de 3 reklamer vi gerne vil have den til at kende. Jeg har til formålet oprettet 3 gif filer med billederne. En animeret gif kan selvfølgelig også anvendes. Det er nu muligt via dynamisk kode at konfigurere AdRotator kontrollen eller statisk via AdvertisementFile propertyen på kontrollen. Jeg vælger den statiske fremgangsmåde, som også vil være den mest almindelige. AdvertisementFile propertyen forventer at modtage en sti til en XML fil. Filen skal være på en ganske særlig form og indholde følgende XML attributter:

1. *ImageUrl*. Dette er et link til en billedfil på serveren.
2. *NavigateUrl*. Adressen på den side indehaveren af reklamen vil have surferen til at se.
3. *AlternateText*. Hvis billedet af en eller anden grund ikke er tilgængeligt kan der vises en tekst som angivet i denne attribut.
4. *Keyword*. Der kan angives forskellige keywords for reklamerne således at AdRotatoren kan vise dem efter keywords. *KeywordFilter* propertyen er default en tom streng, men kan sættes til et givet keyword således at kun reklamer med det angivne keyword vises. Hvis *KeywordFilter* er en tom streng vises alle reklamer også selvom de har forskellige keywords.
5. *Impressions*. Benyttes til at angive hvor ofte en reklame skal forekomme i forhold til de andre reklamer i XML filen.

Ved brug af Add Item adderes en XML fil til projektet. Indholdet adderes direkte i Visual Studio 2005. Filen ser således ud:

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
<Ad>
  <ImageUrl>~/Reklamer/Reklame1.gif</ImageUrl>
  <NavigateUrl>~/Reklame1.aspx</NavigateUrl>
  <AlternateText>Dette er reklame 1</AlternateText>
  <Keyword>Reklame</Keyword>
  <Impressions>60</Impressions>
</Ad>
<Ad>
  <ImageUrl>~/Reklamer/Reklame2.gif</ImageUrl>
  <NavigateUrl>~/Reklame2.aspx</NavigateUrl>
  <AlternateText>Dette er reklame 2</AlternateText>
  <Keyword>Reklame</Keyword>
  <Impressions>30</Impressions>
</Ad>
<Ad>
  <ImageUrl>~/Reklamer/Reklame3.gif</ImageUrl>
  <NavigateUrl>~/Reklame3.aspx</NavigateUrl>
  <AlternateText>Dette er reklame 3</AlternateText>
  <Keyword>Reklame</Keyword>
  <Impressions>10</Impressions>
</Ad>
</Advertisements>
```



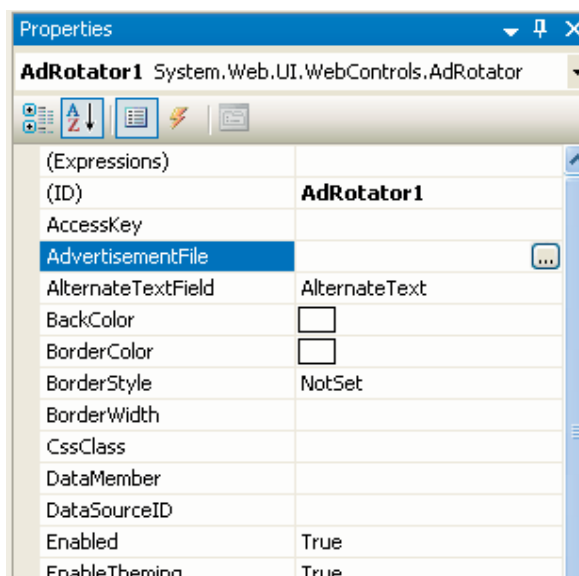
**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

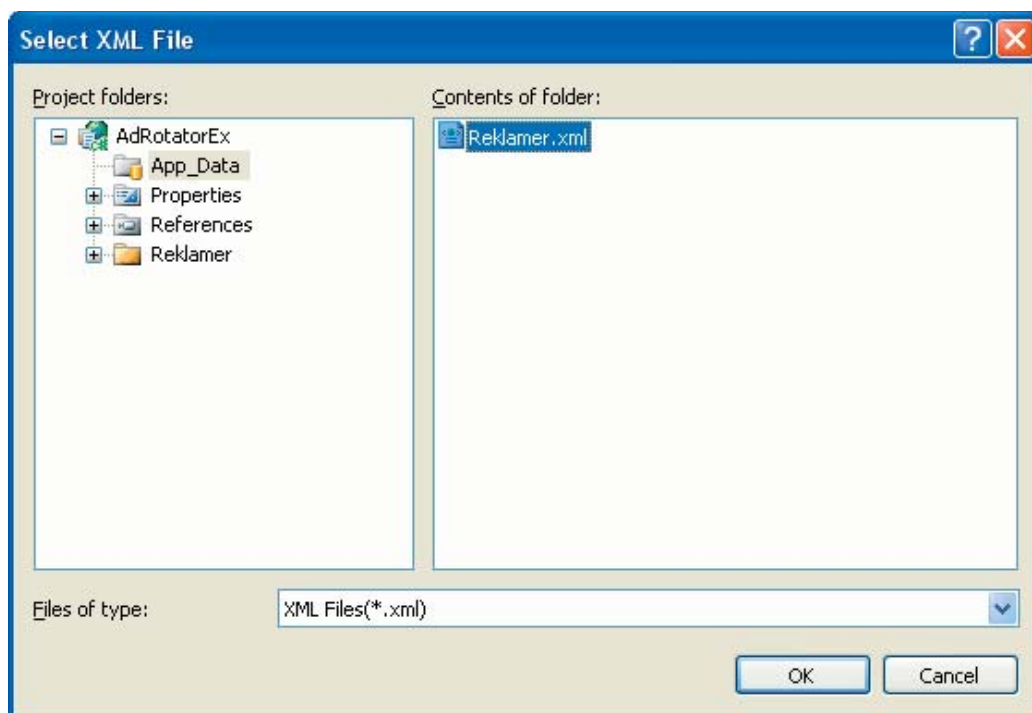


Læg mærke til hvordan det med *Impressions* styres hvor hyppigt en given reklame forekommer. *NavigateUrl* henviser i dette eksempel til Web Forms i roden af vores site, (Tilde '~' angiver roden af sitet) men i en virkelig applikation ville det være adressen på en side defineret af ejeren af reklamen. Jeg har oprettet en folder kaldet *Reklamer* hvor jeg har lagt gif filerne. I *ImageUrl* henvises der til filerne som vist ovenfor. I Visual Studio 2005 henvises der fra AdRotator kontrollen til XML filen som jeg har kaldt Reklamer.xml ved i property vinduet ud for AdvertisementFile propertyen at klikke på de tre punkummer (se nedenstående figur)



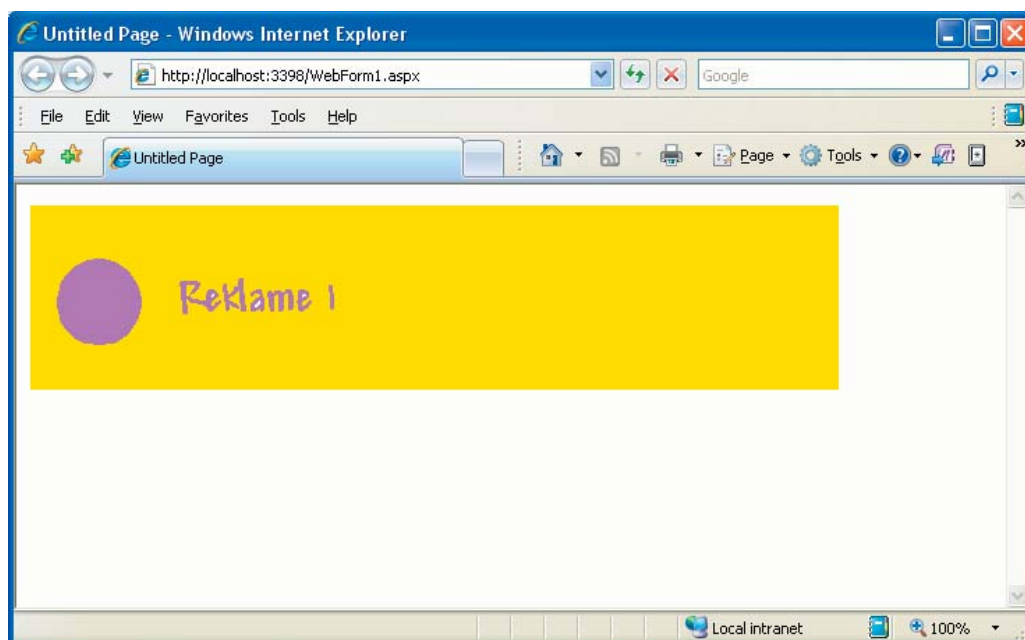
Figur 20. AdvertisementFile propertyen sættes.

Når man har klikket der får man dialogen som vist på Figur 21.



Figur 21. AdvertisementFile XML fil dialog vælger.

Jeg har lagt XML filen ned i en special folder kaldet App\_Data. Reklamer.xml vælges og der klikkes OK. App\_Data er en speciel folder til at gemme data til applikationen, som f.eks. xml filer, databaser etc., men ej billeder. En kørsel af applikationen giver følgende:



Figur 22. AdRotator der viser en reklame.

Dette afslutter eksemplet med AdRotator kontrollen. Jeg har for god ordens skyld prøvet at trykke på refresh knappen hele 112 gange og fandt at reklame 1 fremkom 66 gange, reklame 2 fremkom 35 gange og reklame 3 fremkom 11 gange. Det giver i runde tal at reklame hyppigheden er 59% for reklame 1, 31 % for reklame 2 og 10% for reklame 3, hvilket er i overensstemmelse med de i starten af afsnittet definerede hyppigheder for reklamerne.

## 3.5 Custom Controls

En custom control kan være rigtig nyttig idet man har mulighed for at samle en klump passende grafisk aktivitet i én kontrol. Det giver også en glimrende mulighed for genbrug af kode. Vi har et par forskellige muligheder for at lave en custom kontrol.

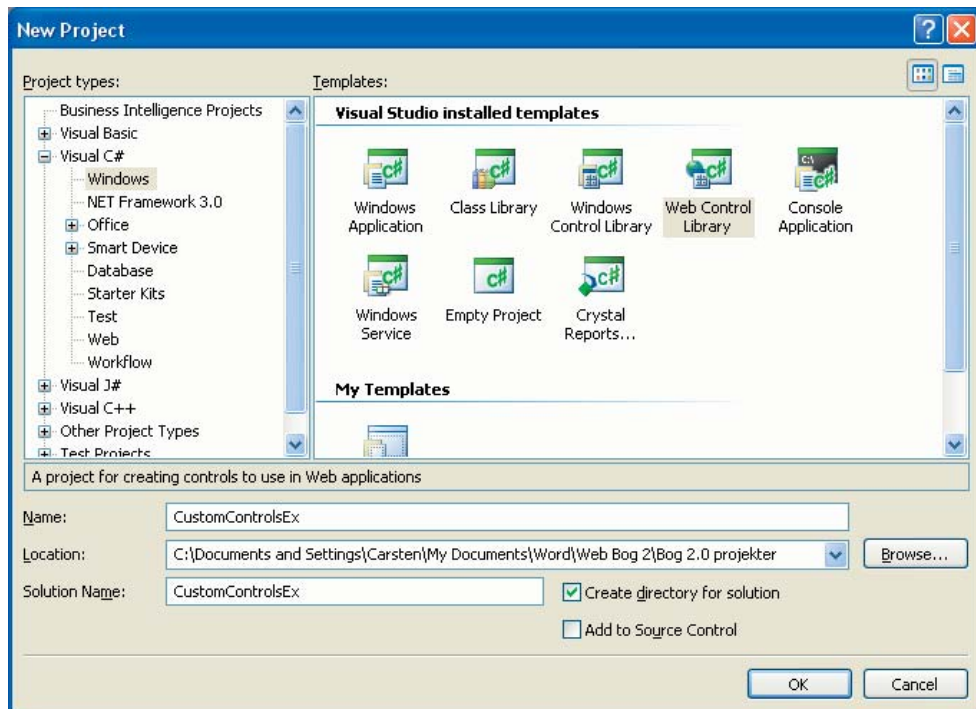
- Nedarve fra System.Web.UI.Control eller System.Web.UI.WebControl og selv skrive HTML koden ved at overskrive Render funktionen. Det kaldes for en *Custom control*.
- Nedarve fra System.Web.UI.UserControl og via den grafiske editor i Visual Studio 2005 indsætte de kontroller man har brug for. Det kaldes for en *User control*. Se også afsnit 0 for et eksempel med en user control.
- Nedarve fra System.Web.UI.WebControls.x hvor x angiver navnet på den web kontrol man ønsker at nedarve fra. Man tilpasser så at sige funktionaliteten af en given kontrol til ens behov.
- Nedarve fra System.Web.UI.Control og implementere interfacet InamingContainer . Dette kaldes også for en *Composite Custom Control*. I praksis gøres det ved at nedarve fra *CompositeControl*.

Forskellen mellem de forskellige kontrol typer er såmen ikke så stor. De største forskelle er nok at en *User Control* designs rent grafisk på en form, hvor de andre typer kodes uden understøttelse af forms. Dernæst genererer en *User Control* en .ascx fil som skal distribueres sammen med den compilerede kontrol, i de tilfælde man ønsker at sælge en sådan kontrol. Det giver et knap så pænt interface til brugeren som de andre kontrol typer, idet de kan compileres til rene .NET dll'er der ikke kræver andet end sig selv. Så længe man blot selv skal bruge kontrollen vil jeg foretrække *User Control* en frem for de andre fordi man kan designe den grafisk på en form. Lad os se på nogle simple eksempler for de forskellige kontrol typer.

### 3.5.1 En simpel Custom Control

Den helt simple custom control vil typisk bruges til blot at præsentere noget data, men man selvfølgelig mulighed for at lave noget interaction med brugeren. Jeg vil dog i dette eksempel blot vise hvordan man laver en Custom Control. Kontrollen skal kunne vise en tekst, der skal kunne sættes via en property.

For at skabe en Custom Control opretter jeg et *Web Control Library* project. Jeg kalder eksemplet CustomControlsEx.



Figur 23. Custom Control eksempel oprettes

Når det er gjort har Visual Studio 2005 oprette noget default kode som ser således ud:

**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
 By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium** is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**

```
1  [DefaultProperty("Text"),
2      ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>")]
3  public class WebCustomControl1 : WebControl
4  {
5      [Bindable(true)]
6      [Category("Appearance")]
7      [DefaultValue("")]
8      [Localizable(true)]
9      public string Text
10     {
11         get
12         {
13             String s = (String)ViewState["Text"];
14             return ((s == null) ? String.Empty : s);
15         }
16
17         set
18         {
19             ViewState["Text"] = value;
20         }
21     }
22
23     protected override void RenderContents(HtmlTextWriter output)
24     {
25         output.Write(Text);
26     }
27 }
28
```

Som det kan ses har Visual Studio 2005 faktisk genereret kode som løser opgaven. Lad mig dog lige forklare hvad koden betyder.

### Linie 1-3:

Man lægger straks mærke til de kantede parenteser som vi kender i forbindelse med arrays i C#. Som de optræder her er betydning dog en anden der også burde kunne se af konteksten. Her optræder de som *attributter*. En attribut<sup>iii</sup> er i denne sammenhæng et deklarations mærke, der fortæller noget om den efterfølgende kode. En attribut afvikles ikke som kode, men benyttes enten af compileren, Visual Studio 2005 IDE eller endda runtime til at ”forstå” hvordan den efterfølgende kode skal behandles. Når det er sagt så er det faktisk muligt direkte i koden at få information om attributterne gennem de objekter der er til rådighed System.Reflection namespace. Microsoft har valgt at kalde det for *Reflection* når man runtime spørger efter attributter. Faktisk dækker begrebet meget mere, men at behandle emnet her er alt for omfattende og jeg vil istedet henvise til MSDN og bøger om C#.

*DefaultProperty* attributten er jo nærmest selvforklarende, idet den fortæller at propertyen Text for denne control er default, altså den der default er valgt når componenten benyttes i designeren. *ToolboxData* attributten benyttes til at fortælle Visual Studio 2005 IDE hvordan kontrollen skal præsenteres sig selv. Man kunne også her angive værdier for de forskellige properties. F.eks. kunne Text propertyen sættes således:

```
ToolboxData("<{0}:WebCustomControl1 Text = 'Hello World'
runat=server></{0}:WebCustomControl1>")
```

### Linie 6-9:

Her har vi også med IDE attributter at gøre. *Bindable* fortæller at propertyen Text skal vises i DataBindings dialog boxen. *Category* fortæller hvor der propertyen skal placeres når der sorteres efter kategorier i property vinduet for kontrollen. *DefaultValue* giver default værdien i IDE designeren. *Localizable* markerer om propertyen er forberedt til lokalisering, dvs. håndtering af flere sprog.

### Linie 24-27:

HtmlTextWriter klassen benyttes til at ”hælde” data ud i. Det som kontrollen skal vise skrives ud i den stream som HtmlTextWriter repræsenterer i form af output variabelen.

## 3.5.2 En Composite Custom Control

En *composite custom control* er en kontrol der udnytter eksisterende kontroller i kombination med hinanden. Et eksempel kunne være en kontrol der indholder en label kontrol, en tekstboks kontrol og en validerings kontrol. Fordelen ved en composite kontrol fremfor en UserControl er nævnt i afsnit 0.

Lad os i dette eksempel skrive en *composite custom control*, der kan validere et telefon nummer.

Kontrollen opbygges af en label, en tekstboks og en validerings kontrol. Kontrollen nedarver fra INamingContainer interfacet og fra System.Web.UI.Control via *CompositeControl* klassen.

INamingContainer interfacet er et interface uden metoder. Der er derfor intet at implementere, men interfacet har selvfølgelig alligevel en funktion. INamingContainer interfacet tjener som markør og sikre at de kontroller der benyttes indenfor ens composite kontrol har helt unikke navne. Det er vigtig når der f.eks. nedarves fra ens composite



kontrol, hvis der benyttes databinding eller hvis der skal dirigeres events ud til child kontrollerne<sup>iv</sup>. For dig er der sådan set ikke noget arbejde forbundet med *INamingContainer* fordi *CompositeControl* allerede nedarver fra interfacet.

Når man implementere en composite kontrol skal man implementere *CreateChildControls* metoden. I den skal man oprette de child kontroller man skal benyttes og addere dem til ens composite kontrol. Det gøres via *Controls* proprieten som ligger på *System.Web.UI.Control* klassen. *Controls* proprieten er en kollektion af de kontroller der udgør ens composite kontrol.

Der er ikke en template til at oprette en *CompositeControl*, så vi benytter Web Custom Control template istedet. Gå ind i Visual Studio og vælg Add New Item. Vælg Web Custom Control og navngiv filen *CompositeCustomControl*.

Gå ind i filen og erstat linien

```
public class CompositeCustomControl : WebControl
```

Med linien

```
public class CompositeCustomControl : CompositeControl
```



**MTHøjgaard**

## BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



Det eneste der er sket er at WebControl er erstattet med CompositeControl. Fjern nu også RenderContents metoden og erstat den ved at override CreateChildControls metoden.

Koden der udgør vores kontrol er vist efterfølgende og en forklaring følger umiddelbart efter:

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;

namespace CustomControlsEx
{
    /// <summary>
    /// Summary description for CompositeCustomControl.
    /// </summary>
    [DefaultProperty("TelefonNummer"),
     ToolboxData("<{0}:CompositeCustomControl  

     runat=server></{0}:CompositeCustomControl>")]
    public class CompositeCustomControl : System.Web.UI.Control,
        INamingContainer
    {
        private string telefonNummer;

        [Bindable(true),
         Category("Appearance"),
         DefaultValue("")]
        public string TelefonNummer
        {
            get
            {
                return telefonNummer;
            }

            set
            {
                telefonNummer = value;
            }
        }

        protected void TjekTelefonNummer(object source,
            ServerValidateEventArgs args)
        {
            try
            {
                long.Parse(args.Value);
                if(args.Value.Length != 8)
                {
                    throw new SystemException("Forkert længde");
                }
            }
        }
    }
}
```

```

        args.IsValid = true;
    }
    catch(System.Exception err)
    {
        args.IsValid = false;
    }
}

protected override void CreateChildControls()
{
    TextBox box = new TextBox();
    Label label = new Label();
    CustomValidator val = new CustomValidator();
    box.ID = this.UniqueID + "box";
    box.Text = telefonNummer;

    label.ID = this.UniqueID + "label";
    label.Text = "Telefon nummer: ";

    val.ID = this.UniqueID + "validator";
    val.ErrorMessage = "Telefon nummeret er ikke korrekt";
    val.ControlToValidate=box.ID;
    val.ServerValidate += new
        ServerValidateEventHandler(TjekTelefonNummer);

    this.Controls.Add(label);
    this.Controls.Add(box);
    this.Controls.Add(val);
}
}
}

```

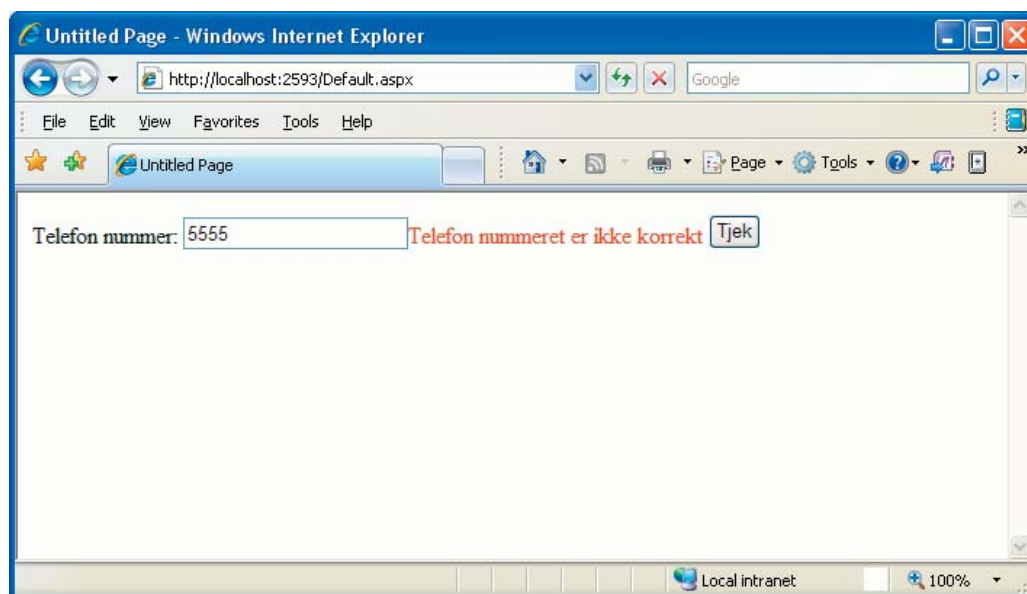
Det interessante i denne custom control sker i CreateChildControls. Læg mærke til at der oprettes ialt tre kontroller, der til sidst i metoden adderes til Controls kollektionen. Rækkefølgen er vigtig i dette tilfælde fordi kontrollerne præsenteres i den rækkefølge de adderes. Da jeg benytter en CustomValidator er jeg nødt til at definere en funktion til at lave validering hvilket gøres ved at tildele ServerValidate delegate funktionen på CustomValidator kontrollen en metode at kalde, nemlig TjekTelefonNummer. I TjekTelefonNummer tjekkes om nummeret kan parses til et tal og om længden af nummeret er ialt 8 tegn. Hvis long.Parse fejler kastes en exception. Så har vi ikke med et tal at gøre, men hvis begge betingelser er opfyldt sættes IsValid til true. Hvis en af betingelserne fejler kastes der en exception, der fanges i catch blokken. Heri sættes IsValid til false. Når IsValid er false skrives ErrorMessage på CuctomValidatoren på skærmen.

Der oprettes et test projekt kaldet TestCustomControlsEx i samme solution. Adder en projekt reference til CustomControlsEx projektet og åbn nu Default.aspx filen og tilføj følgende:

```
<body>
    <form id="form1" runat="server">
        <ccl:CompositeCustomControl runat="server" TelefonNummer="5555"
ID="CompositeCustomControl1" />
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Tjek" />
    </form>
</body>
```

Som det kan ses forsynes vores CompositeControl med et telefonnummer der er forkert. Vi undersøger om kontrollen virker.

En kørsel ser således ud:



Figur 24. Composite custom kontrol.

