

# C# a ASP.NET

# 1. Úvod

## 1.1 Slovo na úvod

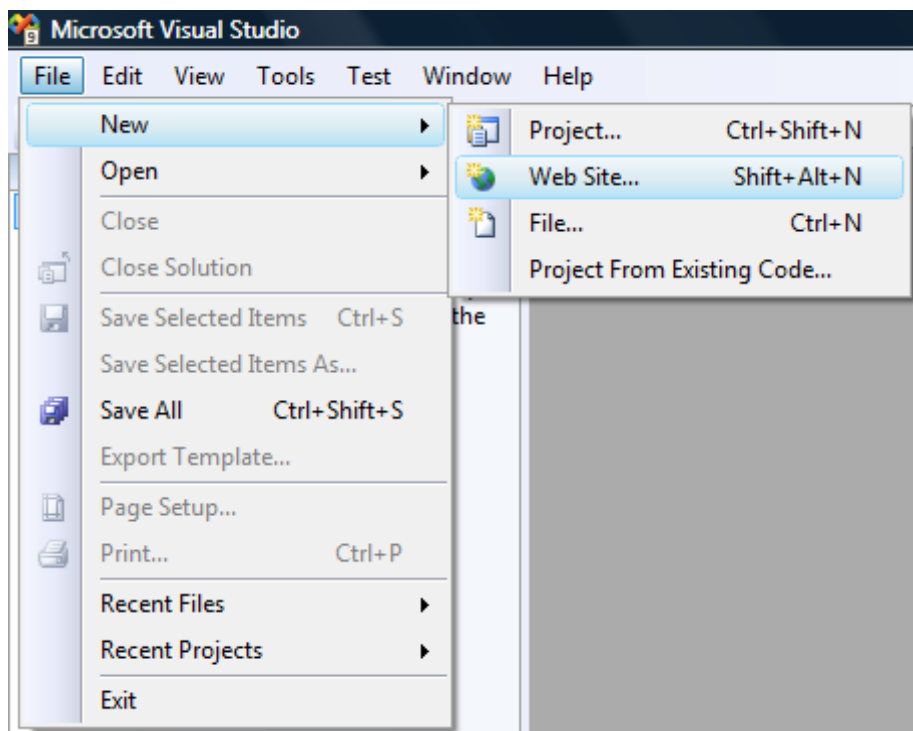
Gratuluji Vám, že jste se rozhodli programovat webové aplikace v ASP.NET. Tento text by Vám měl posloužit jako počáteční krok, který Vás naučí pracovat v programu z dílny Microsoft. Tím programem je Microsoft Visual Studio (dále jen VS). Přeji Vám aby byl tento text nápomocen s rozvíjením Vašich znalostí a dovedností v programování. Microsoft a Microsoft Visual Studio jsou obchodní značky společnosti Microsoft. Další informace naleznete na stránkách společnosti „[www.microsoft.com](http://www.microsoft.com)“.

## 1.2 Poznáváme prostředí Visual Studia

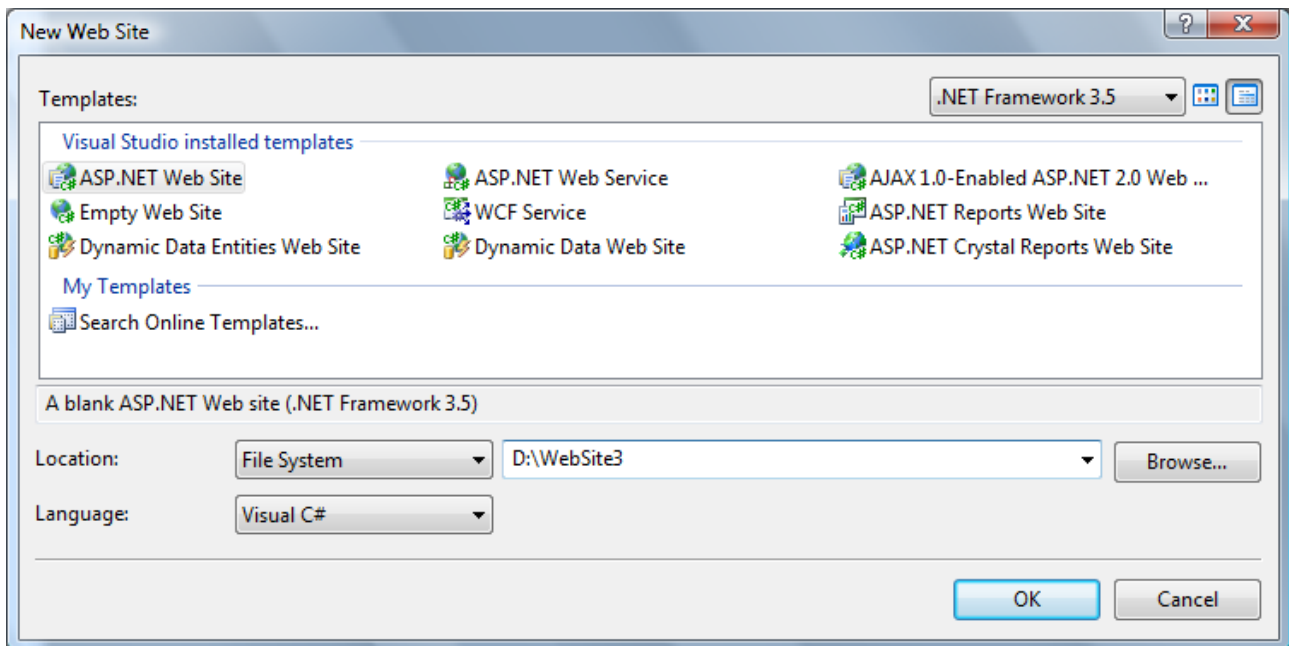
Základním předpokladem je znalost prostředí ve kterém budeme pracovat. Popíšeme si tedy základní operace a prostředky které nám VS nabízí.

### 1.2.1 Vytvoření první webové stránky

Abychom mohli začít pracovat musíme vytvořit nový projekt webové stránky. To provedeme jednoduše přes „Menu=>File=>New=>Web Site...“ jak můžete vidět na obrázku níže.



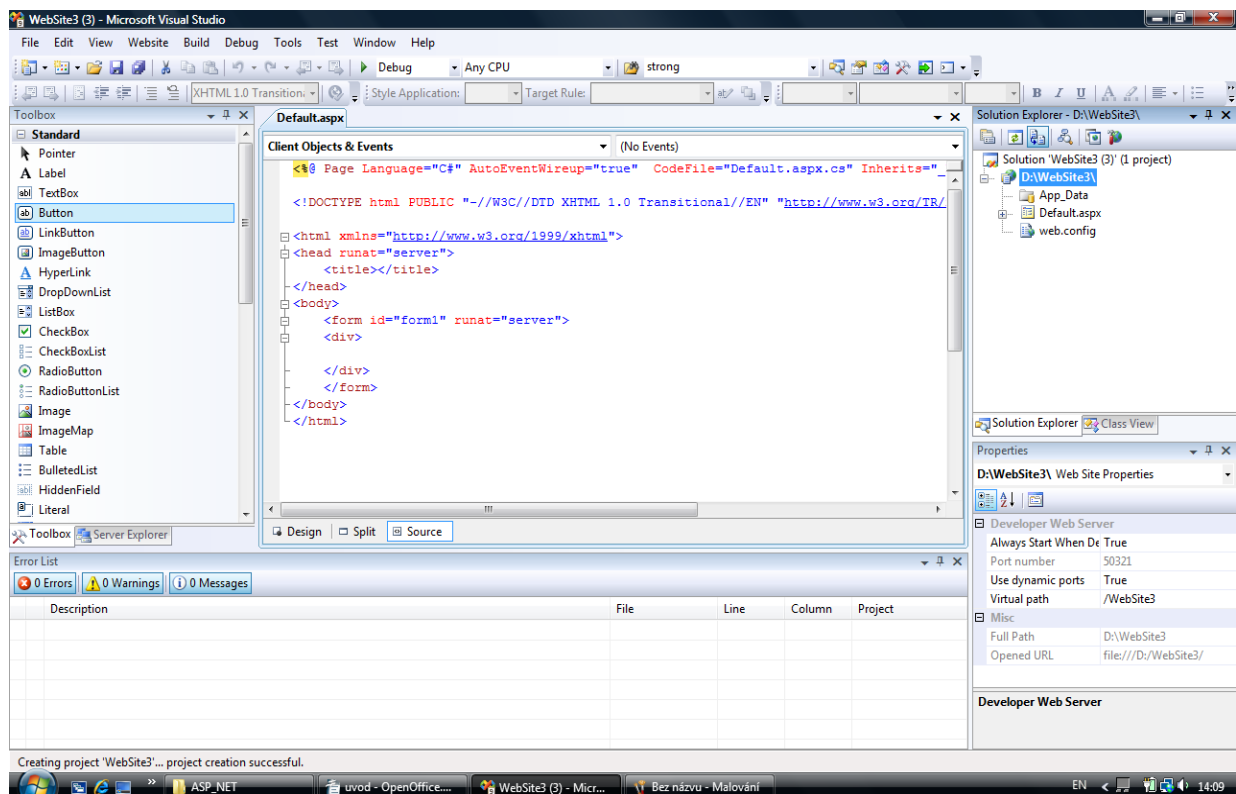
Po stisknutí na tuto volbu, se Vám zobrazí okno ve kterém máte vyplnit parametry Vašeho projektu. Společně si je tedy projdeme podle dalšího obrázku, tak abychom vytvořili šablonu webové stránky.



V horní liště šablon(Templates) musíme vybrat „ASP.NET Web Site“. Další volbou je umístění projektu(Location). Je vhodné si vybrat takové umístění, kde máte úplnou jistotu přidělení práv ke čtení a zápisu souborů. Nedoporučoval bych tedy umístění do dokumentů. Nakonec je na nás výběr programovacího jazyka(Language). Ten by měl být jak předpokládá kniha „Visual C#“. Je například možné programovat i ve „Visual Basicu“ a to hlavně z historických důvodů. Mnoho programátorů na něm začínalo. Již nám stačí potvrdit volbu přes tlačítko „OK“.

### 1.2.3 První příklad „Hello, world!“

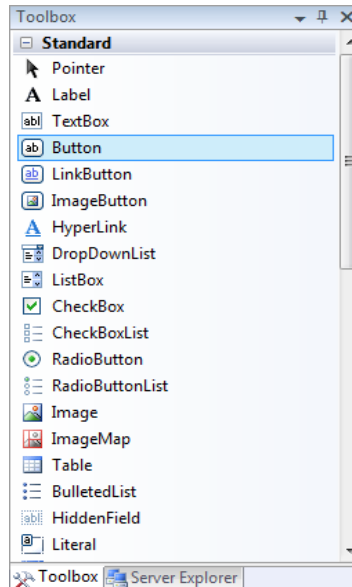
Po úspěšném vytvoření projektu by jste měli vidět okno podobné tomu na obrázku. Rozdělíme si jej na několik částí a zvaž si popíšeme jejich funkci.



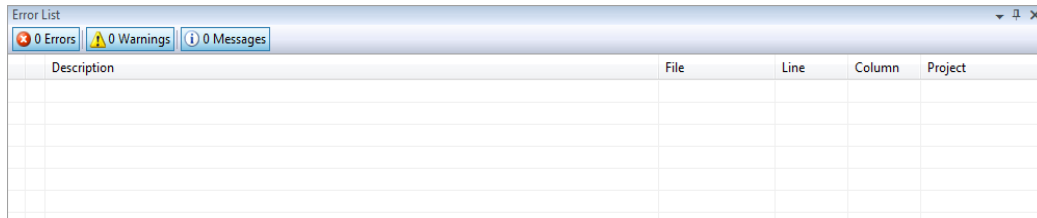
Z horní lišty pro vás bude nejdůležitější pouze jedno tlačítko a to tlačítko určené ke spuštění Vašeho řešení(Solution).



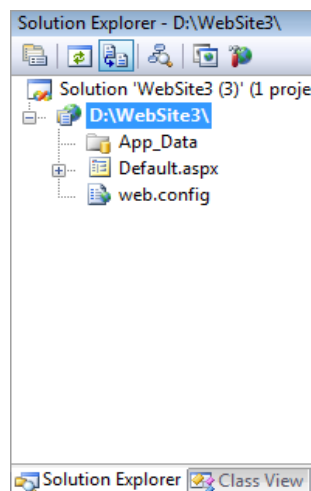
Dalším prvkem který má velký význam je „toolbox“ z něj si můžete vybírat objekty a vkládat je do HTML kódu stránky, nebo do integrovaného WISIWIG editoru kterým upravujete stránku v modu ve kterém vidíte přímo výslednou podobu. (WISIWIG = What I See Is What I Get).



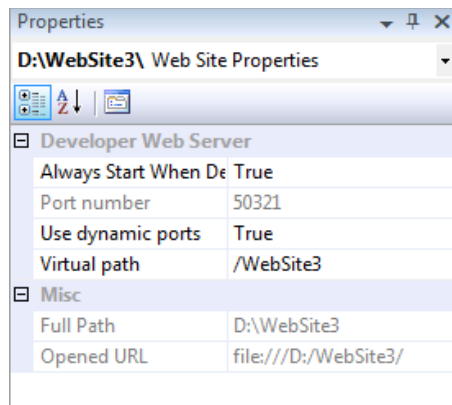
V dolní části obrazovky najdete „Seznam chyb“(Error list) ve kterém můžete sledovat chyby které prostředí VS odhalilo ve Vašem kódu. Za zmínku stojí možnost dvojitého kliknutí na případnou chybovou hlášku, která Vás odkáže přímo na místo kde kód způsobil chybu.



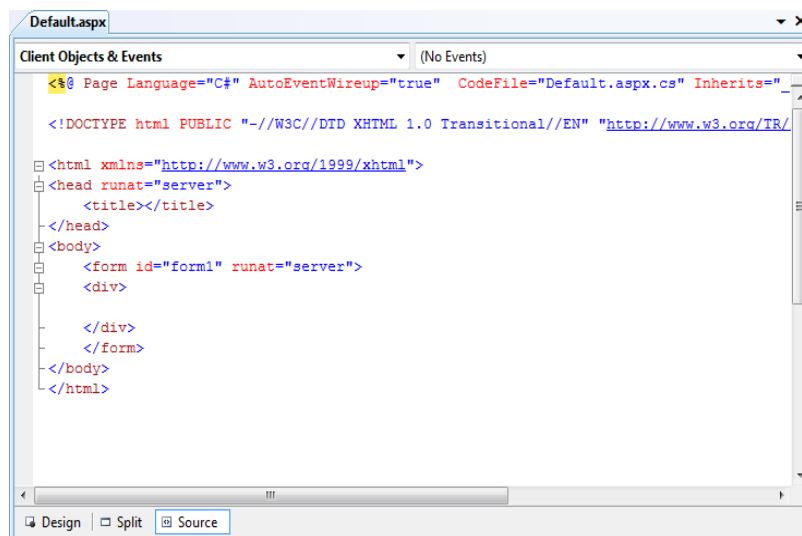
Neméně důležitou částí je také prohlížeč řešení(Solution explorer) v pravé části obrazovky, jenž Vám zobrazuje obsah vašeho řešení a projektů.



