UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Petra Udir

ASP.NET

Diplomska naloga

LJUBLJANA, 2007

PROGRAM DIPLOMSKE NALOGE

V diplomski nalogi prikažite, kako s pomočjo tehnologije ASP.NET razvijamo spletne aplikacije. V ta namen izdelajte praktičen primer spletne aplikacije. S pomočjo tega praktičnega primera predstavite osnovne gradnike ter prijeme pri gradnji tovrstnih programov.

Mentor:

mag. Matija Lokar

Zahvala

Za pomoč pri diplomski nalogi se zahvaljujem mentorju, mag. Matiji Lokarju, ki mi je s potrpežljivostjo popravljal moje osnutke. Za pomoč in spodbudo pri pisanju se zahvaljujem tudi družini in vsem prijateljem, ki so mi stali ob strani.

POVZETEK

Diplomska naloga govori o Microsoftovi tehnologiji ASP.NET, s pomočjo katere izdelujemo spletne strani.
Cilj same diplomske naloge je predstavitev te tehnologije. V ta namen sem izdelala spletno aplikacijo, ki je namenjena vnosu in pregledovanju rezultatov izpitov in kolokvijev. Namen izdelave aplikacije je bil prikaz načina uporabe te tehnologije in ne izdelava neke praktično uporabne aplikacije. Zato je poudarek pri programiranju bil predvsem na prikazu različnih tehnik.
V nalogi si bomo najprej ogledali okolje, v katerem sem izdelala svojo spletno stran. Sledi opis najbolj pogostih gradnikov, ki jih uporabljamo pri razvoju tovrstnih aplikacij. Pri vsakem opisu gradnika je tudi primer iz moje spletne aplikacije. Na koncu so predstavljeni bolj zanimivi deli kode iz moje aplikacije in njihova razlaga. Programska koda je napisana v programske jeziku Visual Basic.

Math. Subj. Class. (2000): 68N19, 68P05, 68U99
Computing Review Class. System (1998): D.1.5, D.1.7, E.1

Ključne besede: spletne aplikacije, .NET, strežniški gradniki
Keywords: web application, .NET, server controls

Uvod

Spletne aplikacije so danes skoraj nepogrešljiva stvar pri poslovanju podjetij, različnih ustanov in tudi posameznikov. Preko spleta se lahko naročimo na različne preglede, uredimo zavarovanje, se prijavimo na izpite, nakupujemo … Spletna aplikacija je najkrajše rečeno primer programa, ki se izvaja preko spleta. Uporabniki za dostop do programa uporabljajo spletni brskalnik, sama koda pa se praviloma izvaja na spletnem strežniku.
Vse te spletne aplikacije so razvite s pomočjo različnih tehnologij (PHP, JSP …). Ena od njih je tudi ASP.NET, o kateri bo govora v moji diplomski nalogi.
Kot praktični primer uporabe te tehnologije, ki je tema moje diplomske naloge, sem razvila spletno aplikacijo za pregledovanje in vstavljanje rezultatov kolokvijev in izpitov.

Ogrodje .NET

Ime .NET je krovno ime za cel skupek izdelkov in tehnologij, ki jih razvija in trži podjetje Microsoft. Poglavitna skupna točka vseh izdelkov in tehnologij je odvisnost od ogrodja .NET (.NET Framework). To ogrodje je sestavni del operacijskega sistema Windows, oziroma ga je temu operacijskemu sistemu mogoče dodati. Ogrodje je sestavljeno iz velikega števila rešitev za različna programska področja kot so gradnja uporabniških vmesnikov, dostop do podatkov, kriptografija, razvoj spletnih rešitev, numerični algoritmi, omrežne komunikacije ... Metode, uporabljene v teh rešitvah, pri razvoju programov programerji kombinirajo z lastno kodo.
Ogrodje .NET določa arhitekturo razvoja programskih rešitev. Temelji na konceptih objektne tehnologije in komponentnega razvoja. Razvijalcem ponuja pregledno razvojno okolje, s katerim lahko na enostaven način gradijo aplikacije. Programi, ki so napisani za ogrodje .NET, se izvajajo v posebnem okolju, ki upravlja zahteve programa med izvajanjem. To okolje, ki je tudi sestavni del ogrodja .NET, se imenuje izvajalnik kode skupnega jezika (CLR – *Common Language Runtime*). Osnovne komponente, ki določajo, kaj je .NET, so:

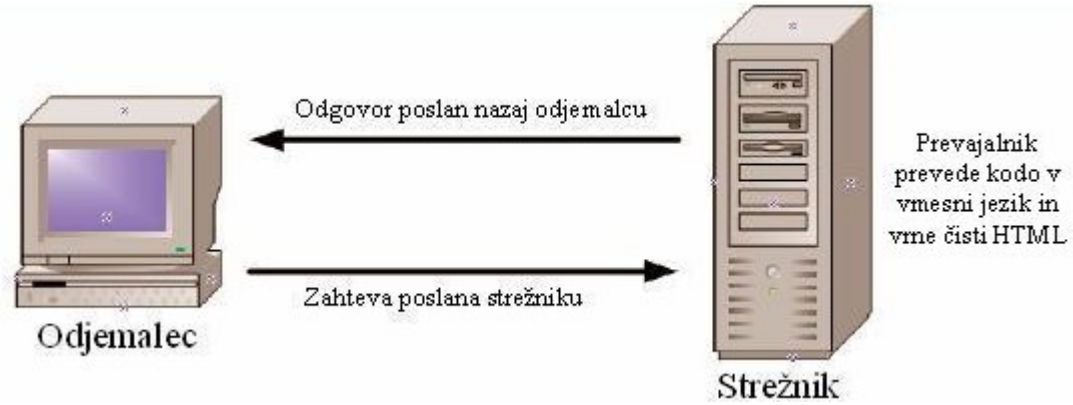
- izvajalnik kode skupnega jezika,
- knjižnice razredov in
- storitve.

Osnova je torej skupno izvajalno okolje za različne programske jezike. To izvajalno okolje predstavlja navidezni računalnik. Zato se programerju ni potrebno ukvarjati z lastnostmi posameznega procesorja in ostalih strojnih delov računalniškega sistema. Prav tako se koda programov ne prevaja neposredno v strojno kodo (kodo, prirejeno za določen procesor), temveč se prevede v tako imenovano vmesno kodo (kodo, osnovano na vmesnem jeziku). V ogrodju .NET se v vmesno kodo lahko prevedejo vsi programski jeziki, prirejeni za okolje .NET. To pomeni, da lahko razvijalci pri pisanju kode uporabljajo kateregakoli od teh jezikov, saj so z uporabo vmesnega jezika zabrisane meje med njimi.
Okolje .NET poleg možnosti izbire med programskimi jeziki omogoča tudi sočasno uporabo več programskih jezikov v enem programu. To pomeni, da lahko razvijalci uporabljajo jezik po svoji izbiri, saj programi, napisani v enem programskem jeziku, lahko kličejo metode (podprograme, funkcije), napisane v drugem jeziku.
Ogrodje .NET vsebuje številne razrede, ki ponujajo bogat nabor vnaprej definiranih funkcionalnosti. Vanj so vključeni razredi, ki vsebujejo različne matematične funkcije, razredi za delo z bazami podatkov, razredi, ki ponujajo gradnike za izdelovanje obrazcev, razredi, ki poenostavljajo ustvarjanje dokumentov XML in delo z njimi ter še mnogi drugi.
Razredi v ogrodju sestavljajo knjižnice, ki so organizirane po imenskih prostorih (*namespaces*). V posamezen imenski prostori združimo razrede, ki omogočajo določene funkcionalnosti (npr. delo s formatom XML, delo z grafiko ...). Imenske prostore zaradi večje preglednosti organiziramo hierarhično. Hierarhija imenskih prostorov poteka od leve proti desni.
Imenski prostori omogočajo boljšo organiziranost razredov in enostavnejše pisanje kode. Če na primer želimo v programski kodi uporabljati metodo `X` iz imenskega prostora `System.Web`, moramo metodo načeloma klicati s `System.Web.X()`. Če pa imenski prostor `System.Web` vključimo v naš program, lahko napišemo le `X()`.

V programu imenski prostor vključimo vedno, ko potrebujemo dostop do želenega imenskega prostora. Če na primer želimo uporabljati imenski prostor `System.Web`, ga v programskem jeziku Visual Basic vključimo z `Imports System.Web()`, v jeziku C# z `using System.Web;` in podobno.

Kaj je ASP.NET in kako deluje?

ASP.NET (*Active Server Pages*) je Microsoftova tehnologija, ki omogoča kreiranje dinamičnih spletnih strani. Dinamične spletne strani se od navadnih spletnih strani razlikujejo po tem, da vsebujejo tudi elemente, ki se dinamično spreminjajo med samim pregledovanjem spletne strani. Ti dinamični elementi ASP se kreirajo na samem strežniku in so naknadno prevedeni v kodo HTML. Dinamične strani omogočajo, da se uporabniku predstavi vsebina, ki jo je na strežniku na podlagi uporabnikovih zahtev in/ali podatkov, ki so zapisani v nekem viru (razne vrste podatkovnih baz, dokumenti XML, oddaljen računalniški sistem, ki komunicira s strežnikom ...) generiral nek program.



Slika 1 - Povezava odjemalec - strežnik

Ko odjemalec (običajno je to spletni brskalnik) želi dostopiti do določene spletne strani, napisane v ASP.NET, se zahteva pošlje strežniku. Strežnik prepozna končnico `.aspx` in stran pošlje posebnemu prevajalniku. Ko prevajalnik pregleduje to stran, razlikuje med navadnimi značkami HTML in strežniškimi gradniki. Če naleti na navadno kodo HTML, jo pusti tako, kot je. Če pa naleti na strežniški gradnik, pokliče metodo `Render`. Ta pripada vsakemu strežniškemu gradniku. Metoda vrne ustrezno kodo HTML, kjer je strežniški gradnik, ki ga je prej opisoval aspx, opisan v jeziku HTML . Ko prevajalnik pregleda vse značke, je rezultat klasična stran v "čistem" jeziku HTML. Strežnik to novo kreirano stran pošlje nazaj odjemalcu, ki spletno stran prikaže.
Oglejmo si enostaven primer.

Primer strani v ASP.NET (datoteka Primer.aspx):

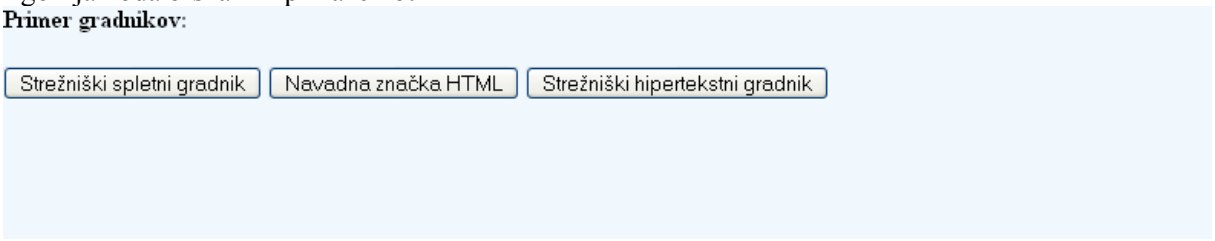
```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Enostavna spletna stran</title>
</head>
<body>
    <form id="frmPrimer" runat="server">
    <div style="background-color: aliceblue">
        <%-- Komentar --%>
        <b>Primer gradnikov:</b><br />
        <br />
        <asp:Button ID="btnStrezniskiAsp" runat="server" Text="Strežniški
                    spletni gradnik" />
        <input id="btnNavadniHtml" type="button" value="Navadna značka
```

```
                HTML" />
        <input id="btnStrezniskiHtml" runat="server" type="button"
                value="Strežniški hipertekstni gradnik" />
    </div>
    </form>
</body>
</html>
```

Prvi dve vrstici kode sestavljata tako imenovan napotek, o katerem bomo povedali več v naslednjem poglavju. Napotke prepoznamo po posebni obliki značke v HTML, ki se začne z `<%`. Na ta način začnemo tudi komentarje, ki jih pišemo znotraj znakov `<%--  --%>`.