## 4. Cookies

Mange har sikkert hørt om cookies, både godt og skidt. Der har i gennem de sidste mange år været skrivi om sikkerheds brister i Internet Explorere, der kompromiterede brugerens maskine. Alt sammen relateret til cookies. Cookies har derfor fået et noget blakket rygte og mange undgår dem bevist. Jeg har tidligere haft en kollega, der var (og stadig er) hysterisk omkring cookies. Ingen får lov at lægge noget der bare dufter lidt hen af cookies på hans maskine og mange har det stadig som ham. Skal man så for alt i verden undgå cookies? Svaret er nej, men sæt dig ind i hvad en cookie er. Læs f.eks. på <http://www.cookiecentral.com/faq/> og find svar på alle spørgsmål du måtte have om cookies. Grundlæggende er en cookie en bit information som din browser lægger på din maskine når en server beder om lov til det. Når du senere vender tilbage til serveren kan cookien benyttes af serveren til at yde en service for dig. Hvis du f.eks. vender tilbage til en site der dækker både windsurfing og kitesurfing kunne en cookie gemme information om hvilken af disse to disipliner du foretrækker og sørger for at præsentere lige præcis den information for dig. En given server kan også kun læse cookies den selv har bedt om at få gemt på din computer og intet andet. Vi har dog set fejl i browserne der har gjort det muligt for servere at læse cookies de ikke selv havde lagt. Det er derfor vigtigt ikke at bruge cookies til at gemme password og følsomme oplysninger. Hvis du alligevel gør det så sørg i det mindste for at kryptere oplysningerne. I det eksempel jeg gav før kan jeg godt leve med at det slap ud at jeg foretrækker windsurfing frem for kitesurfing<sup>v</sup>, men det er dig der skal afgøre hvor langt du vil gå med cookies på din maskine. Grundlæggende er der to slags cookies, nemlig *session cookies* og *persistent cookies*. Session cookies gemmes i klient maskines hukommelse og lever kun så længe browseren er åben, mens persistent cookies skrives på klientens maskines harddisk. Det er god skik at spørge brugeren om lov til at gemme en persistent cookie på maskinen, så husk venligst det næste gang du laver en website der benytter cookies!

Jeg vil nu vise hvordan du kan gemme cookies på en maskine og hvordan du kan læse dem igen.

Eksemplet nedenfor gemmer en cookie på klient maskinen, der indholder information om hvornår klienten sidst var på besøg på siten. Opret et Web projekt kaldet CookiesEx og gå ind i Page\_Load og skriv følgende:

```

1  private void Page_Load(object sender, EventArgs e)
2  {
3      const string name = "test cookie";
4
5      if(Request.Cookies[name] == null)
6      { //Adder cookien, da den ikke findes hos klienten
7          HttpCookie dato = new HttpCookie("dato");
8          dato.Value = System.DateTime.Now.ToString();
9          dato.Expires = DateTime.Now.AddYears(1);
10         dato.Name = name;
11
12         Response.Output.WriteLine("Her på denne site leges der med
13 cookies. ");
14         Response.Output.WriteLine("Der er adderet en cookie ");
15         Response.Output.WriteLine(dato.Value.ToString());
16
17         Response.Cookies.Add(dato);
18     }
19     else
20     { //Læs cookien
21         string sidsteBesøg = Request.Cookies[name].Value;
22
23         Response.Output.WriteLine("Denne site er sidst besøgt " +
24 sidsteBesøg);
25         Response.Cookies[name].Value = DateTime.Now.ToString();
26         Response.Cookies[name].Expires = DateTime.Now.AddYears(1);
27     }

```

**Linie 5:**

Der tjekkes om cookien med navn `test_cookie` findes på maskinen i forvejen. Request propertyen giver adgang til at spørge på information fra klienten bla. på cookies.

**Linie 7-10:**

Cookien oprettes ved brug af `HttpCookie` klassen. Constructoren tager navnet på cookien som parameter hvilket i dette tilfælde er `dato`. Propertyen `value` tildeles serverens tidsinformation. Propertyen `expires` tildeles servens tidsinformation plus 1 år. Dvs. at cookien er gyldig 1 år.

**Linie 12-15:**

Response propertyen benyttes til at skrive tekst direkte til den side som sendes tilbage til klienten.

**Linie 17:**

Cookien adderes til Response objektet og før det sendes tilbage til klienten. **Vær opmærksom på at der IKKE er nogen garanti for at cookien rent faktisk ender på klientens maskine!**

**Linie 21:**

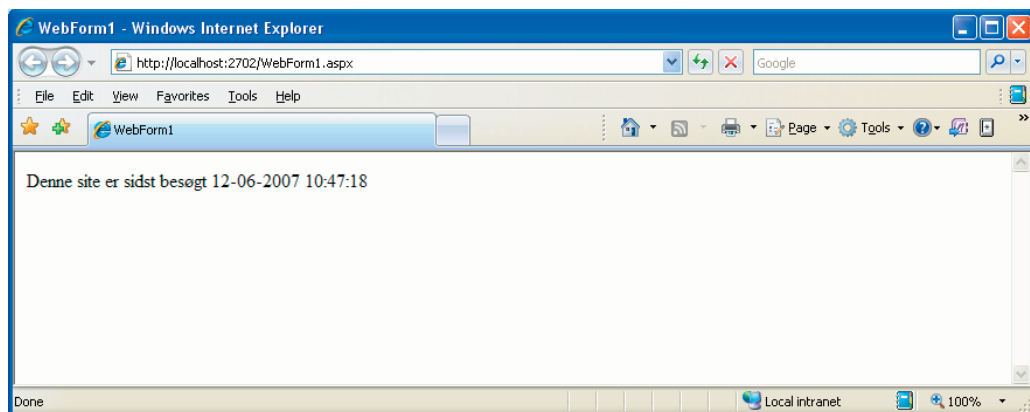
Request propertyen benyttes til at læse værdien af cookien. Hvilket er information om hvornår sidste besøg på siden fandt sted.

**Linie 25-26:**

Cookien opdateres med serverens nuværende tidsinformation.

På `HttpCookie` objektet er der mulighed for at gemme flere værdier i `values` kollektionen. Brug dog cookies med omtanke. Cookies er tiltænkt korte tekst informationer og du bør (læs **skal**) derfor ikke gemme objekter i en cookie.

En kørsel giver følgende:



Figur 25. Cookie eksempel



**Linie 5:**

Der tjekkes om cookien med navn `test_cookie` findes på maskinen i forvejen. Request propertyen giver adgang til at spørge på information fra klienten bla. på cookies.

**Linie 7-10:**

Cookien oprettes ved brug af `HttpCookie` klassen. Constructoren tager navnet på cookien som parameter hvilket i dette tilfælde er `dato`. Propertyen `value` tildeles serverens tidsinformation. Propertyen `expires` tildeles servens tidsinformation plus 1 år. Dvs. at cookien er gyldig 1 år.

**Linie 12-15:**

Response propertyen benyttes til at skrive tekst direkte til den side som sendes tilbage til klienten.

**Linie 17:**

Cookien adderes til Response objektet og før det sendes tilbage til klienten. **Vær opmærksom på at der IKKE er nogen garanti for at cookien rent faktisk ender på klientens maskine!**

**Linie 21:**

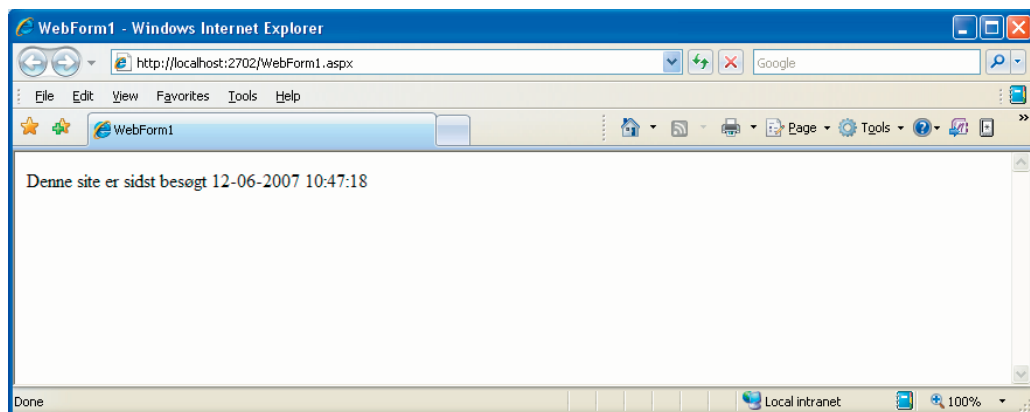
Request propertyen benyttes til at læse værdien af cookien. Hvilket er information om hvornår sidste besøg på siden fandt sted.

**Linie 25-26:**

Cookien opdateres med serverens nuværende tidsinformation.

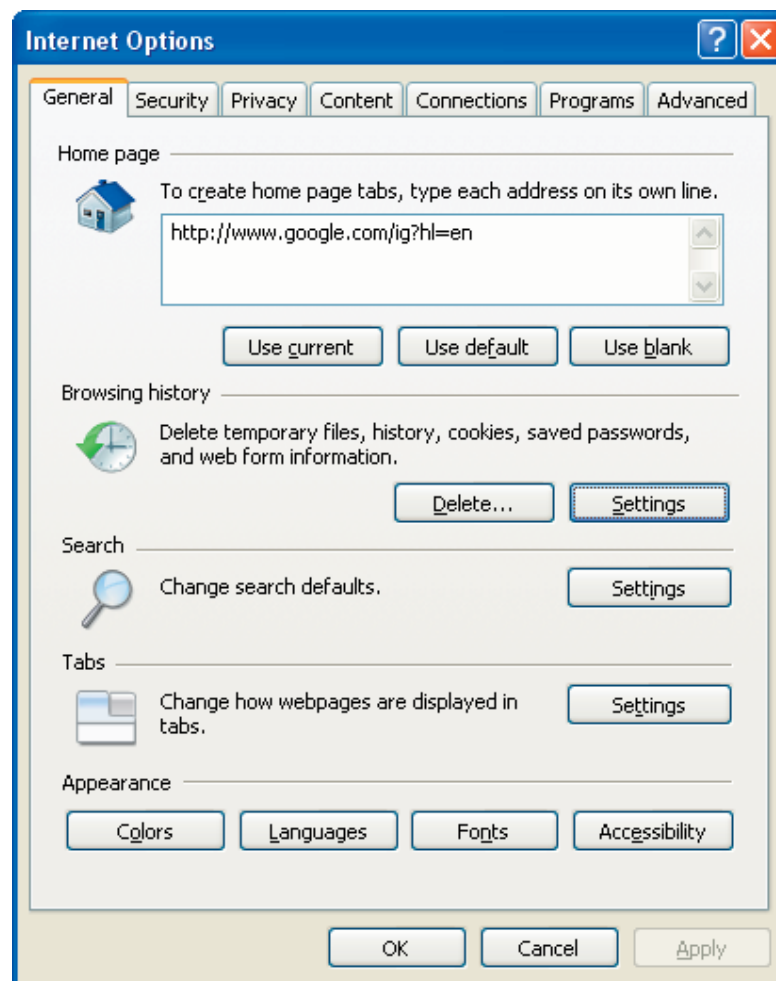
På `HttpCookie` objektet er der mulighed for at gemme flere værdier i `values` kollektionen. Brug dog cookies med omtanke. Cookies er tiltænkt korte tekst informationer og du bør (læs **skal**) derfor ikke gemme objekter i en cookie.

En kørsel giver følgende:

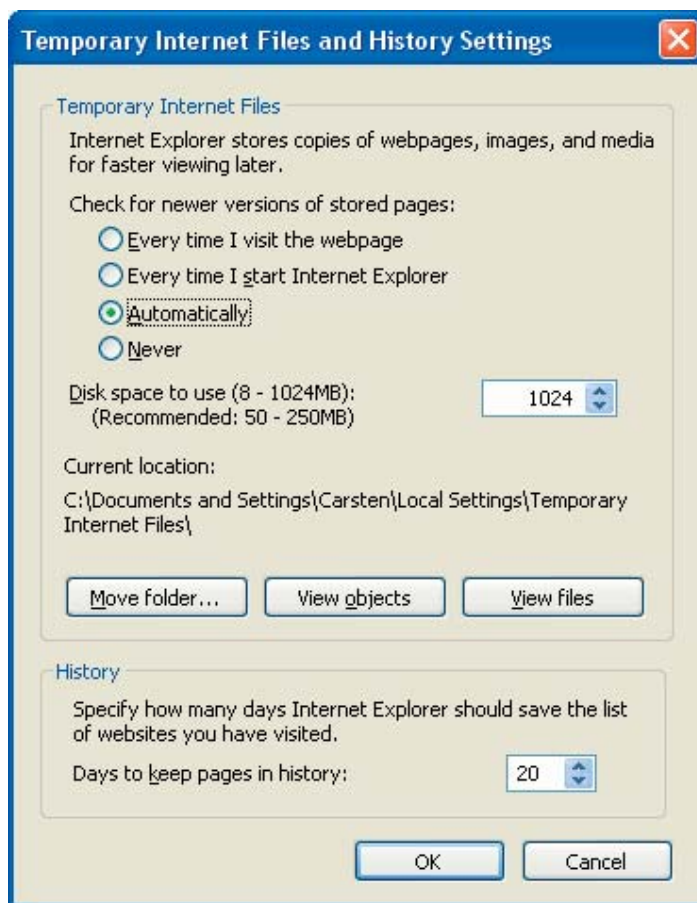


Figur 25. Cookie eksempel

Cookien kan findes ved at gå ind i Explorer under Tools\Internet Options. Det giver dialogen



Vælg nu under Browsing history knappen Settings. Man får nu følgende:



Vælg View Files. Sorter på Last Accessed og din cookie vil stå øverst (el. nederst afhængig af sorteringen). Kopier den ud i en passende folder. Det interessante i filen kan ses her:

```
test cookie
12-06-2007 10:48:00
localhost/
1024
76308480
29936745
1643091072
29863118
```

## 5. En mini shop

Lad os integrere endnu flere Web controller i dette eksempel, der viser en lille mini shop, som også vil gøre brug af en database. Lad os forestille os at vi har en lille butik som sælger DVD film, der nu gerne vil udstille en del af sine vare på nettet og samtidig give brugerne mulighed for at bestille direkte over nettet.

### 5.1 Databasen Mini Shop

Til vores mini shop vil det være hensigtsmæssigt at benytte sig af en database til opbevaring af kundeoplysninger, kundens ordre og varerne som kunden har bestilt. Desuden ville det være praktisk hvis man samtidig kunne holde styr på hvor mange genstande der er tilbage af en given vare<sup>vi</sup>. Der er mange gode databaser på markedet idag og valget af database kan gøres ud fra pengepungen og behov. Hvis behovet for en kraftig superhurtigt database ikke er så stort (som i dette tilfælde) kan en database som Microsoft Acces fint slå til. Er behovet for hurtig adgang til data, men samtidig en billig ja næsten gratis database så kunne MySQL være et bud. Den kan frit downloades og benyttes, men gør du det kommercielt skal du betale for en licens. Jeg har dog valgt at benytte Microsoft SQL Server 2005 fordi den er så let at arbejde med. Den har et lækkert brugerinterface og så er den jo samtidig en kraftig og scalerbar database, der vil kunne dække selv de største behov. De mest fremherskende database typer er idag som det har været i mange år relations databaser. Faderen til relations databasen er fra IBM og hedder iøvrigt Dr. Edgar F. Codd. Relations databasen er iøvrigt baseret på Set Teori, samt Predicate Logic.



**Ses vi til DSE-Aalborg?**

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**



Jeg antager at du er bekendt med databaser i nogen grad og antager at du har banalt kendskab til SQL (Structured Query Language). Så går vi igang.

## 5.2 Oprettelse af databasen

I SQL Server Management Studio opretter vi en database med navnet Mini Shop til brug for eksemplet. Gå ind i Management Studio og højreklik på Databases. Vælg New Database og kald den for MiniShop.

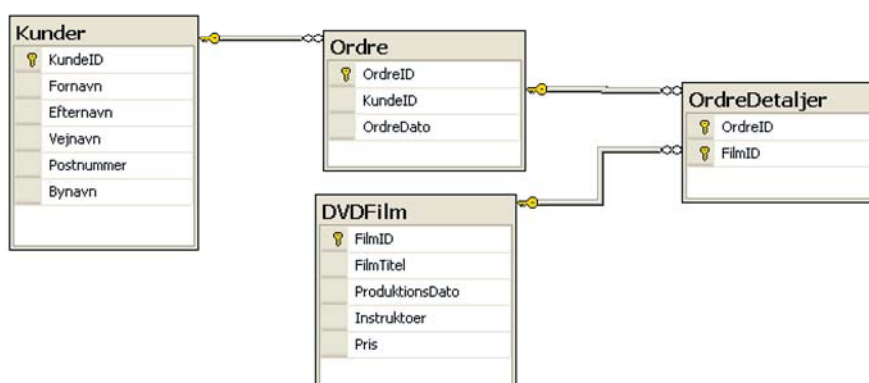
Vi har nu en tom database klar til at blive forsynet med de tabeller og views vi har brug for.

## 5.3 Tabellerne i mini shoppen oprettes

Lad os holde det simpelt (KISS: Keep It Simple and Stupid) og lave en ganske kort analyse af hvad der er behov for.

- Kunde information.
- DVD film
- Order

Det er banalt set hvad der er behov for i databasen. Det giver følgende database design:



Figur 26. Mini shop database design

Som man kan se af Figur 26 er der ialt 4 tabeller og det skyldes at der er inkluderet tabellen *OrdreDetaljer* som en "hjælpe" tabel for at give kunden mulighed for at købe mere end én film af gangen. *Kunder* benyttes til at registrere de kunder som afgiver en ordre. Hver kunde bliver i forbindelse med en ordre registreret i *Ordre* tabellen med *KundeID* fra *Kunder* tabellen. Denne en til mange relation sikre at hver kunde kan afgive mere end én ordre hvilket vi selvfølgelig gerne vil have. Vi vil også gerne give kunden mulighed for at købe mere end én film af gangen og derfor er *OrdreDetaljer* tabellen tilstede. Her finder vi *OrdreID* og id'et på den film som kunden har bestilt. I tabellen *DVDFilm* finder vi informationer om DVD filmene. Vores database er nu komplet.....ja ja, jeg ved godt at vi sagtens

kan lave systemet meget bedre, men det her er kernen og login, email svar og online betaling osv er noget vi kan hæfte på senere for at forbedre service niveauet. Læg iøvrigt mærke til at hvert *field* i databasen er unikt. Det er værd at skrive sig bag øret hvis du ikke allerede har gjort det. Flere steder i databasen forekommer der et ID f.eks. KundeID og OrdreID. Disse to felter (*field*) kunne ligeså godt have heddet ID begge to idet de jo er i hver deres tabel. Sådan er det også i mange databaser idag! Sørg for at hvert felt har et unikt navn og at navnet entydigt siger hvad der skal fyldes i det. Nu burde vi designe en række Views til vores database, men jeg nøjes med et blot for at vise princippet i Views og laver derfor kun et. Der bør også benyttes stored procedures og man bør have et egentlig DAL (Data Access Layer), men det springer vi over her. Kig evt. på MyGeneration ([www.mygenerationsoftware.com](http://www.mygenerationsoftware.com)) og på NHibernate for gratis værktøjer, der kan lave et DAL lag ud fra tabellerne i basen. Istedet for flere Views i databasen vil jeg benytte ADO.NET til at opbygge de queries der er brug for. Grunden til det er at lære dig at benytte ADO.NET.

## 5.4 Views oprettes i Mini Shop databasen

Views er nyttige fordi de giver mulighed for direkte i database designet at implementere hvordan data skal præsenteres. Vi kunne i princippet ligeså godt selv skrive SQL koden fra vores applikation og så sende den til databasen via ADO.NET. Fordelen ved Views er dog imidlertid at de kan forblive uændrede selvom de underliggende tabeller ændre sig. Derudover vil et View give hurtigere respons end hvis vi sendte SQL koden til databasen, idet vi så først skulle generere vores SQL statement og dernæst sende den til databasen der så til sidst udføre den egentlige søgning.



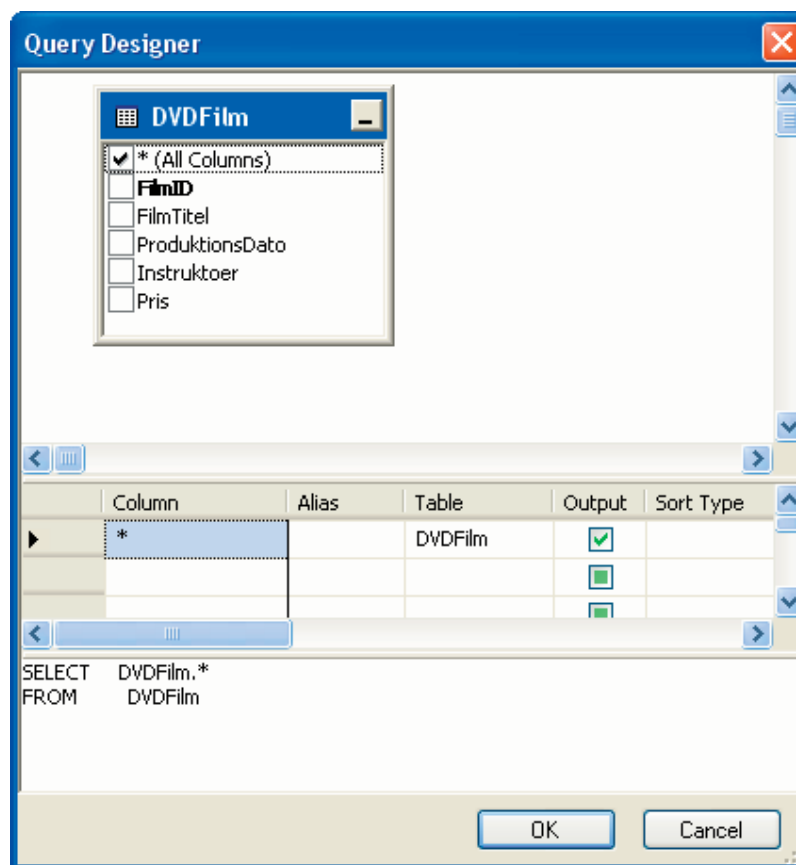
**CISO Conference**  
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**





Figur 27. Film View

I vores Mini Shop har vi behov for et View der kan vise kunden de DVD film der er i shoppen. I første omgang skal han blot kunne se dem alle og sortere på titel, produktions dato og instruktør. For at hente alle film i databasen (hvilket man normalt ikke ville gøre) kan viewet på Figur 27 benyttes.

## 5.5 NET Mini Shoppen oprettes

På tilsvarende vis som vist tidligere oprettes et ASP.NET web projekt i C#. Jeg har kaldt det for *MiniShop*. Enhver website har en velkomst side hvilket vi naturligvis også skal have. Som tidligere nævnt ser vi bort fra fin og flot grafik som det iøvrigt kun er de færreste udviklere der er i stand til at lave alligevel. Jeg vil holde mig fra det og istedet fokusere på kerne funktionaliteten. På vores velkomst side skal brugeren have mulighed for at finde adresse oplysninger og online kontakt information, samt have mulighed for at navigere hen til oversigten over film.

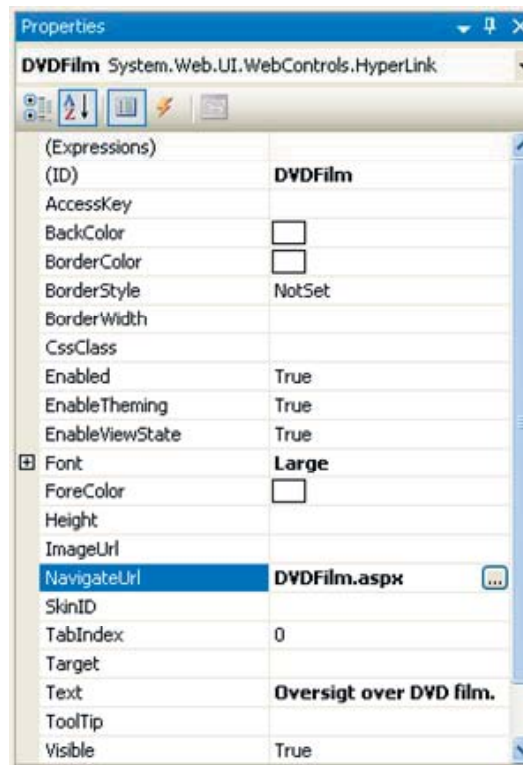
Som det kan ses på Figur 28 er velkomst siden meget primitiv.



Figur 28. Mini Shop velkomst side.

Den er bygget op af *Label* kontroller samt to *Hyperlink* kontroller. *Hyperlink* kontrollerne benyttes til at skifte side når brugeren klikker på dem med musen.

Jeg har udover velkomst siden oprettet en side der viser samtlige DVD film samt en side der viser hvad der er i indkøbsvognen. De to sider skal nu bindes til de to *Hyperlink* kontroller. Det gøres ved at sætte *NavigateUrl* propertyen på *Hyperlink* kontrollen til den side man vil navigere brugeren hen på.



Figur 29. Properties for en hyperlink kontrol



**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**

På Figur 29 kan du se at for *Hyperlink* kontrollen, der navigerer brugeren hen til de DVD film, der kan købes har sat *NavigateUrl* til *DVDFilm.aspx*.

## 5.6 ADO.NET. DataSet

I ADO.NET benyttes ofte et DataSet som kan betragtes som en delmængde af den rigtige database. Du kan benytte et DataSet til at repræsentere de data brugeren har behov for på et givet tidspunkt.