Každý objekt který vložíte do stránky má i své vlastnosti(Properties) a ty se dají upravovat pomocí nástroje v pravé dolní části okna.



Poslední, ale nejdůležitější je editor kódu v prostřední části. Zde měníte obsah všech souborů. Což bude i naším prvním úkolem.



HTML kód stránky upravíme tak aby se nám po spuštění zobrazilo okno s textem „Hello, world!“. Kód bude tedy vypadat takto.

```
<%@ 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></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      Hello, world!
    </div>
  </form>
</body>
</html>
```

Tento kód spustíme pomocí tlačítka run v horní části lišty jak jsme si ukázali dříve. Spustí se prohlížeč a v něm náš text.

## 1.2.4 Některé základní značky HTML

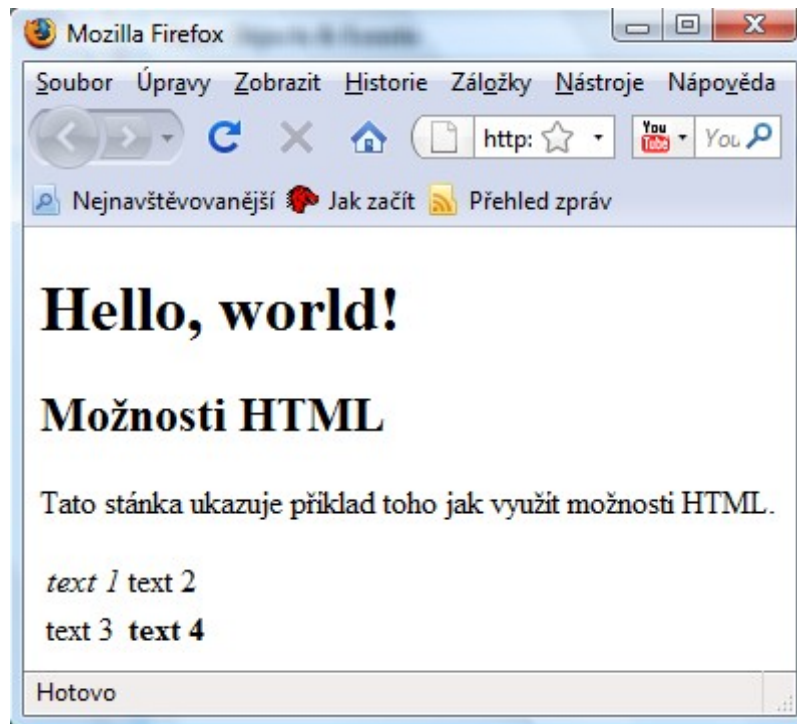
Jelikož jde o webové stránky předpokládá se že zvládáte alespoň základy HTML. Pro případ že si některé nevybavíte jsem zde uvedl pár základních.

<code>&lt;i&gt;text&lt;/i&gt;</code>	Vypíše text kurzivou
<code>&lt;b&gt;text&lt;/b&gt;</code>	Tučně
<code>&lt;p&gt;text&lt;/p&gt;</code>	Odstavec
<code>&lt;h1&gt;text&lt;/h1&gt;</code>	Nadpis (číslo určuje úroveň 1,2,...)
<code>&lt;a&gt;text&lt;/a&gt;</code>	Odkaz
<code>&lt;table&gt;&lt;/table&gt;</code>	Tabulka (obsahuje řádky)
<code>&lt;tr&gt;&lt;/tr&gt;</code>	Řádek tabulky (obsahuje data)
<code>&lt;td&gt;text&lt;/td&gt;</code>	Data tabulky (jednotlivé sloupce)
<code>&lt;br /&gt;</code>	Další řádek
<code>&lt;img /&gt;</code>	Obrázek
<code>&lt;div&gt;&lt;/div&gt;</code>	Blok stránky
<code>&lt;form&gt;&lt;/form&gt;</code>	Formulář

Značek je mnohem více, ale my se prozatím spokojíme s těmito. Ukážeme si malý příklad jak je použít.

```
<%@ 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></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <h1>Hello, world!</h1>
    <h2>Možnosti HTML</h2>
    <p>Tato stránka ukazuje příklad toho jak využít možnosti HTML.</p>
    <table>
      <tr>
        <td><i>text 1</i></td>
        <td>text 2</td>
      </tr>
      <tr>
        <td>text 3</td>
        <td><b>text 4</b></td>
      </tr>
    </table>
  </div>
</form>
</body>
</html>
```



## 2. Začínáme programovat

Zatím jsme se naučili jak vytvořit stránku, která nedokáže nijak dynamicky měnit svůj obsah a reagovat na požadavky uživatele. Tomu bude však po této kapitole konec.

### 2.1 Dynamický web

Každá dynamická stránka by se měla měnit podle funkce kterou vykonává. Nejjednodušším příkladem je například webová kalkulačka. Uživatel zadá čísla a vybere operaci, která se má provést. Aby bylo však možné tuto funkci zprovoznit, musíme ji připravit logiku. Tou logikou bude kód který běží na pozadí stránky. Takovému kódu se říká “code behind”.

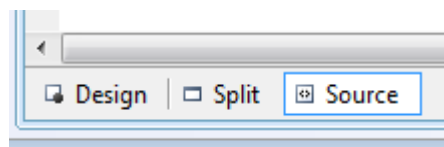
V ASP.NET má každá vytvořená stránka svůj kód který se stará o funkce stránky. V něm můžeme například reagovat na události(Eventy) které vyvolal uživatel kliknutím na tlačítko. V další části této kapitoly se naučíme používat některé komponenty(Controly) a na příkladech si ukážeme jak je můžeme propojit s naším kódem.

#### 2.1.1 Přidáváme komponenty do stránky

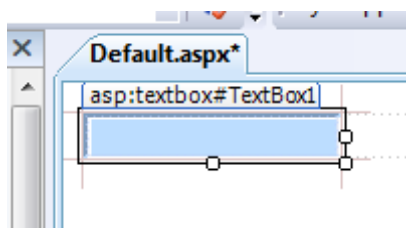
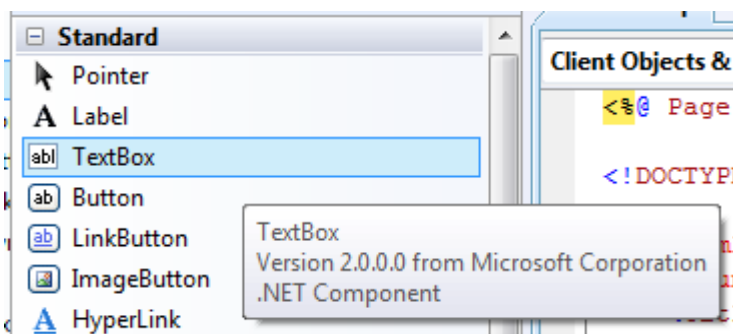
K přidávání komponent do naší stránky se využívá toolboxu v levé části obrazovky. Pro začátek budeme používat jenom základní sadu komponent(Standard). Všimněte si tlačítek v levé spodní části editačního okna. Jsou určeny k přepínání mezi psaním kódu a WISIWIG editorem.

Source = psaní kódu

Design = WISIWIG

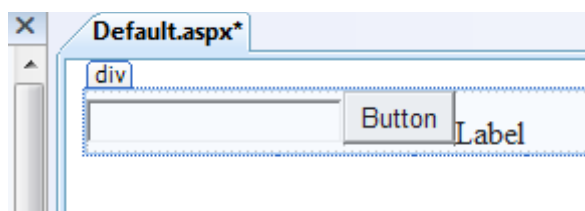


Přepněte si tedy na “Design” a zobrazí se Vám místo textu bílé okno které reprezentuje vzhled stránky (pokud již máte ve stránce nějaký obsah, zobrazí se). Dále si vyberte z toolboxu komponentu TextBox a tahem myši ji přesuňte na plochu stránky.

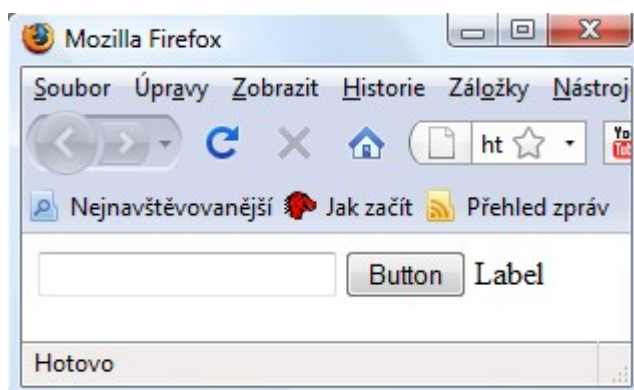


Jistě si všimnete okna Properties(Vlastnosti) v pravé dolní části obrazovky. Její obsah se změnil. Jsou v něm teď vlastnosti komponenty TextBox. Můžete si je po růzku zkoušet měnit a sledovat změny které se projevují okamžitě.

Stejně tak přidejte za “TextBox” i “Button” a nakonec “Label”.



A k čemu jsou tyto komponenty užitečné? TextBox umožní uživateli zadat text který si poté můžeme jako programátoři z něj vybrat. Button je tlačítko. Po jeho stusnutí uživatelem se stránka(Formulář) odešle ke zpracování na server. A label slouží k zobrazování textových dat. Takto vytvořenou stránku si můžeme spustit a podívat se na výsledek v prohlížeči.

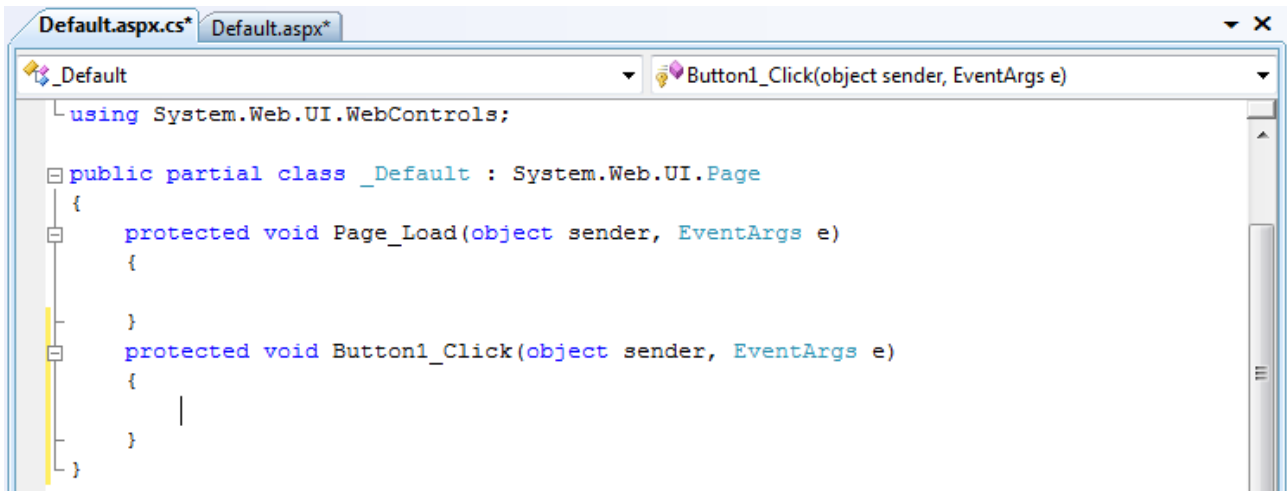


Můžete si zkusit stránku upravit podle svých představ a přidat si další komponenty, které vás v toolboxu zaujaly a zkusit si měnit jejich vlastnosti. Stává se však že se zavřením prohlížeče Vám zmizí toolbox. Je to dané tím že nemůžete editovat stránku kterou máte spuštěnou na serveru. Stačí však stisknout tlačítko “Stop” v menu vpravo nahore. To ukončí běh aplikace.



## 2.1.2 Dáváme stránce život

Máme připravenou stránku která zatím neplní žádnou funkci. Můžeme ji však vdechnout život tím že přiřadíme obsluhu události vyvolané tlačítkem(Co se stane po stisku tlačítka). Dvojklikem na tlačítko se nám zobrazí kód v C# kde se připraví metoda, která obsluhuje událost tlačítka "Click".