Sledijo značke jezika HTML `html`, `head`, `title` ter `body`, `form`, `div` in njihove lastnosti, npr. `style`. Znotraj značke `body` so trije gumbi. Prvi je narejen kot gradnik asp. Prepoznamo ga po začetku `<asp:`, druga dva gumba pa sta iz jezika HTML (z značko `input`), le da smo drugega, ki ima še lastnost `runat="server"`, spremenili v dinamični gradnik.

Zgornja koda brskalnik prikaže kot



Slika 2 - Spletna stran, ustvarjena s kodo na `Primer.aspx`

Brskalnik je od strežnika prejel sledečo kodo, ki jo je na strežniku generiral ustrezni prevajalnik ASP:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1">
<title>Enostavna spletna stran</title>
</head>
<body>
   <form name="frmPrimer" method="post" action="Default.aspx" id="frmPrimer">
      <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
             value="/wEPDwUJODc2NzI0Njk4ZGTCs63PQ9IX+V4EgJkZwXQey6kjCA==" />
      </div>
      <div style="background-color: aliceblue">
      <b>Primer gradnikov:</b><br /><br />
      <input type="submit" name="btnStrezniskiAsp" value="Strežniški
             spletni gradnik" id="btnStrezniskiAsp" />
      <input id="btnNavadniHtml" type="button"
             value="Navadna značka   HTML" />
      <input name="btnStrezniskiHtml" type="button" id="btnStrezniskiHtml"
             value="Strežniški hipertekstni gradnik" />
      </div>
      <div>
      <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
             value="/wEWAwKxjIqODQKulM/0CwKsxaXOA67WwqMBEHXvv4a4950S5Wvt41YL" />
      </div>
   </form>
</body>
</html>
```

Vidimo lahko, da so sedaj vsi gumbi pripravljeni kot običajni gradniki v jeziku HTML.

Več o strežniških gradnikih in samimi razlikami med strežniškimi gradniki in navadnimi hipertekstnimi gradniki bomo povedali v poglavju Strežniški gradniki (stran 17).
Če bi želeli, da se ob kliku na gumb izvede nek dogodek, bi to lahko storili le za strežniška gumba. Namreč le ta dva sta dostopna preko programske kode. Prepoznamo ju po lastnosti `runat`. Strežniškemu spletnemu gradniku lahko preko kode spremenimo npr. barvo ozadja. Sama programska koda, ki ob kliku na gumb spremeni barvo ozadja, je shranjena v datoteki `Primer.aspx.vb` in je lahko videti na primer tako:

```
Partial Class Primer
   Inherits System.Web.UI.Page

   Protected Sub btnStrezniskiAsp_Click(ByVal sender As Object, ByVal e
   As System.EventArgs) Handles btnStrezniskiAsp.Click
      btnStrezniskiAsp.BackColor = Drawing.Color.LightBlue
   End Sub
End Class
```

Aplikacija za spremljanje rezultatov izpitov in kolokvijev

Glavni namen diplomske naloge je opisati, kako s tehnologijo ASP.NET gradimo dinamične spletne strani in spletne aplikacije. V ta namen bom kot zgled prikazala razvoj aplikacije za spremljanje rezultatov izpitov in kolokvijev. Namen priprave aplikacije je zgolj v ilustraciji določenih prijemov in postopkov, ki pogosto nastopajo pri gradnji spletnih aplikacij. Zato poudarek ni na izbiri optimalnih postopkov in na samem načrtovanju aplikacije. Izbrani so taki gradniki in postopki, ki kar najbolje ilustrirajo koncepte, ki so prej razloženi. Praktično vsi zgledi in slike, ki bodo navedeni v nadaljevanju, izvirajo iz same aplikacije.
Sama aplikacija je razdeljena na dva dela. Prvi je administratorski del, kjer lahko administrator vnaša podatke o študentih, kolokvijih, izpitih in rezultatih ter jih ima možnost tudi popravljati.

Slika 3 – Vstopna stran aplikacije

Ko uporabnik vpiše veljavno pravo uporabniško ime in geslo, se odpre stran za pregled rezultatov izpitov in kolokvijev (Slika 4).

Drugi del je namenjen predvsem študentom, ki želijo vpogled v rezultate kolokvijev oz. izpitov in pregled preverjanj, ki še sledijo v študijskem letu. Uporabnik ima več možnosti. Lahko pogleda rezultate vseh študentov v letniku ali pa si ogleda rezultate le za točno določenega študenta. Sam kolokvij oz. izpit je določen z datumom. Na voljo sta dva različna izpisa rezultatov. Tako lahko uporabnik zahteva izpis le končnega seštevka točk, ki jih je študent dosegel pri posameznem testu, ali pa izpis rezultatov po nalogah. Naslednja slika prikazuje podatke za študenta z vpisno številko 1001, ki je pri kolokviju z dne 24.8.2007, dosegel 92%:

Slika 4 - Pregled rezultatov kolokvija za enega študenta

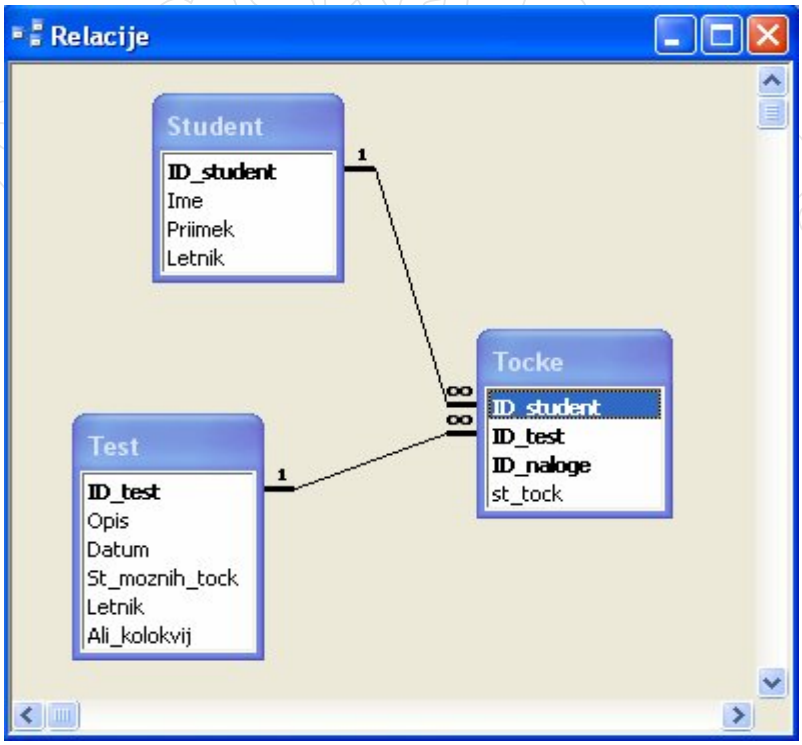
Pri gradnji aplikacije sem uporabljala orodje Visual Web Developer, bazo podatkov pa sem ustvarila v Microsoftovem programu Microsoft Office Access.

Podatkovna baza moje aplikacije

Podatkovna baza, ki jo uporabljam v svoji aplikaciji, je narejena s pomočjo programa Access. Vsebuje tabele `Student`, `Test` in `Tocke`. Podrobnosti o gradnji baze, tabelah in relacijah med njimi ne bom navajala. Podani so le tisti nujni podatki o tabelah in relacijah, ki jih potrebujemo za razumevanje delovanja aplikacije:

Ime stolpca	Tip	Ali je podatek obvezen
STUDENT		
ID_student	število	Da
Ime	niz	Da
Priimek	niz	Da
Letnik	število	Da
TEST		
ID_test	število	Da
Opis	niz	
Datum	datum/čas	Da
Ali_kolokvij	da/ne	Da
Letnik	število	Da
St_moznih_tock	dec.število	Da
TOCKE		
ID_student	število	Da
ID_test	število	Da
ID_naloge	število	Da
St_tock	dec.število	Da

Tabele so med seboj povezane z naslednjimi relacijami:



Slika 5 – Shema podatkovne baze

Visual Web Developer

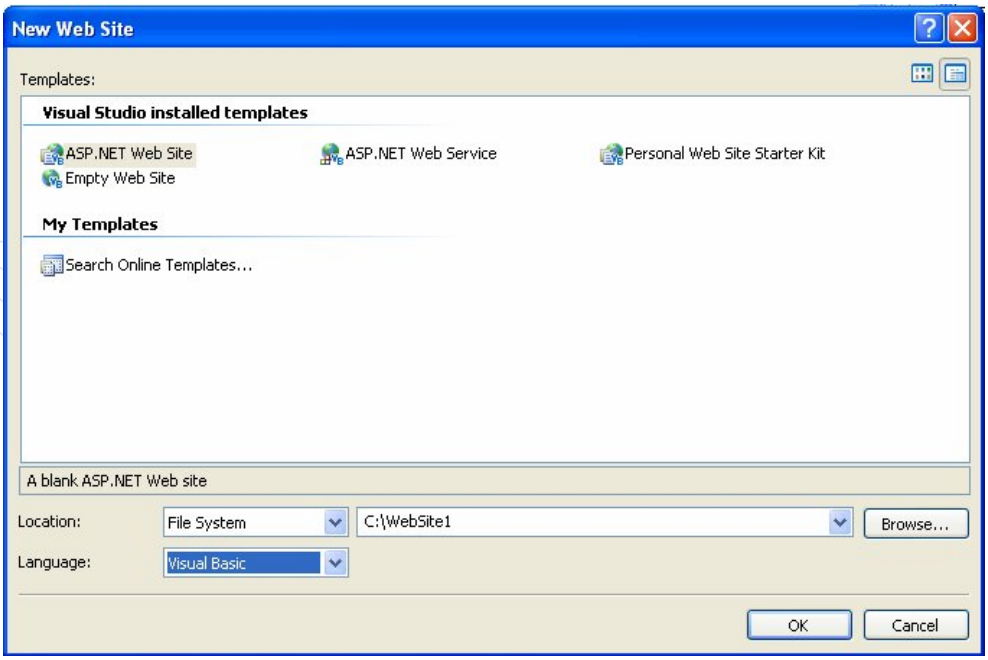
Visual Web Developer 2005 Express Edition je razvojno orodje, namenjeno razvijalcem spletnih aplikacij. Na Microsoftovih spletnih straneh je na voljo brezplačno. Je sestavni del okolja Microsoft Visual Studio, Express Edition. V načelu je to podobno okolje kot "pravi" Visual Studio. Pri različici Express edition gre za skupek razvojnih orodij, ki so prvenstveno namenjene spoznavanju s tehnologijo. Zato določenih funkcionalnosti ni. Še vedno pa gre za močno orodje, ki nam daje dober vpogled v tehnologijo programiranja v okolju .NET.
Ko zaženemo to orodje, se moramo najprej odločiti, kaj bomo razvijali (spletno stran, spletno storitev ...). Na voljo so predloge za izdelavo:

- spletne aplikacije ASP.NET (ASP.NET Web Site),
- spletne storitve (ASP.NET Web Service),
- prazne spletne strani (Empty Web Site)

ali pa uporabimo

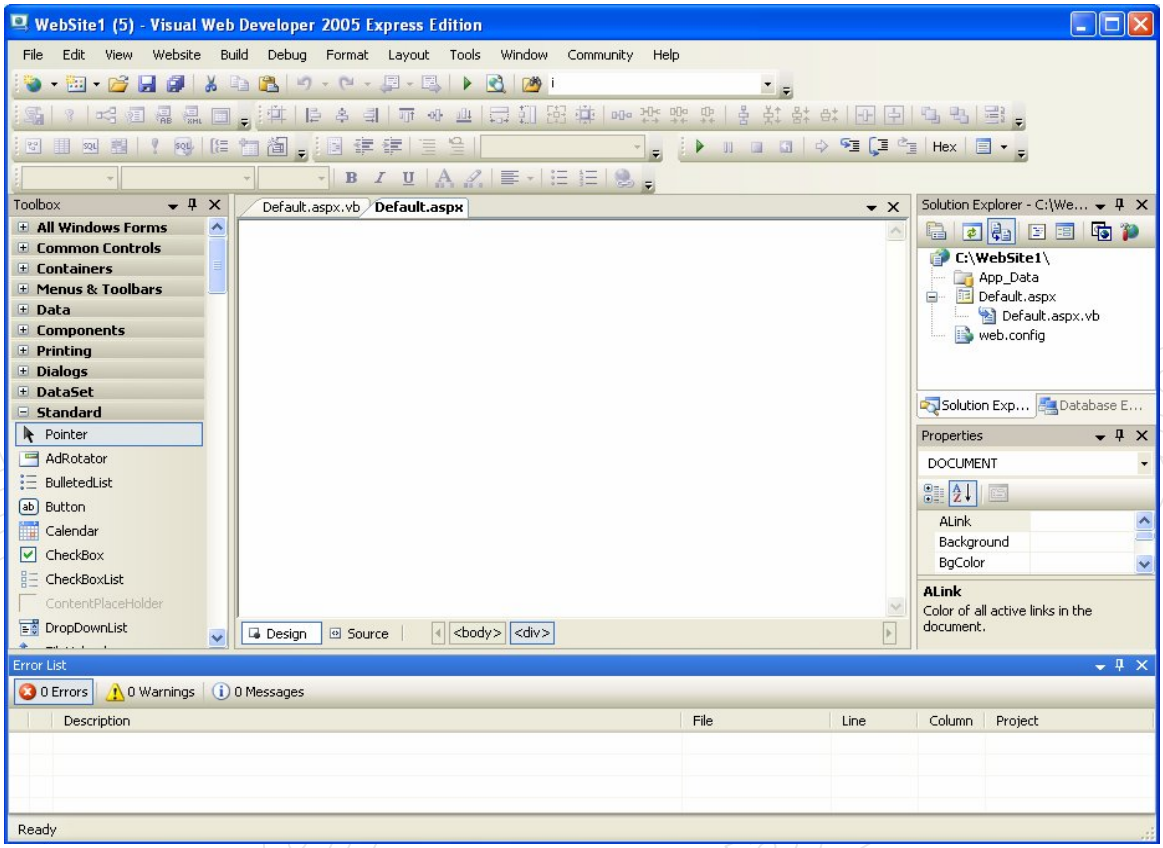
- čarovnika za izdelavo osebne spletne strani (Personal Web Site Starter Kit).

Na začetni strani pri izbiri `Location` imamo možnost, da projekt razvijamo kar v datotečnem sistemu (`File System`). To nam pride prav še posebej takrat, ko nimamo na voljo povezave s spletnim strežnikom, želeli pa bi preizkušati, kako se naša aplikacija obnaša v spletnem okolju. S tem se torej izognemo uporabi spletnega strežnika IIS (*Internet Information Services*), ki je Microsoftov primarni spletni strežnik, oz. katerega od drugih spletnih strežnikov. Ob zagonu rešitve razvijane v lokalnem datotečnem sistemu se samodejno požene vgrajeni spletni strežnik. Taje namenjen le razvoju spletnih strani in se odziva le na zahteve krajevnega računalnika.
Pod `Language` izberemo programski jezik, v katerem želimo programirati. V tem razvojem okolju lahko izbiramo le med jezikoma Visual Basic in Visual C#, čeprav je v splošnem jezikov .NET še veliko več.



Slika 6 - Odpiranje novega projekta v Visual Web Developer 2005

Po opravljenih nastavitvah in kliku na gumb OK, se odpre okolje, v katerem lahko začnemo z izdelovanjem spletne aplikacije.



Slika 7 - Okolje Visual Web Developer 2005

Okolje je sestavljeno iz več podoken, ki so nam pri izdelavi spletne strani v pomoč. Najbolj pomembna okna so Glavno okno, Orodjarna (*Toolbox*), Pregledovalnik rešitve (*Solution Explorer*) in okno z lastnostmi (*Properties*). V glavnem oknu lahko delamo v načrtovalnem načinu (zavihek Načrtovalni način (*Design*)) in v načinu urejanja kode (zavihek Urejevalnik kode (*Source*)). V

načrtovalnem načinu delamo, ko razporejamo gradnike po spletnih straneh in nastavljamo njihove lastnosti, v načinu urejanja kode pa, ko pišemo ustrezno programsko kodo.

Načrtovalni način

V načrtovalnem načinu izdelamo zunanji videz spletne strani. Na spletno stran postavljamo gradnike in jim določamo različne lastnosti. To je le vizualni videz spletne strani in klik na gradnik ne sproži dogodka (o dogodkih več v nadaljevanju na strani 35).

Urejevalnik kode

Urejevalnik kode je pogled, v katerem urejamo kodo spletne strani. Ta se shrani na datoteko s končnico `.aspx`. To je navadna tekstovna datoteka, ki vsebuje hipertekstne elemente in opise gradnikov. Vsaka spletna stran vsebuje tudi napotke, ki jih bomo spoznali v nadaljevanju poglavja. V tej datoteki je lahko vključena tudi programska koda, ki se nahaja znotraj značk `<script runat="server">` in `</script>`. Bolj priročna pa je uporaba tako imenovane tehnike kode za spletnim obrazcem (*Code-Behind*).

Koda za spletnim obrazcem

Koda za spletnim obrazcem je posebna možnost v okolju ASP.NET, ki razvijalcem spletnih strani omogoča, da sta opis strani v HTML, ki določa videz spletne strani, in programska koda ločeni. Tako imamo programsko kodo shranjeno v ločeni datoteki. Ta ima končnico `.aspx.vb`, če uporabljamo programski jezik Visual Basic in končnico `.aspx.cs`, če pišemo v jeziku C#. Na strani v HTML le povemo, kje ta datoteka, ki vsebuje ustrezno kodo, je.
Takšen način je zelo praktičen, saj omogoča, da lahko oblikovalci spletnih strani delajo neodvisno od programerjev, ki pišejo programsko kodo. Prav tako lahko isto programsko kodo uporabljamo še drugje, saj ni direktno vezana na določeno stran `.aspx`.

Napotki

Vsaka spletna stran, razvita v okolju .NET, vsebuje vsaj en napotek. Z njimi uredimo okolje, v katerem izdelujemo strani. Ponavadi napotke, ki jih tako kot ostale sestavne dele ASP.NET v kodi HTML začnemo z znakoma <% zapišemo na začetku strani, in sicer tako, da pred določene rezervirane besede postavimo znak @. Rezerviranim besedam sledi vrednost, ki jo želimo določiti. Napotki so torej neke vrste sistemske konstante, oziroma natančneje rečeno zbirke sistemskih konstant.

Oblika napotka: `<% @[Napotek] [Lastnost=Vrednost] [Lastnost=Vrednost] ... %>`

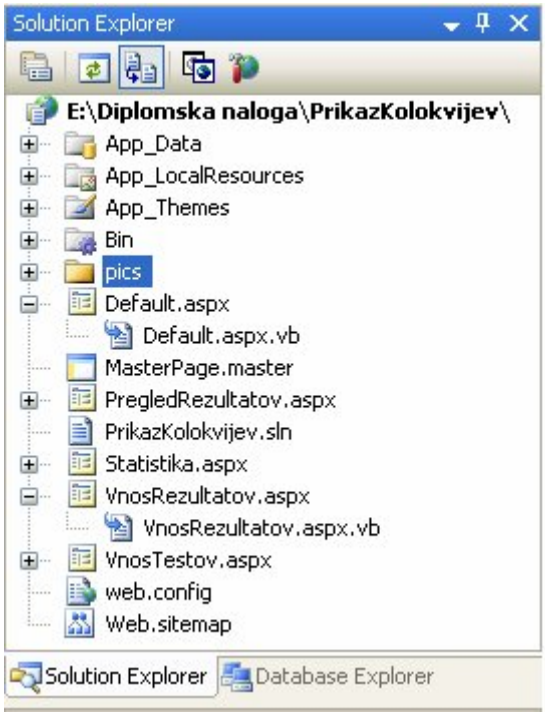
Primer napotka:

```
<% @Page Language="VB" MasterPageFile="~/MasterPage.master"
        CodeFile="PregledRezultatov.aspx.vb" %>
```

V napotku `@Page` so določene lastnosti, ki jih pri prevajanju potrebuje prevajalnik. Z lastnostjo `Language` povemo, kateri programski jezik smo uporabljali pri pisanju te spletne strani. Z lastnostjo `MasterPageFile` določimo predlogo strani (*Master page*), ki ji pripada ta spletna stran (kaj je predloga strani, je razloženo v poglavju o gradnikih na strani 17). Datoteko, v kateri je shranjena programska koda, ki jo uporabljamo na tej strani (tehnika Code Behind), določimo s `CodeFile`. Na voljo je še več drugih lastnosti, ki jih določimo po potrebi.
Z napotkom `@Import` v spletno stran uvozimo imenski prostor. S tem dosežemo, da so vsi razredi tega imenskega prostora tej strani lažje dosegljivi. Ta napotek vsebuje le lastnost `Namespace`.
Uporaben napotek je tudi napotek `@Master`. Določimo mu lahko podobne lastnosti kot napotku `@Page`, le da se uporablja za nastavitve predloge strani.

Pregledovalnik rešitev (Solution Explorer)

Pregledovalnik rešitev je projektno podokno znotraj glavnega okna okolja Visual Web Developer. Običajno se nahaja desno zgoraj. Vsebuje seznam datotek in drugih elementov, ki sestavljajo projekt.



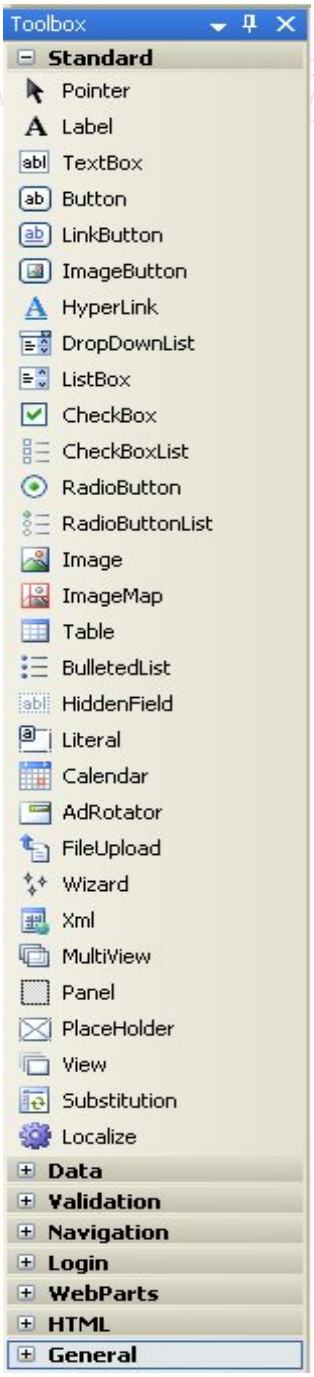
Slika 8 - Solution Explorer

Omogoča nam izbor datoteke, katere vsebina se prikaže v glavnem oknu. V pregledovalniku rešitve tudi dodajamo ali odstranjujemo nove in že izdelane strani, slike, teme … skratka vse vire, ki jih potrebujemo pri razvoju spletne rešitve.
V tem podoknu najdemo tudi datoteko `web.config`. To je datoteka XML, ki vsebuje različne nastavitve spletnega strežnika, ki naj veljajo pri določenem projektu. To so na primer način avtentikacije uporabnikov, povezovalni nizi, uvoženi imenski prostori … . V enem projektu imamo lahko več datotek `web.config`, vendar vsako v svoji podmapi. Nastavitve v datoteki se nanašajo na mapo, kjer je ta datoteka in na vse njene podmape. Nastavitve v `web.config` v podmapah »prekrijejo« nastavitve iz nadmape.

Orodjarna

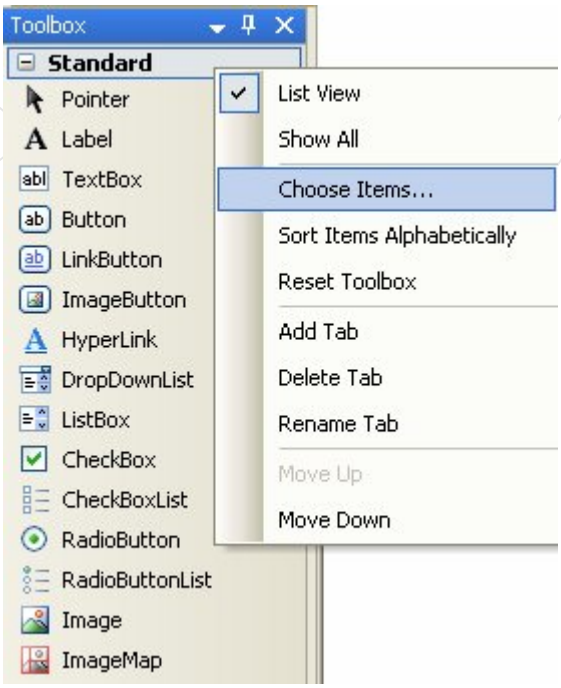
Orodjarna (*toolbox*) vsebuje različne gradnike in komponente, ki jih lahko postavimo na formo. Ponavadi se nahaja na levem delu glavnega okna.
Gradniki so programske komponente. Pravimo jim tudi kontrole ali kontrolniki. Izvirajo iz razreda `Control`, vsak posamezen gradnik, ki ga postavimo na spletno stran, pa je objekt tega razreda.
Gradnike, kot so besedilna polja, gumbe, izbirne gumbe in spustna polja, lahko izberemo v orodjarni in jih povlečemo na obrazec.
Do orodjarne lahko pridemo bodisi iz menija z izbiro možnosti `View` | `Toolbox`, bodisi s pritiskom kombinacije tipk ***Ctrl* + *Alt* + *X*** ali pa kliknemo na ikono `Toolbox`  v orodni vrstici *Standard.*

V razvojnem okolju Visual Web Developer se, ponavadi na levi strani načrtovalnega načina, prikaže okno z najbolj uporabnimi gradniki, ki so razdeljeni v različne skupine (Standard, Data, Validation,…):



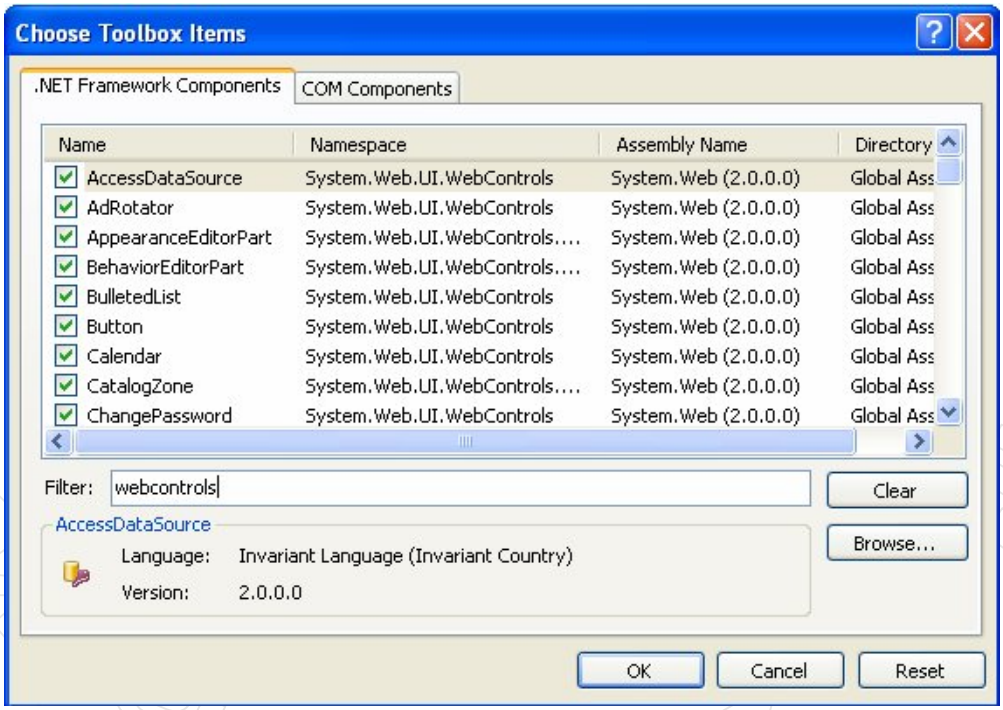
Slika 9 - Orodjarna

V orodjarni na začetku niso prikazani vsi gradniki, ki jih imamo možnost uporabljati. Tiste, ki jih še potrebujemo, v orodjarno dodamo tako, da se znotraj orodjarne postavimo na kartico, kamor želimo dodati gradnik (npr. na kartico `Standard`). Kliknemo desni gumb na miški in izberemo možnost `Choose items`:



Slika 10 – Dodajanje gradnikov v orodjarno

Izberemo zavihek `.NET Framework Components` in odkljukamo gradnike, ki jih želimo imeti prikazane v orodjarni. Imamo tudi možnost izbire filtra, s katerim prikažemo le gradnike, ki pripadajo določenemu imenskemu prostoru. Npr. če želimo izbirati le med gradniki imenskega prostora `WebControls`, v besedilno polje vpišemo besedo `webcontrols`.

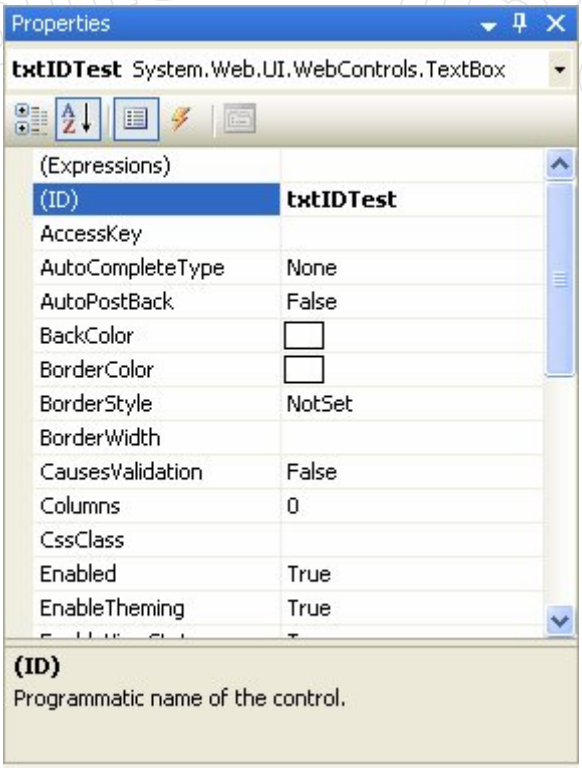


Slika 11 - Okno za dodajanje / odstranjevanje gradnikov iz orodjarne

Lastnosti

V načrtovalnem načinu gradnikom lastnosti spreminjamo na enostaven način . Ko z miško izberemo določen gradnik, se lastnosti tega gradnika prikažejo v podoknu Lastnosti. V prvem stolpcu okna so imena lastnosti, v drugem pa trenutne vrednosti teh lastnosti. Če kliknemo na določeno lastnost,

imamo možnost, da jo spremenimo. Če sprememba vpliva na videz gradnika, to takoj vidimo na delovni površini.



Slika 12 – Lastnosti

Med lastnostmi posebej omenimo lastnost `ID`. Gre za poimenovanje gradnikov. Vsi gradniki na isti spletni strani morajo imeti različna imena, torej različno vrednost lastnosti `ID`. V programski kodi se z ID sklicujemo na gradnik. Ko gradnik postavimo na okno, mu sistem sam priredi ime. Ponavadi je to ime gradnika in številka. Ker pa nam ta imena običajno ne pomenijo kaj dosti, tistim gradnikom, na katere se bomo pogosteje sklicevali, to lastnost nastavimo po svoje.

Strežniški gradniki (*Server controls*)

Do strežniških gradnikov lahko dostopamo preko programske kode in vsebujejo lastnosti in metode, ki jih lahko programiramo. Da je gradnik strežniški, povemo z lastnostjo `runat`, ki ima vrednost `"server"`. Kot vsi drugi gradniki imajo tudi ti obvezno lastnost `ID`, s katero gradnik poimenujemo.

Primer strežniškega spletnega gradnika:

```
<asp:Button ID="btnRezultati" runat="server" Height="50px"
            Style="z-index: 106; left: 343px; position: absolute"
            Text="PRIKAŽI REZULTATE" />
```

Z zgornjo kodo smo naredili strežniški gradnik tipa `Button`. Poimenovali smo ga `btnRezultati` in mu z lastnostjo `Style` določili videz. Z lastnostjo `Text` smo povedali, kaj naj na tem gradniku piše (`PRIKAŽI REZULTATE`).

Slika 13 - Primer strežniškega gumba

Če bi želeli, da se ob kliku na gumb tekst na gumbu spremeni v tekst `REZULTATI PRIKAZANI`, to storimo programsko z naslednjo kodo:

```
Protected Sub btnRezultati_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles btnRezultati.Click
   btnRezultati.Text = "REZULTATI PRIKAZANI"
End Sub
```

Ob kliku na gumb se tekst spremeni:

Slika 14 – Gumb po kliku

Gradniki se odzivajo na dogodke in lahko tudi sami sprožijo različne dogodke. O tem bomo več povedali v naslednjem poglavju. Tu povejmo le to, da pri večini gradnikov za del dogodkov, ki se lahko zgodijo (npr. klik na gumb, izbira datuma na koledarju, izbira elementa v spustnem seznamu ...) napišemo ustrezno metodo, kjer povemo, kako naj se gradnik odzove na ta dogodek.

Strežniške gradnike delimo na strežniške hipertekstne gradnike in na strežniške spletne gradnike. Prvi so namenjeni kot nadomestilo za gradnike, ki jih poznamo predvsem iz obrazcev v HTML (gumbi, vnosne vrstice, oznake, ..), drugi pa so novi gradniki, ki nam omogočajo dodatne možnosti.

Hipertekstni gradniki (HTML controls)

Na prvi pogled so strežniški hipertekstni gradniki videti enako kot gradniki, ki jih v HTML uporabljamo z značko `input` z izjemo tega, da imajo še lastnost `runat="server"`. Začnejo se torej z `<input` in ne z `<asp` kot ostali strežniški gradniki.
Lastnost `runat="server"` jih naredi programljive. Zato jih obravnavamo kot strežniške gradnike. To pomeni, da nanje lahko vplivamo s kodo, ki se izvaja na spletnem strežniku, v nasprotju z običajnimi gradniki iz HTML, ki se obravnavajo v samem brskalniku. Običajnim gradnikom iz HTML med samim izvajanjem aplikacije ne moremo spreminjati lastnosti in nad njimi izvajati metod, s strežniškimi hipertekstnimi gradniki pa to lahko počnemo.

Primer gumba kot značke HTML:

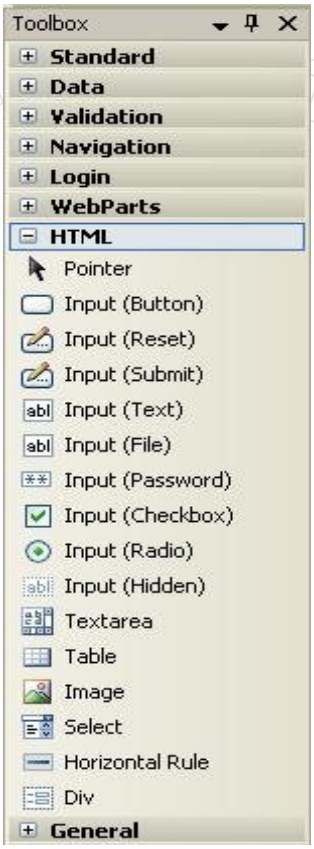
```
<input id="btnZnacka" type="button" value="button" />
```

Primer strežniškega hipertekstnega gumba:

```
<input id="btnStrezniski" type="button" value="button" runat="server" />
```

Koda v HTML je praktično enaka, le da prvi gradnik ne vsebuje lastnosti `runat`. To pomeni, da gre za standardni hipertekstni gradnik. Zato mu lastnosti po prikazu v brskalniku ne moremo spreminjati .

V primeru, da hipertekstni gradnik povlečemo na obrazec iz orodjarne v načrtovalni način (*design view*), še nima dodane lastnosti `runat`. Če ga torej želimo uporabiti kot strežniški gradnik, mu moramo to lastnost dodati naknadno. Če tega ne storimo, gre za standardni hipertekstni gradnik, ki mu v kodi ne moremo spreminjati lastnosti.



Slika 15- Hipertekstni gradniki v orodjarni

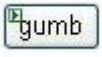
Primer gumba:

```
<input id="btn" type="button" value="gumb" />
```

V taki obliki gre za običajni gradnik, kot ga pozna HTML. V strežniški gradnik ga spremenimo tako, da se v načrtovalnem načinu z miško postavimo nanj, kliknemo desni gumb na miški in v meniju izberemo možnost `Run As Server Control`. Druga možnost je, da gremo v urejevalnik kode in tam na ustrezno mesto dopišemo lastnost `runat="server"`:

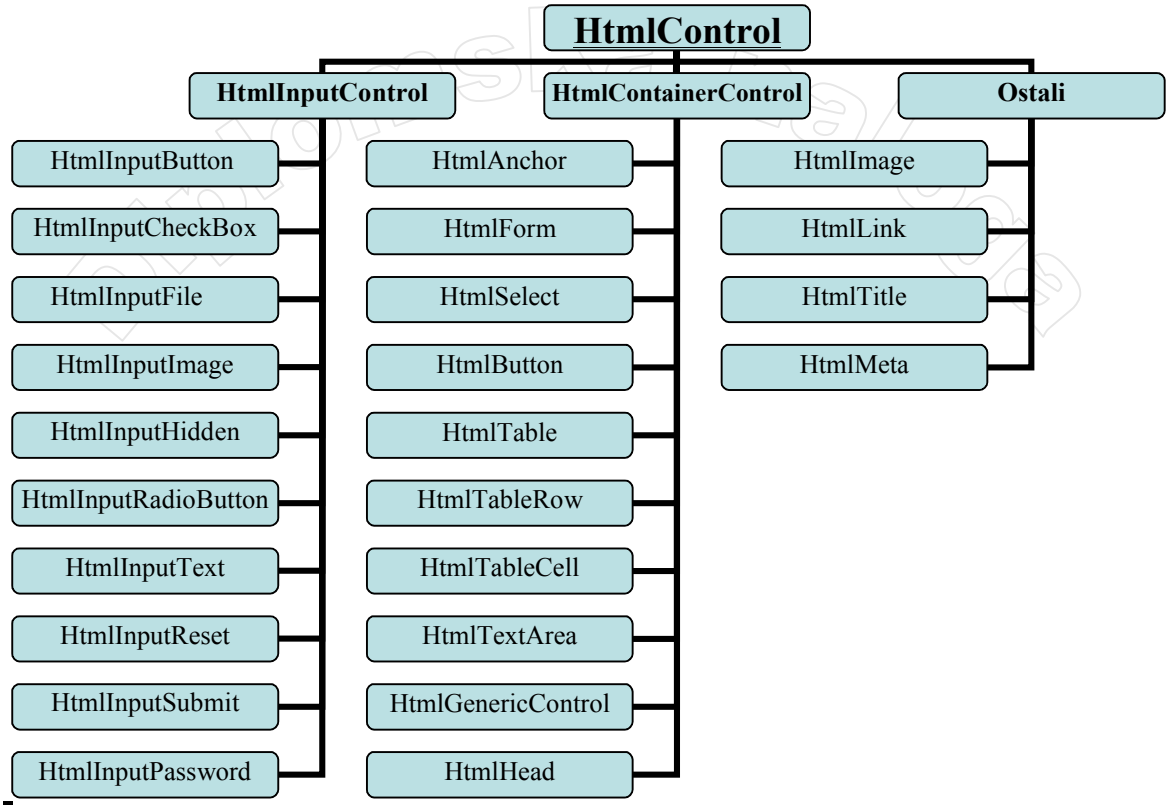
```
<input id="btn" runat="server" type="button" value="gumb" />
```

V načrtovalnem načinu strežniški gradnik prepoznamo po zelenem trikotniku v levem zgornjem kotu gradnika.



Slika 16 - Primer strežniškega gradnika

Večino hipertekstnih strežniških gradnikov lahko razporedimo v dve skupini – gradnike, ki vsebujejo druge gradnike (*container controls*) in vnosne gradnike (*input controls*). Nekaj gradnikov ne spada v nobeno od teh dveh skupin. Taki so npr. gradniki `HtmlImage`, `HtmlLink`, `HtmlMeta`, `HtmlTitle`. V okolju ASP.NET ti gradniki predstavljajo dvojnike značk `<img>`, `<link>`, `<meta>` in `<title>`.



Slika 17 - Delitev hipertekstnih gradnikov

Spletni gradniki (Web controls)

Ker so hipertekstni gradniki večinoma nadomestek ustreznih značk iz HTML, med njimi ne najdemo bolj kompleksnih gradnikov, kot so npr. koledar, gradniki za preverjanje, podatkovni gradniki … . Prav tako bi včasih potrebovali možnost, da bi značkam iz HTML (oziroma hipertekstnim gradnikom kot njihovim nadomestkom) dodelili še določene druge lastnosti. V ta namen nam služijo strežniški gradniki, ki jih imenujemo spletni gradniki.
Spletne gradnike v kodi prepoznamo po tem, da se njihova značka začne z `<asp` .
Del teh gradnikov je zelo podobnih hipertekstnim gradnikom in jih lahko uporabljamo namesto njih. Imajo pa več dodatnih lastnosti in metod, s katerimi lahko še dodatno spreminjamo njihov videz in delovanje. Tako hipertekstnim spletnim gradnikom ne moremo preko programske kode določiti na primer barve ozadja (lastnost `BackColor`), barvo teksta (`ForeColor`) … spletnim gradnikom pa. Njihova pomanjkljivost je ta, da je koda zaradi njihove kompleksnosti lahko nekoliko počasnejša kot če uporabimo hiperteksne gradnike.
Drugi del spletnih gradnikov pa sestavljajo posamezni zapletenejši gradniki kot so koledar, gradniki za preverjanje in podobno.

Razred WebControl

Razred `WebControl` je nadrazred vseh spletnih gradnikov, sam pa deduje iz razreda `Control`. V njem je definiranih mnogo lastnosti in metod. Ker te lastnosti podedujejo ostali spletni gradniki, si jih oglejmo kar tukaj, saj veljajo za vse spletne gradnike.

Nekaj lastnosti v razredu WebControl:

- **AccessKey**: S to lastnostjo določimo tako imenovano hitro tipko (*hot-key*). V lastnosti je zapisana črka, ki ob pritisku skupaj s tipko ***Alt*** povzroči, da se izbira (fokus) prenese na ta gradnik. To pomeni, da je sedaj izbran ta gradnik (gumb, vnosno polje ...). Če je to vnosno polje, se bodo znaki, ki jih takoj nato tipkamo, vnašali v to vnosno polje, če je to gumb, se bo ob pritisku na tipko Enter izvedel dogodek klik ...

- **BackColor**: V tej lastnosti je v obliki niza v šestnajstiškem sistemu zapisana koda barve ozadja gradnika.
- **Enabled**: Lastnost nam pove, ali je gradnik omogočen. Ko je gradnik omogočen, se odziva na dogodke.
- **Font**: S to lastnostjo nastavimo lastnosti pisave na gradniku (velikost, družino pisave, barvo …).
- **ForeColor**: Lastnost pove barvo ospredja, ki je v gradniku največkrat barva teksta.
- **Height**: Preko te lastnosti nastavimo višino gradnika.
- **TabIndex:** S tem številom določimo zaporedje, po katerem se s tipko ***TAB*** premikamo med posameznimi gradnikih.
- **ToolTip:** Tu določimo besedilo, ki se prikaže, ko gre uporabnik z miško preko gradnika.
- **Width:** S to lastnostjo določimo širino gradnika.

Najbolj uporabni spletni gradniki

V tem razdelku si bomo ogledali nekaj najbolj uporabnih spletnih gradnikov. Pri vsakem bomo navedli njegove najbolj tipične lastnosti. Seveda pa ima vsak gradnik še številne druge lastnosti. A ker jih manj uporabljamo, jih ne bomo navajali.

Oznaka *(Label)*

Oznaka je osnovni gradnik, ki ga uporabljamo za prikaz besedila na spletnem vmesniku. Prikazan tekst nadzorujemo z lastnostjo `Text`. V ASP.NET 2.0 so oznaki dodali še koristno novost. Po novem ima gradnik oznaka tudi lastnost `AssociatedControlID`. Ta omogoča, da je ob kliku na to oznako, oziroma ob uporabi k oznaki pripadajoče hitre tipke, izbran (dobi fokus) v tej lastnosti navedeni gradnik (gumb, vnosno polje ...).
Poglejmo primer uporabe. Pri tem bomo uporabili tudi gradnik besedilno polje, ki ga bomo opisali takoj za tem. Denimo, da imamo pred besedilnim poljem še oznako:

Slika 18 - Oznaka in besedilno polje

Ta dva gradnika smo ustvarili z