*Advarsel: Forsøg aldrig at loaded en hel produktions database i et DataSet.*

Et DataSet er disconnected fra DataBasen. Det giver en masse fordele, idet databasen ikke behøver at holde styr på en masse forbindelser på en gang. Omvendt kræver det noget arbejde at flætte de disconnectede ændringer i DataSettet sammen med eksisterende data. En fysisk instance af DataSet klassen består af en kollektion af DataTable objekter. Dvs. at et DataSet for vores database f.eks. kunne indeholde data fra tabellerne Kunder og Ordre. DataSet objektet taler ikke direkte med databasen, idet det overlades til en data adapter. En data adapter virker tovejs, idet den både kan anvendes til at til at afvikle SELECT udtryk og INSERT, UPDATE og DELETE udtryk. For nuværende findes der en SqlDataAdapter, en OleDbDataAdapter, en OdbcDataAdapter og en OracleDataAdapter, men der kommer sikkert flere. SqlDataAdapteren er optimeret til SQL Server, mens OleDbDataAdapteren benyttes til alle OLE Db databaser dvs. at også SQL Server kan kontaktes via denne. Odbc og Oracle dataadapterne benyttes ikke overraskende til henholdsvis odbc data kilder og Oracle baser. Data adapteren står for kontakten til databasen og giver mulighed for at ”fylde” et DataSet via Fill metoden. En DataTable består af DataRows og et DataRow består af DataColumnns. I henhold til SQL standarden er dette meget fornuftigt idet den foreskriver at man skal benytte ordene *table*, *row* og *column* for at henvise til strukturen i en database. I det daglige benytter jeg dog stadig af og til ordene *table*, *recordset* og *field* om de samme strukturer.

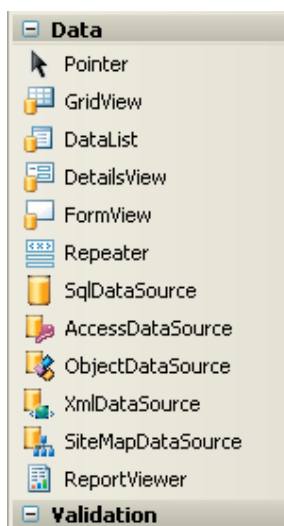
*Generelt gælder det at data til en Web Form skal hentes så hurtigt som muligt og derfor er det ikke altid fornuftigt at vælge et dataset som fyldes for hvert round trip. Hvis du vælger et dataset så forsøg at benytte cache.*

Typisk vil du benytte en eller flere dataadapters til at fylde data i dit dataset og så binde datasettet til en kontrol. Det var også det man gjorde i ASP.NET 1.1, men nu er det blevet endnu lettere. Bemærk at du stadig kan benytte dataset og dataadapters og at du bør overveje at benytte det hvor det giver mening. Det kan f.eks. være hvis data fra flere tabeller skal vises, hvis du implementere et DAL lag, hvis data skal udveksles med en Web Service, hvis du skal lave beregninger på hver række og der er sikkert flere eksempler. For brugen af datasets og dataadapters henvises til MSDN.

## 5.7 SqlDataSource og GridView

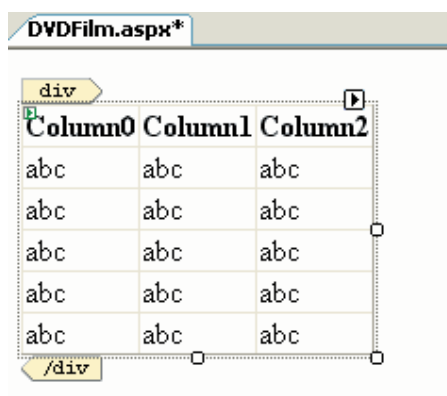
Ved brug af SqlDataSource og GridView kontrollerne kan du uden at skrive én linie kode hente data fra en underliggende kilde og få den præsenteret i GridView kontrollen.

Opret en ny Web Form og kald den DVDFilm.aspx. Gå til toolboxen og under Data sektionen tag fat i GridView (se Figur 30)



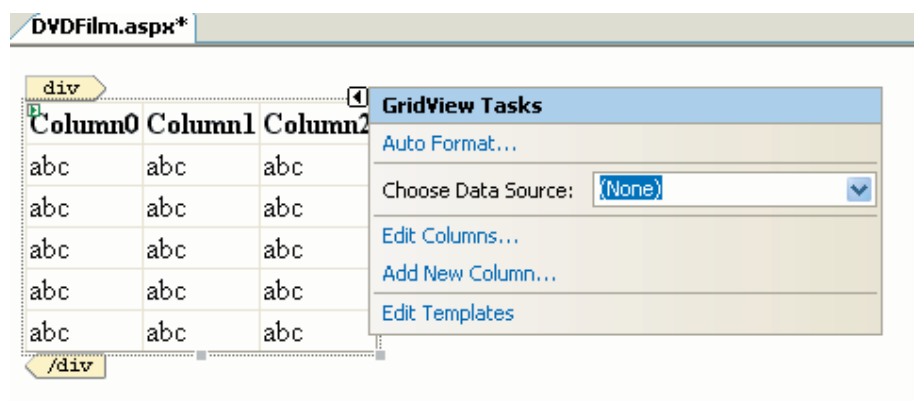
Figur 30. Data sektionen i Toolboxen

og drag den over på Web Formen (se Figur 31).



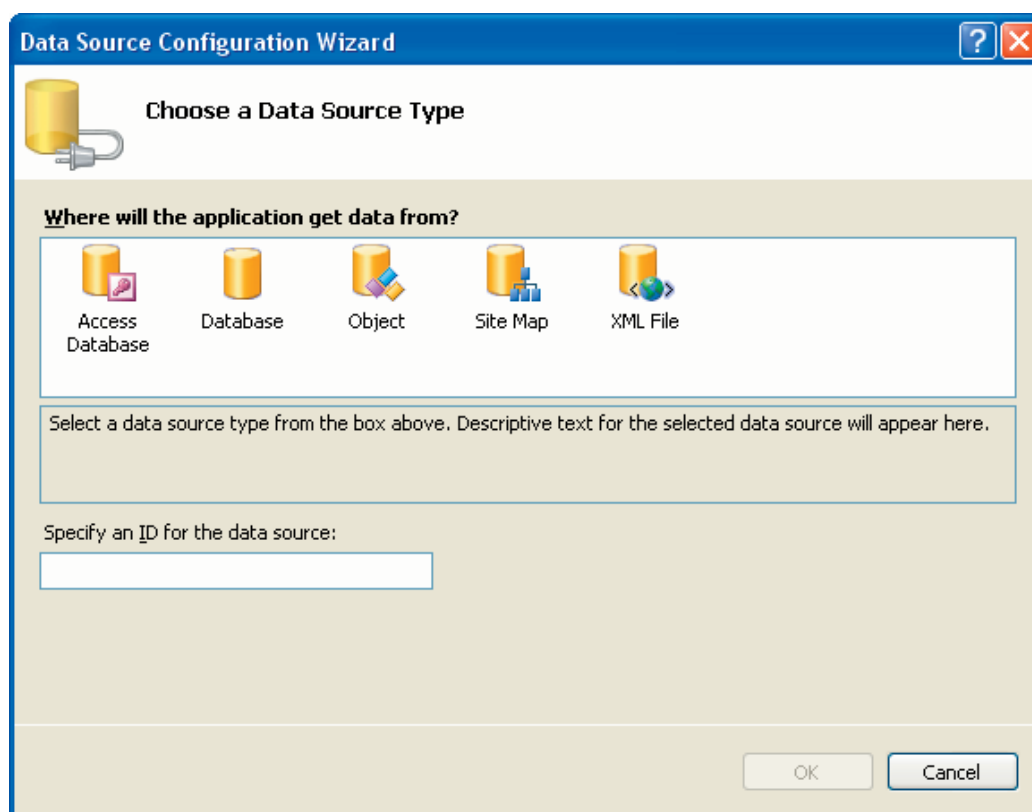
Figur 31. GridView kontrol

Som det kan ses er der en lille pil i øverste højre hjørne af kontrollen. Det er en smart tag. Ved at vælge den vil en wizard nu hjælpe os med at vælge vores datakilde. Når der klikkes på pilen ser det således ud (se Figur 32):



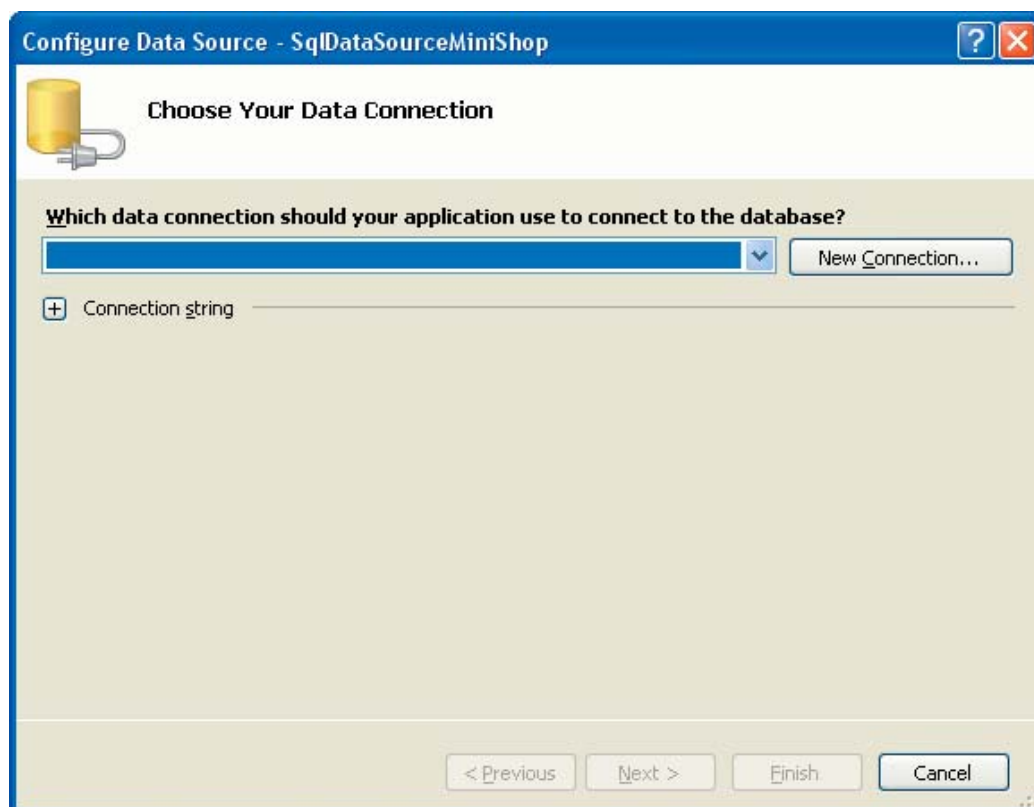
Figur 32. GridView valg af data kilde

Under Choose Data Source vælges *New Data Source* og man får følgende muligheder



Figur 33. Valg af data kilde

Da vi skal hente data fra SQL serveren vælges Database. Under *Specify an ID for data source* skriver du `SqlDataSourceMiniShop` og klikker ok. En ny dialog til specificering af connection strengen til databasen popper nu op (se Figur 34).



Figur 34. Specificering af connection streng



**MTHøjgaard**

## BEDRE LØSNINGER

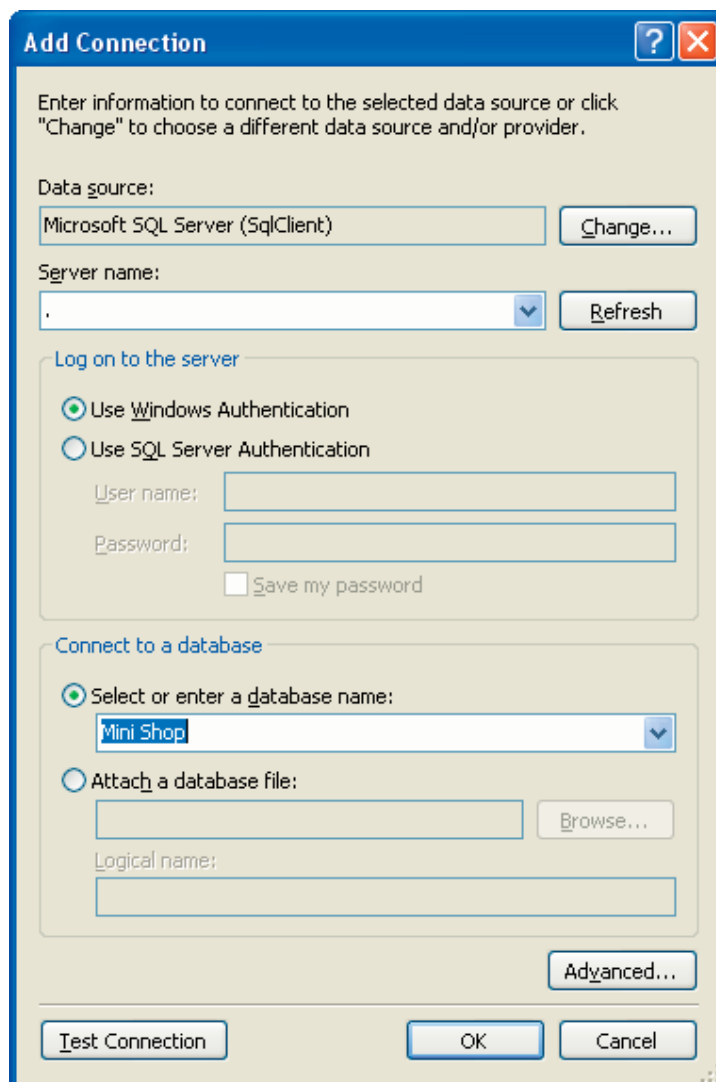
I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



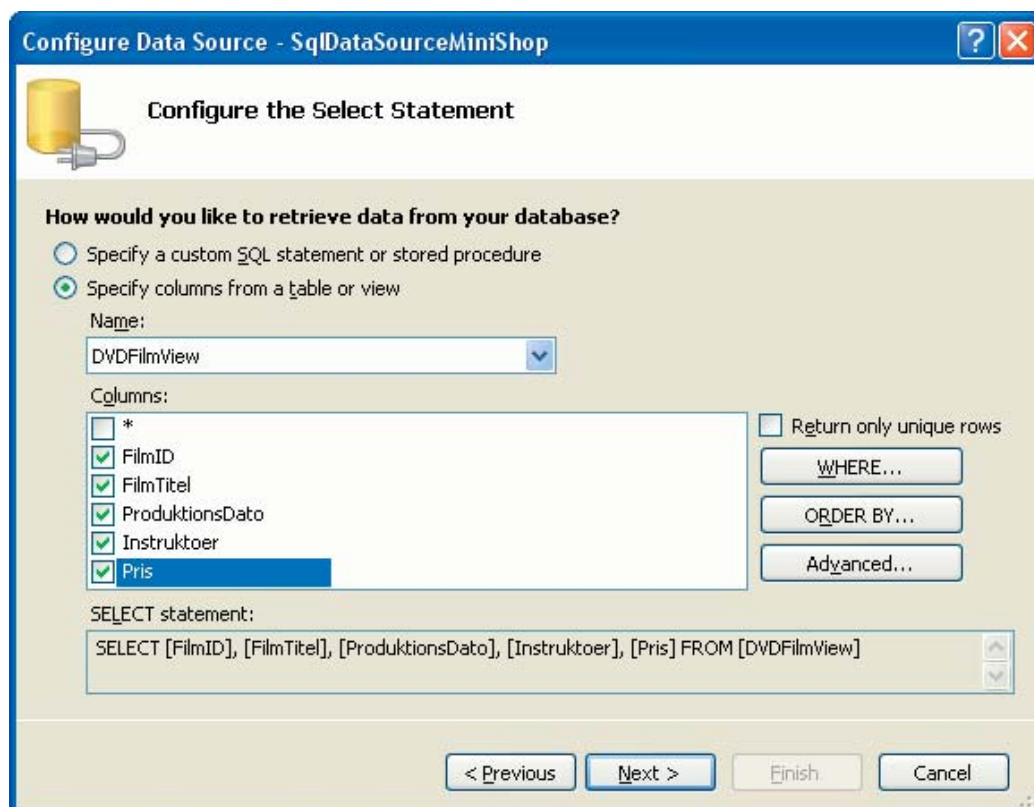


Klik på *New Connection* og i den nye dialog der popper op vælger du server og database (se Figur 35).



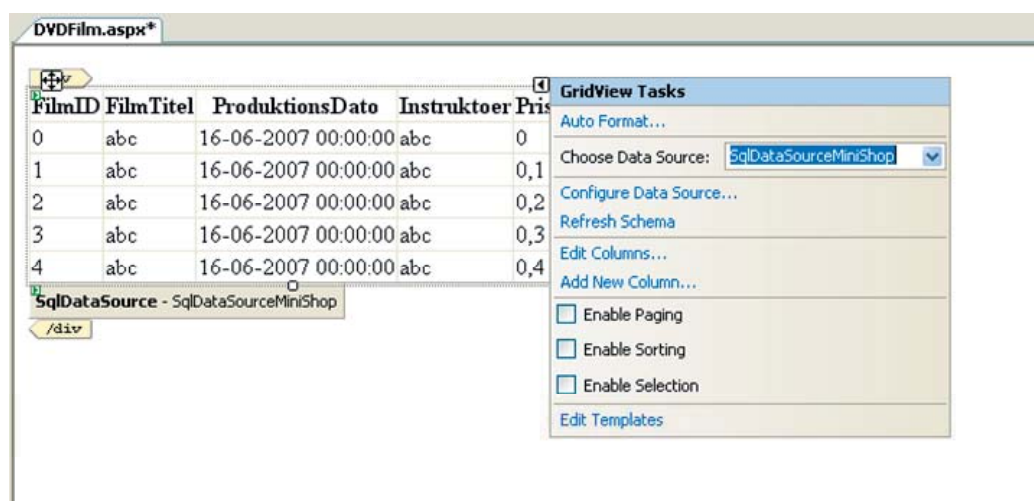
Figur 35. Valg af server, samt database

Følg wizarden og accepter navnet på connection strengen. Nu er vi nået til at vælge hvad vi vil hente fra databasen. Se Figur 36.



Figur 36. Hvad er hentes fra databasen.

Her vælges vores database view samt de kolonner vi er interesseret i. Vi kunne her benytte nogle af mulighederen man får på de 3 knapper, men det springer vi over og afslutter wizarden. På vores Web Form skifter vores Grid View udseende og ser nu ud som på Figur 37.



Figur 37. Grid View kontrol med data source

Der lægges en label på med en overskrift, samt et hyperlink der peger på default.aspx siden. Bemærk vi ikke har skrevet noget kode endnu! En kørsel giver nu følgende (se Figur 38)



Figur 38. Oversigt over film i mini shoppen.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

banedanmark



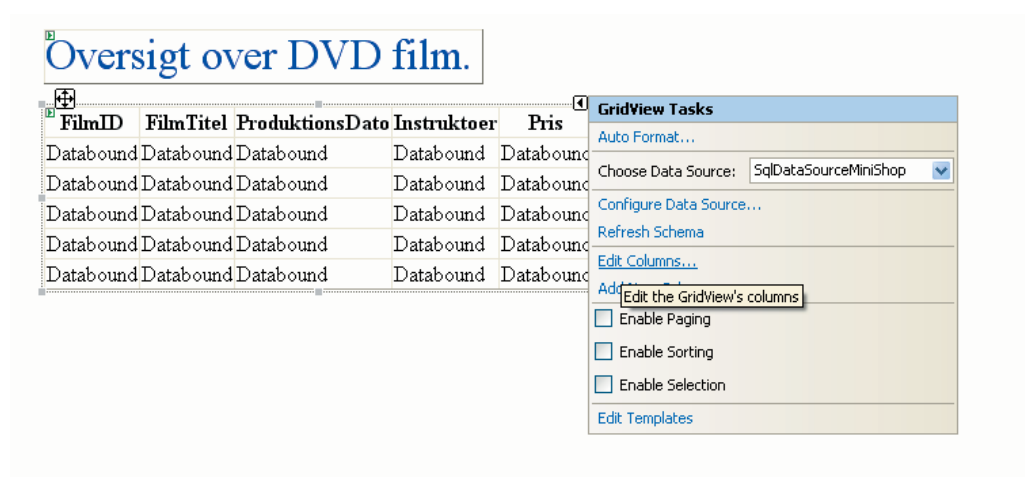
Som det kan ses på Figur 38 har GridView kontrollen automatisk hentet kolonne titlerne fra vores SqlDataSource. Selve fremtoningen af GridView kontrollen er lidt kedelig og som det fremstår nu kan man heller ikke sådan umiddelbart købe film ved at vælge dem i GridView kontrollen. Det vil vi råde bod på i næsten afsnit.

## 5.8 GridView kontrollen konfigureres

Som nævnt i afsnit 0 kan man ikke bestille en vare sådan som filmene er præsenteret. Det vil vi lave om på og desuden konfigurere vores GridView så det fremstår lidt mere imødekommende.

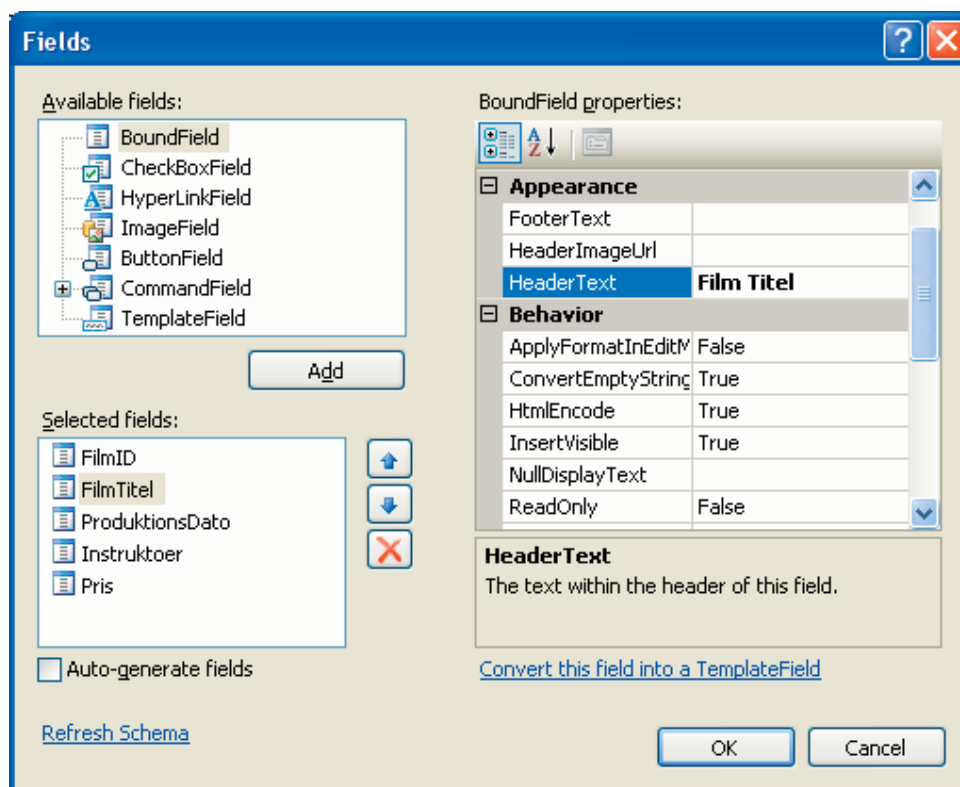
Først vil jeg ændre GridView kontrollen så overskrifterne til kolonnerne fremstår i almindelig tekst og ikke som navne på kolonner i datasettet.

I smart tag vinduet for GridView kontrollen er der et link til en *Field dialog boks* via Edit Columns menu punktet (se Figur 39) .



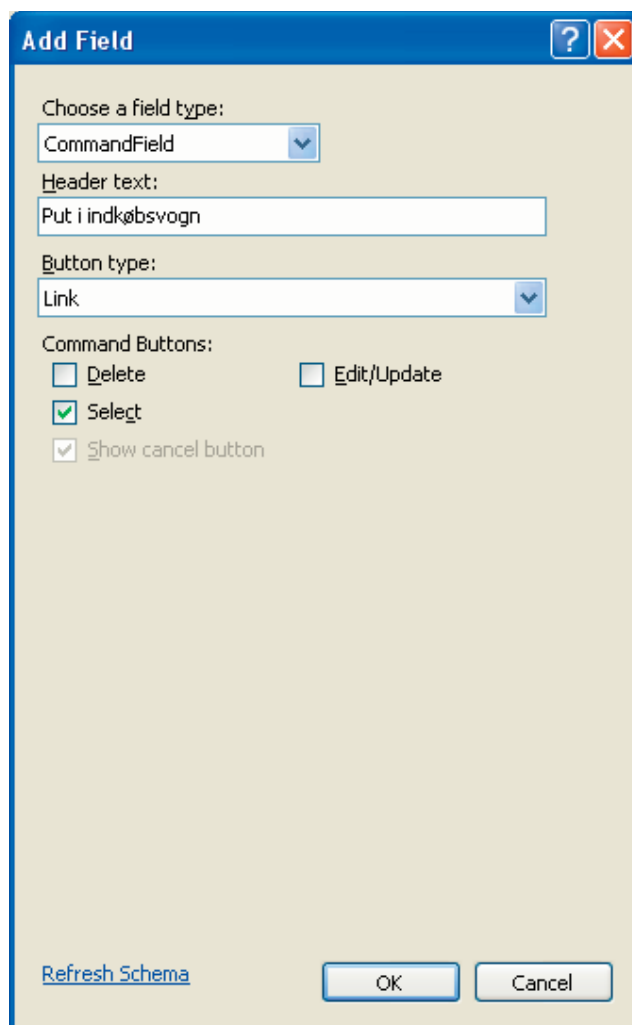
Figur 39. GridView smart tag muligheder.

Når Edit Columns er valgt har man adgang til at ændre overskrifterne for hver kolonne.



Figur 40. Kolonne titler ændres.