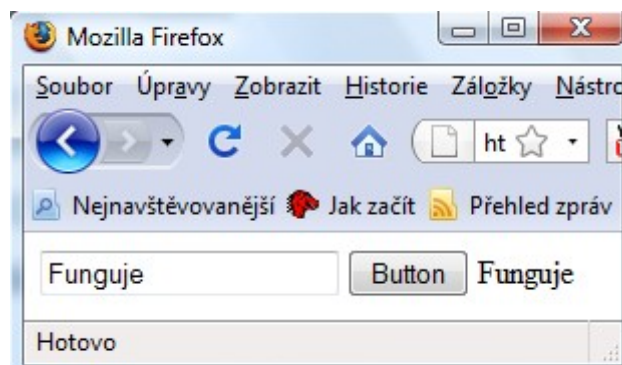
Do této metody napíšeme kód který po stisknutí tlačítka vybere z komponenty TextBox text a vloží jej do Label. Což vypadá asi takto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

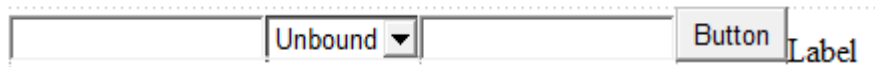
    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Label1.Text = this.TextBox1.Text;
    }
}
```

Doslova přiřadíme vlastnosti Text z komponenty Label text z komponenty TextBox. Takto napsaný kód můžete okamžitě vyzkoušet.

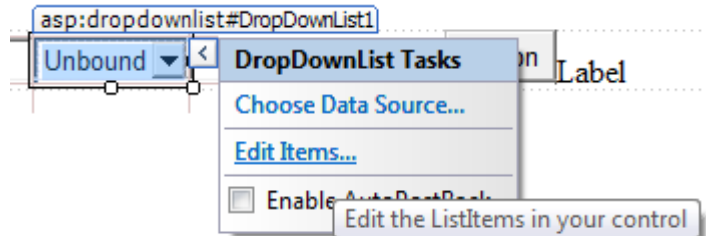


### 2.1.3 Příklad Kalkulačka

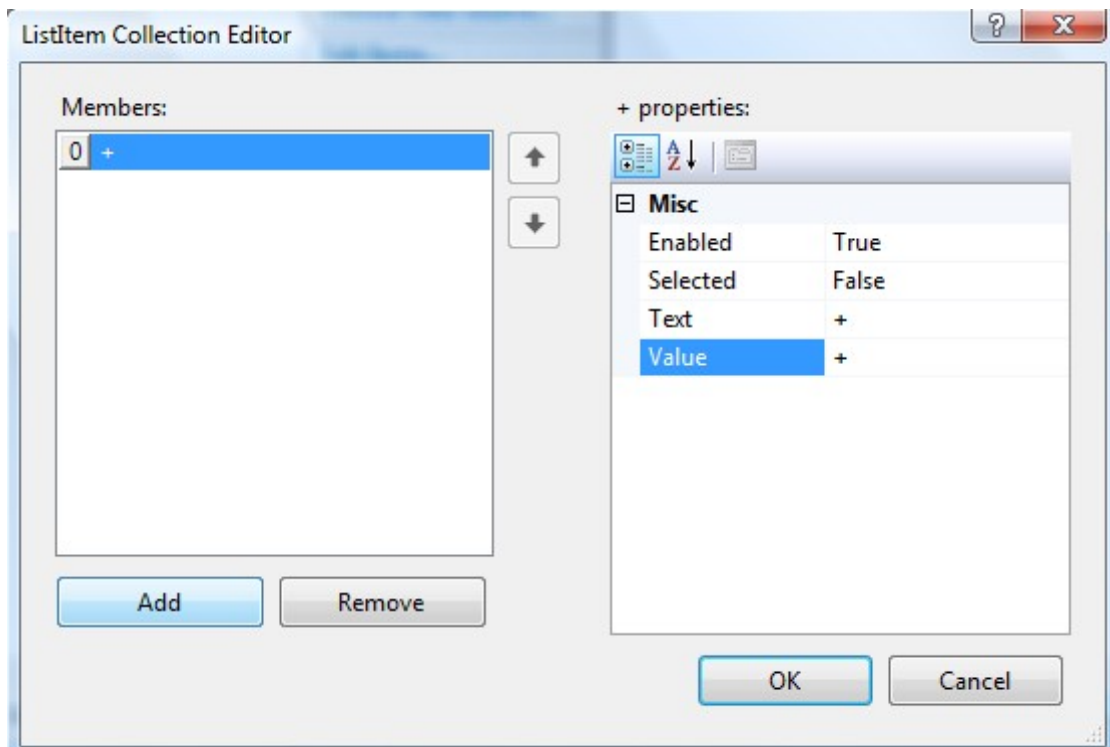
Nyní již známe vše pro to, abychom mohli zkusit již dříve zmiňovaný příklad kalkulačka. Využijeme při tom komponent TextBox, DropDownList, Button a Label. Tyto komponenty vložíme do stránky stejně jako na obrázku.



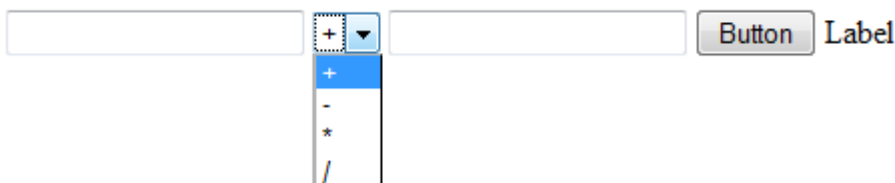
Teď přiřadíme DropDownListu hodnoty. To provedem najetím myši na něj a stisknutím šipky která se objeví na jeho pravém horním okraji.



Dále stiskneme Edit Items... což nám zobrazí dialogové okno ve kterém můžeme přidávat položky.



Stisknutím tlačítka Add se nám přidá položka u níž změníme Text a Value na "+". Pokračuje přidáním dalších operací jako "-", "\*", "/". Pote stiskneme "OK". Na stránce se pak komponenta chová jako výběr operace.



Máme tedy připraven vzhled stránky a můžeme pokračovat psáním kódu. Opět dvojklikem na tlačítko vygenerujeme metodu do které napíšeme náš kód.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

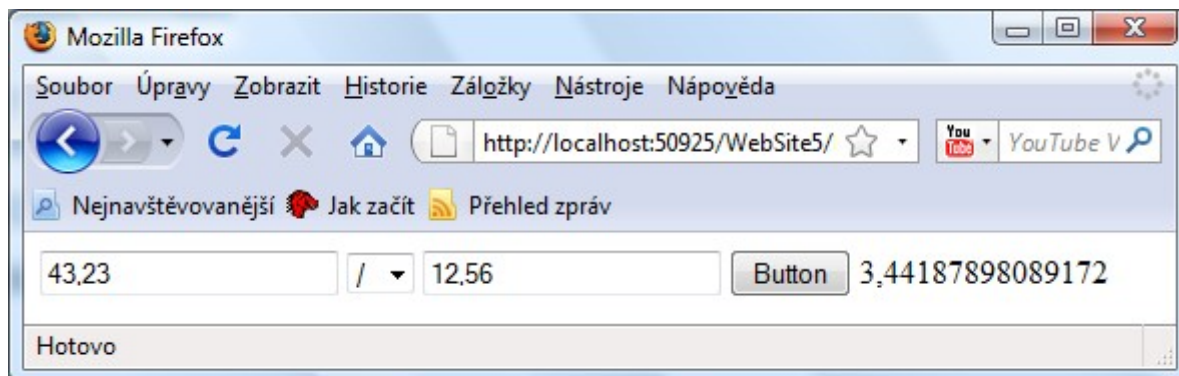
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        double a, b;
        string result;
        a = Convert.ToDouble(this.TextBox1.Text);
        b = Convert.ToDouble(this.TextBox2.Text);
        string operace = this.DropDownList1.Text;
        switch (operace)
        {
            case "+": result = (a + b).ToString(); break;
            case "-": result = (a - b).ToString(); break;
            case "*": result = (a * b).ToString(); break;
            case "/": result = (a / b).ToString(); break;
            default: result = ""; break;
        }
        this.Label1.Text = result;
    }
}

```

Tento kód převezme čísla z TextBoxů a převede je z textu na desetinná čísla. Dále zjistí z DropDownListu jakou operaci uživatel zvolil a po vypočtení dané hodnoty jej zapíše do Labelu.



Tento kód není zcela dokonalý, ale můžete je rozšířit například o kontrolu dělení nulou, nebo můžete přidat další operace jako třeba mocninu “^”, Zjišťovat rovnost, nerovnost, menší větší a podobně. Pokud by jste snad chtěli zkrášlit kalkulačku obrázkem, můžete tak učinit jednoduchým přetáhnutím obrázku z vaší složky přímo na stránku v Design módu.

## 3. Validace vstupů

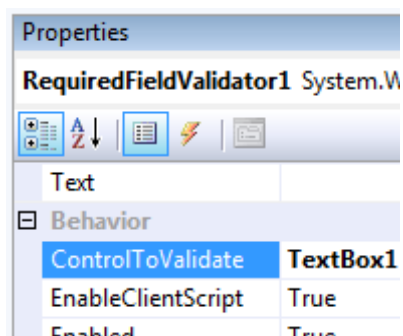
V ASP.NET jak jste již jistě pochopili je celá stránka formulář a každý formulář se musí zkontrolovat zda je vyplněný správně. Špatný vstup může znamenat vyvolání chyby, která normálního uživatele může zmást.

Ke kontrole vstupů nám slouží validační komponenty, které se vážou na jiné komponenty které mají kontrolovat. Mezy ty které nám nabízí VS jsou tyto:

- RequiredFieldValidator      Kontrola neprázdného vstupu
- RangeValidator              Kontrola rozsahu
- RegularExpressionValidator    Kontrola regulárním výrazem
- CompareValidator            Porovnávací kontrola
- CustomValidator            Nastavitelný(Vlastní) Kontrola

### 3.1 Nastavení validace komponenty

Každá komponenta kterou chceme kontrolovat musí být přiřazena danému validátoru pomocí vlastnosti "ControlToValidate", kterou najdete v okně "Properties" v prave spodní části obrazovky. A před zpracováním dat z formuláře se musí zkontrolovat validnost zavoláním metody validátoru IsValid která v případě správného vstupu "true" a v opačném "false". Projdeme si tedy nastavení jednotlivých validátorů.



#### 3.1.1 RequiredFieldValidator

Toto je nejjednodušší validátor. Stačí nastavit komponentu kterou má kontrolovat a on už sám zjišťuje zda je dané pole vyplněno nebo ne. Reakce na stisk tlačítka vypadá takto:

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (RequiredFieldValidator1.IsValid)
    {
        // toto provede pokud je validni
    }
    else
    {
        // toto pokud neni validni
    }
}
```

Toto pole je povinne

### 3.1.2 RangeValidator

Tento validátor slouží ke kontrole rozsahu. Můžeme v něm nastavovat jakého typu se má rozsah kontrolovat pomocí vlastnosti “Type”. Nejčastější zřejmě bude typ Integer, kdy se kontroluje v jakém rozsahu je číslo. Například od 0 do 100. To nastavíme pomocí vlastností “MaximumValue” a “MinimumValue”.

MaximumValue	100
MinimumValue	0
SetFocusOnError	False
SkinID	
ToolTip	
Type	Integer

### 3.1.3 RegularExpressionValidator

Toto je komponenta pomocí které můžete vstup kontrolovat regulárním výrazem. Nastavení regulárního výrazu se provádí pomocí vlastnosti “ValidationExpression” do které přímo výraz zapíšete. Například výraz ([0-9a-zA-Z])+ dovoluje napsat jen malé a velké písmena a čísla.

ToolTip	
ValidationExpression	([0-9a-zA-Z])+ ...
ValidationGroup	

### 3.1.4 CompareValidator

Tento validátor je určený k porovnávání dvou hodnot. Na rozdíl od předchozích vyžaduje zadání další komponenty porovnávací a to nastavením vlastnosti “ControlToCompare”. Opět můžeme nastavit typ a navíc i funkci kterou má plnit vlastností Operátor, například “equal” (Ekvivalentní). Nejčastěji je asi aplikován v registračním formuláři, kde se zjišťuje zda uživatel zadal obě hesla stejně.

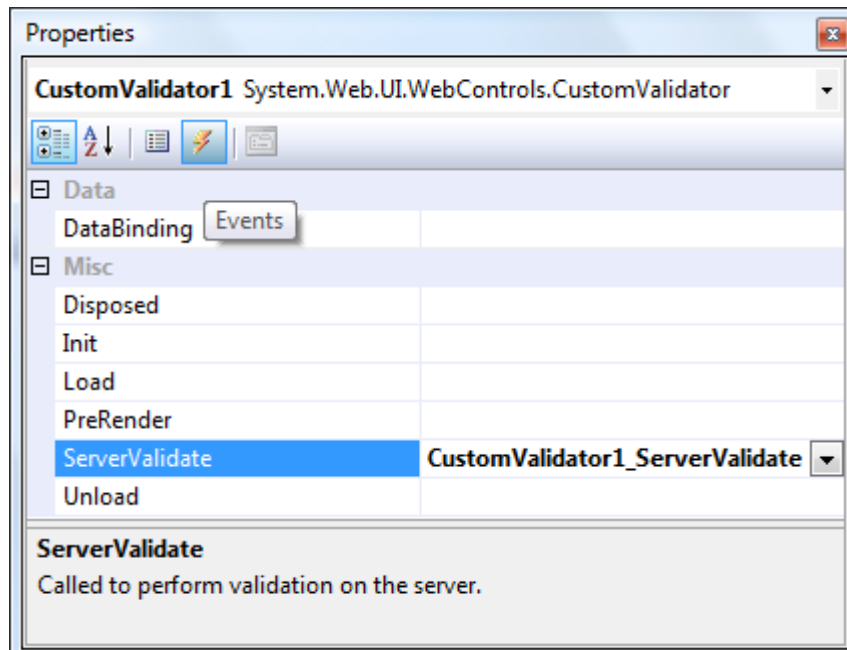
Behavior	
ControlToCompare	TextBox5
ControlToValidate	TextBox4
CultureInvariantValues	False
EnableClientScript	True
Enabled	True
EnableTheming	True
ableViewState	True
Operator	Equal ▼
SetFocusOnError	False
SkinID	
ToolTip	
Type	String

### 3.1.4 CustomValidator

Všechny předchozí validátory jsou orientovány na nejběžnější aplikace. Co ale dělat v případě že chceme po uživateli aby zadal jenom čísla dělitelná třemi. Pro takové případy je tu CustomValidátor kde si validační funkci píšeme sami.

Prvním krokem je nastavení události “ServerValidate” kterou najdeme pod kartou “events”

v okně “Properties” v pravé spodní části obrazovky.



Můžeme si nechat vygenerovat funkci dvojitým kliknutím do zadávacího pole. V té pak vstup kontrolujeme přímo sami.

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
    int hodnota = Convert.ToInt32(args.Value);
    if (hodnota % 3 == 0)
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
```

Hodnotu získáme pomocí argumentu “args” přes vlastnost “Value” tu si převedeme na celá čísla a zkontrolujeme zda je hodnota dělitelná třemi nebo ne. Výsledek pak předáme opět do “args” na vlastnost “IsValid”.

Pokud bychom chtěli provádět validaci na straně klienta musíme nastavit vlastnost validatoru “ClientValidationFunction” na název funkce v JavaScriptu a javascriptový kód přidat přímo na stránku kde vstup ověřujeme.