```
<asp:Label ID="lblIdTest" runat="server" AssociatedControlID="txtIdTest"
           Text="ID testa:" AccessKey="I"></asp:Label>
<asp:TextBox ID="txtIdTest" runat="server"></asp:TextBox>
```

Z `AssociatedControlID` smo oznako povezali z besedilnim poljem. Če kliknemo na napis na oznaki ali pa če pritisnemo na ***Alt+I***, se fokus prenese na besedilno polje. Zato lahko takoj zatem v besedilno polje vtipkamo ustrezno besedilo.

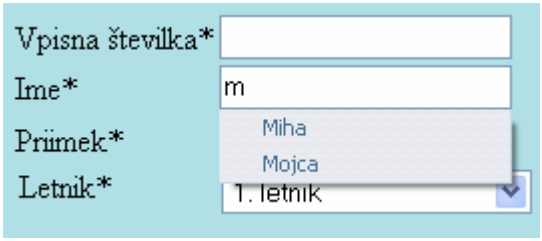
Besedilno polje (*TextBox*)

Besedilno polje je v spletnih obrazcih zelo pogosto uporabljen gradnik. Kot pove že ime, je to polje, v katerega uporabnik lahko vpiše besedilo. Do tega besedila dostopamo preko lastnosti `Text`. Uporabljamo ga lahko kot enovrstično ali kot večvrstično polje. Če ga želimo uporabljati kot večvrstičnega, lastnosti `TextMode` pripišemo vrednost `MultiLine`.
V besedilna polja vnašamo tudi gesla. Takrat lastnost `TextMode` nastavimo na `password`. Besedilo se potem zakodira, uporabnik pa ob vnosu namesto natipkanih znakov vidi zvezdice (*).
Če želimo uporabniku omogočiti, da čim hitreje izpolni določene podatke na naši spletni strani, mu pri tem lahko pomagamo z lastnostjo `AutoCompleteType`. Z njeno pomočjo ustvarjamo seznam vrednosti, ki jih uporabnik lahko vpiše v besedilno polje in jih ob ponovnem vnosu lahko le izbere (glej Slika 19) .

Ko želimo pri nekem besedilnem polju uporabljati samodejno dopolnjevanje, mu kot lastnost `AutoCompleteType` dodelimo eno od kategorij (`FirstName`, `LastName`, `Email`, `Company`, `Search` …). Kategorijo najlažje izberemo iz kombiniranega polja v podoknu Lastnosti, kjer imamo naštete vse možnosti. Svojih kategorij žal ne moremo tvoriti. Med obstoječimi nam najbolj pogosto pridejo prav `FirstName` (ime), `LastName` (priimek), `Email` (elektronski naslov), `Gender` (spol), `Cellular` (mobilni telefon). Seveda pa jih je na voljo še več (skupaj 30).
Vsa besedilna polja z isto izbrano kategorijo si delijo isti seznam že vnesenih vrednosti. To velja tudi, če se nahajajo na različnih spletnih straneh.
Privzeta vrednost lastnosti `AutoCompleteType` je `None`. Ta pove, da si to besedilno polje deli seznam s tistimi besedilnimi polji na drugih spletnih straneh, ki imajo enako lastnost `ID`. Seveda na isti strani ne moremo imeti dva taka gradnika, saj dva gradnika na isti strani ne moreta imeti enakega `ID`-ja. Če pa imamo na dveh različnih spletnih straneh besedilni polji, ki imata obe ID na primer enak `txtIme`, si seznam, torej vrednosti, ki jih uporabnik vnese v poljubno od teh dveh besedilnih polj, delita gradnika na obeh straneh. Torej si seznam vnesenih vrednosti delijo vsa besedilna polja na vseh spletnih straneh, ki imajo za lastnost `AutoCompleteType` izbrano isto kategorijo.
To lastnost onemogočimo tako, da izberemo vrednost `Disabled`. S tem preprečimo ustvarjanje seznama.
Vendar se ne smemo preveč zanašati na to lastnost, saj je odvisna od brskalnika, ki ga uporablja uporabnik. Vsi brskalniki namreč ne podpirajo možnosti `AutoComplete`. Prav tako jo uporabnik v svojem brskalniku lahko tudi izklopi.

Primer besedilnega polja:

```
<asp:TextBox ID="txtIme" runat="server"  AutoCompleteType= "FirstName">
</asp:TextBox>
```



Slika 19 - Uporaba lastnosti AutoCompleteType

Ko je uporabnik v besedilno polje zraven oznake Ime vpisal črko m, so se v seznamu izpisala vsa tista imena, ki se začno na črko M in so bila pred tem enkrat vpisana v to besedilno polje oz. v katerokoli besedilno polje na drugih spletnih straneh, ki imajo lastnost `AutoCompleteType` nastavljeno na `FirstName`.

Gumb (*Button*)

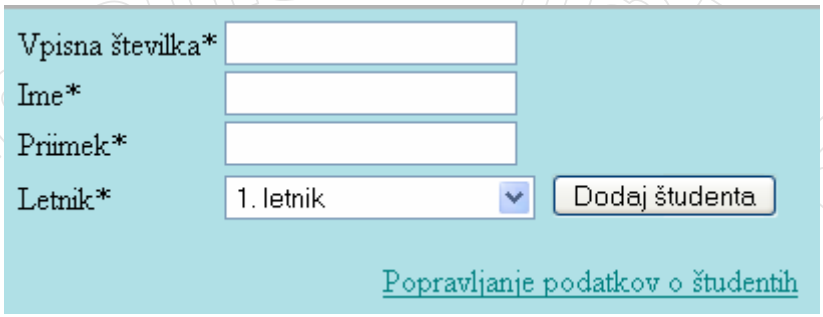
Na spletnih obrazcih pogosto srečamo tudi gumbe. Navadni gumb ASP ustvarimo z

```
<asp:Button ID="btnVnesiTest" runat="server" Text="VNESI TEST"/>
```

Slika 20 – Gumb

Gumbi ASP se od hipertekstnih razlikujejo po tem, da jim lahko določimo veliko več lastnosti, kot so npr. lastnosti `ForeColor`, `CssClass`, `SkinID` …
Poleg gumbov, ki jih naredimo s pomočjo HTML in njihovih strežniških različic v ASP, imamo na voljo tudi povezavne gumbe (`LinkButton`). Ti se od prej omenjenih razlikujejo po tem, da so na videz podobni hiperpovezavi, obnašajo pa se kot gumbi. Pripravni so na obrazcih, kjer je prisotno

veliko število gumbov, saj s tem naredimo stran bolj razgibano. V moji aplikaciji uporabnik klikne na povezavni gumb, kadar želi popravljati podatke:



Slika 21 - Primer povezavnega gumba (spodaj desno)

Tak gumb ustvarimo ga z

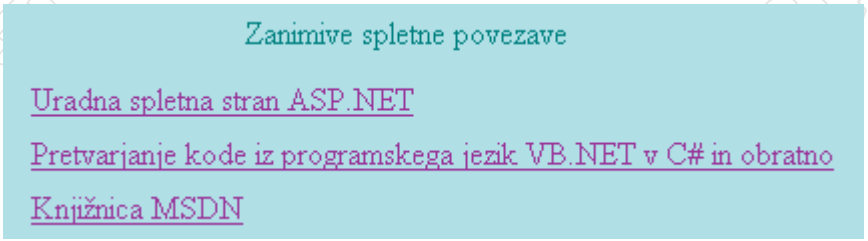
```
<asp:LinkButton ID="lbPopravljanje" runat="server" ForeColor="Teal"
                Width="214px">Popravljanje podatkov o študentih
</asp:LinkButton>
```

Gumb s sliko je še ena od različic gumba. Od navadnega gumba se razlikuje le v tem, da lahko namesto tipičnega izgleda gumba, ki ga vidimo na večini obrazcev, zanj uporabimo sliko. Kje se nahaja slika, ki jo uporabimo za gumb, povemo z lastnostjo `ImageUrl`.
Ob kliku na gumb se sproži dogodek `Click`. Sicer ni nujno, da napišemo odzivno kodo za vsak dogodek, a običajno pri vsakem gumbu napišemo, kaj naj se zgodi ob dogodku klik. To pomeni, da v kodi, ki pripada gumbu, napišemo, kaj se mora zgoditi v trenutku, ko uporabnik klikne na ta gumb.

Med strežniškimi gradniki imamo torej na voljo več različnih gumbov, ki jih lahko uporabimo pri gradnji spletnih strani. Če gumbu ne želimo spreminjati oblikovnih lastnosti, se odločimo za hipertekstni gumb, v nasprotnem primeru pa za gumb ASP. V primeru, da želimo na gumbu imeti sliko, izberemo gumb s sliko. Na voljo pa imamo tudi povezavne gumbe, ki so po videzu podobni hiperpovezavi.

Hiperpovezava (HyperLink)

Hiperpovezava omogoča uporabniku enostavno sprehajanje med različnimi spletnimi stranmi. Z lastnostjo `NavigateUrl` določimo naslov strani v internetu, kamor se prestavimo ob kliku na ta gradnik.
Od navadne značke HTML (torej značke A), ki ima isti končni učinek, se razlikuje v tem, da ji lahko med izvajanjem aplikacije na primer zamenjamo lastnost `Text`, spremenimo barvo besedila …
Za razliko od povezavnega gumba ne moremo določiti obnašanja hiperpovezave pri kliku nanjo, ampak klik vedno povzroči, da se prestavimo na stran, kamor kaže ta hiperpovezava.
Zanimiva lastnost hiperpovezave je tudi, da lahko namesto prikazanega teksta uporabimo sliko, katere lokacijo določimo z lastnostjo `ImageUrl`. Stran, na katero naj se ob kliku brskalnik preusmeri, pa še vedno določimo za lastnostjo `NavigateUrl`.



Slika 22 - Primeri hiperpovezav

Zgornjo sliko dobimo s pomočjo kode

```
<asp:Label ID="lblPovezave" runat="server" ForeColor="Teal"
           Text="Zanimive spletne povezave"></asp:Label>
<asp:HyperLink ID="hlAsp" runat="server"
               NavigateUrl="http://www.asp.net/">
               Uradna spletna stran ASP.NET</asp:HyperLink>
<asp:HyperLink ID="hlPretvarjanje" runat="server"
               NavigateUrl="http://www.developerfusion.co.uk/utilities/
               convertcsharptovb.aspx">Pretvarjanje kode iz programskega
               jezik VB.NET v C# in obratno</asp:HyperLink>
<asp:HyperLink ID="hlKnjiznica" runat="server"
               NavigateUrl="http://msdn2.microsoft.com/en-
               us/library/default.aspx">Knjižnica MSDN</asp:HyperLink>
```

Potrditveno polje (CheckBox)

Potrditvena polja uporabljamo, kadar želimo uporabniku dati na izbiro potrditev določene možnosti. Gradnik je v osnovi le prazen kvadratek. Z lastnostjo `Text` ga opremimo z besedilom, ki razloži pomen kvadratka. Lastnostjo `Checked`, ki sprejme vrednost tipa `boolean`, pove, ali je polje na začetku odkljukano ali ne. Kasneje v samem programu pa z vpogledom v to lastnost preverimo, če je uporabnik to polje odkljukal ali ne:

```
If chkKolokvij.Checked Then ' uporabnik je izbral kolokvije
   ' Koda
End If
```

Potrditveno polje ustvarimo z:

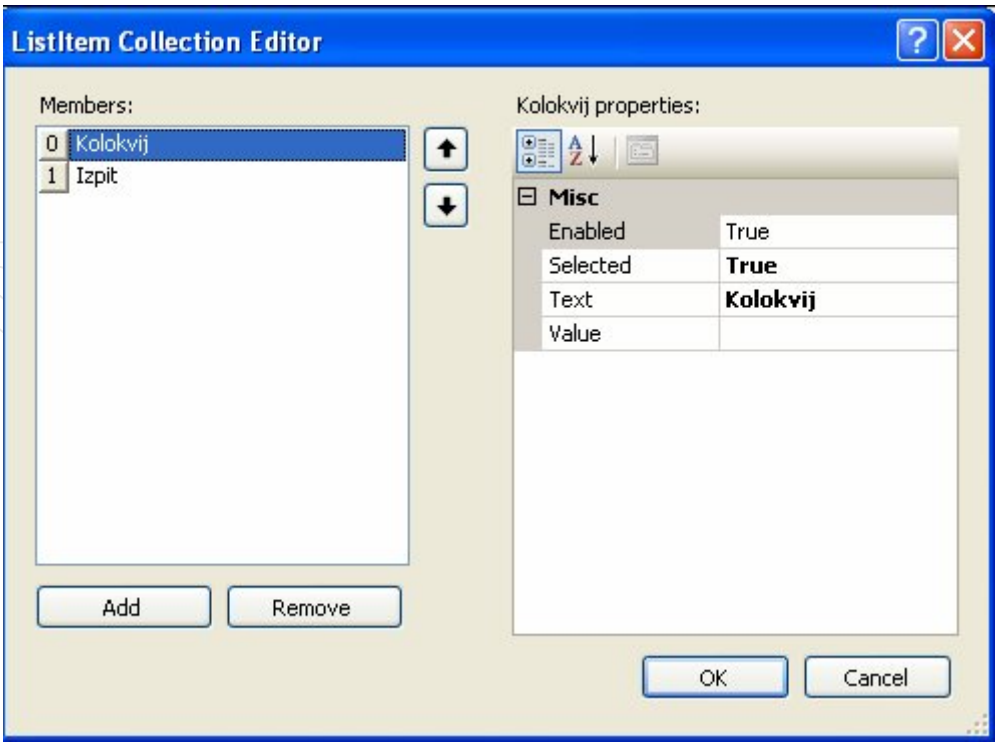
```
<asp:CheckBox ID="chkKolokvij" runat="server" Checked="True"
              Text="Kolokvij:" TextAlign="Left" />
```

Slika 23 - Potrditveno polje

Ustvarili smo potrditveno polje, ki ima besedilo poravnano levo od kvadratka. Kvadratek je odkljukan, kar smo dosegli tako, da smo lastnost `Checked` nastavili na `True`.

Seznam izbirnih gumbov (RadionButtonList)

Seznam izbirnih gumbov uporabljamo, kadar želimo uporabniku dati na izbiro več možnosti, izbrati pa sme le eno samo. Najlažji način za dodajanje elementov v seznam izbirnih gumbov je s pomočjo čarovnika. Gradnik povlečemo iz orodjarne in ga postavimo na obrazec. Kliknemo mali trikotnik v zgornjem desnem kotu in izberemo možnost `Edit Items…` S klikom na gumb `Add` dodamo nov element. Z lastnostjo `Text` določimo besedilo, ki se izpiše zraven izbirnega gumba. Lastnosti `Selected` določimo vrednost `True`, če naj bo gumb izbran, oz. `False` v nasprotnem primeru. Seveda je treba upoštevati, da je v seznamu sočasno lahko izbran le en gumb.
Določimo lahko tudi lastnost `Value`, ki pove, kakšna je vrednost gumba, če je izbran. Več o tej lastnosti bomo povedali v naslednjem podpoglavju Spustni seznam.



Slika 24 - Dodajanje izbirnih gumbov v seznam

V gornjem seznamu izbirnih gumbov sta dva izbirna gumba, med katerima je izbran izbirni gumb z napisom Kolokvij:

Slika 25 - Seznam izbirnih gumbov

Ali je izbirni gumb odkljukan ali ne, lahko preverimo na dva načina:

```
If rblTest.Items(0).Selected Then ' Je gumb z indeksom 0 izbran?
    ' Koda
End If
```

ali

```
If rblTest.SelectedIndex = 0 Then ' Ali ima izbrani gumb indeks 0?
    ' Koda
End If
```

Pri obeh načinih preverjamo, če je izbran prvi element. Razliko med načinoma pojasnjujeta komentarja.

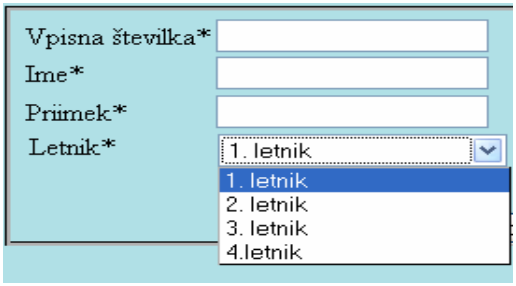
Spustni seznam (DropDownList)

Spustni seznam je zelo podoben seznamu izbirnih gumbov, le da v uporabniškem vmesniku na začetku vidimo le eno izbiro. Tako sam gradnik zasede manj prostora. Ko kliknemo na puščico na desni strani gradnika, se odpre seznam možnosti, med katerimi lahko izbiramo.

Primer spustnega seznama:

```
<asp:DropDownList ID="ddlLetnik" runat="server">
   <asp:ListItem Value="1">1. letnik</asp:ListItem>
   <asp:ListItem Value="2">2. letnik</asp:ListItem>
   <asp:ListItem Value="3">3. letnik</asp:ListItem>
   <asp:ListItem Value="4">4. letnik</asp:ListItem>
</asp:DropDownList>
```

Slika 26 - Spustni seznam



Slika 27 - Spustni seznam med izbiranjem

Polnimo ga na isti način kot seznam izbirnih gumbov. Z lastnostjo `Text` povemo opis posameznega elementa, ki bo viden pri izbiranju. Lastnost `Value` določa njegovo dejansko vrednost, ki jo uporabimo med programiranjem. Poglejmo primer uporabe.
Polje Letnik je v bazi podatkov določeno kot tipa število. Zato lastnosti `Value` posameznih izbir priredimo na `1, 2 …,` v samem opisu pa lastnost `Text` nastavimo na `1. letnik, 2. letnik ….`

Slika 28 - Čarovnik za dodajanje elementov v spustni seznam

Ker je izbrani element sedaj po svoji vrednosti število, jo lahko kot tako uporabimo. Denimo, da moramo s pomočjo stavka SELECT iz podatkovne baze izbrati vse študente določenega letnika. Letnik uporabnik izbere iz spustnega seznama, kot smo ga naredili na način, ki ga prikazuje Slika 28.

```
Dim strStudent As String = "SELECT IME, PRIIMEK " & _
                           "FROM STUDENT " & _
                           "WHERE LETNIK = " & ddlLetnik.SelectedValue
```

Koledar (Calendar)

Gradnik koledar prikazuje mesečni koledar in omogoča, da s pomočjo miške izberemo datum. Pri tem se lahko premikamo nazaj in naprej po mesecih in letih. Po mesecih v koledarju se premikamo s pomočjo puščic v zgornji vrstici. Če kliknemo na letnico, pa lahko izberemo tudi leto. Ko imamo odprt pravi mesec, kliknemo na želen dan in ta dan bo izbran. Z lastnostjo `SelectedDate` ugotovimo datum, ki je na koledarju izbran.
Stil koledarja (način prikaza mesecev, dni, izbrane barve, obliko puščic za premikanje ...) lahko tudi spreminjamo. To najlažje storimo v načrtovalnem načinu. Postavimo se na koledar in kliknemo na majhen trikotnik v desnem zgornjem kotu koledarja ter izberemo možnost `AutoFormat`. Odpre se novo okno, kjer izberemo stil, ki ustreza naši aplikaciji.

Primer koledarja iz moje spletne aplikacije:

```
<asp:Calendar ID="Calendar1" runat="server">
   <TitleStyle BackColor="CornflowerBlue" />
</asp:Calendar>
```

Slika 29 – Koledar

Gradnik za izbiranje in nalaganje datotek (FileUpload)

Gradnik nam omogoča, da na strežnik prenesemo določeno datoteko. Sestavljen je iz besedilnega polja, ki mu sledi gumb (glej Slika 30).. S klikom na gumb se odpre standardno pogovorno okno za izbiranje datotek

Primer gradnika za izbiranje in nalaganje datotek:

```
<asp:FileUpload ID="fuTest" runat="server" />
```

Slika 30 - Gradnik za izbiranje datotek in gumb

Sam gradnik omogoča le, da izberemo datoteko, za vse ostalo pa je treba poskrbeti s programsko kodo. Zato običajno zraven tega gradnika dodamo še gumb, s pomočjo katerega izbrano datoteko prenesemo na strežnik. Na sliki (Slika 30) je to gumb z napisom Dodaj. Ob kliku nanj naj se izvede naslednja koda

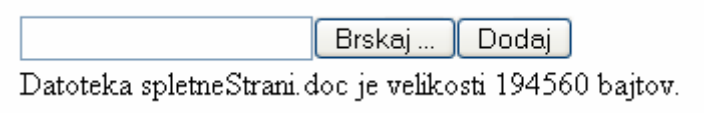
```
Protected Sub btnDodaj_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnDodaj.Click
   If fuIzberi.HasFile Then 'v gradniku je vpisana datoteka
      fuIzberi.SaveAs("E:\\Diplomska naloga\\ " & fuIzberi.FileName)
      lblPodatki.Text = "Datoteka " & fuIzberi.PostedFile.FileName & " & _
      je velikosti " & fuIzberi.PostedFile.ContentLength & " byte-ov."
   Else
      MsgBox("Nobena datoteka ni izbrana.", MsgBoxStyle.Exclamation)
   End If
End Sub
```

Z izvedbo te kode dosežemo, da se datoteka prenese na strežnik. V primeru uspešnega prenosa se s pomočjo oznake lblPodatki izpišejo še podatki o preneseni datoteki. Če pa uporabnik ni izbral nobene datoteke in je le kliknil na gumb Dodaj, se prikaže opozorilno okno z ustreznim obvestilom.
Seveda uporabniku ni potrebno klikniti na gumb Brskaj ..., ampak lahko pot do datoteke (če jo seveda pozna) vpiše neposredno v besedilno polje in potem klikne na gumb Dodaj.

Razlaga kode:
Najprej preverimo, če je bila datoteka res izbrana (pogledamo lastnost `HasFile`). Če ni bila izbrana nobena datoteka, se prikaže sporočilo:

Slika 31 – Sporočilo o napaki

Izbrana datoteka se s pomočjo metode `SaveAs` shrani na strežnik. Metoda sprejme parameter tipa niz, ki nam pove mesto, kamor želimo shraniti datoteko. V našem primeru se bo datoteka shranila v mapo `E:\Diplomska naloga\`. V kodi smo napisali \\, ker \ napoveduje posebni znak znotraj niza in moramo zato napisati dva. Datoteka ohrani enako osnovno ime, kot ga ima v originalu.
Na strani je tudi oznaka, vendar ta na začetku ni vidna. Ko se stran naloži, ima namreč lastnost `Text` nastavljeno na prazen niz.
S pomočjo lastnosti `FileName` in `ContentLength` pridobimo podatke o datoteki, ki smo jo naložili na strežnik in te podatke izpišemo na to oznako.
V primeru na sliki se izpiše tekst, ki nam pove, da je ime naložene datoteke `spletneStrani.doc` in da je njena velikost 194.560 bajtov.



Slika 32 - Graniki po kliku na gumb Dodaj

Vsebovalnik strani (ContentPlaceHolder)

Pri projektih, ki so sestavljeni iz več spletnih strani, pogosto želimo, da imajo vse strani enotno zgradbo. Takrat si pomagamo s predlogami strani (*Master pages*). Te nam omogočajo, da v aplikaciji določimo skupne dele spletnih strani. To so ponavadi razne slike, meniji, logotipi …
Vsebovalnik strani je vsebovan le v predlogi strani. Z njim določimo prostor, na katerem bo vsaka vsebovana stran (*Content page*) dodala svojo vsebino.

Predloga strani je posebna spletna stran, ki je shranjena v datoteki s končnico `.master`. Sam projekt lahko vsebuje več predlog strani, ki jih lahko tudi gnezdimo (ena predloga strani je vsebovana v drugi).
Vsebovana stran je vsaka spletna stran, ki uporablja predlogo strani. Vsebovani spletni strani določimo predlogo strani s pomočjo napotkov. V napotku `Page` lastnosti `MasterPageFile` dodelimo pot do želene predloge strani:

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master"
    CodeFile="Login.aspx.vb" Inherits="Prijava" title="Untitled Page" %>
```

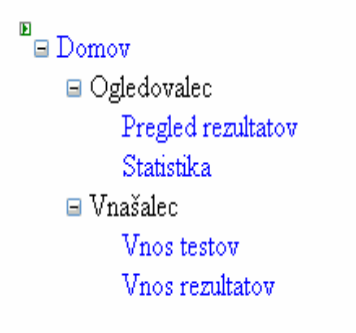
Vsakič, ko odjemalec (spletni brskalnik) zahteva vsebovano stran, strežnik naloži predlogo strani, jo spoji z vsebovano stranjo in rezultat pošlje nazaj odjemalcu.

Če želimo, da je predloga strani razdeljena na več delov, postavimo na spletno stran tabelo. To storimo tako, da v orodni vrstici izberemo `Layout` in nato `Insert Table`. Lahko pa tudi uporabimo že narejene šablone, ki so primerne prav za predloge strani. V tem primeru označimo možnost `Template` in navedemo ustrezno šablono. Za svojo spletno stran sem uporabila šablono `Header and side` in tako razdelila stran na tri dele. Zgoraj je vrstica, v kateri je »ime« spletne strani, na levi strani del za drevesni pogled (več o tem v naslednjem podpoglavju), vse ostalo pa je namenjeno vsebovalniku strani. Zgornji in levi del torej določata nespremenljivo vsebino vseh spletnih strani, ki bodo uporabljale to predlogo.