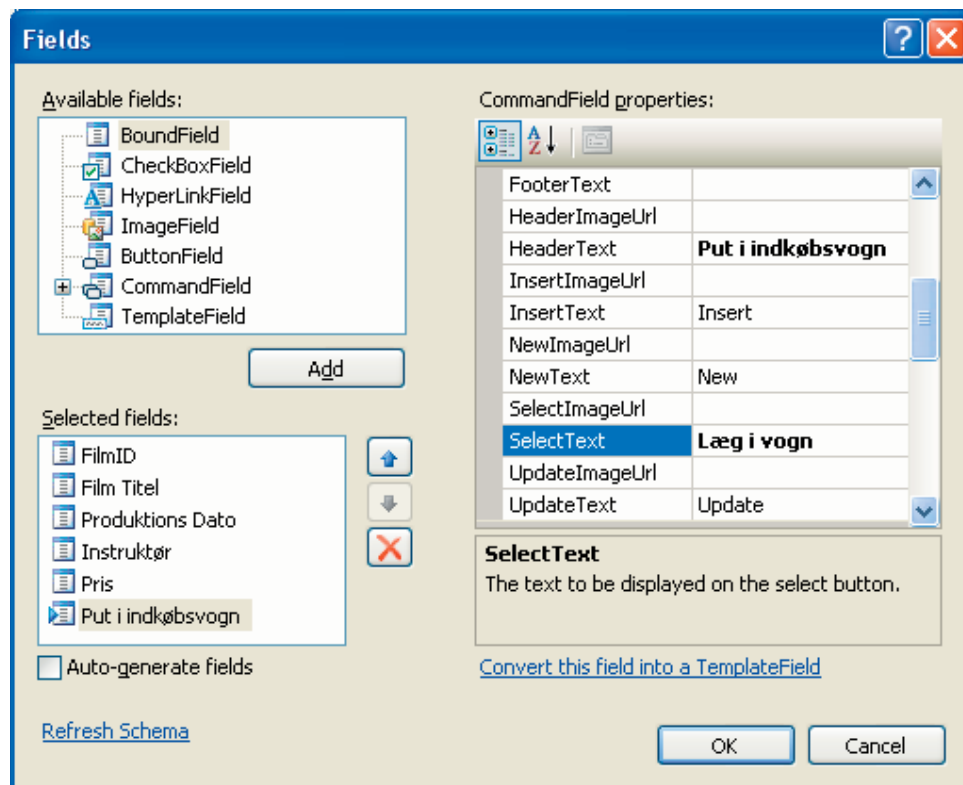
Som det kan ses på Figur 40 har man adgang til formatering af bla. kolonne overskrifter. Jeg har valgt at ændre FilmTitel til Film Titel, hvilket gøres ved at i den venstre kontrol under *Selected Fields* at vælge FilmTitel og i *BoundFields* at ændre *HeaderText* til "Film Titel". Jeg ændre en par af de andre overskrifter på lign. Via Smart tag editoren vælges nu *Add New Column*.



Figur 41. Add new column dialog boksen.

Som det kan ses af Figur 41 har jeg valgt at addere en kolonne af typen *CommandField* kaldet *Put i indkøbsvogn* som en *Link* button. Det giver os mulighed for at håndtere et event når der klikkes på en række i GridViewet.

Når det nu er gjort står der *Select* i alle rækkerne i GridView og jeg vil gerne have teksten *Læg i vogn* istedet. Det kan jeg få ved gå tilbage til *Edit Columns* under GridViews smart tag. Under *SelectText* skrives den ønskede tekst (se Figur 42).



Figur 42. Ændring af SelectText

**CISO Conference**  
Produced by **Inspired**

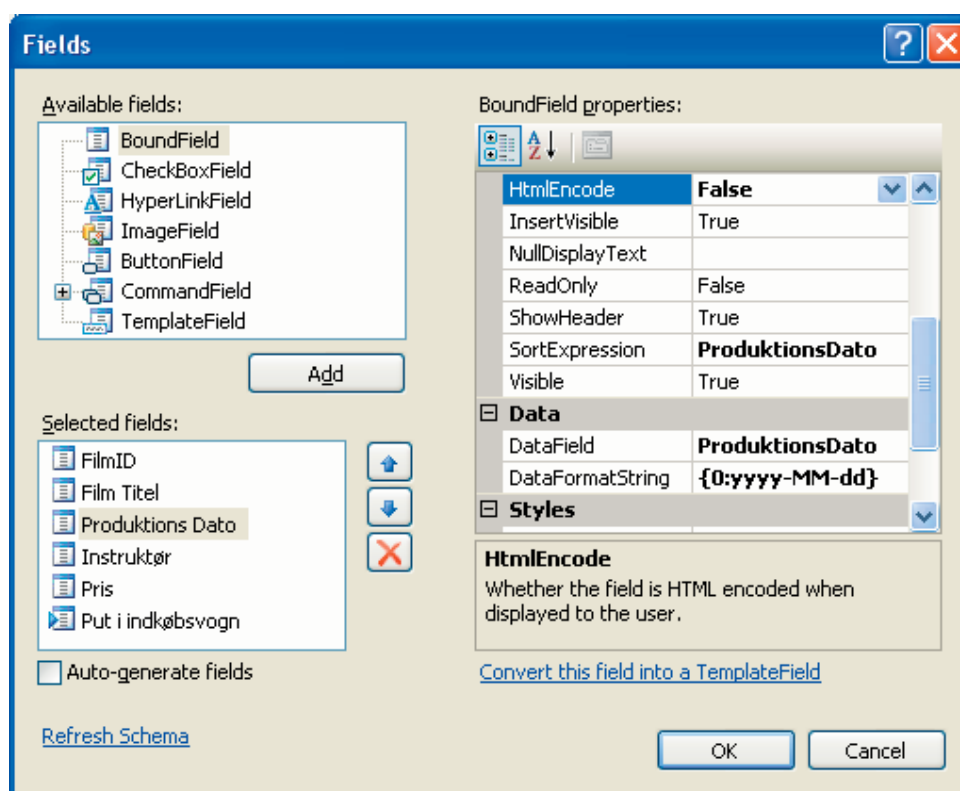
**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

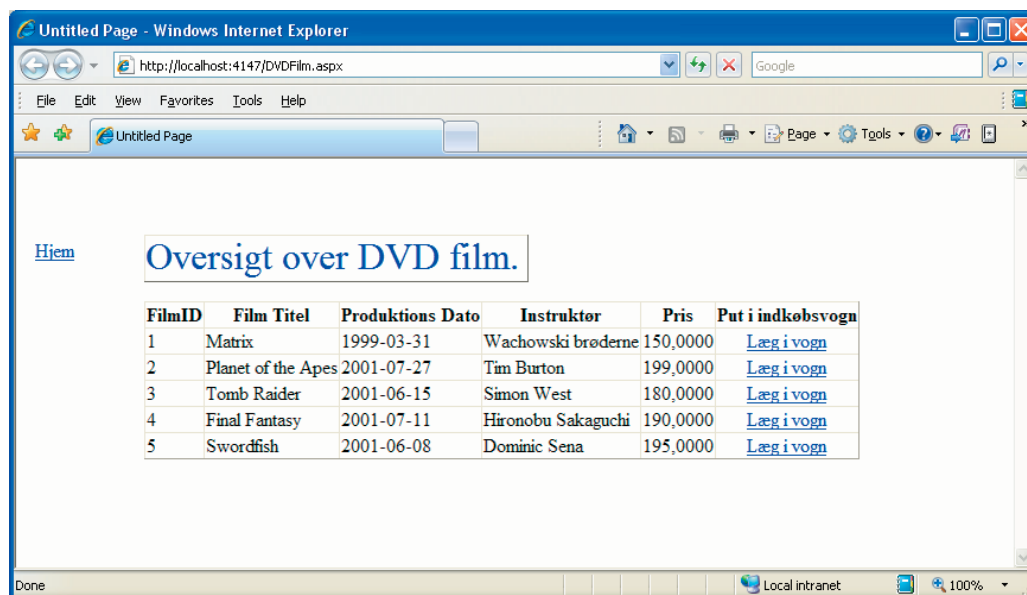


Selve dato formatet er også ændret til dansk standard dvs. år-måned-dag f.eks. 2002-01-17, se Figur 43. Bemærk især at formatet styres af {0:yyyy-MM-dd}. Syntaksen er lidt kryptisk idet man angiver udtrykket i tuborg paranteser. Dette er for at skelne en formaterings streng fra en literal streng (altså noget konstant). Tallet før kolonnet angiver hvilket index i en parameter liste af data der skal formateres. Tallet kan i dette tilfælde kun være 0 idet en celle kun kan indeholde en værdi. Efter kolonnet er der et udtryk der fortæller at jeg vil have præsenteret årstal som år-måned-dag og med bindestreg imellem. **For at *DataFormatString* slår igennem skal *HtmlEncode* sættes til *False*!** For at forhindre Cross site scripting angreb er *HtmlEncode* default sat til *True* og da html encodingen bliver afviklet før formatteringen har den ingen effekt. (Det ligner da en bug i .NET frameworket). Under *Styles/ItemStyle* sættes *HorizontalAlign* til *Center*. Effekten af det er at data bliver centreret i kolonnen. Det kan man selvfølgelig gøre i de resterende kolonner også.



Figur 43. Dato kolonne

Siden ser nu ud som vist på Figur 44.



Figur 44. Den lidt ændrede GridView kontrol.

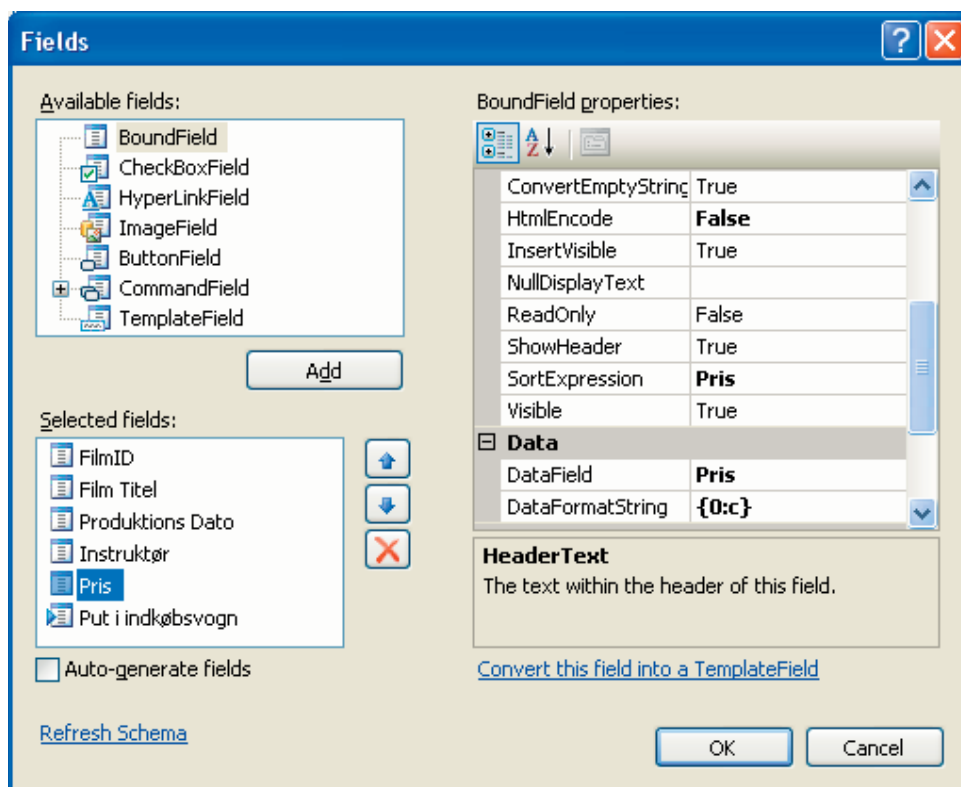
Prisen er konfigureret på ligende vis. På Figur 45 kan det ses at formaterings strengen er sat til {0:c}. c angiver currency (valuta) og benytter den valuta der er angivet i Windows control panel under Regional Settings.

**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

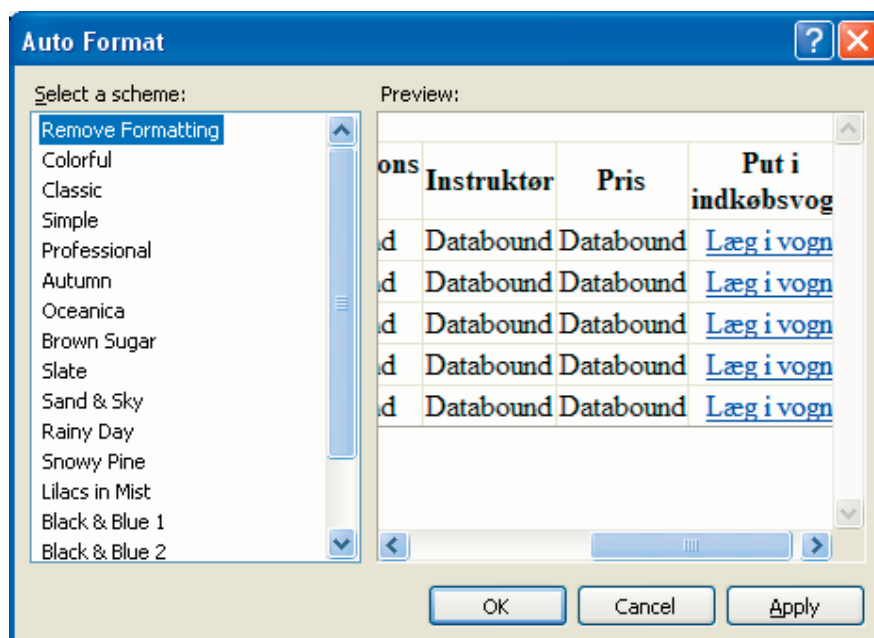
2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**



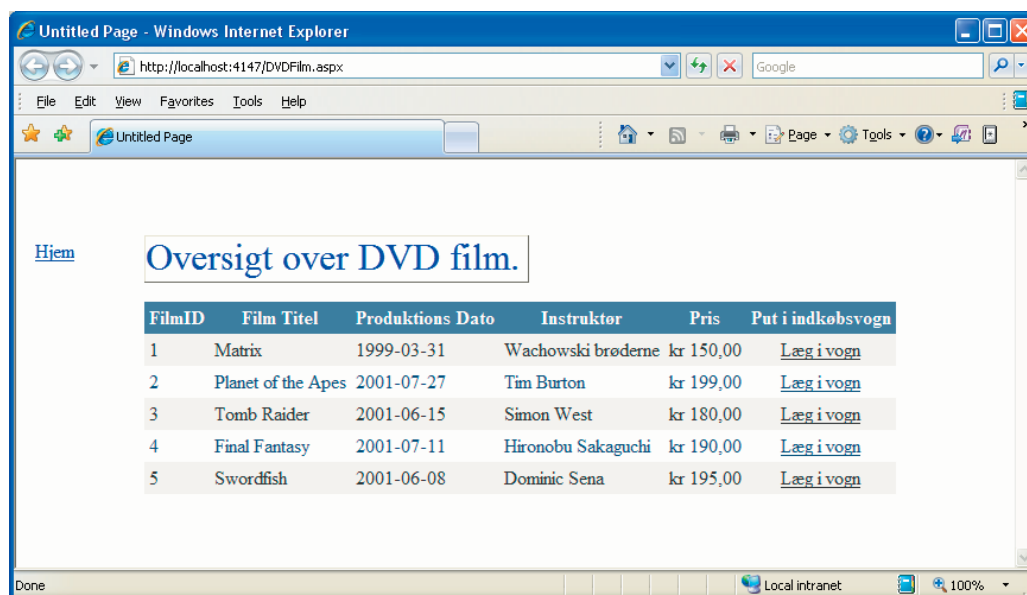
Figur 45. DataFormatString for prisen.

Sidst vil jeg ændre selve udseendet af GridView kontrollen via *Auto Format* linket der er vist på Figur 39. Når man klikker på linket fås dialogen vist på Figur 46.



Figur 46. Autoformat af GridView.

Jeg vælger *Professional* og resultatet er vist på Figur 47. Du kan selvfølgelig opnå det samme resultat ved selv at sætte baggrundsfarver osv på din GridView. Her er blot en hurtig standard løsning.



Figur 47. GridView med Auto Format Professional.

Der er mange andre konfigureringsmuligheder som du kan læse om i MSDN.

## 5.9 Session variabel. Shopping Cart/Indkøbsvogn

Når brugeren klikker på *Læg i vogn* knappen genereres et *RowCommand* event der sender en parameter med af typen *GridViewCommandEventArgs*. Det er i denne parameter vi undersøger om *CommandName* er *Select*. Som i almindelig ASP har man stadig Session variable til at hjælpe med at holde styr på tilstande, idet HTTP protokollon jo er tilstandsløs. Jeg vælger at benytte en Session variabel som jeg vælger at kalde *ShoppingCart* til at gemme id'erne, tittel og prisen for de film brugeren vælger at lægge i sin indkøbsvogn. Jeg benytter mig af en DataTable til at gemme disse oplysninger og denne DataTable gemmes i en Session variabel. Simpelt og rigtig smart!

I afsnit 0 gennemgik jeg filen Global.asax. I denne fil adderes Session variabelen i Session\_Start eventet. Variablen nedlægges igen i Session\_End Eventet. Koden ser således ud:

```
protected void Session_Start(Object sender, EventArgs e)
{
    DataTable cart = new DataTable();
    cart.Columns.Add("FilmID", typeof(long));
    cart.Columns.Add("FilmTitel", typeof(string));
    cart.Columns.Add("Pris", typeof(double));
    Session["ShoppingCart"] = cart;
}

protected void Session_End(Object sender, EventArgs e)
{
    Session["ShoppingCart"] = null;
}
```

Når brugeren klikker på *Læg i vogn* benyttes Session variabelen til at addere information fra den valgte Row til session variabelen. På min web form har jeg adderet en metode der samler eventet RowCommand op. I den tjekker jeg på om kommanden er *Select* (og det er den, fordi der i dette eksempel kun er én kommando) og hvis det er tilfældet overføres data fra DataGridden til Session variabelen. Koden ser således ud:

```
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
{
1.   if (e.CommandName == "Select")
    {
2.       DataTable cart = (DataTable)Session["ShoppingCart"];
3.       DataRow dataRow = cart.NewRow();
        //Det valgte rows index
4.       int index = Convert.ToInt32(e.CommandArgument);
5.       GridView dvdFilmGridView = (GridView)e.CommandSource;
        //Den valgte row
6.       GridViewRow row = dvdFilmGridView.Rows[index];

7.       dataRow[0] = Convert.ToInt64(row.Cells[0].Text); //Film ID
8.       dataRow[1] = row.Cells[1].Text; //Film Titel
9.       Char[] krRemove = new Char[3] { 'k', 'r', ' ' };
10.      dataRow[2] = Convert.ToDouble(row.Cells[4].Text.TrimStart(krRemove));
        //Pris
11.      cart.Rows.Add(dataRow);
12.      Session["ShoppingCart"] = cart;
    }
}
```

1. Som det kan ses af oventsående kode er der på *GridViewCommandEventArgs* en property kaldet *CommandName* der benyttes til at identificere hvad brugeren har klikket på.
2. Session variabelen *ShoppingCart* typecastes til en *DataTable*.
3. For at addere den valgte film til indkøbsvognen oprettes først en ny *DataRow*, der så senere adderes til variabeln *cart*.
4. *CommandArgument* indeholder det valgte index som et object. Derfor kaldes *Convert.ToInt32*.
5. *CommandSource* indeholder vores *GridView*. Det kunne også tilgås direkte istedet via navnet på vores *GridView*.
6. *Row* indeholder nu den valgte *GridViewRow* i vores *GridView*.
7. FilmID fra datagridden findes med *row.Cells[1].Text* der så castes til en *Int64* (long). 0 angiver colonne nummeret der er 0 indekseret. (dvs. første kolonne er kolonne 0)
8. Film titel graves frem fra row i kolonne 1.
9. Prisen på filmen er angivet med kr. som en del af teksten. Jeg ønsker at fjerne kr. inden teksten konverteres til en double. *krRemove* er et hjælpe array som benyttes i næste linie. Arrayet indeholder de chars som skal fjernes fra strengen.
10. Prisen graves frem med *e.ItemCells[4].Text* og dernæst trimmes strengen så alle k'er, r'er og mellemrum (spaces) fjernes med commandoen *TrimStart*.
11. *DataRow*'en med værdierne fra DataGridden adderes til DataTablen *cart*.
12. DataTablen *cart* lægges tilbage i Session variabelen *cart*.

Jeg vil godt lige bede dig tænke over linie 12. Er det nødvendigt at assigne *cart* til session variabelen igen og hvad er det egentlig der sker når jeg sætter de to lig hinanden? Stop op og tænk det igennem.....

Hvis du læser denne linie er det enten fordi du har tænkt det igennem og fundet et svar eller også har du tænkt det igennem men ikke fundet et svar eller du har ignoreret min opfordring til at tænke!

Svaret er at linie 12 er overflødig og hvorfor så det? Jo, i linie 2 oprettes der en reference til indholdet af session variablen. Det betyder at `cart` er en reference der arbejder direkte på indholdet af session variablen og derfor behøver jeg ikke assigne `cart` til session variablen igen. Det betyder så også at `DataTablen` IKKE kopieres fra session variablen til `cart`! Det er meget vigtigt at forstå. Du kan læse mere om referencer og om begreber som boxing i bøger om C#. Bogen "Programming C#" af Jesse Liberty er glimrende til at forstå disse detaljer i sproget, som jeg ikke vil komme nærmere ind på, da du sikkert alligevel har en C# bog liggende og hvis ikke, synes jeg du skulle se at få købt dig en.

Hvis man insisterede på at have linie 12 med i sin kode (hvilket ikke gør nogen skade) burde teksten derfor lyde "Session variablen sættes til at "pege på" det samme som referencen `cart`".

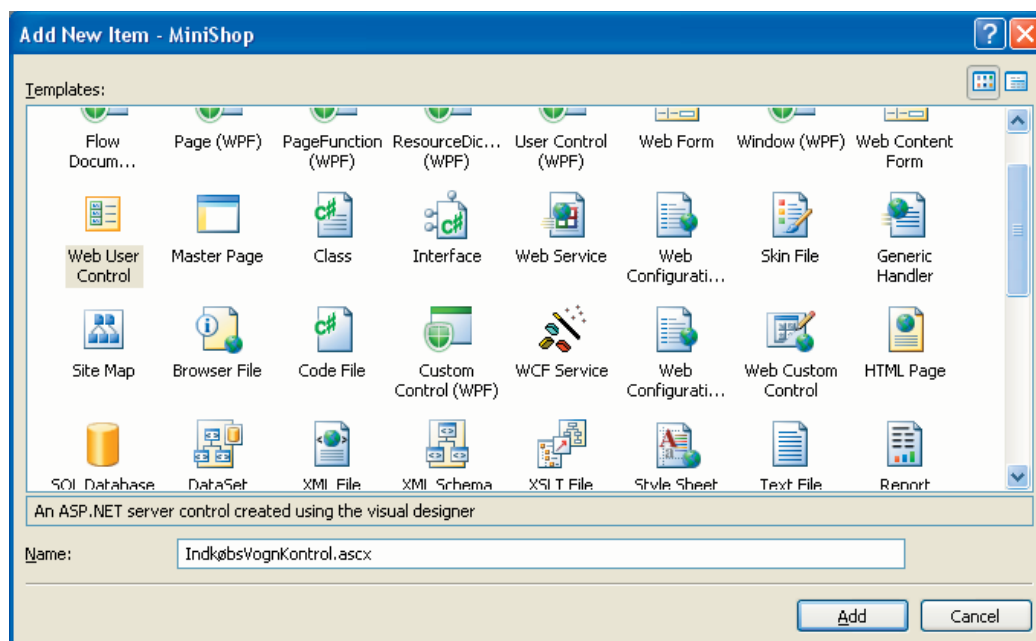
Jeg tog linie 12 med for at kunne stille spørgsmålet som jeg hermed lige har gjort og samtidig besvaret.

Til visning af indkøbsvognen vil jeg benytte en user control. Opbygningen af denne vises i næste afsnit.

## 5.10 En Web user control. Indkøbsvognen

En Web user kontrol er din egen hjemmelavde kontrol som du kan benytte til dine Web Forms præcis som var den en standard Web Form kontrol. Se også afsnit 0. Jeg vil nu lave en Web user kontrol til visning af indholdet i indkøbsvognen. På web kontrollen skal man kunne se de varer der er lagt i indkøbsvognen, samt slette dem man alligevel ikke ønsker. Lad os gå igang.

Først skal vi have adderet Web user kontrollen. Vælg Add New Item som vi har gjort mange gange tidligere. Man får nu dialogen vist på Figur 48.



Figur 48. Adding af Web User control dialog.