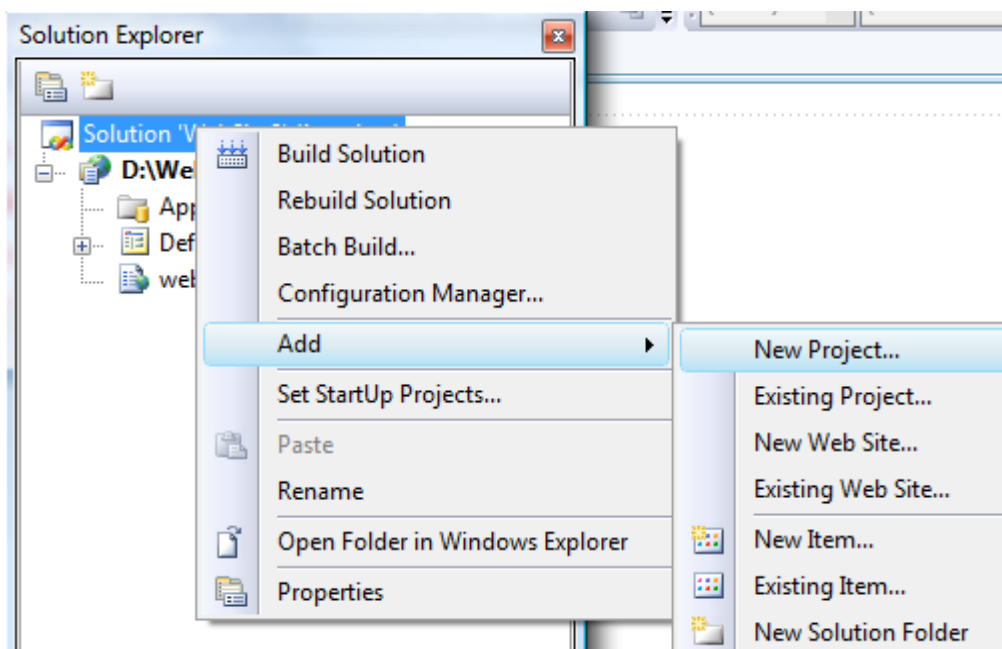
```
<script language="javascript">
    function ClientValidate(source, args) {
        if (args.Value % 3 == 0)
            args.IsValid = true;
        else
            args.IsValid = false;
    }
</script>
```

## 4. Dynamické knihovny

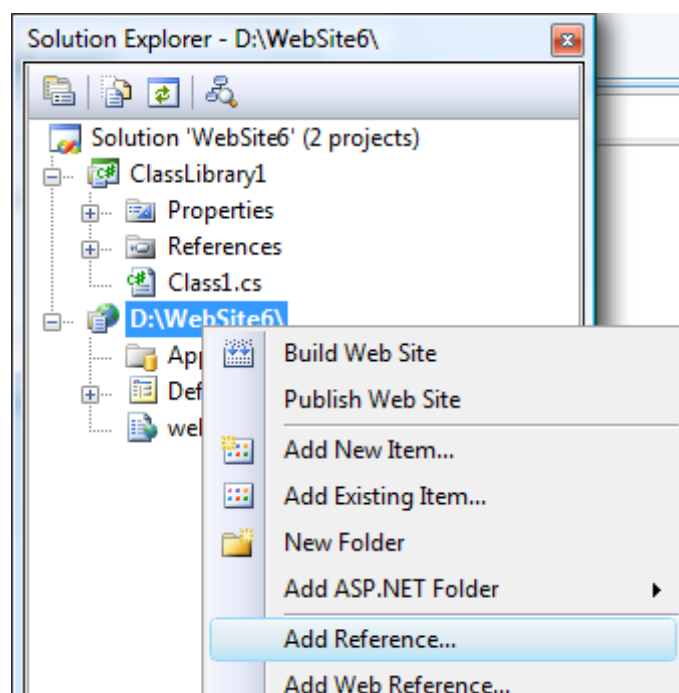
Samotný projekt velice rychle roste a programovat veškerou logiku do jednoho by se brzy mohlo vymstít. Lepším řešením je rozdělit jej na logické části. VS našťastí nabízí toto dělení a aplikace jako celek označuje jako “Solution”(Řešení), které obsahuje několik vzájemně propojených projektů.

## 4.1 Vytvoření nové třídní knihovny

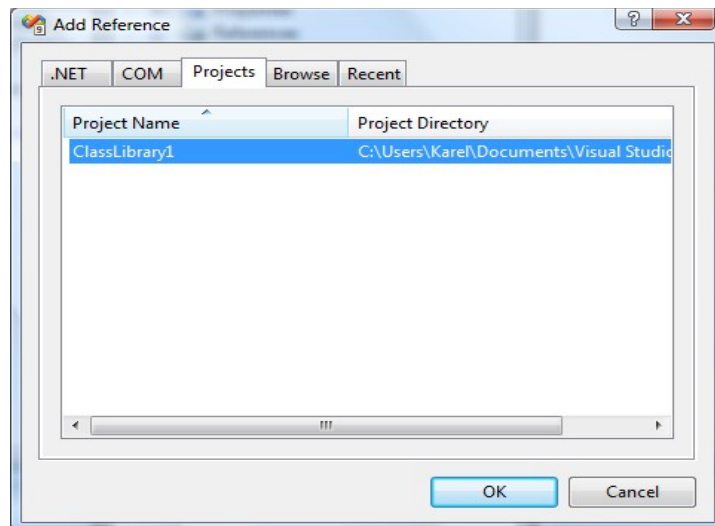
Začneme tím, že klikneme pravým tlačítkem myši na naše řešení(Solution) v Solution exploreru v pravé horní části obrazovky a přidáme nový projekt (Add=>New Project...).



Zobrazí se dialogové okno ve kterém vybereme “Class Library” nastavíme jméno a stiskneme “OK”. Abychom měli k třídám nové knihovny přístup musíme přidat do našeho projektu na ní referenci(odkaz).



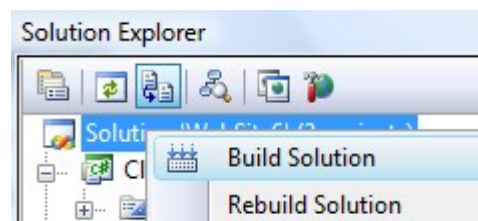
V dalším dialogovém okně vybereme katalog Projekty(Projects) a přidáme naši knihovnu. Po tomto kroce bychom měli ještě celé řešení sestavit(Build) aby se vytvořily vazby.



Poté se již můžeme pustit do psaní kódu naší knihovny. Pro ukázkou můžeme ve třídě která nám byla vytvořena ze šablony vytvořit veřejnou statickou metodu která vrací řetězec “Knihovna”.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassLibrary1
{
    public class Class1
    {
        public static string getText()
        {
            return "Knihovna";
        }
    }
}
```

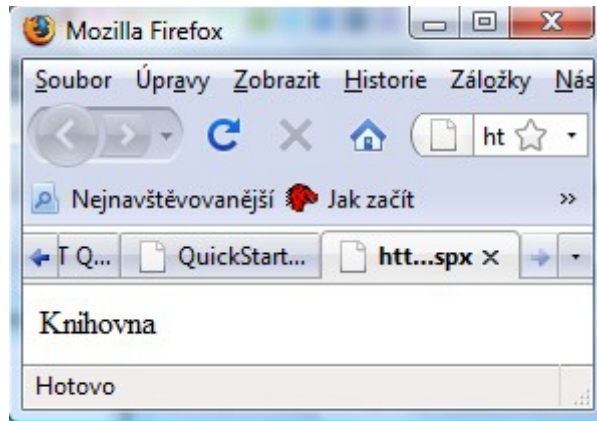


tento řetězec vložíme do Labelu hned při načtení stránky. To znamená že se přepneme do kódu stránky a v předgenerované metodě Page\_Load to provedeme.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = ClassLibrary1.Class1.getText();
    }
}
```

```
}
```



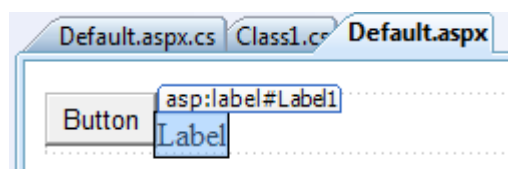
### 4.1.1 Příklad třídní knihovny na náhodná čísla

Využijeme stejného postupu jako dříve, s tím rozdílem, že vytvoříme metodu která bude vracet generický seznam celých čísel.

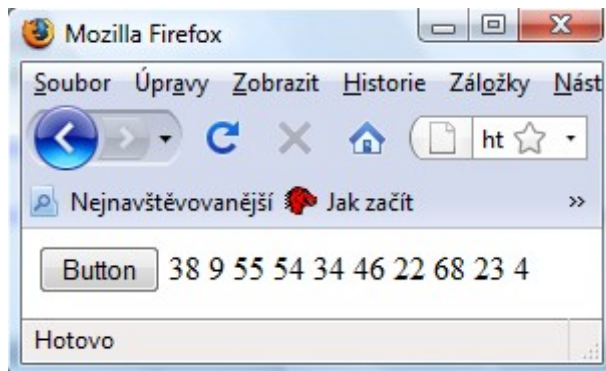
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassLibrary1
{
    public class Class1
    {
        public static List<int> getRandom()
        {
            List<int> list = new List<int>();
            Random rand = new Random();
            for (int i = 0; i < 10; i++)
            {
                list.Add(rand.Next(100));
            }
            return list;
        }
    }
}
```

Pro zobrazení na stránce využijeme komponenty Label a komponentu Button jako tlačítko spouštějící generování náhodných čísel.



```
protected void Button1_Click(object sender, EventArgs e)
{
    List<int> list = ClassLibrary1.Class1.getRandom();
    Label1.Text = "";
    foreach (int i in list)
    {
        Label1.Text += " " + i;
    }
}
```



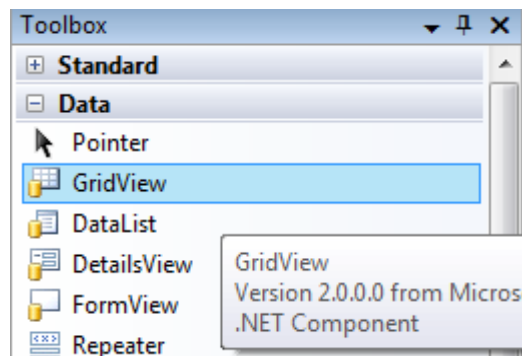
## 5. Vybrané komponenty pro prezentaci dat

Většina dat se kterými se v praxi setkáváme je strukturovaná a není vhodné je prezentovat nestrukturalizovaně. K zobrazení dat je vhodné použít například tabulky. Přístup k datovým komponentám přístupných ve VS je jednotný. K přidělení datového zdroje se používá vlastnost DataSource a k projevení změn se používá metoda DataBind. Komponenty na které se podíváme podrobněji jsou:

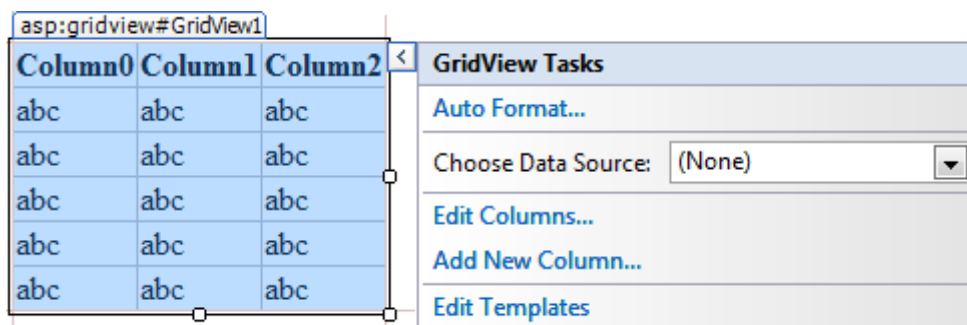
- GridView zobrazuje data v tabulce
- DetailsView používá se k zobrazení jednotlivých záznamu(detailů) z tabulky
- Repeater zobrazuje data podle šablony kterou si definujete

### 5.1 GridView

Pokud potřebujeme data zobrazit v tabulce je GridView nejlepší komponentou. První vložíme komponentu z toolboxu ze záložky data.

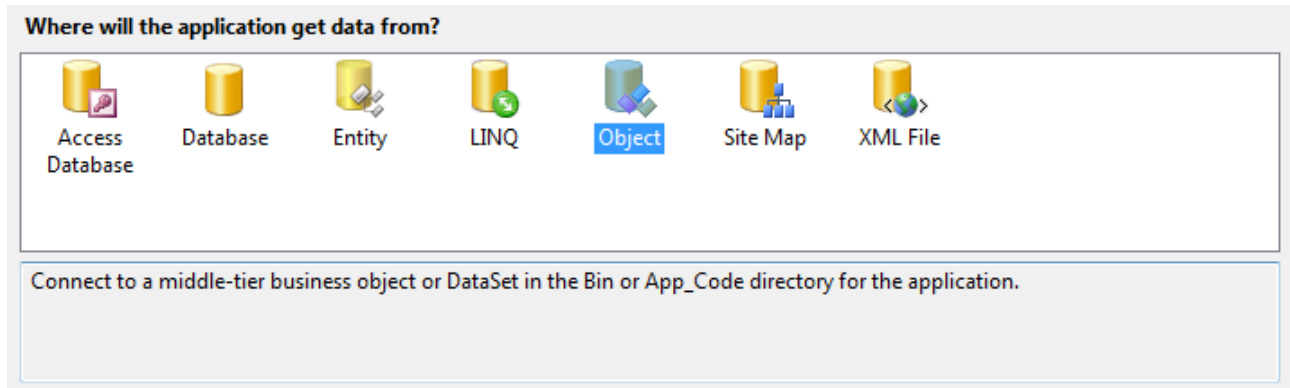


Po vložení se nám zobrazí základní vzhled tabulky a nabídka ve které můžeme upravit vzhled, obsah a datový zdroj.

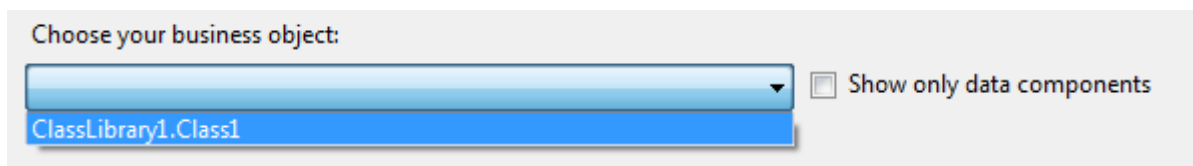


Pro začátek se postaráme o přidělení datového zdroje(Choose Data Source) a využijeme k tomu třídu, kterou jsme v předchozím příkladu použili ke generování náhodných čísel. Vybereme volbu <New data source...>.

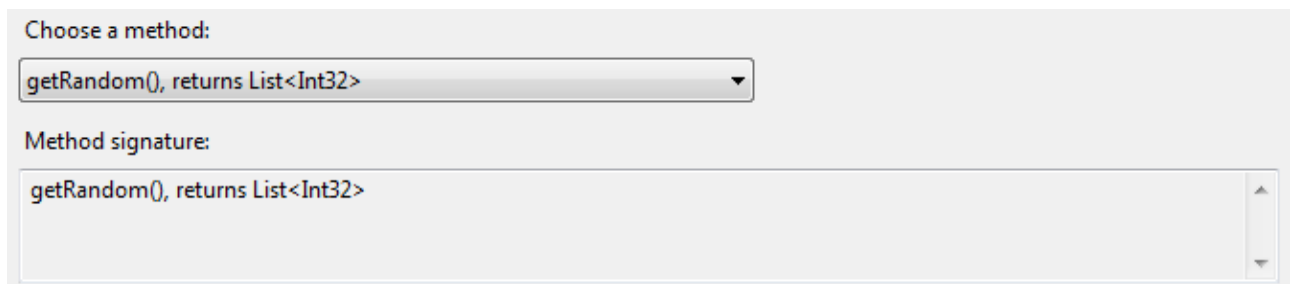
Zobrazí se nám průvodce, který nás provede vytvořením nového datového zdroje. Kde máme možnost vybrat si z několika možností. Pro nás je teď zajímavá pouze volba Object, která získává data z třídních metod.



Stiskneme “OK” a v dalším kroku vybereme třídu ze které budeme data získávat. Nesmíme zapomenout na přidělení reference knihovny do našeho projektu a následnou kompilaci, jinak se nám třída ve volbě nezobrazí.



Další krok je výběr metody, která vrací data. Pokud jste napsali novou metodu, zobrazí se opět až po zkompilování.



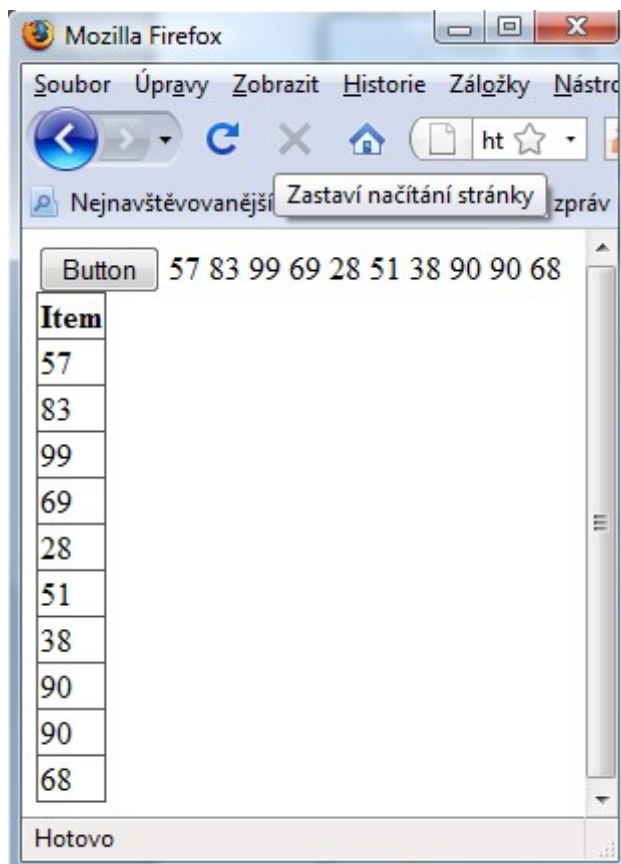
Po tomto kroku se vytvoří nový objekt ”ObjectDataSource” který reprezentuje připojení komponenty GridView k metodě v naší knihovny.

Naším dalším krokem bude přepsání kódu stránky tak, aby se po každém jejím načtení generovala nová čísla. Napíšeme tedy do metody Page\_Load tento kód:

```
protected void Page_Load(object sender, EventArgs e)
{
    ObjectDataSource2.Select();
    GridView1.DataBind();
}
```

V metodě první zavoláme Select což způsobí načtení dat s pomocí metody a následně komponentě GridView řekneme aby si načetla nová data. Výsledkem je stránka s tabulkou

vyplněnou náhodnými čísly.



### 5.1.1 [Serializable] třída

Předpokládáme-li strukturu pod jakou se má třída ukládat (Serializovat) můžeme před její definicí přidat přepínač [Serializable].

Řekněme, že chceme ukládat osoby s atributy {jméno, příjmení}. Vytvoríme tedy knihovnu která bude obsahovat třídu s metodou Select, která vrací záznamy typu vnitřní serializovatelné třídy Osoba.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassLibrary1
{
    public class Class1
    {
        [Serializable]
        public class Osoba
        {
            public string jmeno { set; get; }
            public string prijmeni { set; get; }
        }

        public static List<Osoba> Select()
        {
            List<Osoba> list = new List<Osoba>();
            Random rand = new Random();
            Osoba o = new Osoba();
```

```

o.jmeno = "Karel";
o.prijmeni = "Mozdren";
list.Add(o);
Osoba o2 = new Osoba();
o2.jmeno = "Pepa";
o2.prijmeni = "Skocdopole";
list.Add(o2);
return list;
}
}
}

```

Pokud toto použijeme jako datový zdroj komponenty GridView, tak si ObjectDataSource sám zajistí rozdělení dat do jednotlivých sloupců tabulky.

<b>jmeno</b>	<b>prijmeni</b>
Karel	Mozdren
Pepa	Skocdopole

## 5.2 DetailsView

DetailsView ukazuje detaily záznamů. Jde o výpis jednoho záznamu se svými položkami pod sebou. Vyzkoušet si jej můžete prostým přidáním datového zdroje z předchozího příkladu. Doporučuji pak v tom případě zapnout volbu “Enable Paging”, která nam umožní přepínat mezi jednotlivými záznamy.

<b>jmeno</b>	<b>prijmeni</b>
Karel	Mozdren
Pepa	Skocdopole

<b>jmeno</b>	Pepa
<b>prijmeni</b>	Skocdopole
<u>1</u> 2	

## 5.3 Repeater

Pokud chceme mít naprostou kontrolu nad vzhledem našich dat, použijeme komponentu Repeater. K zobrazení dat je třeba připravit šablonu, která určuje vzhled.

Pro editaci šablony je třeba se přepnout z režimu “Design” do “Source”. K vypsání dat se použije DataBinder s metodou eval do které se jako parametry zapíšou názvy sloupců a Kontejner s daty.

```

<asp:Repeater ID="Repeater1" runat="server" DataSourceID="ObjectDataSource1">
<ItemTemplate>
  <i><%=# DataBinder.Eval(Container.DataItem, "jmeno") %></i>
  <b><%=# DataBinder.Eval(Container.DataItem, "prijmeni") %></b><br />
</ItemTemplate>
</asp:Repeater>

```

*Karel Mozdren*  
**Pepa Skocdopole**

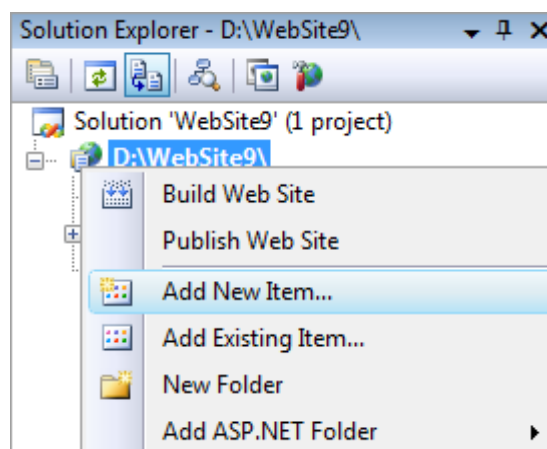
Znalosti o těchto komponentách si rozšíříme v dalších kapitolách podle potřeby.

## 6. Databáze

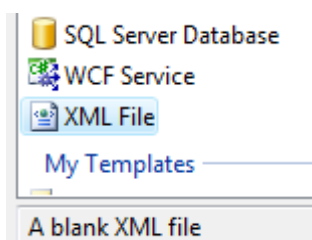
Pro trvalejší uložení dat budeme používat externí datové úložiště. Máme několik možností. Mezi nejčastěji používané patří používání Databázových serverů jako je MSSQL server a nebo ukládání do souborů formátu XML.

### 6.1 XML

Soubory XML jsou strukturované a proto se dobře čtou člověku i počítači. Předpokládáme že se skládají ze značek(tagů), parametrů a dat. Dobře strukturovaný soubor má počáteční a konečný tag. Případně není párový a je tedy zakončen lomítkem. Parametry se zapisují přímo do počátečního tagu ve tvaru název="hodnota". Protože má být XML univerzální úložiště dat, je potřeba definovat tvar ve kterém se data do souboru ukládají. K tomu nám slouží doprovodné soubory DTD ne XSL(Schema). Podrobnější informace můžete najít na internetu. Podíváme se tedy nejprve na ukázkou.



Do našeho projektu si přidáme nový objekt stisknutím pravého tlačítka myši na něj. A v levé spodní části otevřeného dialogového okna nalezneme XML File.

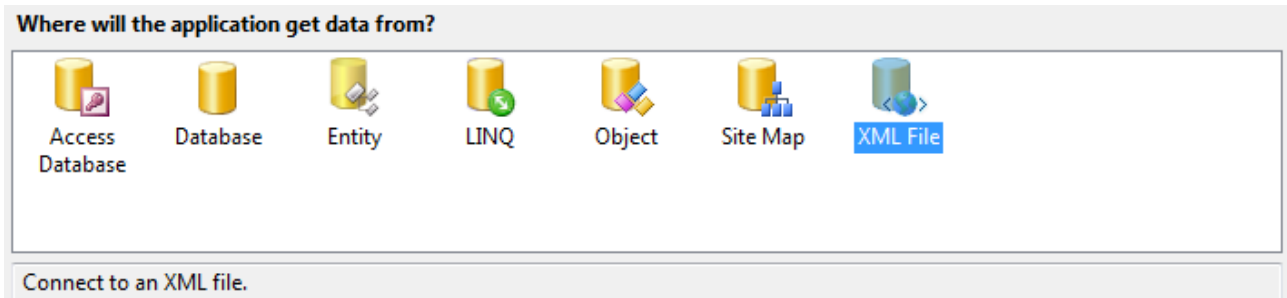


Přeskočíme část tvorby XML Scheme, nebo DTD souboru, protože si ji můžete zkusit sami zvašť. Na internetu jistě naleznete mnoho příkladů. Zajímavější jistě bude zápis dat do XML.

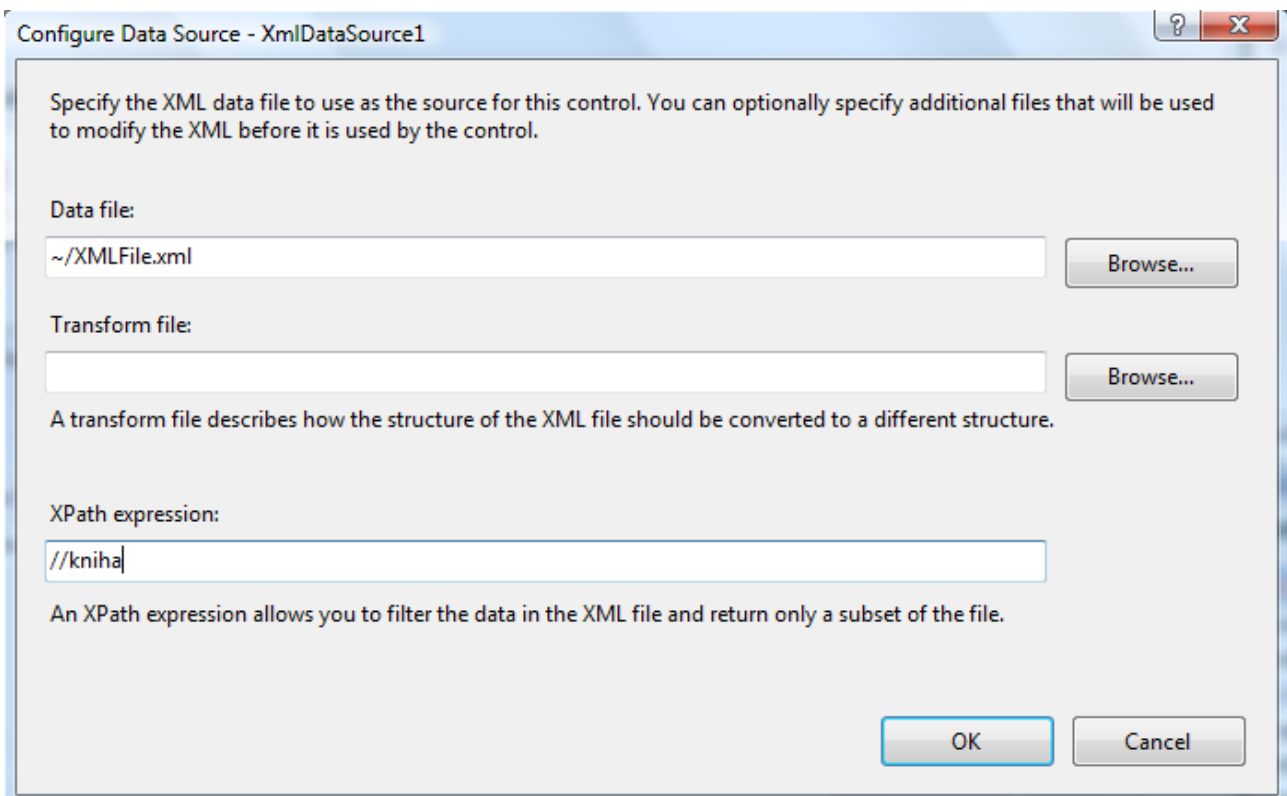
```
<?xml version="1.0" encoding="utf-8" ?>
<knihovna>
  <kniha vytisku="10000">
    <nazev>C#</nazev>
    <autor>Pepa Skocdopole</autor>
  </kniha>
  <kniha vytisku="20000">
    <nazev>Uvod do ASP.NET</nazev>
    <autor>Karel Mozdren</autor>
  </kniha>
</knihovna>
```

Zde můžeme vidět obsah XML souboru kde jsou zapsány dvě utorké knížky a údaje jako počet výtisků, název a jméno autora. Jakmile jsme tento soubor vytvořili, ukážeme si nejjednodušší způsob jak zobrazit údaje na naší stránce.

Přidáme si komponentu Repeater a nastavíme si datový zdroj XMLDataSource.



V dalším kroku zadáme cestu k našemu XML souboru do textového pole Data file a zadáme příkaz jazyka XPath pro výběr dat z XML souboru.