**MTHøjgaard**

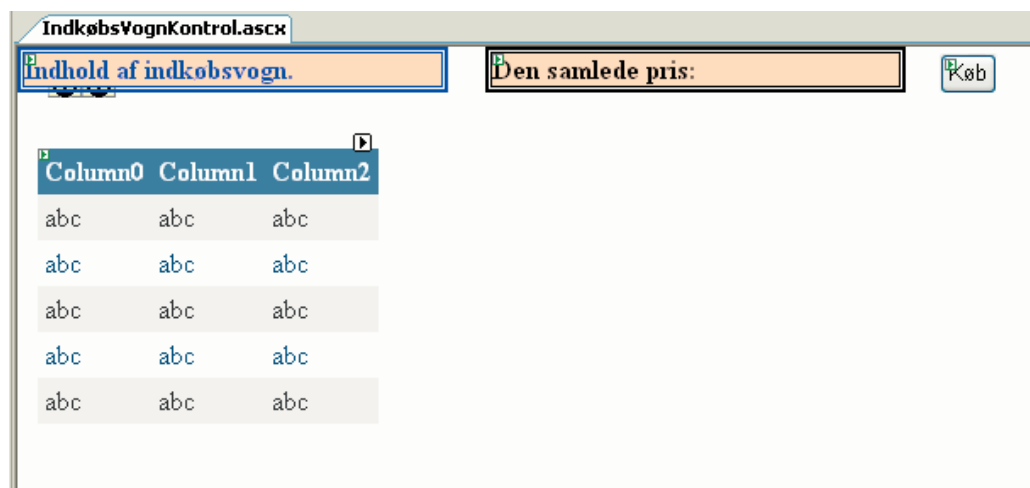
## BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



Jeg omdøber default navnet til `IndkøbsVognKontrol.ascx`. Hvis vi ønsker at kunne placere kontroller i absolut position på user kontrollen kan man ændre layout som beskrevet i afsnit 0. Jeg vil benytte en `GridView` kontrol til at vise indholdet af den i sidste afsnit nævnte session variabel, der fungerer som datalager for indkøbsvognen. Design siden for indkøbsvognen ser ud som vist på Figur 49.



Figur 49. Indkøbsvogn kontrol design side.

Figuren viser to label kontroller samt et `GridView`. I dette eksempel vælger jeg at binde data sammen dynamisk så du kan se en anden måde at fylde data i en `GridView` kontrol. Først må vi dog lige gennemgå hvad der sker når kontrollen bliver benyttet på en Web Form. Som du kan se nedenfor er der en række events der kan abonneres på under levetiden af user kontrollen.

- `OnInit`
- `Init`
- `InitComplete`
- `PreLoad`
- `Load`
- `LoadComplete`
- `PreRender`
- `PreRenderComplete`
- `Unload`

De mest interessante for os er `Load` og `PreRender`. `Load` har vi set før og vi ved allerede at det affyres når en Web Form loades (det er sikkert derfor eventet hedder *load*). Det samme gælder for en user kontrol. `PreRender` affyres efter `load`, men umiddelbart før kontrollen tegnes. Det er vigtigt at forstå disse forskelle og udnytte dem under kodningen, fordi kontrollen jo lever på en Web Form der har lignende events. Når du skal abonnere på et af ovenstående events kan det gøres ved i kode filen at skrive `Page_` foran eventnavnet. F.eks. skal man for at samle `PreRender` eventet op i sin kode skrive:

```
protected void Page_PreRender(object sender, EventArgs e)
```

Ovenstående events er automatisk hooket op til siden. Det skyldes at `AutoEventWireup="true"` som kan findes i aspx filen for user kontrollen. Hvis `AutoEventWireup` sættes til false skal man selv sættes sin eventhandler op. Det er jo ikke ligefrem svært og man bestemmer selv hvad eventhandleren skal hedde, så jeg kan ikke lige se den store fidus i `AutoEventWireup`.

Jeg har på Figur 50 adderet IndkøbsvognKontrollen til web formen med DVD film. Det gøres ved at trække den fra Solution Explorer over på formen. Jeg har lagt den i en Panel kontrol på formen, fordi Panel kontrollen kan flyttes rundt i designeren.



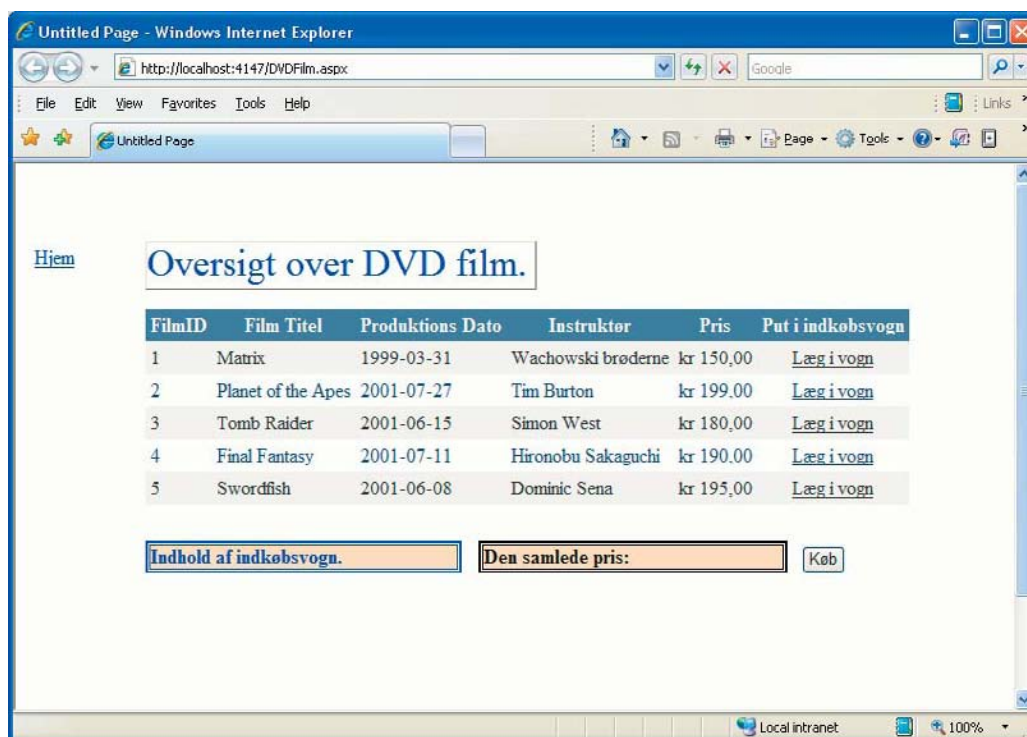
Figur 50. DVD film oversigt med indkøbsvogn kontrol (DVDFilm.aspx).

Lad os se på affyrrings rækkefølgen af de for os relevante events, der sendes under kontrollernes levetid. Koden der ligger bag bliver forklaret bagefter, idet det er lettere at forstå når vi forstår rækkefølgen af events.

Når brugeren loader siden DVDFilm.aspx med indkøbsvogn kontrollen affyres events i denne rækkefølge (tiden er ned af siden):

DVDFilm.aspx.cs	IndkøbsVognKontrol.ascx.cs
Load	
	Load
PreRender	
	PreRender

Dvs. formen hvorpå web kontrollen befinder sig er hele tiden et skridt foran. I browseren ser det ud som vist på Figur 51. Læg mærke til at vores Indkøbsvogn user kontrol er placeret umiddelbart under film GridViewet.

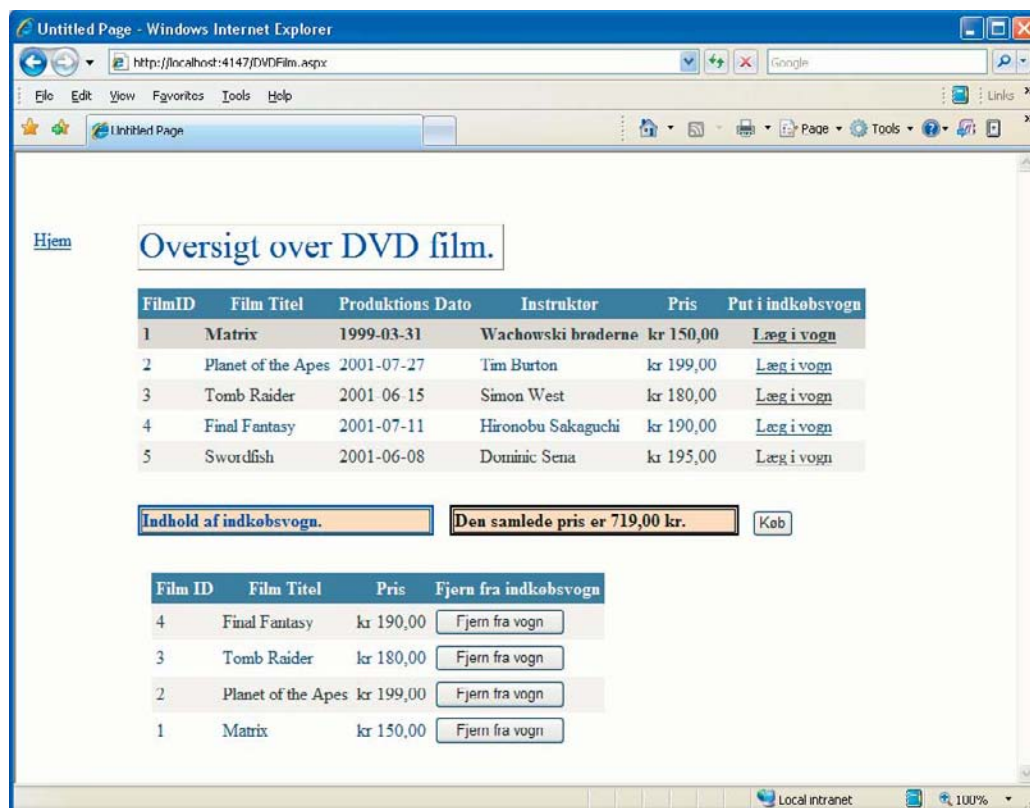


Figur 51. Oversigt med indkøbsvogn.

Når brugeren klikker på *Læg i vogn* sendes events i denne rækkefølge:

DVDFilm.aspx.cs	IndkøbsVognKontrol.ascx.cs
Load	
	Load
GridView1_RowCommand (Læg i vogn)	
PreRender	
	PreRender

Det giver følgende i browseren. Ja, jeg ved godt vi ikke har kigget kode endnu, men det gør vi bagefter. Når vi forstår rækkefølgen af events er det lettere at forstå koden.



Figur 52. Film oversigt med film i indkøbsvognen.

Sidst men ikke mindst har brugeren mulighed for at fjerne film fra sin indkøbsvogn. Når han gør det sendes følgende events:

DVDFilm.aspx.cs	IndkøbsVognKontrol.ascx.cs
Load	
	Load
	IndkøbsvognGridView_RowCommand (Fjern fra vogn)
PreRender	
	PreRender

Som det ses af de to sidste tabeller sendes load før siden tegnes og før RowCommand. Tilgængelig sendes PreRender efter RowCommand. Den viden skal vi udnytte til at lægge data i og fjerne data fra vores indkøbsvogn på det rigtige tidspunkt.

Under load oprettes kolonnerne for GridView kontrollen i indkøbsvognen. Koden ser således ud:

```
1 protected void Page_Load(object sender, EventArgs e)
2 {
3     IndkøbsvognGridView.AutoGenerateColumns = false;
4     IndkøbsvognGridView.Columns.Clear();
5
6     //Adder kollonerne dynamisk til GridViewet
7     BoundField dataCol = new BoundField();
8     dataCol.HeaderText = "Film ID";
9     dataCol.DataField = "FilmID";
10    IndkøbsvognGridView.Columns.Add(dataCol);
11
12    dataCol = new BoundField();
13    dataCol.HeaderText = "Film Titel";
14    dataCol.DataField = "FilmTitel";
15    IndkøbsvognGridView.Columns.Add(dataCol);
16
17    dataCol = new BoundField();
18    dataCol.HeaderText = "Pris";
19    dataCol.DataField = "Pris";
20    dataCol.HtmlEncode = false;
21    dataCol.DataFormatString = "{0:c}";
22    IndkøbsvognGridView.Columns.Add(dataCol);
23
24    ButtonField col = new ButtonField();
25    col.HeaderText = "Fjern fra indkøbsvogn";
26    col.CommandName = "Fjern";
27    col.Text = "Fjern fra vogn";
28    col.ButtonType = ButtonType.Button;
29    IndkøbsvognGridView.Columns.Add(col);
30
31    //Hook vores Fjern command op
32    IndkøbsvognGridView.RowCommand += new
GridViewCommandEventHandler(IndkøbsvognGridView_RowCommand);
33 }
```

Bemærk at eventhandleren for *Fjern fra vogn* knappen manuelt sættes op i Load metoden. Det er for sent at lægge det i PreRender da det event kaldes umiddelbart før siden tegnes og efter initialisering er overstået. Her er en forklaring på hvad der sker i koden.

3. Automatisk generering af kolonner slås fra, da vi selv sætter det op.

4. Kolonne listen i GridViewet slettes.

6-22. Der lægges kolonner på af typen BoundField. Bemærk at DataField propertyen modsvarer navnet på kolonnerne i tabellen i vores session variabel *ShoppingCart*. HeaderText propertyen sættes til et læsbart navn. Bemærk at Pris kolonnen er forsynet med en formaterings streng og at HtmlEncode derfor sættes til false.

24-32. En kolonne af typen ButtonField lægges på. Den findes ikke i *ShoppingCart* og er en knap der giver mulighed for at fjerne vare der er lagt i indkøbsvognen. CommandName sættes til *Fjern* som er et custom navn. Hvis jeg havde sat den til delete kunne RowDeleting eventet benyttes, men her hooker jeg op på RowCommand istedet som det ses i linie 32.

Nu er data strukturen oprettet og tilbage er at lave den egentlige data binding mellem GridView kontrollen og data i *ShoppingCart*. Lad os se koden for PreRender af vores indkøbsvogn kontrol:

```
protected void Page_PreRender(object sender, System.EventArgs e)
{
    DataTable Cart = ((DataTable)Session["ShoppingCart"]);
    IndkøbsvognGridView.DataSource = Cart;
    IndkøbsvognGridView.DataBind();

    double SamledePris = 0;
    foreach (DataRow row in Cart.Rows)
    {
        SamledePris += (double)row[2];
    }
    SamledePrisLabel.Text = string.Format("Den samlede pris er {0:0.00} kr.",
    SamledePris);

    Køb.Visible = (Cart.Rows.Count != 0) && (VisKøbeKnappen);
}
```



Ses vi til DSE-Aalborg?

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

banedanmark



Som det ses kaldes `DataBind` hvilket aktiverer at data præsenteres i `GridView`et. Efter `DataBind` beregnes værdien af varene i indkøbsvognen ved at benytte `Pris` kolonnen og addere samtlige priser i `DataGrid`et. Værdien skrives ud i lablen `SamledePrisLabel`.

Hvis brugeren vil slette en film fra indkøbsvognen gøres det ved at klikke på *Fjern fra vogn* knappen. Det udløser eventet `RowCommand`. Kommandehandleren ser således ud:

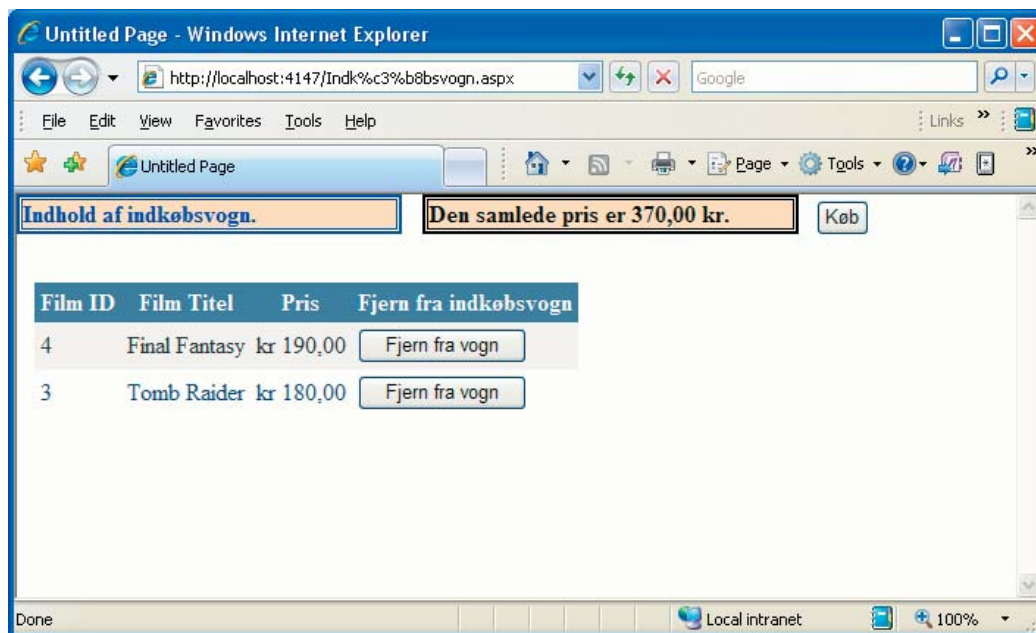
```
void IndkøbsvognGridView_RowCommand(object sender, GridViewCommandEventArgs e)
{
    if (e.CommandName == "Fjern")
    {
        DataTable Cart = ((DataTable)Session["ShoppingCart"]);
        Cart.Rows.RemoveAt(Convert.ToInt32(e.CommandArgument));
    }
}
```

Parametren `e` som er af typen `GridViewCommandEventArgs` indholder indexet på den item i `GridView` kontrollen der er trykket på (via `e.CommandArgument`) og det benytter jeg så til at fjerne hele den række i *ShoppingCart* som brugeren gerne vil af med. Vi mangler nu blot at give brugeren mulighed for at kunne bestille det der ligger i indkøbsvognen, hvilket vi ser på i næste afsnit.

## 5.11 Ordenen modtages

Jeg tror du nu er ved at være selvkørende men vil alligevel færdiggøre eksemplet og udbygge det med at vise hvordan du direkte fra koden og uden om wizarden kan skabe kontakt til databasen. Derudover vises hvordan du holder dit dataset opdateret når du adderer nye rows (records) til databasen. Her tænkes specielt på at man i masser af tilfælde lader databasen tildele en row et autonummer. Dette autonummer skal efter at data er skrevet til databasen læses igen. Kontrollen fra sidste afsnit udbygges med en købe knap, der linker til en ordre side. Jeg vil også vise hvordan du kan komme i kontakt med user kontrollen inde fra koden på den side hvorpå kontrollen benyttes. På ordre siden skal man kunne indtaste sit navn og adresse ligesom vi så i afsnit 0 og derudover skal man godkende ordren.





Figur 53. Indkøbsvognen med købe knap.

På Figur 53 kan du se indkøbsvogn kontrollen forsynet med en købe knap. Knappen er kun tilstede dvs. synlig hvis der er film i indkøbsvognen. Om knappen er synlig eller ej styres i PreRender i koden for indkøbsvogn kontrollen (se kode listning i forrige afsnit).

**CISO Conference**  
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

Når brugeren trykker på Køb sendes han videre til en ordre side hvor han skal angive navn og adresse inden han til slut skal acceptere indholdet i kurven. Koden er simpel og ser således ud:

```
private void Køb_Click(object sender, System.EventArgs e)
{
    Response.Redirect("OrdreSide.aspx");
}
```

Som det kan ses er der kun én linie kode og alt hvad den gør er at dirigere brugeren videre til ordre siden.

BEMÆRK: at ovenstående er skidt kode fordi der er en direkte binding fra en user kontrol til en web form uden for denne. Det skal naturligvis laves generisk f.eks. således at user kontrollen blot forwarder eventet til den container den ligger på og det er så den der på sørge for logikken, men det her er blot demo kode..

Lad os nu kigge på selve OrdreSide.aspx.cs filen. Det er her koden til modtagelse af ordren gemmer sig. Jeg viser her hvordan man dynamisk og uden de mange wizard selv kan styre kontakten til databasen. Selve ordre siden ser således ud i en browser:

Untitled Page - Windows Internet Explorer

http://localhost:4147/OrdreSide.aspx

File Edit View Favorites Tools Help

Untitled Page

[Hjem](#)

Fornavn

Efternavn

Adresse

Post nummer

By

Godkend Fortryd

**Indhold af indkøbsvogn.** **Den samlede pris er 370,00 kr.**

Film ID	Film Titel	Pris	Fjern fra indkøbsvogn
4	Final Fantasy	kr 190,00	<input type="button" value="Fjern fra vogn"/>
3	Tomb Raider	kr 180,00	<input type="button" value="Fjern fra vogn"/>

Local intranet 100%

Figur 54. Ordre siden.

På ordre siden er indkøbsvogn kontrollen genbrugt, men nu uden købe knappen. Når man indsætter indkøbsvogn kontrollen genereres et tagname som registreres for siden. Koden kan findes på Source tabben i design view for Web Formen.

```
<%@ Register Src="IndkøbsVognKontrol.ascx" TagName="IndkøbsVognKontrol"
TagPrefix="uc1" %>
```

Man kan se at @Register erklærer indkøbsvogn kontrollen som *IndkøbsVognKontrol*. For at få fat i selve kontrollen erklæres inde i *code behind* filen (cs) eller blot kode filen en variabel således:

```
protected global::MiniShop.IndkøbsVognKontrol IndkøbsVognKontrol1;
```

Jeg kan nu via *IndkøbsVognKontrol1* variablen komme i kontakt med indkøbsvogn kontrollen inde fra koden.

Da jeg har lavet *VisIndkøbsVogn* til en property kan den sættes til false i designeren. I indkøbsvogn kontrollen tjekkes om købeknappen skal vises eller ej.

Mere interessant er det at se hvad der sker når man trykker på Godkend. Det som skal ske er jo at der skal oprettes en ny ordre og de film som findes i indkøbsvogn kontrollen og dermed i session variabelen *ShoppingCart* skal registreres i databasen sammen med kundens navn. Rækkefølgen vil være således:

- 1) tjek om kunden findes i forvejen.
  - a. Hvis ja
    - i. Find hans KundeID.
  - b. Hvis nej,
    - i. Opret kunden og læs hans af SQL Server tildelte KundeID
- 2) Opret ordren i Ordre tabellen og læs det af SQL Server tildelte OrdreID
- 3) Opret Rows i OrdreDetaljer tabellen med sammenhørende værdier af OrdreID og FilmID. FilmID hentes i session variabelen *ShoppingCart*.

Det lyder jo ret nemt så lad os kigge koden line for line. Først lige en bemærkning og det er at nedenstående er demo kode og ikke velegnet til produktion. For det første er connection strengen hårdkodet og for det andet er metoden for lang. Detaljerne skal flyttes ud i andre metoder eller endnu bedre i et service lag for GUI'en bør ikke kende til databasen. Ligeledes bør nedenstående foregå i en transaction, f.eks. via et *TransactionScope* for at sikre konsistens i data. Selve koden listes herunder og gennemgås line for line:

```
1 protected void Godkend_Click(object sender, EventArgs e)
2 {
3     //Vi vil kun acceptere at gå videre hvis navn og adresse er udfyldt.
4     if (IsValid)
5     {
6         DataSet ordreDataSet = new DataSet();
7         using (SqlConnection connection = new SqlConnection())
8         {
9             connection.ConnectionString = "data source=.;initial catalog=Mini
Shop;integrated security=True";
```

```

10
11     using (SqlCommand cmd = connection.CreateCommand())
12     {
13         string sql = string.Format("SELECT * FROM Kunder WHERE Fornavn='{0}' AND
Efternavn='{1}' AND Vejnavn='{2}' AND Postnummer='{3}' AND Bynavn='{4}'",
14             Fornavn.Text, Efternavn.Text, Adresse.Text, PostNummer.Text,
By.Text);
15
16         SqlDataAdapter db = new SqlDataAdapter(sql, connection);
17         SqlCommandBuilder cmdBuilder = new SqlCommandBuilder(db);
18
19         connection.Open();
20
21
22 ////////////////////////////////////////
db.Fill(ordreDataSet, "Kunder");
23     DataTable kunder = ordreDataSet.Tables["Kunder"];
24     DataRow kunde;
25
26     if (kunder.Rows.Count == 0)
27     { //Vi antager i dette eksempel at søgekriterierne giver et unikt svar
28         //Opret ny kunde
29         kunde = kunder.NewRow();
30         kunde["Fornavn"] = Fornavn.Text;
31         kunde["Efternavn"] = Efternavn.Text;
32         kunde["Vejnavn"] = Adresse.Text;
33         kunde["Postnummer"] = PostNummer.Text;
34         kunde["Bynavn"] = By.Text;
35         kunder.Rows.Add(kunde);
36         db.Update(ordreDataSet, "Kunder");
37         //Læs nu det af SQL server tildelte autonummer for KundeID
38         cmd.CommandText = "SELECT @@IDENTITY";
39         kunde["KundeID"] = (Decimal)cmd.ExecuteScalar();
40
41         ordreDataSet.AcceptChanges();
42     }
43     else
44     {
45         kunde = kunder.Rows[0];
46     }
47     ////////////////////////////////////////
48
49     //Opret ordren i Ordre tabellen
50     db.SelectCommand.CommandText = "SELECT * FROM Ordre WHERE l=0"; //dvs
vælg ingen rows
51     //Da vi har ændre SelectCommand på DataAdapteren skal vi kalde
RefreshSchema
52     //SqlCommandBuilder så den er ajour.
53     cmdBuilder.RefreshSchema();

```

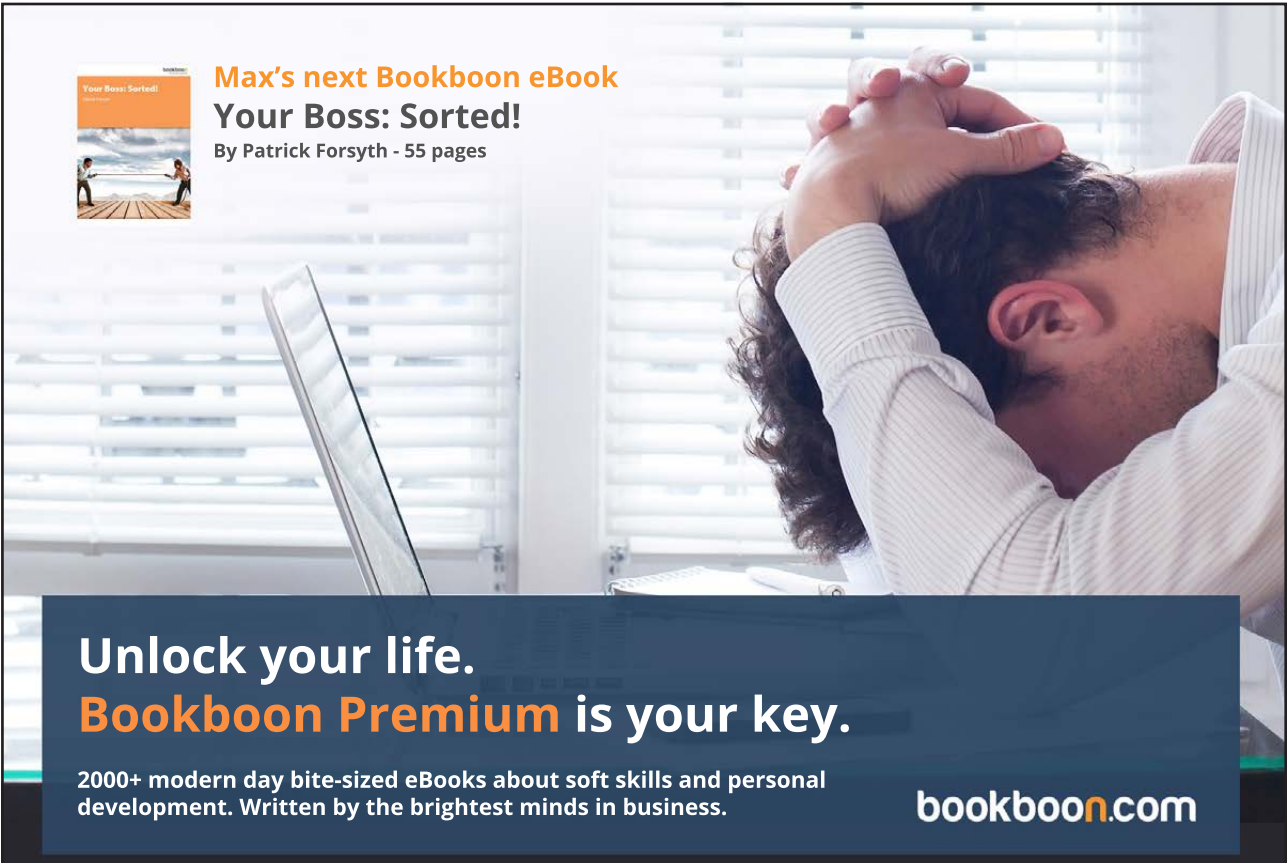
```


54     db.Fill(ordreDataSet, "Ordre");
55     DataTable ordre = ordreDataSet.Tables["Ordre"];
56     DataRow ordreRow = ordre.NewRow();
57     ordreRow["KundeID"] = kunde["KundeID"];
58     ordreRow["OrdreDato"] = DateTime.Now;
59     ordre.Rows.Add(ordreRow);
60     db.Update(ordreDataSet, "Ordre");
61     //Læs nu det af SQL server tildelte autonummer for OrdreID
62     cmd.CommandText = "SELECT @@IDENTITY";
63     ordreRow["OrdreID"] = (Decimal)cmd.ExecuteScalar();
64
65     ordreDataSet.AcceptChanges();
66
67     //////////////////////////////////////
68     //Indsæt film id'er i OrdreDetaljer tabellen
69     db.SelectCommand.CommandText = "SELECT * FROM OrdreDetaljer WHERE 1=0";
//dvs vælg ingen rows
70     //Da vi har ændre SelectCommand på DataAdapteren skal vi kalde
RefreshSchema
71     //på SqlCommandBuilder så den er ajour.
72     cmdBuilder.RefreshSchema();
73     db.Fill(ordreDataSet, "OrdreDetaljer");
74     DataTable ordreDetaljer = ordreDataSet.Tables["OrdreDetaljer"];
75     DataTable Cart = ((DataTable)Session["ShoppingCart"]); //vores
indkøbsvogn session variabel
76     foreach (DataRow row in Cart.Rows)
77     {
78         DataRow ordreDetaljeRow = ordreDetaljer.NewRow();
79         ordreDetaljeRow["OrdreID"] = ordreRow["OrdreID"];
80         ordreDetaljeRow["FilmID"] = row["FilmID"];
81         ordreDetaljer.Rows.Add(ordreDetaljeRow);
82     }
83     db.Update(ordreDataSet, "OrdreDetaljer");
84     ordreDataSet.AcceptChanges();
85 }
86 }
87 Response.Redirect("OrdreBekræftelse.aspx");
88 }
89 }

```

Koden gennemgås nu linie for linie, dog springer jeg tomme linier og selvforklarende linier over:

4. IsValid er en property på Page klassen der er sand hvis alle validator kontroller ikke har fundet fejl. Validator kontrollerne har vi set på i afsnit 0.
- 7: Der oprettes et selvstændigt Connection objekt til at etablere kontakten til databasen. Det er fordi jeg skal bruge objektet flere gange ellers kunne jeg blot have angivet en connection string til SqlDataAdapteren erklæret i linie 19.
- 9: Erklæring af den connection string der giver adgang til min SQL database.
- 13-14: En SQL statement, der skal undersøge om kunden findes i databasen opbygges.
- 16: Variablen db indeholder SqlDataAdapteren. Husk på at DataAdapteren er bindeled mellem vores database og vores dataset.
- 17: En SqlCommandBuilder erklæres. I constructoren er vores SqlDataAdapter db parameter. SqlCommandBuilder klassen er nyttig idet den ud fra den givne SelectCommand på SqlDataAdapteren selv opbygger InsertCommand, DeleteCommand og UpdateCommand statements der benyttes når man kalder Update på SqlDataAdapteren. Hvis man ikke benytter SqlCommandBuilderen kan man selv opbygge de tre nævnte commandoer.



 **Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**



- 29-35: En ny kunde oprettes i Kunder tabellen i ordreDataSet og data fra Web Form kontrollerne indsættes.
- 36: SqlDataAdapteren db bedes om at opdatere databasen med de nye informationer fra ordreDataSet. Det er her at SqlCommandBuilderens `magi` kommer i spil fordi `InsertCommand` skal benyttes. Hvis vi ikke havde en `SqlCommandBuilder` og vi ikke selv havde defineret en `InsertCommand` på db ville kaldet fejle.
- 38: `CommandText` opdateres. Den skal hente det af SQL server tildelte `KundeID` således at vores `ordreDataSet` er opdateret. `@@IDENTITY` indeholder det sidst generede autonummer dvs. vores `KundeID`.
- 39: Den Row i vores `DataSet` der indeholder kunden opdateres med det tildelte `KundeID`. `ExecuteScalar` metoden på `SqlCommand` er meget hurtig og returnerer den første kolonne i den første række i resultatsættet.
- 41: `AcceptChanges` får `DataSet` til at godkende alle ændringer der er lavet. I dette tilfælde er det ikke nødvendigt at lave dette kald, men i tilfælde med f.eks. et `GridView` er det en god ide at vænne sig til at kalde `AcceptChanges` når man har lavet ændringer i et `DataSet` som er "permanente".
- 45: *kunde* sættes til den første row der returneres fra tabellen. Det antages at der kun returneres en row, idet navn og adresse antages at være unikke for alle kunder.
- 50: Der defineres en ny `SELECT` statement for `DataAdapteren`. Vi henter ingen records, men gør klar til at tilgå `Ordre` tabellen.
- 53: `RefreshSchema` kaldes på `SqlCommandBuilderen`, idet vi har ændret `SelectCommand` på `DataAdapteren`. Herved sørger `SqlCommandBuilderen` selv for at genererer de nødvendige `InsertCommand`, `DeleteCommand` og `UpdateCommand` for den nye tabel `DataAdapteren` nu tilgår.
- 54: `Fill` commandoen kaldes nu på `DataAdapteren` og derved opdateres `DataSet` med `Ordre` tabellen, selvom der ikke returneres nogen records. En anden mulighed var at kalde `FillSchema` istedet. `FillSchema` returnerer ingen rows, men overføre tabel strukturen til et `DataSet`.
- 55-60: `Ordren` oprettes og databasen opdateres. Koden er principelt den samme som da vi adderede en ny kunde til databasen.
- 62-63: Det af SQL server tildelte `OrdreID` læses via en `SELECT @@IDENTITY` statement ligesom vi gjorde for `KundeID`.
- 65: `AcceptChanges` kaldes igen selvom det egentlig ikke er nødvendigt, men som tidligere nævnt er det en god ide at gøre.
- 69-84: `OrdreDetalje` tabellen forsynes med nye rækker der indholder samhörende værdier af `OrdreID` og `FilmID`. `OrdreID` blev hentet i linie 63 og `FilmID` hentes fra session variabelen *ShoppingCart*.
- 87: Data er nu i databasen og kunden ledes videre til en ordrebekræftelses side.

På ordrebekræftelses siden vil man slette indholdet af indkøbsvognen og vise kunden hvilke film han har bestilt samt hvor leveringen sendes hen. Jeg overlader trygt implementeringen af denne side til læseren.



Det virker måske som en stor mundfuld, men egentlig er de skridt der tages små og ret logiske når man ser det lidt fra oven. *Jeg vil igen understrege at ovenstående er demo kode og at man i produktions sammenhæng vil skrive koden noget anderledes. Her handler det blot om at vise mulighederne i de forskellige system klasser. Hvis der er for mange lag vil det være mere kompliceret at beskrive.*

Dette afslutter eksemplet på en lille online web butik. Jeg ved godt der er meget der ikke er sagt og meget der ikke er gjort, men der skulle jo skæres et sted. Jeg håber at have ramt nogle nyttige sider af .NET teknologien som du kan få stor glæde af siden hen.



 **MTHøjgaard**

**BEDRE  
LØSNINGER**

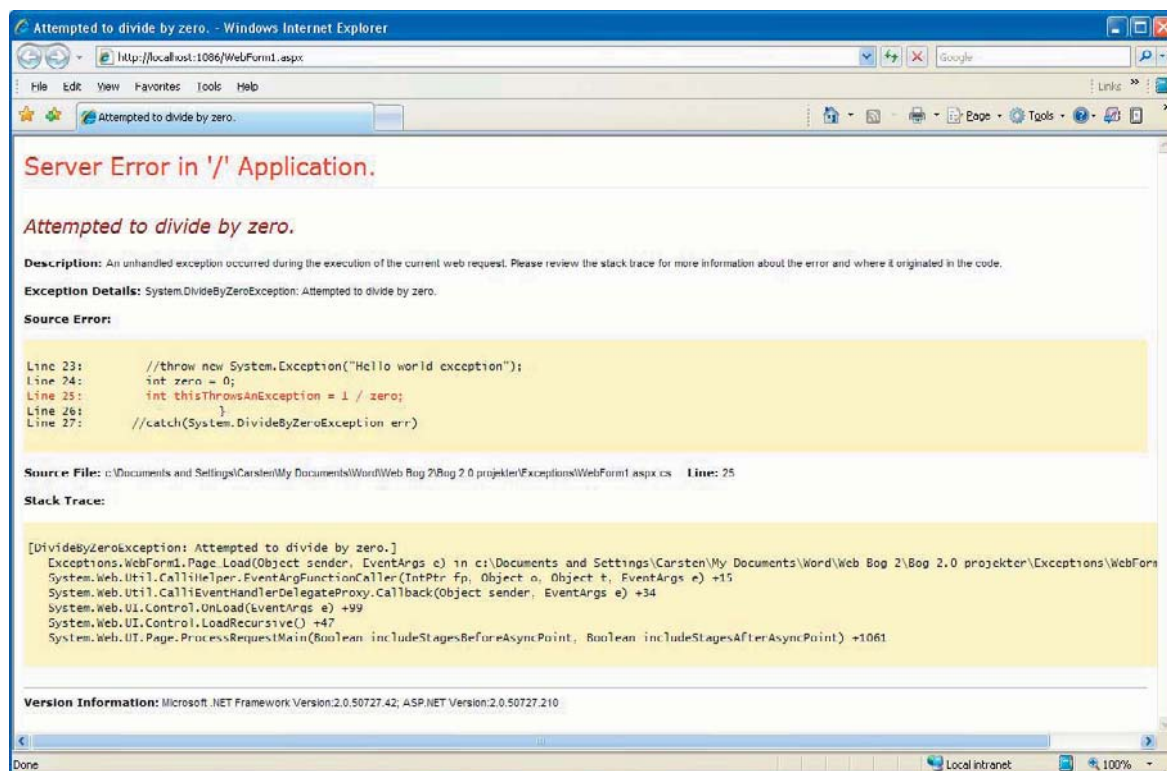
I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



## 6. Exception handling

Exception handling eller fejl håndtering som det hedder på dansk har jeg fuldstændig ignoreret hvilket gør at eventuelle fejl ikke bliver håndteret af koden men af webserveren. Exceptions kan optræde i en hel række forskellige sammenhænge som f.eks. disk access, database fejl, division med nul med mere. Det kan selvfølgelig være ok at webserveren håndtere fejl sålænge man udvikler, men på et tidspunkt skal websiden benyttes af kunder og så duer det ikke med en besked som vist i Figur 55.



Figur 55. Uhåndteret exception besked til brugeren

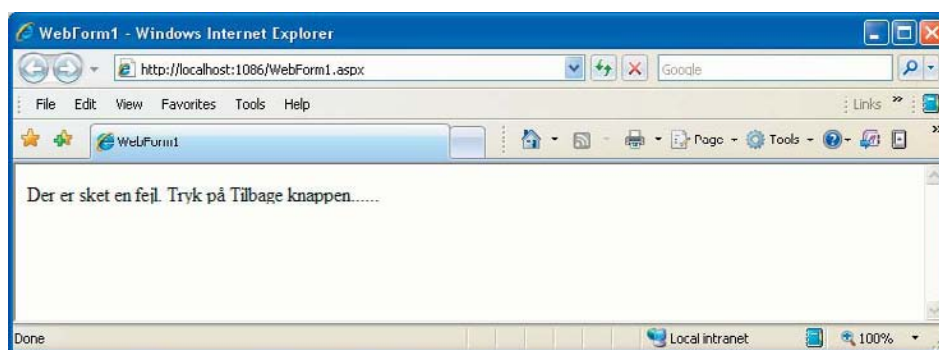
Som det kan ses af eksemplet divideres der med 0 hvilket fører til at der kastes en exception. Koden er kun synlig i browseren fordi den er compileret som debug code, men det ændrer ikke ved at man skal have styr på sin exception handling! Koden der genererer ovenstående er placeret i forbindelse med load eventet og ser således ud:

```
private void Page_Load(object sender, System.EventArgs e)  
{  
    int zero = 0;  
    int thisThrowsAnException = 1/zero;  
}
```

Når man har statements, der kan kaste en exception bør man altid definere en exception handler. I dette tilfælde kan koden nemt modificeres så vi fanger den exception, der kastes og vi kan så fortælle brugeren på en pæn måde at der er gået noget galt og give ham mulighed for at vende tilbage til f.eks. home for den pågældende web site.

```
private void Page_Load(object sender, System.EventArgs e)
{
    try
    {
        int zero = 0;
        int thisThrowsAnException = 1/zero;
    }
    catch
    {
        Response.Write("Der er sket en fejl. Tryk på Tilbage  
knappen.....");
    }
}
```

Til at fange exceptions benyttes en try catch blok. I dette eksempel bekymre jeg mig ikke om typen på den exception der er kastet. Jeg definerer at alle exceptions der måtte kastes i de to liner kode der er i try blokken skal fanges i catch blokken. Hvis man køre ovenstående kode ser resultatet således ud:



Figur 56. Sempel catch blok

Hvis man har en ide (og det har man som regel) om hvad det er for en type exception der kastes er det den man skal definere en handler for og reagere hensigtsmæssigt på det. I dette eksempel ved jeg at det er en division med 0 der er skyld i at der kastes en exception og min handler bør derfor afspejle det.

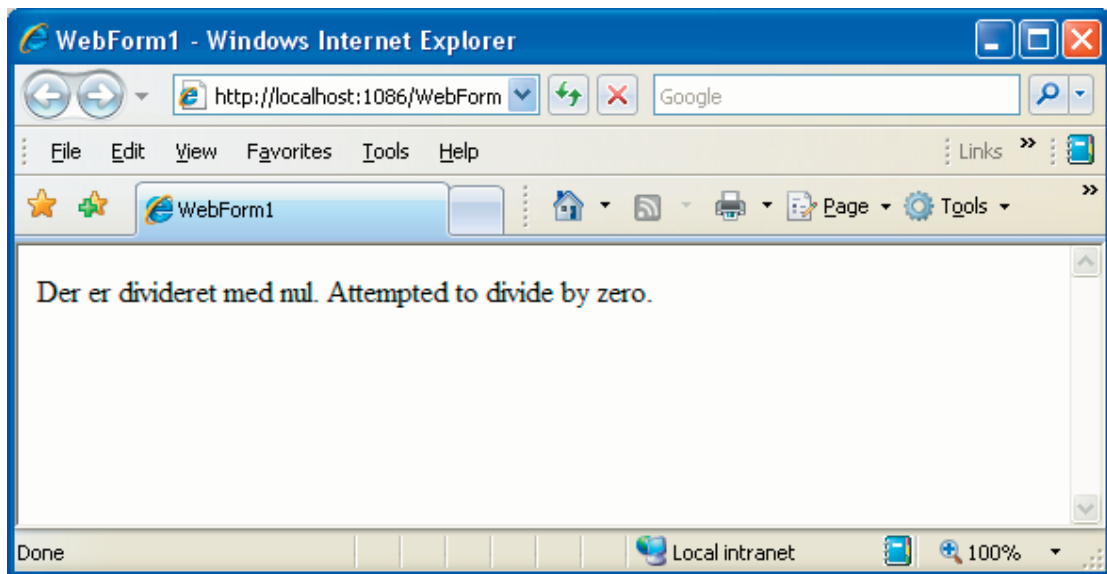
```
try
{
    int zero = 0;
    int thisThrowsAnException = 1/zero;
}
catch(DivideByZeroException err)
{
    Response.Write("Der er divideret med nul. " + err.Message);
}
```

```

}
catch
{
    Response.Write("Der er sket en fejl. Tryk på Tilbage knappen.....");
}

```

Rækkefølgen af de to viste catch blokke er ikke tilfældig. Den der ikke tager parametre skal være til sidst. Læg mærke til at man i System namespace har adgang til en hel række exception klasser. I dette tilfælde benyttes DivideByZeroException klassen. Hvis man køre ovenstående kode fås følgende:



Figur 57. catch blok med type.

Man har også mulighed for selv at kaste exceptions med en throw statement.

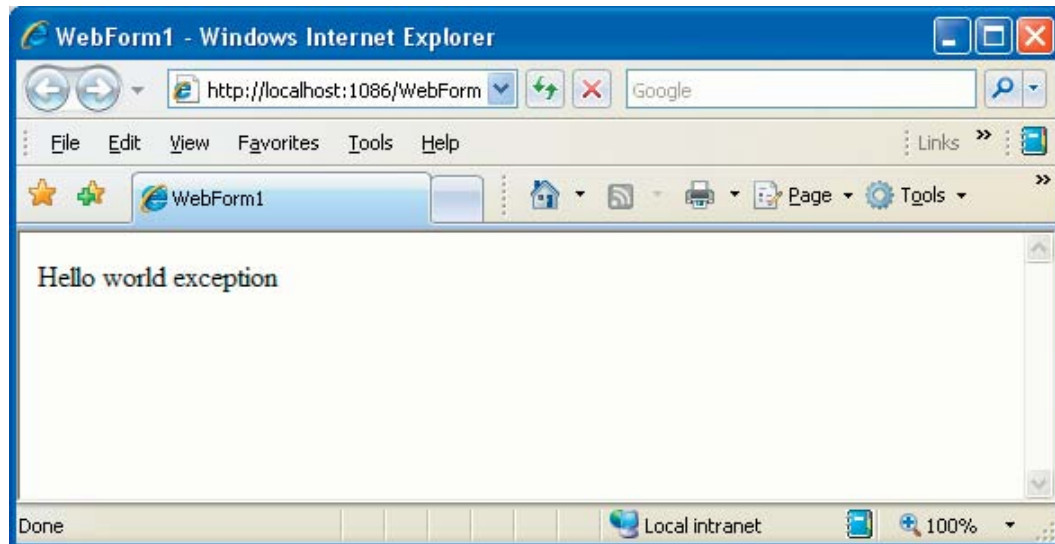
```

try
{
    throw new System.Exception("Hello world exception");
    int zero = 0;
    int thisThrowsAnException = 1/zero;
}
catch(DivideByZeroException err)
{
    Response.Write("Der er divideret med nul. " + err.Message);
}
catch(Exception err)
{
    Response.Write(err.Message);
}
catch
{
}

```

```
Response.Write("Der er sket en fejl. Tryk på Tilbage knappen.....");  
}
```

I ovenstående eksempel kastes der nu en `System.Exception` og de to næste linie i try blokken køres derfor ikke. Istedet fanges den kastede exception i den til formålet definerede catch blok og resultatet ser således ud:



Figur 58. Exception fanget.



**Ses vi til DSE-Aalborg?**

Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**

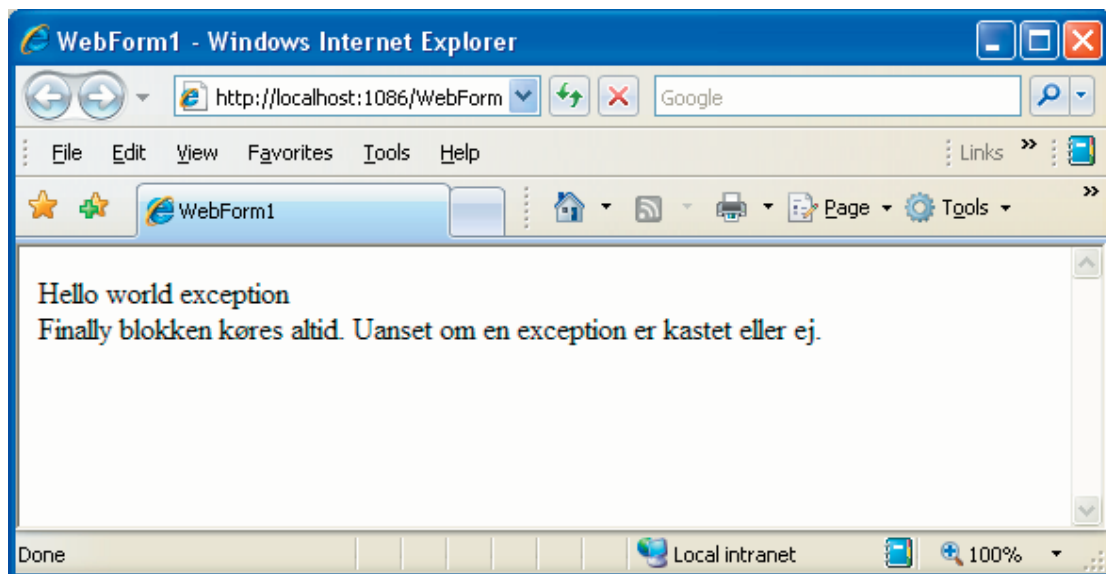


Det sidste jeg vil nævne er finally blokken. Man kan udvide sin try – catch blok med en finally statement, hvis der er noget kode man altid vil have kørt uanset om der er kastet en exception eller ej. Selv hvis du i en catch blok har en return statement køres finally blokken alligevel. Eksemplet udvidet med en finally blok ser således ud:

```
try
{
    throw new System.Exception("Hello world exception");
    int zero = 0;
    int thisThrowsAnException = 1/zero;
}
catch(DivideByZeroException err)
{
    Response.Write("Der er divideret med nul. " + err.Message);
}
catch(Exception err)
{
    Response.Write(err.Message);
    return ;
}
catch
{
    Response.Write("Der er sket en fejl. Tryk på Tilbage knappen.....");
}
finally
{
    Response.Write("<br>Finally blokken køres altid. Uanset om en exception er kastet eller ej.");
}
```

Som det kan ses har jeg indsat en return statement i den catch blok der fanger den exception der kastes af throw statementen. finally blokken skal afvikles ifølge C# standarden og en kørsel giver da også det forventede resultat:





Figur 59. finally blok eksempel

Efter vi indsatte en `catch(Exception err)` blok i koden blev `catch` blokken uden parametre overflødig. Meget mere om exception handling kan læses i både MSDN library og i bøger om C#.