Jakmile jsme toto provedli, musíme ještě připravit šablonu pro zobrazení. V tuto chvíli nám nebude fungovat metoda Eval z dataBinderu, ale můžeme použít funkci XPath().

```
<asp:Repeater ID="Repeater1" runat="server" DataSourceID="XmlDataSource1">
  <ItemTemplate>
    Kniha: <b> <%# XPath("nazev") %></b>,
    <i><%# XPath("autor") %></i><br />
  </ItemTemplate>
</asp:Repeater>
```

**Kniha: C#, Pepa Skocdopole**

**Kniha: Uvod do ASP.NET, Karel Mozdren**

Tento způsob je velice jednoduchý, avšak velice nepraktický v další části si ukážeme složitější, prakticky mnohem lépe použitelnou metodu čtení, ukládání, změnu a mazání záznamů z XML souboru.

## 6.1.2 Implementace přístupu k datům v XML

Teď si ukážeme jak se dají získávat a vkládat data do XML souboru. Na Ostatní operace, jako úpravy a mazání již jistě přijdete sami.

V první řadě si vytvoříme třídní knihovnu, s třídou “Knihy” a metodami INSERT a SELECT. V této třídě taktéž založíme vnitřní třídu “Kniha” s veřejnými udaji “Název a autor”.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Data;

namespace Knihovna
{
    public class Knihy
    {
        [Serializable]
        public class Kniha
        {
            public string nazev { set; get; }
            public string autor { set; get; }
        }

        public static List<Kniha> Select()
        {
            List<Kniha> list = new List<Kniha>();
            return list;
        }

        public static void Insert(Kniha k)
        {
        }
    }
}
```

První sepíšeme funkci Select, která nám vrátí seznam knih. Ke čtení využijeme objekt XmlDocument to znamená že do části using budeme muset přidat položku ”using System.Xml;”.

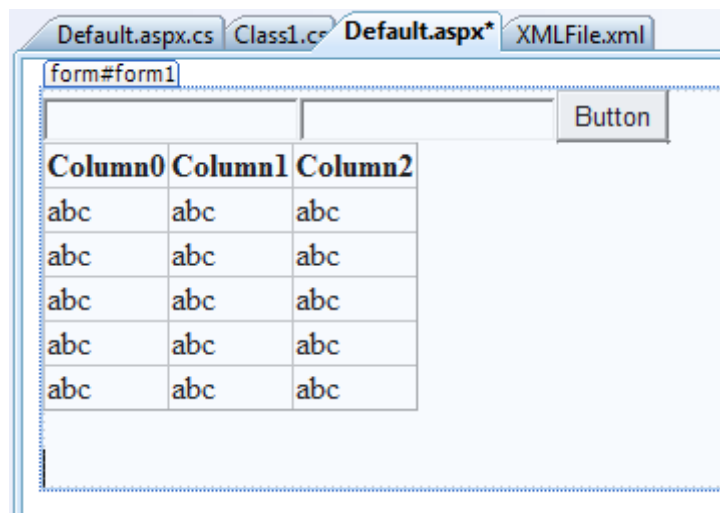
```
public static List<Kniha> Select()
{
    List<Kniha> list = new List<Kniha>();
    XmlDocument doc = new XmlDocument();
    doc.Load("D:\\WebSite9\\XMLFile.xml");
    XmlNode knihovna = doc.SelectSingleNode("/knihovna");
    foreach (XmlNode node in knihovna.ChildNodes)
    {
        XmlNode nazev = node.SelectSingleNode("nazev");
        XmlNode autor = node.SelectSingleNode("autor");
        Kniha k = new Kniha();
        k.nazev = nazev.InnerText;
        k.autor = autor.InnerText;
        list.Add(k);
    }
    return list;
}
```

Tato metoda otevře XML dokument na disku, najde uzel Knihovna a pro všechny její potomky zjistí hodnoty “nazev” a “autor” které zapíše do seznamu knih jenž posléze předá.

V předchozí metodě jsme strom procházeli, teď jej musíme vytvořit. Metoda Insert vytvoří uzel knihovna a do ní přidává knihy a v nich jsou uloženy elementy název a autor. Po vytvoření celého stromu se uloží data do souboru XML.

```
public static void Insert(Kniha k)
{
    List<Kniha> list = Select();
    list.Add(k);
    XmlDocument doc = new XmlDocument();
    XmlNode knihovna = doc.CreateNode(XmlNodeType.Element, "knihovna", null);
    foreach (Kniha kniha in list) {
        XmlNode nova = doc.CreateNode(XmlNodeType.Element, "kniha", null);
        XmlNode nazev = doc.CreateNode(XmlNodeType.Element, "nazev", null);
        nazev.InnerText = kniha.nazev;
        XmlNode autor = doc.CreateNode(XmlNodeType.Element, "autor", null);
        autor.InnerText = kniha.autor;
        nova.AppendChild(nazev);
        nova.AppendChild(autor);
        knihovna.AppendChild(nova);
    }
    doc.AppendChild(knihovna);
    doc.Save("D:\\WebSite9\\XMLFile.xml");
}
```

Teď si odzkoušíme funkčnost. Vytvoříme stránku na kterou vložíme GridView, 2x TextBox a jedno tlačítko.



Po stisku tlačítka se vloží nová kniha s názvem v TextBoxu1 a autorem v TextBoxu2 do souboru XML. Aby se změna projevila, nesmíme zapomenout přiřadit datový zdroj do GridView a zavolat metodu DataBind.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        GridView1.DataSource = Knihovna.Knihy.Select();
        GridView1.DataBind();
    }
}
```

```
protected void Button1_Click(object sender, EventArgs e)
{
    Knihovna.Knihy.Kniha k = new Knihovna.Knihy.Kniha();
    k.nazev = TextBox1.Text;
    k.autor = TextBox2.Text;
    Knihovna.Knihy.Insert(k);
    GridView1.DataSource = Knihovna.Knihy.Select();
    GridView1.DataBind();
}
}
```

Výsledkem našeho snažení je jednoduchá evidence knih.

bbb	bbb	Button
nazev	autor	
C#	Pepa Skocdopole	
Úvod do ASP.NET	Karel Mozdren	
aaa	aaa	
bbb	bbb	

## 6.2 Data a MS-SQL (T-SQL)

Data, se kterými se konečný uživatel setkává, nejsou jak se na první pohled může zdát, z jednoho stroje. Databáze rozsáhlých informačních systémů jsou uloženy na samostatných strojích, které jsou přímo určeny k jejich uchování a neplní žádnou jinou funkci. Takovému stroji říkáme databázový server, což je zároveň označení programu, jenž nám tyto data poskytuje.

Abychom mohli s tímto strojem komunikovat, musíme znát jazyk, kterým se budeme na data dotazovat. Tímto jazykem je SQL (Structured question language). Hlavními prvky tohoto jazyka jsou dotazy. Mezi nejdůležitější patří SELECT (Dotaz na data), INSERT (Vložení dat), DELETE (Mazání) a UPDATE (Změna).

Ačkoliv se zdá, že tyto dotazy jsou zcela postačující, existuje řada dalších, které se starají například o vytvoření tabulek, databází, pohledů, vložených funkcí a procedur, triggerů a podobně. Těmito dotazy se však nebudeme zabývat a přenecháme to aplikaci MS-SQL Server od fy. Microsoft, kterou si můžete stáhnout z webu microsoftu v express verzi, pokud možno s toolkitem.

### 6.2.1 SELECT, INSERT, UPDATE a DELETE

Nyní si ukážeme základní syntaxi těchto dotazů a ukážeme si na příkladu, jak s nimi pracovat.

```
SELECT [sloupce] FROM [tabulky] WHERE [podmínky];
```

```
INSERT INTO [tabulka] (sloupce) VALUES (hodnoty);
```

```
UPDATE [tabulka] SET [sloupec] = [hodnota] WHERE [podmínky];
```

```
DELETE FROM [tabulka] WHERE [podmínky];
```

Ve většině případů budete nuceni vytvářet skripty, které Vám vytvoří databázovou strukturu dat. Takový skript pak budete mít k dispozici se zdrojovým kódem své aplikace a budete jej postupně podle potřeb měnit. Přesto že si navrhnete databázi sebeděle, počítejte s tím že Váš návrh není konečný! Připojím zde skript, který poslouží jako ukázka pro vytvoření evidence knih.

Bude značně zjednodušena, protože nebudu vytvářet samostatnou tabulku autorů, což odborníci jistě potvrdí je redundantní a může vést k nekonzistenci.

## 6.2.2 Příklad SQL Skriptu

```
CREATE DATABASE Example;
```

```
USE Example;
```

```
-- Vytvori a použije databazi Example
```

```
CREATE TABLE Knihy (  
    ISBN int NOT NULL PRIMARY KEY,  
    nazev varchar(255),  
    autor varchar(255)
```

```
);
```

```
-- Vytvori tabulku Knihy
```

```
INSERT INTO Knihy (ISBN,nazev,autor)
```

```
VALUES (0, 'ASP.NET a C++', 'Karel Mozdren');
```

```
INSERT INTO Knihy (ISBN,nazev,autor)
```

```
VALUES (1, 'ASP.NET a Visual Basic .NET', 'Karel Mozdren');
```

```
INSERT INTO Knihy (ISBN,nazev,autor)
```

```
VALUES (2, 'C++ tutorial', 'Jezisek');
```

```
-- Vlozi ukazkove hodnoty
```

```
UPDATE Knihy
```

```
SET nazev = 'ASP.NET a C#'
```

```
WHERE ISBN = 0;
```

```
-- Zmeni zaznam s ISBN 0
```

```
DELETE FROM Knihy WHERE ISBN = 1;
```

```
-- Smaze zaznam s ISBN 1
```

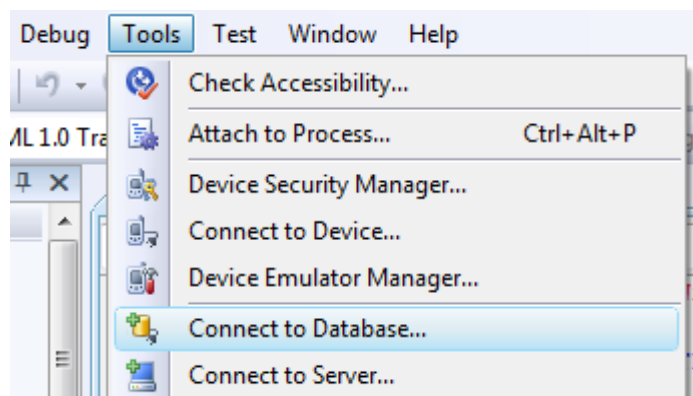
```
SELECT * FROM Knihy WHERE autor like '%Mozdren%';
```

```
-- Zobrazi vsechny zaznamy kde autor obsahuje podretezec Mozdren
```

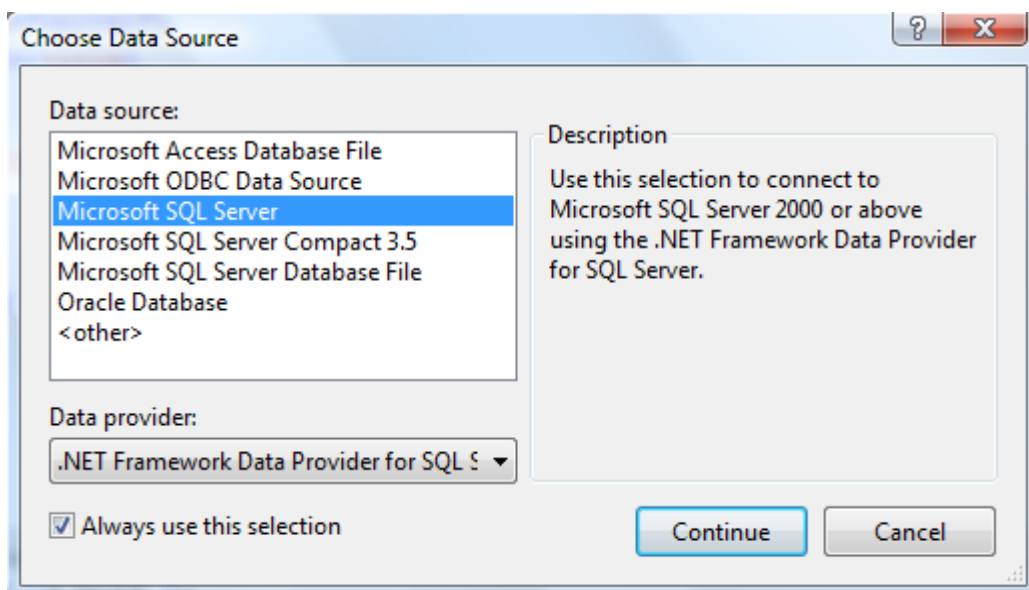
Tímto jsme vytvořili novou databázi Example ve které jsme vytvořili novou tabulku Knihy a vložili do ní nějaké záznamy. Dále jsme si ukázali jak záznamy upravovat, mazat a zobrazovat.

## 6.2.3 Připojení databáze do aplikace

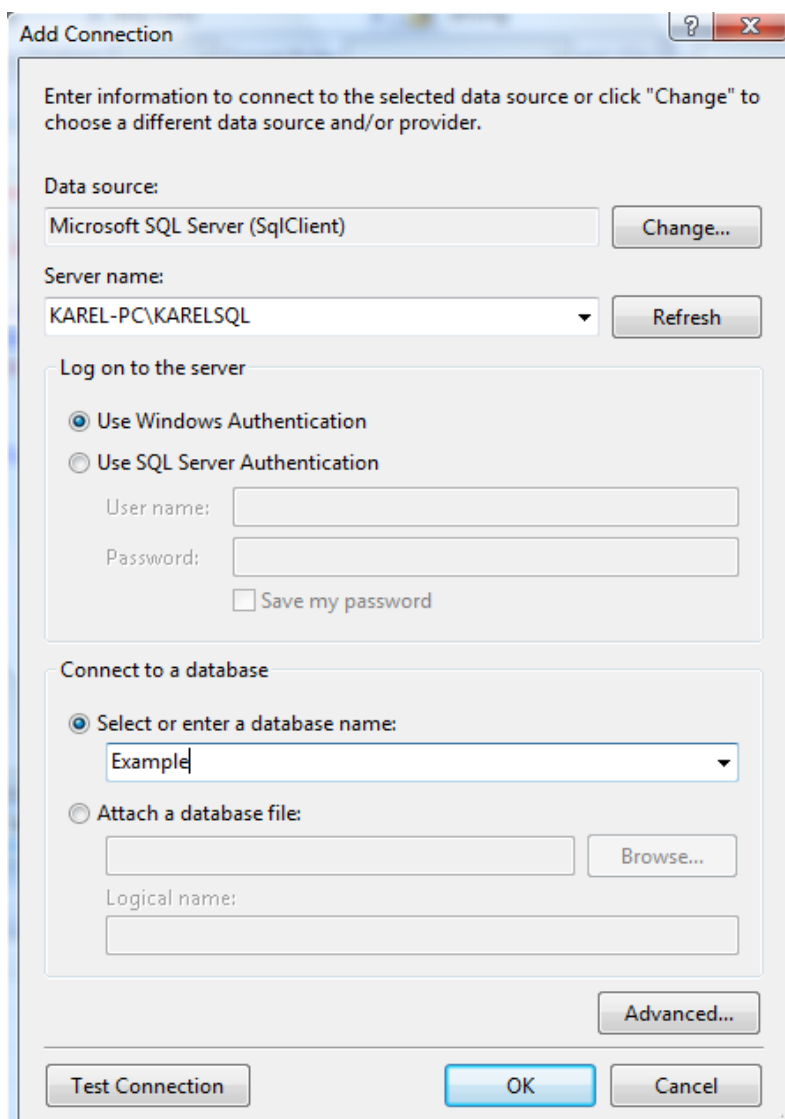
Nyní předpokládám, že mám na straně DB Serveru vytvořenou databázi a chci si ji připojit do své aplikace. Stisknu tedy v menu “Tools” (Nástroje) volbu “Connect to the database” (Připojit se k databázi).