**CISO Conference**  
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

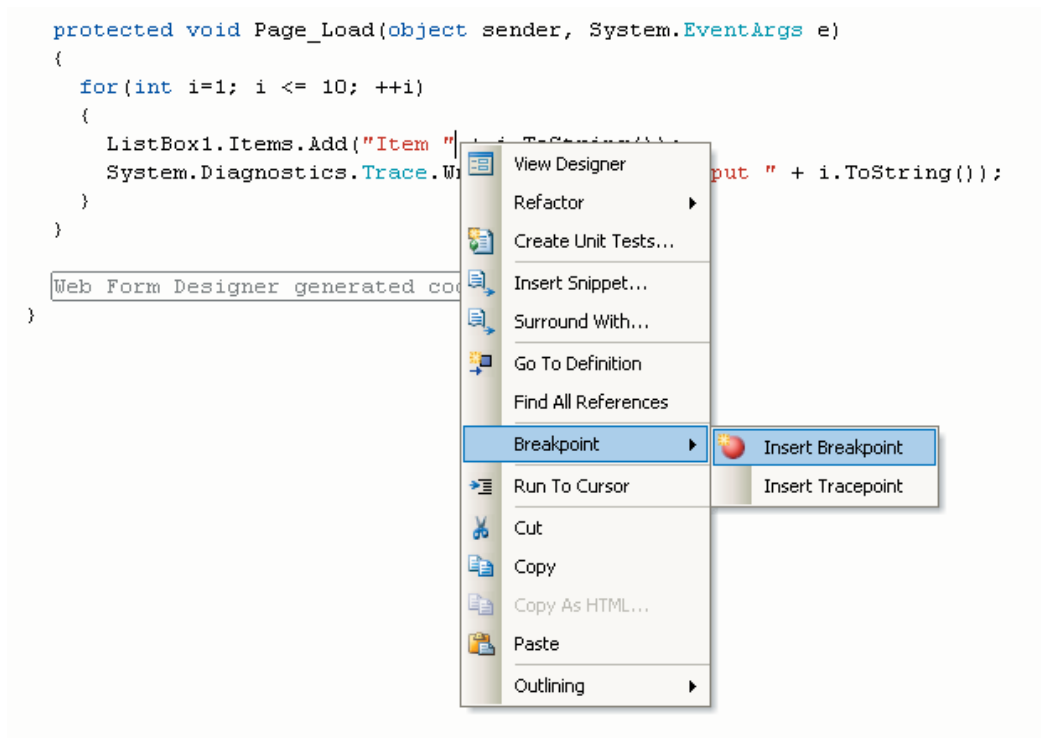
**Inspired**



## 7. Debugging

Debugging er utroligt nyttigt når man udvikler kode. Det uanset om det er standard windows software, komponenter eller web software. Med den nye Visual Studio 2005 har man virkelig fået et solidt værktøj til rådighed i form af et lækkert udviklingsmiljø, der i en grad som aldrig tidligere gør det muligt at debugge web applikationer.

Først og fremmest er der muligheden for at sætte breakpoints direkte i koden. Breakpoints kan indsættes på en given kode linie ved blot at trykke F9 (hvis du altså kører med standard opsætningen) eller ved at højre klikke på den kode line man ønsker at indsætte breakpointet og så vælge "Insert breakpoint". Det ser således ud:



Figur 60. Højre klik for at indsætte et breakpoint.

Efter breakpointet er indsat er linien markert og highlighted. Det ser således ud:

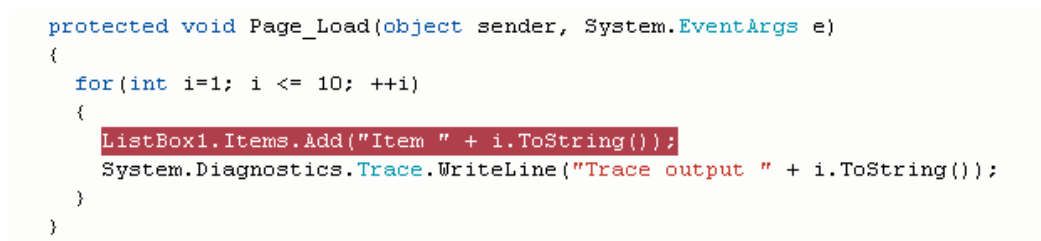
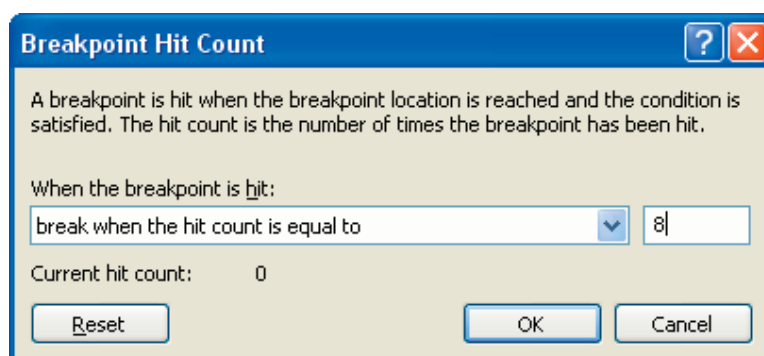


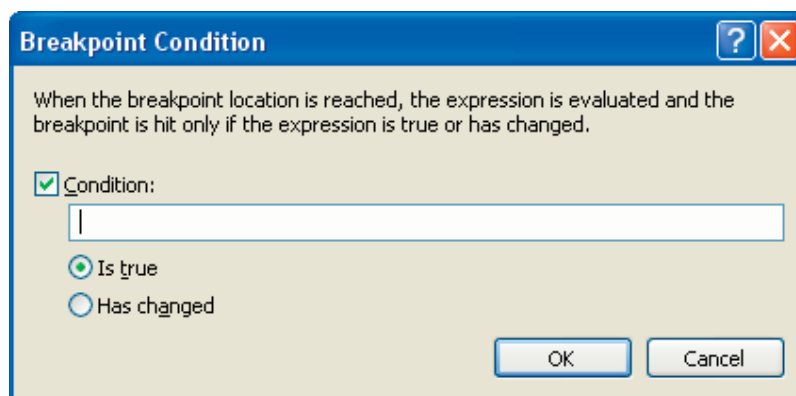
Figure 61. Breakpoint highlighted.

Hvis man afvikler koden nu stoppes afviklingen på den viste kode linie. I bunden af udviklings miljøet kan man f.eks. kigge i *Autos* for at finde information om den eller de variable man ønsker eller man kan højreklikke og vælge QuickWatch for at få en stor dialog hvor man rigtig kan kigge en variabel efter. For at fortsætte er det letteste blot at trykke F5. I ovenstående eksempel vil eksekveringen stoppe ialt 10 gange. Lad os nu f.eks. sige at man vidste at de første 7 gange skete der ikke noget interessant, men den 8 og 9 gang ville man gerne følge med. Hvad gør man så? Man kan selvfølgelig selv tælle hvor mange gange man har været ved breakpointet, men hvis det nu var 1000 gange der skulle passeres inden der skete noget interessant blev det lidt at en opgave. Istedet kan Visual Studio 2005 debuggeren hjælpe os. Man kan sætte properties på sit breakpoint. Under Breakpoints menuen er der flere muligheder. Vi vælger "Hit Count" ved at højreklikke op breakpointet. Man får følgende dialog:



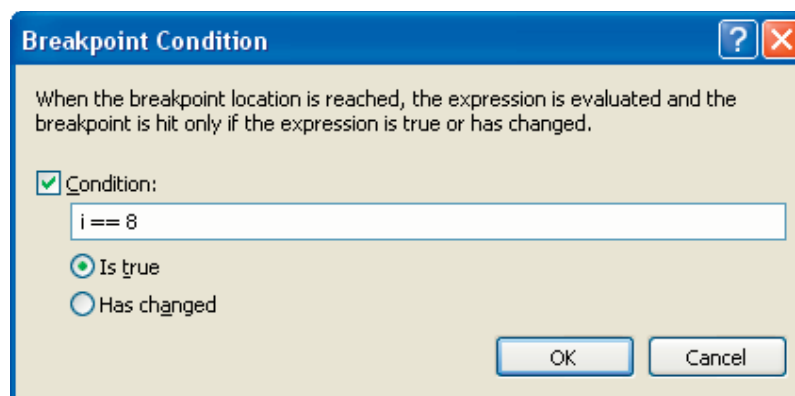
Figur 62. Sæt hitcount til 8.

Som det kan ses på Figur 62 har jeg i comboboxen valgt at fortælle debuggeren at der skal breakes den 8 gang. En anden mulighed ville være sætte en Condition altså en betingelse for hvornår der skal breakes. Ved at vælge Condition fås følgende dialog:



Figur 63. Breakpoint condition.

Her skriver man så hvad der skal være opfyldt for at der breakes. I mit eksempel ville betingelsen være at  $i=8$  så det sætter jeg ind:



Figur 64. Condition i=8.

Læg mærke til at man skal benytte ligheds (equals to) operatoren og ikke blot skrive `i=8`.

Breakpoints kan være og er meget meget nyttige, men nogen gange vil man blot gerne have logget hvad der skete uden at det hele afbrydes af et breakpoint. Heldigvis har man også den mulighed. *I en rigtig applikation kan et logging framework som log4net varmt anbefales.*

I namespace System.Diagnostics findes der en hel række klasser med en række statiske funktioner man kan kalde for at udskrive debug information. Informationen kan udskrives til både output vinduet i Visual Studio 2005 editoren og til en "omfattende" rapport over hvad der er sket på siden en side blev loaded. Lad os se på det sidste først.

**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**

For at generere en kørsels rapport med trace information skal man sætte trace=true. Det kan gøres ved at sætte trace="true" i @Page direktivet på HTML tabben på Web Form ens design flade. F.eks. som i dette eksempel:

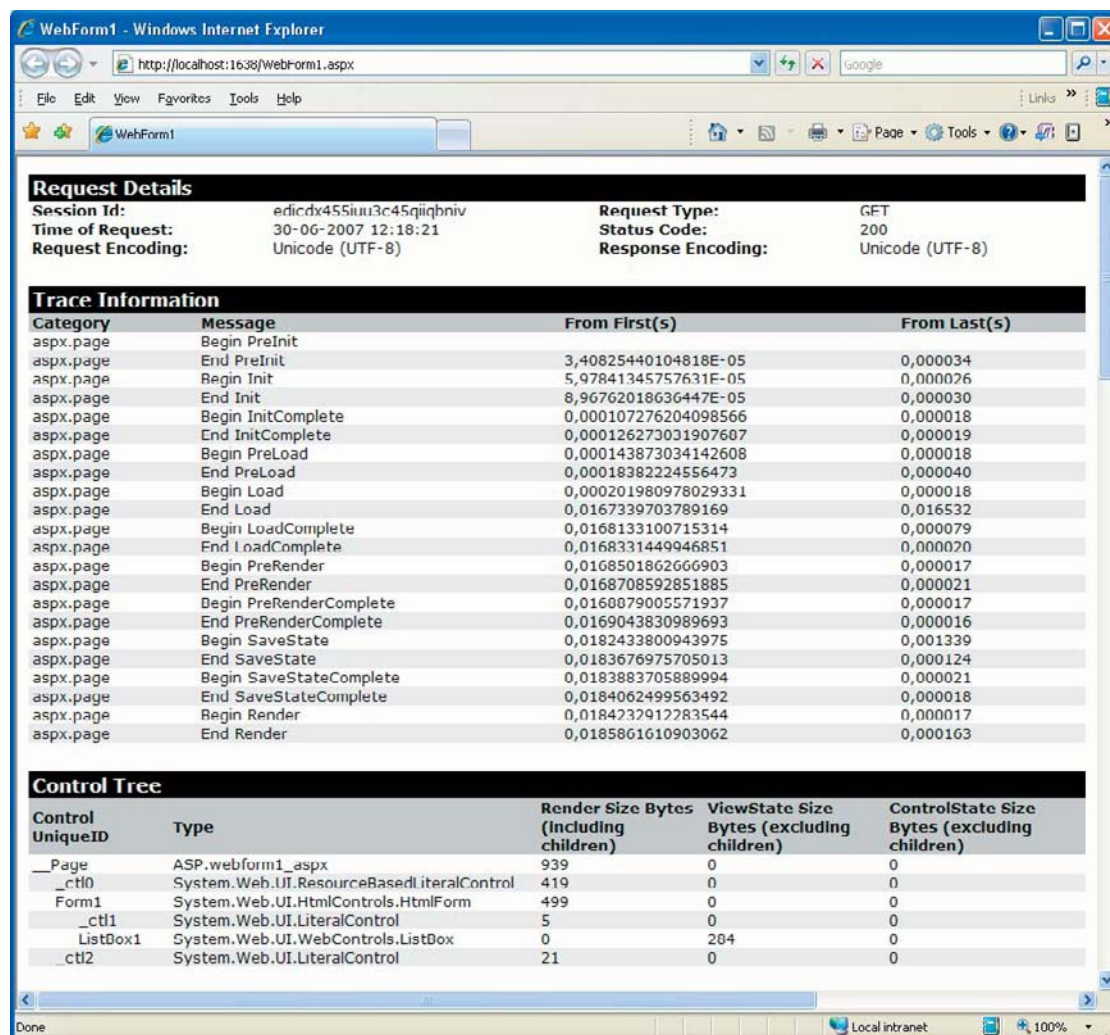
```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
Inherits="DebugEx.WebForm1" trace="true" %>
```

Man kan i koden aflæse om trace er sat ved at kalde Trace.IsEnabled idet der på Page objektet er en Trace property. Man kan ikke i koden sætte trace til true! (endnu)

Hvis man vil have skrevet noget ud kan det gøres ved blot at kalde Trace.Write. Det kunne f.eks. se således ud:

```
private void Page_Load(object sender, System.EventArgs e)
{
    for(int i=1; i <= 10; ++i)
    {
        ListBox1.Items.Add("Item " + i.ToString());
        Trace.WriteLine("Trace output " + i.ToString());
    }
}
```

Trace.Write skriver til trace rapporten. Afvikling af ovenstående kode på en web form med trace sat til true giver følgende:



Figur 65. Trace rapport.

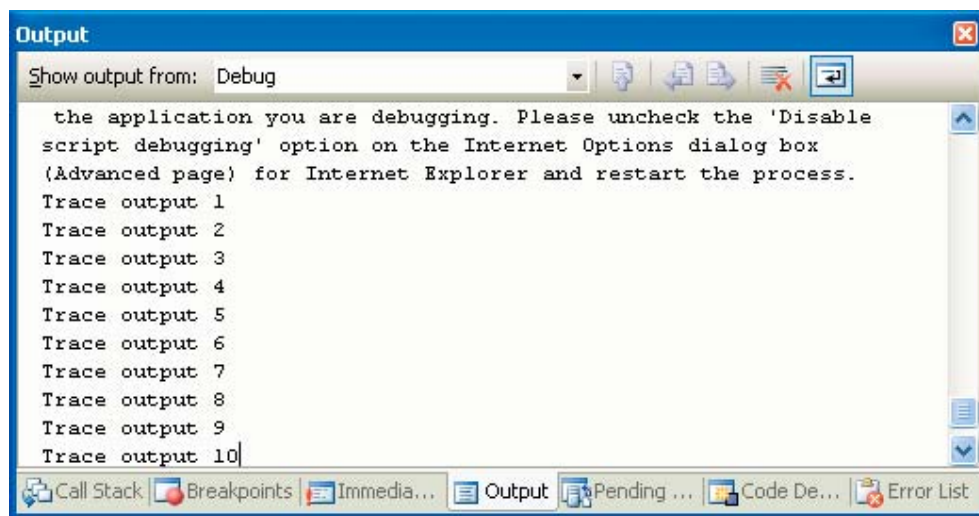
Som man kan se af Figur 65 kan man virkelig følge med i hvilke events der bliver sendt og hvornår samt man kan se sin egen trace kode som en del af rapporten. Rapporten har følgende hoved overskrifter.

- Request Details
- Trace Information
- Control Tree
- Session State
- Application State
- Cookies Collection
- Headers Collection

- Form Collection
- QueryString Collection
- Server Variables

Læs selv mere om dem i MSDN eller på [msdn.microsoft.com](http://msdn.microsoft.com).

Output vinduet er også særdeles nyttig og det vil typisk være det du benytter istedet for at generere en trace rapport. I namespace System.Diagnostics findes der en hel række klasser der kan hjælpe med at debug information. Jeg benytter WriteLine istedet for Write i dette eksempel idet man i output vinduet skal huske selv at skrive til ny line. Informationen i output vinduet for ovenstående eksempel ser således ud:



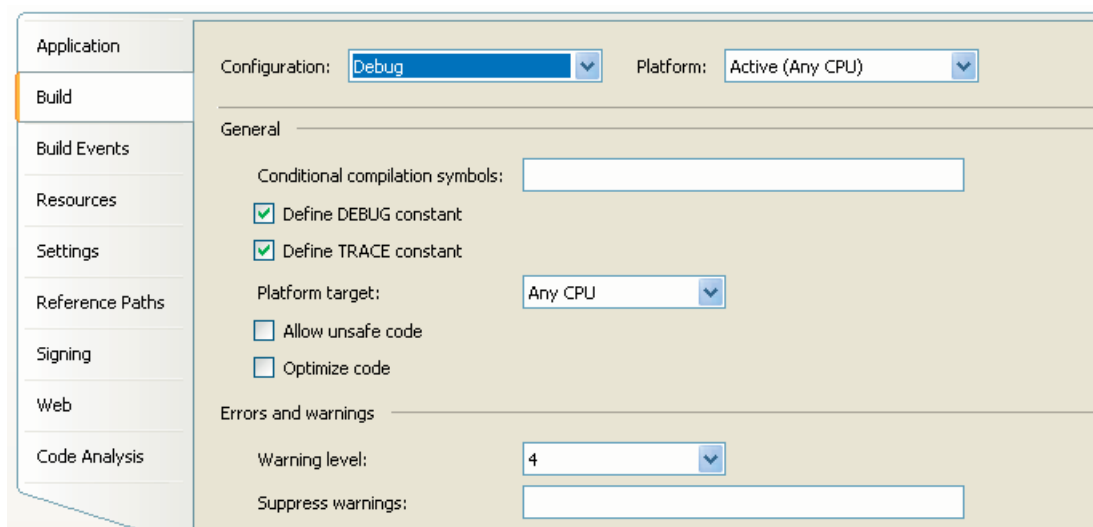
Figur 66. Trace information i output vinduet.

Man har også mulighed for at styre om der skal skrives til output vinduet ved at teste om en betingelse er opfyldt. Hvis jeg f.eks. kun ønsker output når i er lige kan jeg vælge at benytte `WriteLineIf`:

```
System.Diagnostics.Trace.WriteLineIf(i % 2 == 0, "Trace output " + i.ToString());
```

Ovenstående kode tjekker om i modulo 2 er lig 0. Dvs. er i lige? Hvis ja, udskrives strengen efter kommaet, ellers udskrives der intet. Der er en tilsvarende Debug klasse som kan det samme som Trace, men den er default kun aktiv i en debug udgave af applikationen. En release udgave der benytter Debug klassen bliver derfor ikke sløvet af trace informationen. Normalt vil jeg derfor foretrække at benytte Debug klassen, men Trace klassen der default er enablet i både debug og release builds, kan også disables i både release og debug builds ved at sætte trace til false.





Figur 67. Projekt Properties.

Ved at fjerne TRACE fra release builden afvikles Trace statement ikke. Om man benytter den ene eller den anden afhænger derfor meget af ens behov.



**MTHøjgaard**

## BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)



## 8. Web services

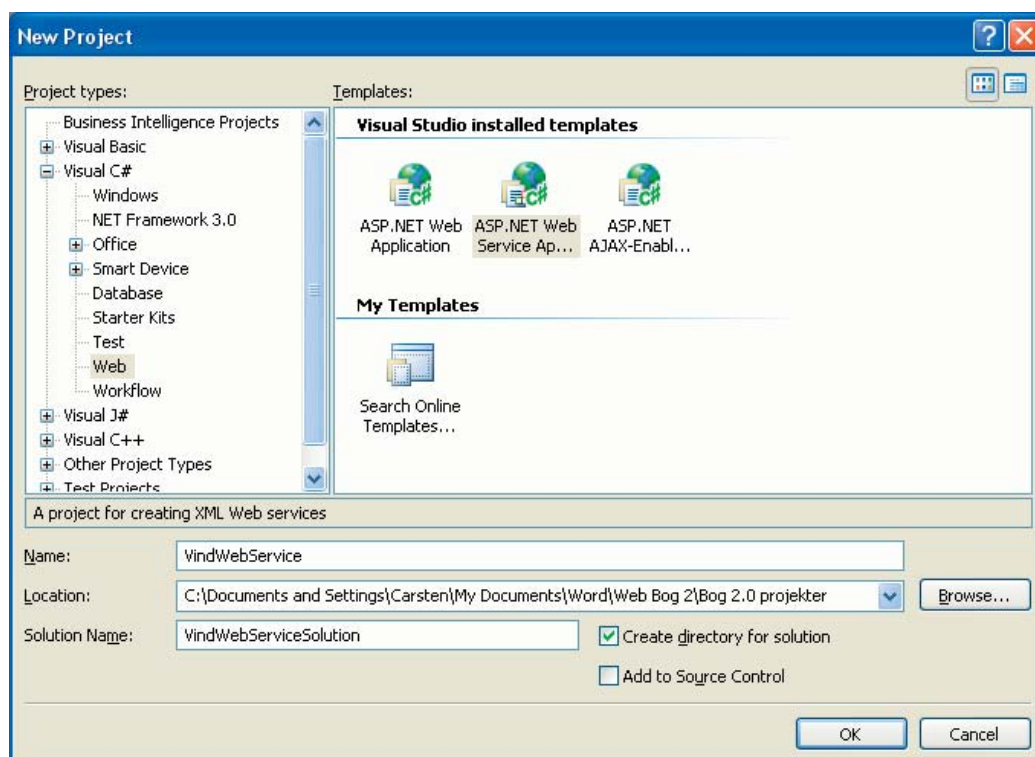
En web service giver mulighed for at udveksle information mellem en server og en client på en standardiseret måde via HTTP protokollen. Et eksempel kunne f.eks. være en web service der tilbød den aktuelle vindhastighed og retning som funktion af en position, hvilket ville være af relevans for sejlerfolket. De kunne f.eks. som en del af en applikation til beregning af kurs og sejltid have indbygget et kald til en web service der gav den relevante information for vindhastighed og retning. Et andet eksempel kunne være en web service der gav en given aktie kurs som funktion af et aktive symbol.

Web services udveksler data over det globale internet via HTTP protokollen, men pakker data ind via SOAP protokollen (Simple Object Access Protocol), der er baseret på XML.

### 8.1 En simpel web service

Jeg vil nu lave en simpel web service der giver clienter mulighed for at hente den aktuelle vindhastighed og vindretning som funktion af en position. Positionen angives som en længde og en bredde grad. Jeg vil blot sende nogle random værdier tilbage.

I project wizarden vælges et *ASP.NET Web Service* project og jeg kalder servicen for *VindWebService*.



Figur 68. Vind Web Service.

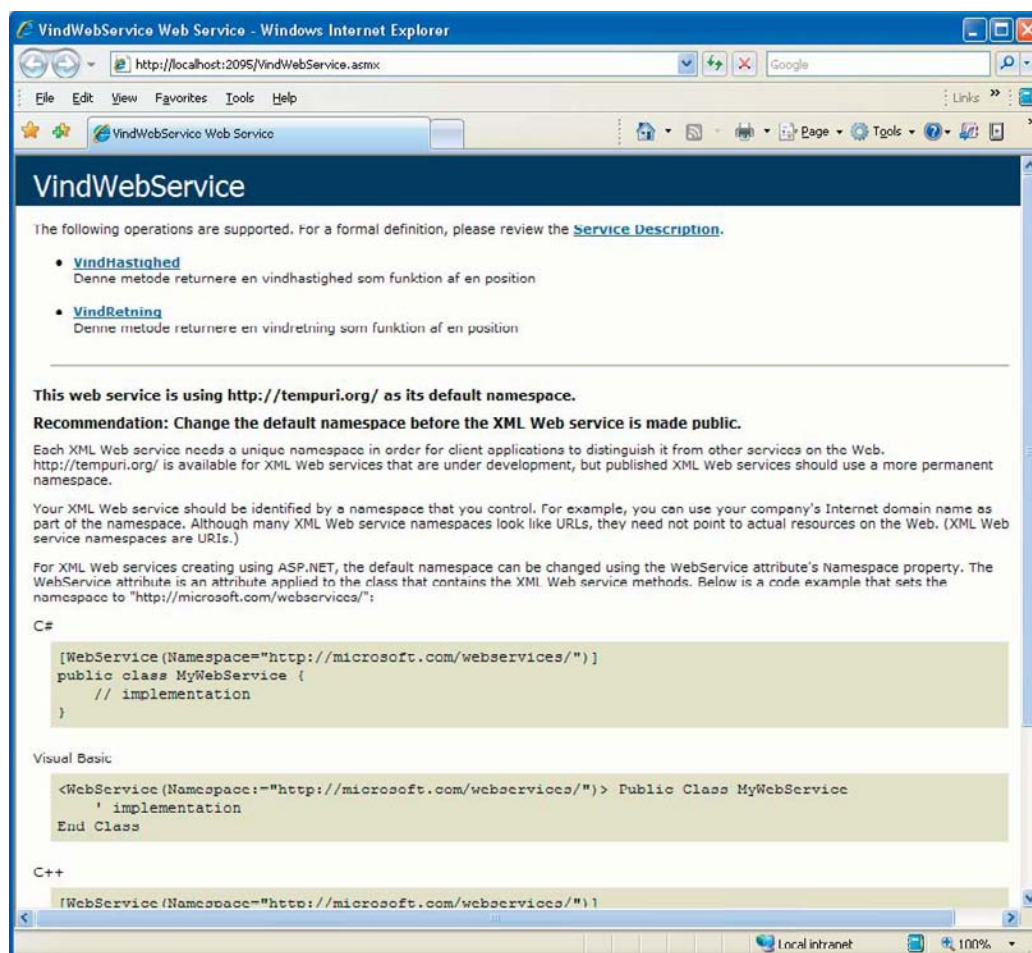
Som det kan ses af Figur 68 oprettes en web service præcis som en almindelig web applikation. I modsætning til en web applikation har man ikke nogen grafisk bruger flade til clienten. Clienten benytter web servicen som en instance af en klasse i sin applikation. I virkeligheden benytter clienten en proxy der så kommunikerer med en stub på web serveren. Wizarden opretter en klasse der nedarver fra `System.Web.Services.WebService` og giver klassen navnet `Service1`. Jeg omdøber til `VindWebService` og indsætter kode:

```
/// <summary>
/// Kald denne funktion for at få vindhastigheden som funktion af bredde-
længdegrad i m/s
/// </summary>
[WebMethod(Description="Denne metode returnere en vindhastighed som funktion af en
position")]
public double VindHastighed(double BreddeGrad, double LængdeGrad)
{
    Random rand = new Random();
    return rand.NextDouble()*20;
}

/// <summary>
/// Kald denne funktion for at få vindretningen som funktion af bredde- længdegrad
i m/s
/// </summary>
///
[WebMethod(Description="Denne metode returnere en vindretning som funktion af en
position")]
public double VindRetning(double BreddeGrad, double LængdeGrad)
{
    Random rand = new Random();
    return rand.NextDouble()*360;
}
```

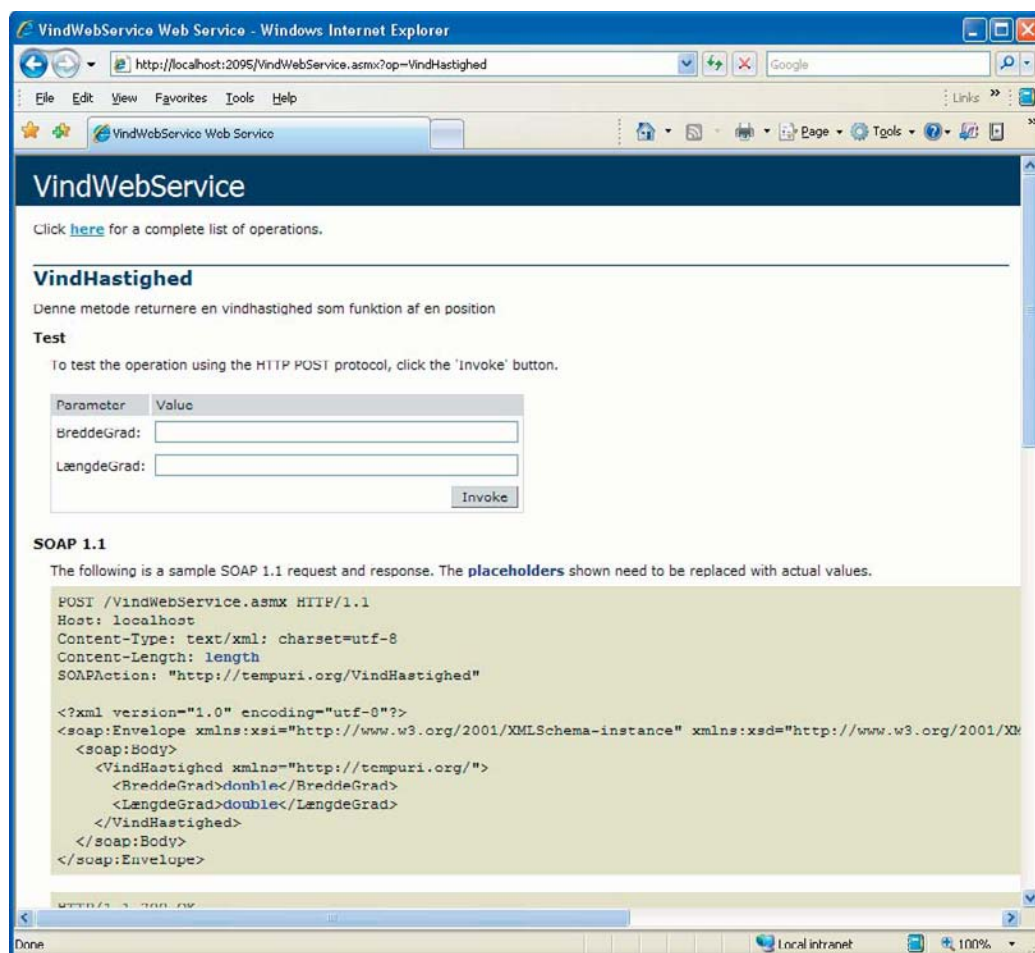
Der er indsat to metoder der begge som parameter tager en bredde og en længdegrad. De returnere begge en værdi af typen *double*. Som det kan ses benyttes `Random` klassen der via kaldet til `NextDouble` returnere en værdi mellem 0 og 1. Umiddelbart ovenover metoderene er attributten `WebMethod` indsat for at definere metoderne som værende tilgængelige over nettet, altså som en web service. Attributten `WebMethod` har 6 properties der også kan konfigureres. Jeg har valgt at beskrive hvordan metoderne virker via *Description* attributten.

For at teste webservicen kan man bare starte debuggeren og accesse servicen via browseren. Når det gøres dannes en default web side der ser således ud:



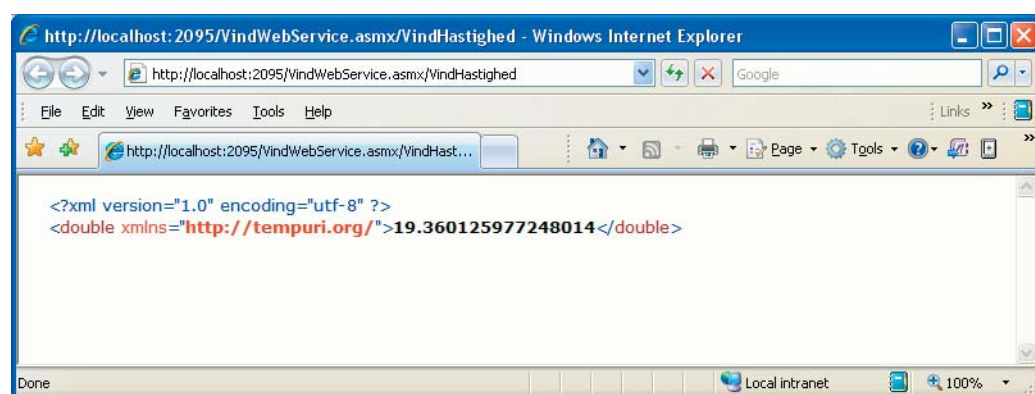
Figur 69. Default web side for en web service

Man kan se at specielt namespace bør ændres inden man releaser web servicen, men man kan også se de to metoder VindRetning og VindHastighed, samt en beskrivelse af dem. Beskrivelsen er den jeg definerede med attributten *Description*. Hvordan man ændre default namespace er beskrevet på default siden. Hvis man klikker på f.eks. VindHastighed havner man igen på en default genereret side:



Figur 70. Default side for VindHastighed metoden.

Man har her mulighed for at teste metoden ved at angive parametre og kalde Invoke. Når det gøres fås igen en default genereret side som i dette tilfælde ser således ud:



Figur 71. Resultat fra VindHastighed metoden.

Man kan se at metoden har returneret 19.36 m/s for vindhastigheden.

Ovenstående er glimrende til at teste om ens web service virker. I praksis vil man dog skrive noget code som integrerer web servicen i en applikation.

For at teste web servicen skriver jeg en lille windows test applikation i C#. Ved brug af Visual Studio 2005 Windows Applikation wizard og ved brug af toolboxen akkurat som hvis det var en ASP.NET applikation dannes en form indholdende kontroller. For at integrere web servicen i applikationen benyttes Solution Explorere og der højre klikkes på References og i menuen vælges Add Web Reference som vist på Figur 72.



**Ses vi til DSE-Aalborg?**

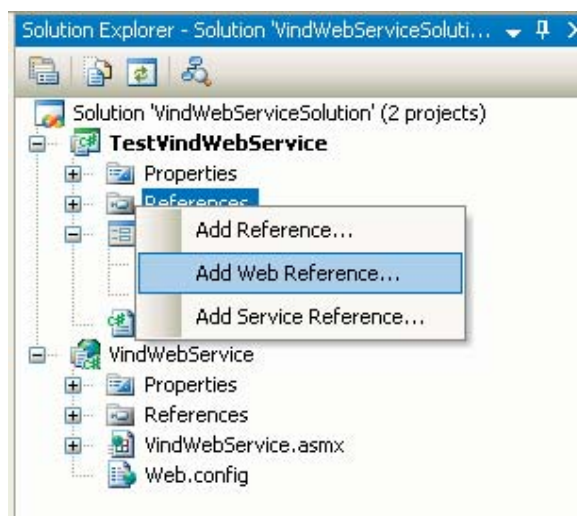
Kom forbi vores stand den  
9. og 10. oktober 2019.

Vi giver en is og fortæller  
om jobmulighederne hos  
os.

**banedanmark**

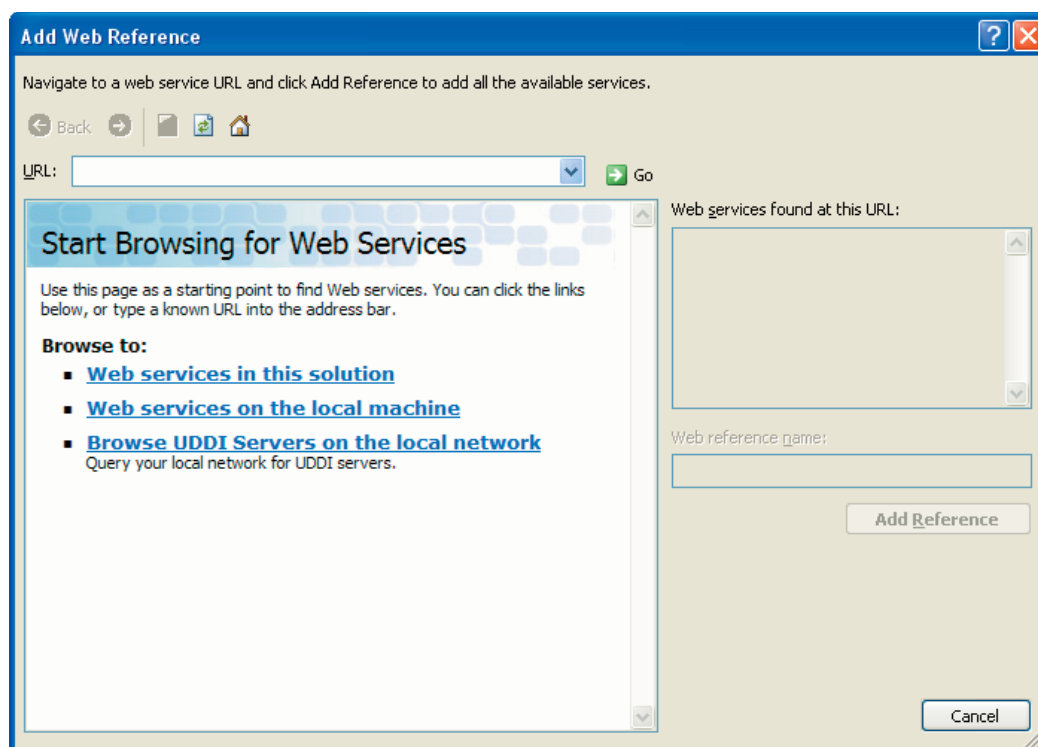






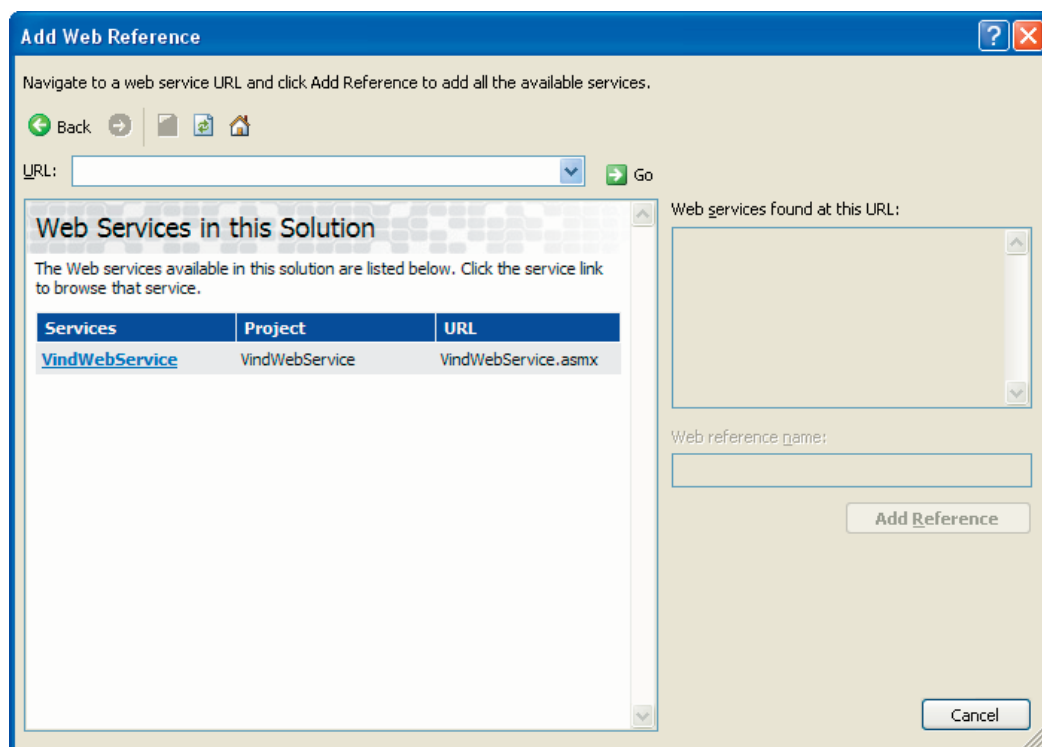
Figur 72. Addering af en Web service til projektet.

Når det gøres bliver man bedt om en URL (http:// adressen) for den web service man ønsker at benytte:



Figur 73. Addering af web service.

Hvis du har flere maskiner kan du indtaste NetBios adressen, endnu bedre IP adressen, eller bedst www adressen (DNS) på maskinen du ønsker at hente web servicen fra. Som det kan ses af Figur 73 er der flere andre muligheder og jeg vælger *Web Services in this solution*.

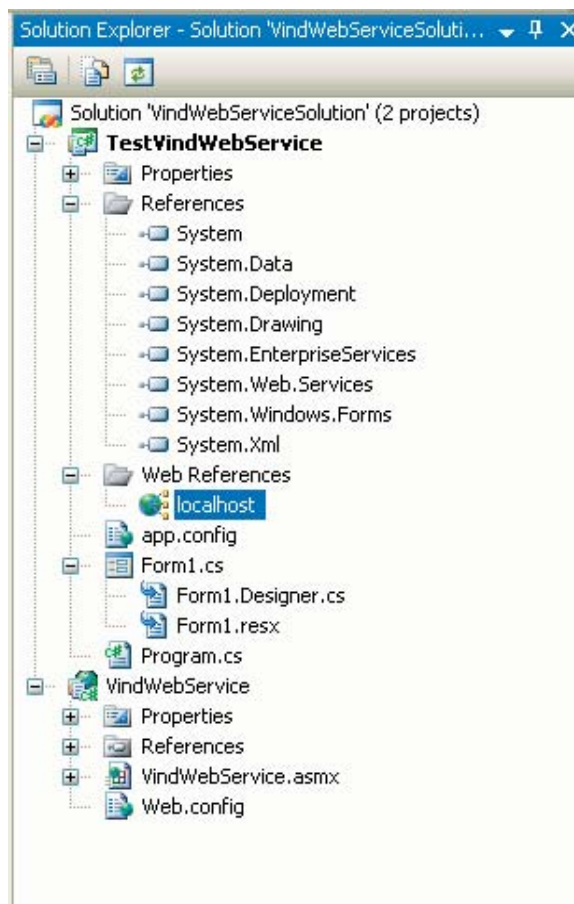


Figur 74. Valg af web service

Nu vælges VindWebService og der trykkes på Add reference knappen der i mellemtiden er enablet.

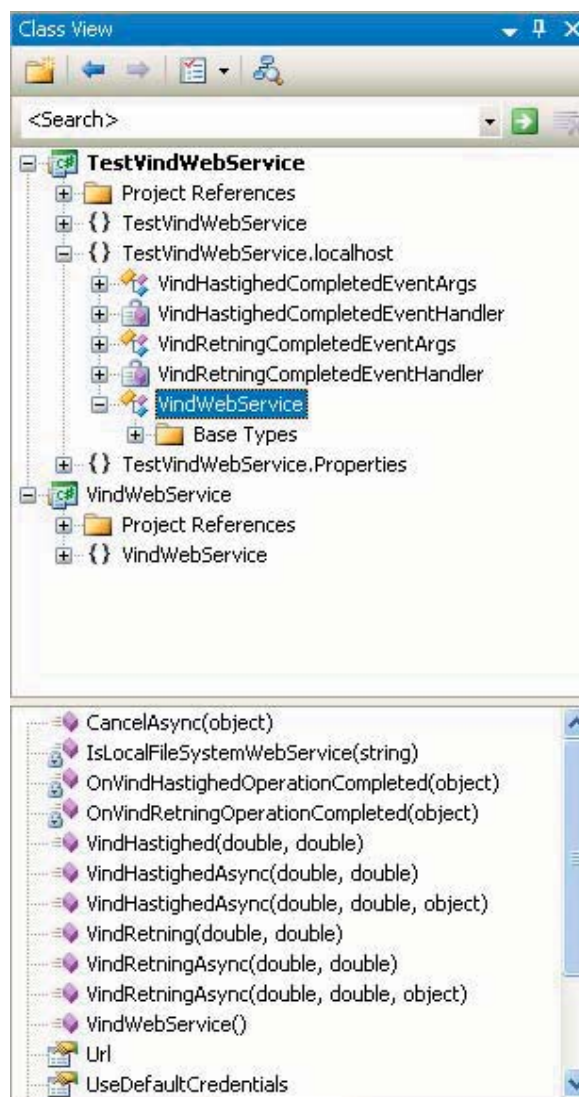
I Solution Explorer er web servicen nu synlig som vist på Figur 75.





Figur 75. Web service indsat i client programmet.

Også i Class View har man adgang til Web servicen. Den er nu præsenteret pænt som en klasse med metoder og parametre, samt evt. base klasser og interfaces<sup>vii</sup>.



Figur 76. Web servicen i Class View.

Nu kan man oprette et objekt af typen `VindWebService` hvor man vil, akkurat som alle andre C# klasser.

Jeg har en form der ser ud som vist på Figur 77 og vælger at instantiere et objekt af typen `VindWebService` lokalt i de metoder der kaldes når der trykkes på henholdsvis `Vind Hastighed` og på `Vind Retning`.

Figur 77. Test form for VindWebService.

Eventhandleren for Vind Hastighed ser således ud:

```
private void vindHastighed_Click(object sender, System.EventArgs e)
{
    VindWebService vws = new VindWebService();

    double BreddeGrad = Convert.ToDouble(breddeGrad.Text);
    double LængdeGrad = Convert.ToDouble(længdeGrad.Text);
    resultat.Text = vws.VindHastighed(BreddeGrad,
        LængdeGrad).ToString();
}
```

**CISO Conference**  
Produced by **Inspired**

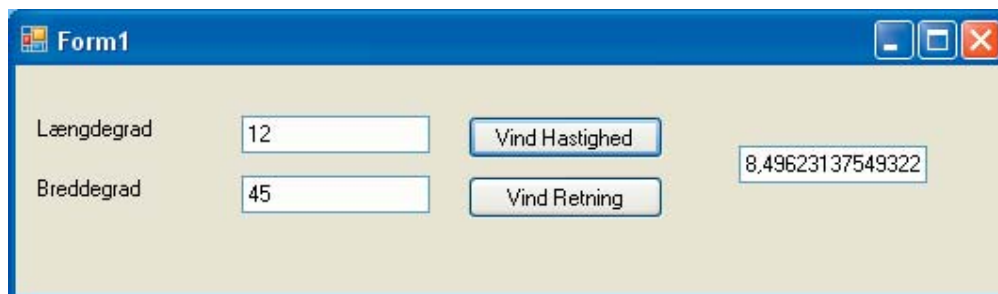
**Apollo Hotel 1, Groenlandsekade  
Vinkeveen, Amsterdam, NL  
Dec 5th 2019**

**Listen, learn & build relationships with our  
Network of CISOs & Cyber Security Leaders**

**Inspired**

Som det kan ses instantieres web services akurat som en normal C# klasse og metoder kaldes og valideres som andre C# klasser. Dette er virkelig helt suverænt!

En kørsel giver følgende:



Figur 78. Kørsel af TestVindWebService.

Webservicen startes automatisk op i den indbyggede Web server i Visual Studio og svare fint tilbage som det kan ses.

## 9. Afslutning

Vi har været igennem meget af det grundlæggende, men der er stadig meget at sætte sig ind i endnu. Af vigtige emner som ikke er berørt i denne bog er:

- Lokalisering (håndtering af flere sprog)
- Components/COM interfaces/C# interfaces
- Template kontroller
- Application Domains
- COM+, transactions og object pooling
- Caching
- Sikkerhed
- Master Pages
- Themes and skins
- Personalization
- Membership og Role Management

Jeg håber du får nytte af bogen og kommer igang med at kode.



**Max's next Bookboon eBook**  
**Your Boss: Sorted!**  
By Patrick Forsyth - 55 pages

**Unlock your life.**  
**Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

**bookboon.com**



## 10. Index

absolut position.....	22, 82
AdRotator.....	41, 43, 44, 45
AdvertisementFile .....	41, 43, 44
<i>AlternateText</i> .....	41
ASP.NET.....	6, 9, 15, 41, 64
<i>assembly manifest</i> .....	17
<i>attributter</i> .....	48
Attributter.....	41
Bindable .....	48
breakpoints .....	103
callback.....	25
Category .....	48
<i>CheckBoxList</i> .....	24, 28
Codebehind .....	106
<i>CommandName</i> .....	78, 79
<i>Composite Custom Control</i> .....	45, 48
Condition.....	104, 105
ControlToValidate .....	38, 51
cookies.....	53, 54, 55
CreateChildControls .....	49, 51
custom control.....	45, 48, 51
<i>CustomValidator</i> .....	38
Debugging .....	103
DefaultProperty .....	47, 48
DefaultValue .....	48
<i>delegate</i> .....	25, 51
Diagnostics .....	105, 108
<i>DropDownList</i> .....	24, 28
exception.....	51, 97, 98, 99, 100, 101, 102
finally .....	101, 102
Global.asax .....	18, 19, 78
Hello World.....	7, 10, 11, 12, 13
HtmlTextWriter.....	47, 48
<i>ImageUrl</i> .....	41, 43
<i>Impressions</i> .....	41, 43
INamingContainer .....	45, 48, 50
Internet Explorer.....	12, 30
Internet Information Server .....	7
label kontrol.....	10, 13, 14, 48
modulo .....	40, 108
namespace .....	15, 27, 50, 112
<i>NavigateUrl</i> .....	41, 43, 62, 64

persistent cookies .....	53
<i>PhysicalApplicationPath</i> .....	28, 34
<i>RadioButtonList</i> .....	24, 28
<i>Reflection</i> .....	48
Regulære udtryk .....	24
<i>RegularExpressionValidator</i> .....	24
<i>Request</i> .....	28, 30, 34, 54, 55, 107
<i>RequiredFieldValidator</i> .....	24
Response .....	27, 30, 54, 55, 90, 98, 99, 100, 101
ServerValidate .....	38, 40, 51
ServerValidateEventArgs .....	40, 50
Solution Explorer .....	17, 20, 21, 31, 116
SQL Server .....	6, 58, 59, 64, 91
<i>StreamWriter</i> .....	27, 28, 29, 34
<i>TextBox</i> .....	24, 28
throw .....	50, 99, 101
ToolboxData .....	47, 48
trace .....	106, 107, 108
try catch .....	98
<i>User Control</i> .....	45
validering .....	25, 38, 40
VIEWSTATE .....	15, 16
web service .....	110, 111, 112, 114, 115
Web services .....	110
Web.Config .....	17
WebMetod .....	111
XML .....	41, 42, 43, 44, 110



## 11. Noter

- <sup>i</sup> Det er nu svært at forestille sig der skulle kunne opstå økonomiske tab ud fra det der er præsenteret i denne bog, men ok, nu er jeg da dækket ind.
- <sup>ii</sup> Microsoft har også en AJAX teknologi der giver mulighed for asynkrone kald til webserveren. Det vil vi dog ikke komme nærmere ind på her.
- <sup>iii</sup> Metadata , instruktion.
- <sup>iv</sup> De kontroller der benyttes til at sammensætte ens composite kontrol kaldes for child kontroller.
- <sup>v</sup> Man ser ofte kitesurfere kunne det hele i løbet af 1 år. På 1 år kan man næsten ikke engang nå at lære at vandstarte en windsurfer. Windsurfing er bare meget mere udfordrende end kitesurfing! ☺
- <sup>vi</sup> Det vises dog ikke i eksemplet.
- <sup>vii</sup> Der er ikke multipel nedarvning af klasser, kun af interfaces.



 MTHøjgaard

**BEDRE  
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

[mth.dk/vorestilgang](http://mth.dk/vorestilgang)