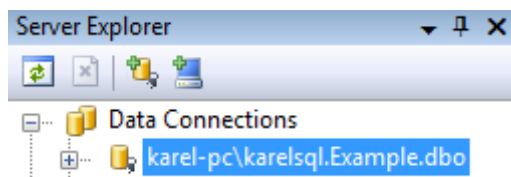
V dialogovém okně vyberu volbu “Microsoft SQL Server”.



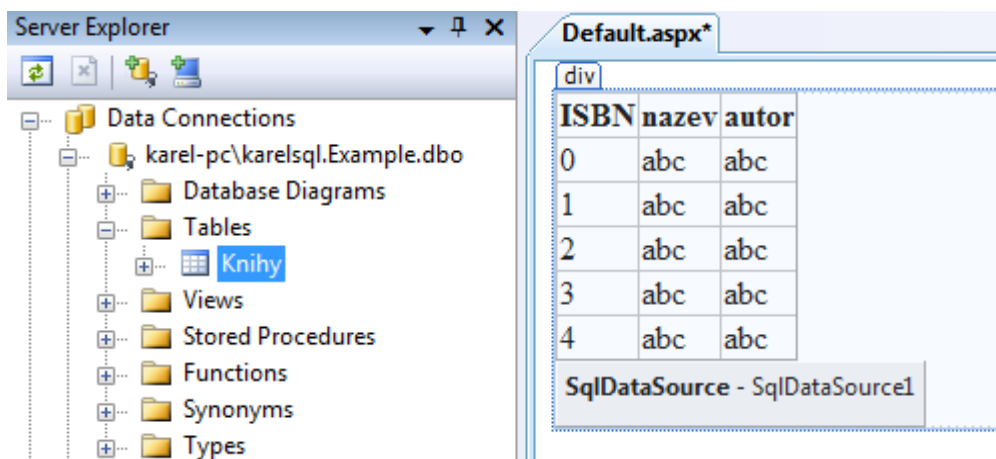
Dále musím zadat cestu k serveru na síti což je v mém případě “KAREL-PC\KARELSQL” a jméno databáze ke které se chci připojit. (“Example”).



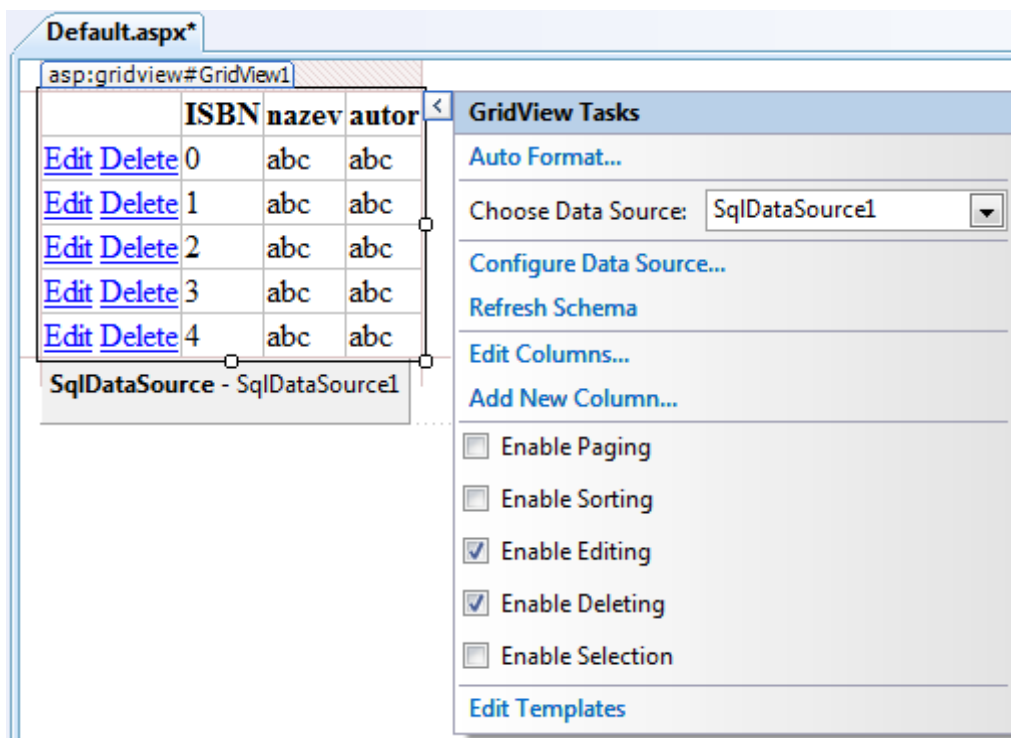
V levé části okna se nám zobrazí “Server explorer” ve kterém by jste měli vidět své připojení na server stejně jako na obrázku.



V takovém případě již můžeme přikročit k vložení dat do naší stránky a to jednoduchým tahem myši z tabulky v Server Exploreru do naší stránky v Design modu.



Dále můžete na gridView povolit funkce update a delete a tím vytvořit téměř plnohodnotnou databázi knih.



Ve výsledku bude již aplikace schopna komunikovat se serverem, tak že vlastně nic nepoznáte.

	ISBN	nazev	autor
<a href="#">Aktualizovat</a> <a href="#">Storno</a>	0	ASP.NET a C#	Karel Mozdren
<a href="#">Upravit</a> <a href="#">Odstranit</a>	2	C++ tutorial	Jezisek

## 6.2.4 Přístup k datům pomocí ObjectDataSource bez SQLDataSource

Jak jsme viděli, použití SQLDataSource je jednoduché, ale celkově vzato nepraktické pro větší projekty. Abychom mohli použít ObjectDataSource musíme implementovat metody SELECT, UPDATE, INSERT a DELETE. Vytvoříme si tedy knihovnu která bude zprostředkovávat připojení na databázi a práci s daty.

To odpovídá standardně v praxi používanému 3 vrstvému dělení informačních systémů na prezentační(ASP), logickou(C# kód aplikace) a datovou(ADO.NET). Naše knihovna bude mít jméno Knihovna a bude obsahovat třídu knihy ve které bude definovaná serializable třída Kniha obsahující vlastnosti ISBN, nazev, autor přesně jako na DB Serveru. Dále bude mít třída Knihy již dříve zmíněné funkce SELECT, UPDATE...

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Knihovna
{
    public class Knihy
    {
        [Serializable]
        public class Kniha
        {
            public int ISBN { set; get; }
            public string nazev { set; get; }
            public string autor { set; get; }
        }

        // Vratit vsechny knihy
        public static List<Kniha> Select()
        {
            List<Kniha> list = new List<Kniha>();
            return list;
        }

        // uprav zaznam podle ISBN
        public static void Update(Kniha k)
        {
        }

        // smaz zaznam podle ISBN
        public static void Delete(Kniha k)
        {
        }

        // Vloz nový zaznam
        public static void Insert(Kniha k)
        {
        }
    }
}
```

Nyní se zaměříme na implementaci jednotlivých funkcí. Setkáváme se zde s novými objekty. Jako jsou DbDataAdapter, DbCommand a DbConnection, které nás budou provázet při práci s daty. Jmenovite nas ovšem zajímají spíše objekty, které umožňují SQL přístup k datům a dědí z nich.

K tomu abychom mohli přistupovat k serveru potřebujeme řetězec ve kterém bude uložena adresa serveru a parametry připojení k němu. Takovému řetězci říkáme connection string. Příkladem může být tento řetězec.

```
private static string connectionString = "Data Source=KAREL-PC\\KARELSQL;Initial Catalog=Example;Integrated Security=True";
```

Ten jsem si přidal do třídy knihovna, abych jej mohl mnohonásobně použít. Dalším krokem je tedy už jenom vytvoření připojení k serveru. K tomu poslouží třída SqlConnection v jejímž konstruktoru je parametrem právě zmíněný connectionString.

```
using (SqlConnection con = new SqlConnection(connectionString))  
{
```

Tato zatím nová konstrukce, která využívá using je náhradou za dvojici try-catch a v případě chyby se postará o uzavření spojení.

Jelikož máme vytvořené spojení, můžeme vytvořit dotaz, který na server odešleme.

```
SqlCommand command = con.CreateCommand();  
command.CommandText = "SELECT * FROM Knihy";
```

dále pak přiřadíme objektu DataAdapter tento příkaz a zavoláme funkci Fill která naplní data do Objektu DataSet. Můžete vracet již objekt DataSet místo List<Kniha> a tím by tedy veškerá práce pro vás již končila. My se však rozhodli že budeme vracet Knihy přímo, tak musíme projít všechny řádky(DataRow) v DataSet a přiřadit jednotlivé hodnoty do objektu knihy. Celý kód pak vypadá takto:

```
// Vratit vsechny knihy  
public static List<Kniha> Select()  
{  
    List<Kniha> list = new List<Kniha>();  
    using (SqlConnection con = new SqlConnection(connectionString)){  
        SqlCommand command = con.CreateCommand();  
        command.CommandText = "SELECT * FROM Knihy";  
        SqlDataAdapter adapter = new SqlDataAdapter();  
        adapter.SelectCommand = command;  
        DataSet ds = new DataSet();  
        con.Open();  
        adapter.Fill(ds);  
        foreach (DataRow dr in ds.Tables[0].Rows)  
        {  
            Kniha kn = new Kniha();  
            kn.ISBN = Convert.ToInt32(dr[0].ToString());  
            kn.nazev = dr[1].ToString();  
            kn.autor = dr[2].ToString();  
            list.Add(kn);  
        }  
    }  
    return list;  
}
```

Použití všech těchto tříd navíc vyžaduje další usingy.

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```
using System.Text;
using System.Data.Common;
using System.Data.SqlClient;
using System.Data;
```

Zjímavost při implemetaci metody Update je to že budeme používat parametrizované příkazy a to vyžaduje použití třídy SqlParameter. V žádném případě se nesnažte parametry vkládat spojováním řetězce, protože se tím vystavujete riziku injekce kódu. Výsledná funkce pak vypadá takto:

```
// uprav zaznam podle ISBN
public static void Update(Kniha k)
{
    using (SqlConnection con = new SqlConnection(connectionString))
    {
        SqlCommand command = con.CreateCommand();
        command.Parameters.Add(new SqlParameter("@isbn", k.ISBN));
        command.Parameters.Add(new SqlParameter("@autor", k.autor));
        command.Parameters.Add(new SqlParameter("@nazev", k.nazev));
        command.CommandText = "UPDATE Knihy SET nazev=@nazev, autor=@autor WHERE ISBN=@isbn;";
        con.Open();
        command.ExecuteNonQuery();
    }
}
```

Stejným způsobem se postupuje i u dalších 2 funkcí.

```
// smaz zaznam podle ISBN
public static void Delete(Kniha k)
{
    using (SqlConnection con = new SqlConnection(connectionString))
    {
        SqlCommand command = con.CreateCommand();
        command.Parameters.Add(new SqlParameter("@isbn", k.ISBN));
        command.CommandText = "DELETE FROM Knihy WHERE ISBN=@isbn;";
        con.Open();
        command.ExecuteNonQuery();
    }
}
```

```
// Vloz nový zaznam
public static void Insert(Kniha k)
{
    using (SqlConnection con = new SqlConnection(connectionString))
    {
        SqlCommand command = con.CreateCommand();
        command.Parameters.Add(new SqlParameter("@isbn", k.ISBN.ToString()));
        command.Parameters.Add(new SqlParameter("@nazev", k.nazev));
        command.Parameters.Add(new SqlParameter("@autor", k.autor));
        command.CommandText = "INSERT INTO Knihy (ISBN, nazev, autor)"+
            "VALUES (@isbn, @nazev, @autor);";
        con.Open();
        command.ExecuteNonQuery();
    }
}
```

Nyní máte připravenou knihovnu pro práci s databází a můžete vzít příklad s SQLDataSource a nahradit jej ObjectDataSource kde přiřadíte všechny funkce UPDATE, DELETE, SELECT, INSERT. Získáte tak naprosto stejný projekt, ale máte nad ním maxiální

kontrolu.

	ISBN	nazev	autor
<a href="#">Upravit</a> <a href="#">Odstranit</a>	0	Dopis Jeziskovi	Karel Mozdren
<a href="#">Upravit</a> <a href="#">Odstranit</a>	2	C++ tutorial	Jezisek

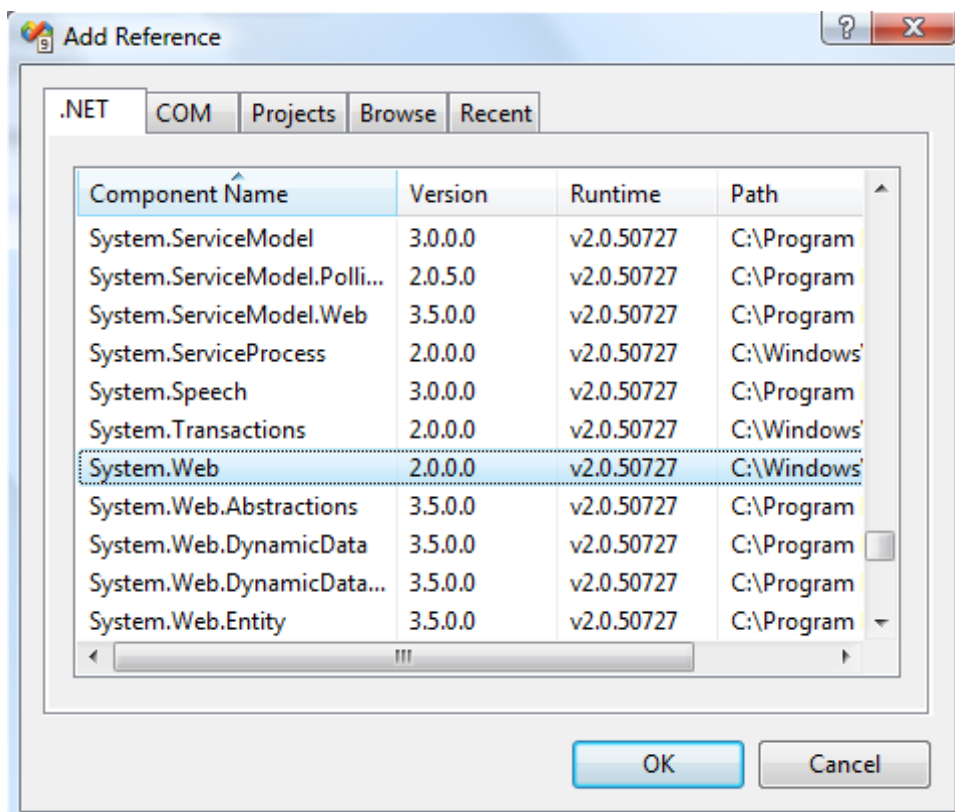
## 7. Vlastní komponenty (Controly)

V našich příkladech jsme využili již velké množství komponent, které máme standardně v toolkitu. Co ale dělat v případě, že jsme nenašli žádný vyhovující pro naši aplikaci. Odpověď je jednoduchá. Vytvoříme si ji sami.

### 7.1 Control

Všechny komponenty jsou děděny z tříd Control, nebo WebControl a stejně tak i my musíme naši novou komponentu dědit z těchto tříd. Postupujeme tedy tak, že si pro začátek vytvoříme třídní knihovnu ve které bude třída, která bude dědit z Control.

Abychom si zpřístupnili toto rozhraní musíme použít System.Web. To provedeme přidáním reference do projektu třídní knihovny. A vyhledáme v položce .NET rozhraní System.Web.



Po přidání této reference již můžeme použít rozhraní System.web.UI. A následovně dědit naši třídu z Control.

```
using System.Web.UI;
```

Dědičnost nám umožňuje eliminaci nutnosti implementovat celou třídu a měnit pouze části, které jsou pro danou komponentu specifické.

Metody, které budeme měnit mají speciální definici slovem `override`, jež nám umožňuje přepsat původní definici zděděnou z třídy původní. Pro nás je momentálně důležité přepsat metodu `render`. Ta se volá při vytváření HTML kódu stránky.

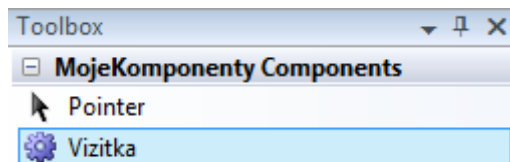
Pro ukázkou implementujeme tuto komponentu jako vizitku s jménem, příjmením a bydlištěm.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.UI;

namespace MojeKomponenty
{
    public class Vizitka : Control
    {
        public string jmeno = "Karel";
        public string prijmeni = "Mozdren";
        public string bydliste = "Pod mostem";

        protected override void Render(HtmlTextWriter writer)
        {
            writer.Write("Vizitka <b>" + prijmeni + "</b> " +
                jmeno + "<br />" + bydliste);
        }
    }
}
```

Teď musíme přidat referenci tohoto projektu do projektu webové stránky a buildnout naše řešení. Pokud nyní přejdete do režimu design měla by se vám zobrazit Vaše komponenta v toolboxu.



Přetáhneme-li naši komponentu na stránku zobrazí se stejně jako na dalším obrázku.

Vizitka **Mozdren Karel**  
Pod mostem

## 7.2 Skládané komponenty (Composite controls)

Při tvorbě složitějších komponent jako funkčních celků vkládaných do stránky je předchozí postup nedostatečný. Musíme šáhnout po dalších třídách které nám umožní spojovat komponenty. Třída, nebo lépe interface, který použijeme nyní se nazývá `INamingContainer`. S jeho pomocí jsme schopni implementovat funkce, které přidávají na stránku množství “dětských” komponent. Další výhodou je možnost odchylování událostí, nebo jejich tvorba.

Postup vytvoření takové komponenty je stejný, ale je rozšířený o několik jednoduchých kroků. Mezi ty patří například implementace metody `CreateChildControls`, kterou použijeme ke vložení komponent.

Jako ukázkou si vytvoříme jednoduchou sčítačku.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace DalsiKomponenty
{
    public class Scitacka : Control, INamingContainer
    {
        // delegat a eventa oznamujici konec vypoctu
        public delegate void spoctenoEvent(object sender, EventArgs a);
        public event spoctenoEvent spocteno;

        // komponenty
        private TextBox t1, t2;
        private Button b;
        private Label vysledek;

        // vytvoreni detskych komponent
        protected override void CreateChildControls()
        {
            t1 = new TextBox();
            t2 = new TextBox();
            b = new Button();
            b.Click += spocti; // prirazeni obsluhy odalosti funkci spocti
            b.Text = "=";
            vysledek = new Label();
            Controls.Add(t1);
            Controls.Add(new LiteralControl("+"));
            Controls.Add(t2);
            Controls.Add(b);
            Controls.Add(vysledek);
        }

        public void spocti(object source, EventArgs arg)
        {
            int a = Convert.ToInt32(t1.Text);
            int b = Convert.ToInt32(t2.Text);
            int v = a + b;
            vysledek.Text = v.ToString();
            spocteno(this, null); // vyvolani udalosti spocteno
        }
    }
}

```

V kódu si všimněte jakým způsobem se vytvářejí události, nebo se přiřazují obslužné metody reagující na události vyvolané uživatelem.

První za zmínku stojí přidělení obsluhující metody pro tlačítko, které vyvolá výpočetní funkci `spocti()` a ta následně získá hodnoty z obou `TextBoxů`, sečte je a vloží do `Labelu` výsledek.

```
b.Click += spocti; // prirazeni obsluhy odalosti funkci spocti
```

Neméně zajímavé je vytvoření delegáta a jeho využití k tvorbě události specifické pro danou komponentu. Vyvoláme ji přesně v době, kdy skončí výpočet.

```

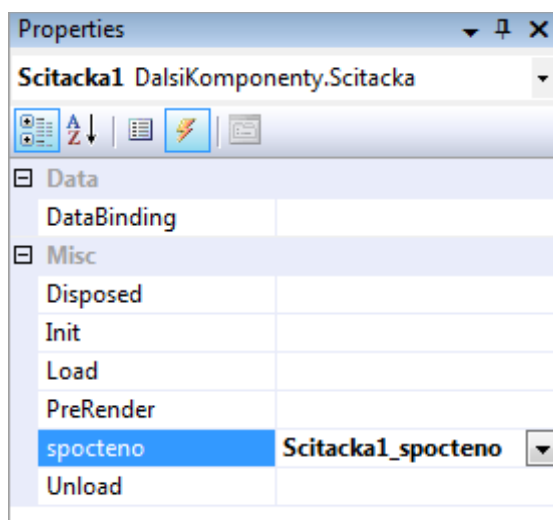
// delegat a eventa oznamujici konec vypoctu
public delegate void spoctenoEvent(object sender, EventArgs a);
public event spoctenoEvent spocteno;

public void spocti(object source, EventArgs arg)
{
    int a = Convert.ToInt32(t1.Text);
    int b = Convert.ToInt32(t2.Text);
    int v = a + b;
    vysledek.Text = v.ToString();
    spocteno(this, null); // vyvolani udalosti spocteno
}

```

Při vkládání této komponenty v design modu se může stát, že nepůjde vidět. V takovém případě si můžete zkontrolovat vložení v modu Source.

Po vložení můžeme využít události spočteno. Vztvoříme si label a metodu, která bude na událost spočteno reagovat. Ta zapíše do Labelu text Spočteno.



```

protected void Scitacka1_spocteno(object sender, EventArgs a)
{
    this.Label1.Text = "Spočteno";
}

```

Výsledek našeho snažení je komponenta jako samostatný funkční celek, kterou si můžeme do stránky opakovaně vkládat.



## 8. ViewState a Session

V některých případech potřebujeme uložit data pro pozdější zpracování, jako počet zobrazení stránky, login uživatele, jeho roli v informačním systému atd.. K tomu nám slouží ViewState a Session.

ViewState je hodnota která je uložena ve stránce a lze ji na klientském prohlížeči pozorovat. Obsahuje informace o pohybu uživatele na stránce, jako akce a podobně. Tento způsob ukládání je pro server výhodný, neboť je nemusí udržovat v paměti a šetří tak místo. Při přechodu

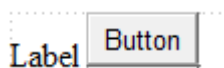
na jinou stránku se strácí.

Naproti tomu Session je uložena po nějakou dobu a je udržována dokud neuplyne čas nečinnosti uživatele a ukončí se, nebo se vypne prohlížeč.

## 8.1 Příklad použití ViewState a Session

Abychom lépe pochopili rozdíl mezi ViewState a Session uděláme si příklad ve kterém budeme počítat zobrazení stránky.

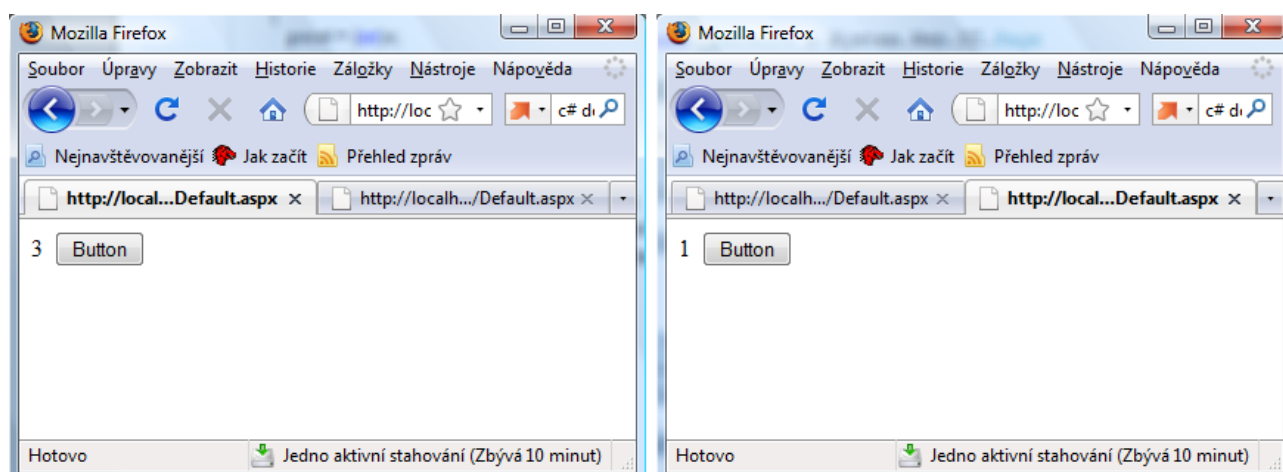
Nejprve si vytvoříme jednoduchou stránku s komponentami Label a button.



Pak přejdeme do psaní logiky stránky a zaměříme se na událost Page\_Load. Ve které si získáme hodnotu z ViewState pocítadlo pokud existuje, a pokud ne, tak ji vytvoříme, inkrementujeme a vložíme do Label.

```
protected void Page_Load(object sender, EventArgs e)
{
    object o = ViewState["pocítadlo"];
    int pocet = 0;
    if (o == null)
    {
        ViewState["pocítadlo"] = pocet;
    }
    else
    {
        pocet = (int)o;
        pocet++;
        ViewState["pocítadlo"] = pocet;
    }
    Label1.Text = pocet.ToString();
}
```

Výsledkem je stránka na které s každým stisknutím tlačítka zvlší hodnota o 1. Pokud vytvoříte nový panel na stejném serveru, začíná výpočet zase od nuly. Hodnoty se tedy liší, protože každá zobrazená stránka ma vlastní ViewState.



Obrázky vedle sebe jsou snímky jednotlivých panelů jednoho prohlížeče.

Pro ukázkou Session stačí v předchozím kódu přepsat všechna ViewState na Session a zkusit si v jednotlivých panelech prohlížeče klikat na tlačítka. Teď si obě stránky čtou data z jednoho Session a jsou tak na sobě závislé.

# Obsah

1. Úvod.....	2
1.1 Slovo na úvod.....	2
1.2 Poznáváme prostředí Visual Studio.....	2
1.2.1 Vytvoření první webové stránky.....	2
1.2.3 První příklad „Hello, world!“.....	3
1.2.4 Některé základní značky HTML.....	6
2. Začínáme programovat.....	7
2.1 Dynamický web.....	7
2.1.1 Přidáváme komponenty do stránky.....	7
2.1.2 Dáváme stránce život.....	9
2.1.3 Příklad Kalkulačka.....	10
3. Validace vstupů.....	12
3.1 Nastavení validace komponenty.....	12
3.1.1 RequiredFieldValidator.....	12
3.1.2 RangeValidator.....	13
3.1.3 RegularExpressionValidator.....	13
3.1.4 CompareValidator.....	13
3.1.4 CustomValidator.....	13
4. Dynamické knihovny.....	14
4.1 Vytvoření nové třídní knihovny.....	15
4.1.1 Příklad třídní knihovny na náhodná čísla.....	17
5. Vybrané komponenty pro prezentaci dat.....	18
5.1 GridView.....	18
5.1.1 [Serializable] třída.....	20
5.2 DetailsView.....	21
5.3 Repeater.....	21
6. Databáze.....	22
6.1 XML.....	22
6.1.2 Implementace přístupu k datům v XML.....	24
6.2 Data a MS-SQL (T-SQL).....	26
6.2.1 SELECT, INSERT, UPDATE a DELETE.....	26
6.2.2 Příklad SQL Skriptu.....	27
6.2.3 Připojení databáze do aplikace.....	27
6.2.4 Přístup k datům pomocí ObjectDataSource bez SQLDataSource.....	30
7. Vlastní komponenty (Controly).....	33
7.1 Control.....	33
7.2 Skládané komponenty (Composite controls).....	34
8. ViewState a Session.....	36
8.1 Příklad použití ViewState a Session.....	37