Slika 33 – Stran, ki ima določeno predlogo strani

Drevesni pogled (TreeView)

Drevesni pogled nam omogoči enostavno navigacijo med spletnimi stranmi, ki sestavljajo spletno aplikacijo. Skupaj z gradnikom izvor podatkov za načrt spletne strani in ustrezno datoteko XML ustvari hiperpovezave do spletnih strani. Zato ga pogosto uporabljamo kot meni spletne aplikacije.
Hierarhijo spletnih strani opišemo v jeziku XML v datoteki `web.sitemap`. Gradnik izvor podatkov za načrt spletne strani (*SiteMapDataSource*) predstavlja vez med datoteko XML in drevesnim pogledom. Drevesni pogled povežemo z gradnikom izvor podatkov za načrt tako, da `ID` podatkovnega drevesnega pogleda pripišemo lastnosti `DataSourceID`, ki pripada drevesnemu pogledu. Gradnik za izvor podatkov sam poišče datoteko `web.sitemap` in podatke poda drevesnemu pogledu. Ta jih prikaže. Ko je ta povezava narejena, lahko datoteko XML spreminjamo. Vse spremembe bodo avtomatsko vidne na drevesnem pogledu.
Največkrat drevesni pogled postavimo na predlogo strani in igra vlogo menija, preko katerega se premikamo po spletnih straneh, ki sestavljajo aplikacijo.



Slika 34 - Drevesni pogled

Datoteka XML za zgornji drevesni pogled je videti kot:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode url="Default.aspx" title="Domov"  >
   <siteMapNode  title="Ogledovalec"  >
      <siteMapNode url="PregledRezultatov.aspx" title="Pregled rezultatov" />
      <siteMapNode url="Statistika.aspx" title="Statistika"  />
      <siteMapNode url="DatumiPreverjanj.aspx" title="Datumi preverjanj"  />
   </siteMapNode>
   <siteMapNode  title="Vnašalec" >
      <siteMapNode url="VnosTestov.aspx" title="Vnos testov"  />
      <siteMapNode url="VnosRezultatov.aspx" title="Vnos rezultatov"  />
  </siteMapNode>
  </siteMapNode>
</siteMap>
```

Znački `<siteMap>` in `</siteMap>` določata začetek in konec dokumenta XML. Znotraj značk `<siteMapNode>` določimo povezave na želene spletne strani. V lastnosti `url` napišemo spletni naslov, kamor nas preusmeri klik na ta element. Samo besedilo, ki bo prikazano kot ime v drevesnem pogledu, pa nam določa lastnost `title`. Če določeno vozlišče (`siteMapNode`) lastnosti `url` nima (kot v našem zgledu npr. tisto z naslovom Vnašalec), ga uporabimo kot naslov za lepši prikaz samega menija,

Slika 35 - Spletna stran, ki vsebuje drevesni pogled

Gradniki za preverjanje *(Validation controls)*

Ti gradniki so namenjeni preverjanju veljavnosti podatkov, ki jih vnese uporabnik. Preverjajo npr. ali določeno polje sploh vsebuje podatek, ali so podatki v določenem obsegu, ali se podatki ujemajo z določenimi vrednostmi … V nasprotnem primeru sprožijo sporočilo o napaki. Gradniki, razen sporočila, ki ga lahko prikažemo ob napačnem vnosu, nimajo vidne podobe – torej med samim izvajanjem niso vidni na uporabniškem vmesniku. Ena pomembnejših lastnosti je lastnost `ErrorMessage`, saj z njo uporabnika opozorimo na napačen vnos podatkov. Več o prikazovanju opozoril je opisano pri opisu gradnika za prikaz opozoril.
Preverjanja se ne zgodijo avtomatsko, ampak jih sproži določen dogodek. V večini primerov je to klik na gumb. Gumbi, povezavni gumb, gumbi s sliko so primeri gradnikov, ki lahko sprožijo dogodek, ki povzroči preverjanje.
Vsak gumb ima privzeto vrednost `CausesValidaton` nastavljeno na `True`. To pomeni, da klik na tak gumb sproži preverjanje. Pri tem se izvedejo vsa preverjanja, kot jih določajo vsi gradniki za preverjanje na tej spletni strani. Če imamo na strani pet gradnikov za preverjanje, se ob kliku na poljuben gumb na tej strani izvede vseh pet preverjanj. Če želimo, da klik na določen gumb ne sproži preverjanja, mu moramo lastnost `CausesValidaton` nastaviti na `False`.
Lastnost `ValidationGroup` nam omogoča, da gradnike za preverjanje razdelimo v skupine. Isto lastnost imajo tudi gumbi. Če gumb dodamo v isto skupino kot gradniki za preverjanje, dosežemo, da se ob kliku na ta gumb sproži preverjanje le tistih gradnikov za preverjanje, ki so v tej isti skupini. Denimo, da kliknemo na gumb, ki ima lastnost `ValidationGroup` nastavljeno na `ObveznaPolja`. `Ob kliku` se bo sprožilo izvajanje preverjanj v tistih gradnikih za preverjanje, ki imajo lastnost `ValidationGroup` prav tako nastavljeno na `ObveznaPolja`. Seveda pa to velja le, če gumbu z postavitvijo lastnosti `CausesValidation` na False nismo onemogočili sprožanje preverjanja.

Nekaj pogostih lastnosti, ki jih določimo gradnikom za preverjanje:

- **ControlToValidate**: Tej lastnosti nastavimo `ID` gradnika, na katerem želimo preveriti uporabnikov vnos.
- **ErrorMessage**: V tej lastnosti napišemo besedilo, ki se bo prikazalo kot sporočilo o napaki, torej kadar bo preverjanje neuspešno.
- **ForeColor**: Določa barvo besedilo v sporočilu o napaki.
- **ValidationGroup**: S to lastnostjo nastavimo skupino za preverjanje, ki ji gradnik pripada.

Na voljo imamo šest tipov gradnikov za preverjanje. Najdemo jih orodjarni v razdelku `Validation`:

Slika 36 – Del orodjarne z gradniki za preverjanje

Gradnik za preverjanje obveznega vnosa (*RequiredFieldValidator*)

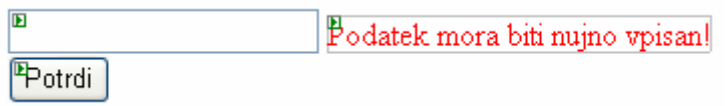
S tem gradnikom preverimo, če je v določenem gradniku (običajno je to besedilno polje) res vpisan podatek. V primeru da podatek ni vpisan, se izpiše opozorilo.

Večkrat besedilnemu polju že vnaprej določimo privzeto vrednost. To je določeno besedilo, s katerim pojasnjujemo, kakšen podatek naj uporabnik vnese ali pa vrednost, ki jo pričakujemo, da jo bo vnesel uporabnik. V tem primeru nas ne zanima, če je vrednost vnesena (saj zagotovo je), ampak če je uporabnik spremenil to (praviloma neuporabno) začetno vrednost. Če želimo preveriti, ali je v polju spremenjena vrednost, lahko z lastnostjo `InitialValue` določimo začetno vrednost, ki je vpisana besedilno polje, ki ga preverjamo. Če je vrednost v vpisanem gradniku ob kliku na gumb s katerim sprožimo preverjanje, še vedno enaka, kot je vrednost v `InitialValue`, se prav tako prikaže opozorilo, kot bi se, če podatek sploh ne bi bil vnesen.

Primer:

Na obrazec dodamo besedilno polje, gumb in gradnik za preverjanje obveznega vnosa in jim določimo lastnosti:

```
<asp:TextBox ID="txtPodatek" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvVpisanPodatek" runat="server"
                            ErrorMessage="Podatek mora biti nujno vpisan!"
                            ControlToValidate="txtPodatek">
</asp:RequiredFieldValidator><br/>
<asp:Button ID="btnPotrdi" runat="server" Text="Potrdi" />
```



Slika 37 – Preverjanje obveznega polja

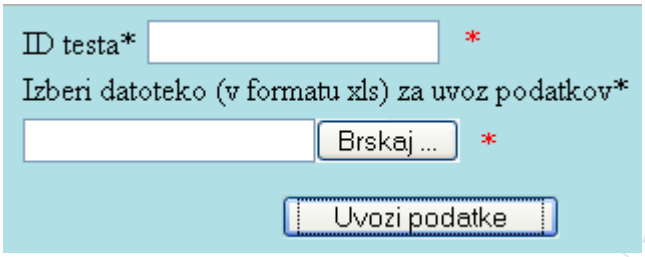
Vidimo, da skupine za preverjanje nismo določali.
Uporabnik na začetku na spletni strani vidi le besedilno polje in gumb. Če v trenutku, ko klikne na gumb Potrdi, podatek ni vpisan, se prikaže tekst, ki uporabnika opozori na manjkajoči podatek.

Primerjalnik vrednosti (*CompareValidator*)

Ta gradnik primerja vrednost, ki jo uporabnik vnese v vnosni gradnik, s konstantno vrednostjo ali z vrednostjo v nekem drugem vnosnem gradniku na isti spletni strani. Vnosni gradniki so besedilna polja, gradniki za izbiranje in nalaganje datotek ... Če bi želeli preveriti lastnosti vnosa in ne le enakost (npr. če je vneseno število sodo), moramo uporabiti drug gradnik za preverjanje, gradnik za preverjanje po meri (*CustomValidator*).
Ta primerjalni gradnik uporabimo za različne vrste preverjanj. Katera vrsta preverjanja se uporabi, je odvisno od nastavljenih lastnosti.
Če želimo primerjati, če je vnesena vrednost enaka vrednosti v nekem drugem vnosnem gradniku, uporabimo lastnost `ControlToCompare`. Z njo povemo `ID` gradnika, katerega vrednost želimo primerjati z gradnikom, ki smo ga nastavili z lastnostjo `ControlToValidate`. Na ta način torej primerjamo, če sta vrednosti dveh gradnikov enaki. Med seboj lahko primerjamo le vrednosti vnosnih gradnikov, ne moremo pa vnesene vrednosti primerjati npr. s tekstom, ki je izpisan na oznaki.
Druga možnost primerjanja je uporaba lastnosti `ValueToCompare`. Če jo nastavimo, se v to lastnost vpisana vrednost primerja z v gradnik vneseno vrednostjo. Izogibati se moramo tega, da bi gradnik sočasno imel nastavljeni tako lastnost `ControlToCompare` kot tudi lastnost `ValueToCompare`. Če sta nastavljeni obe lastnosti, ima lastnost `ControlToCompare` prednost in se preverja samo ta.
Če nastavimo lastnost `Operator`, se v prej opisanih primerih ne preverja enakost, ampak ali je vrednost, ki jo je uporabnik vnesel v vnosno polje, manjše (`LessThan`) / večje (`GreaterThan`) / manjše ali enako (`GreatherThanEqual`) / ... od vrednosti v nekem drugem besedilnem polju oz. od določene vrednosti. Privzeta vrednost je enakost (`Equal`) , ki preverja, če sta dve vrednosti enaki.
Uporabna lastnost je tudi `Type`. Z njo določimo, kakšnega tipa mora biti vnesena vrednost. Izbiramo lahko med številom (`Integer`), decimalnim številom (`Double`), nizom (`String`), ki je privzeta vrednost, datumom (`Date`) in valuto (`Currency`).

Poglejmo primer: Če imamo lastnost `Operator` nastavljeno na `LessThan` in `ValueToCompare` na 23, nam v primeru, da je lastnost `Type` nastavljena na `String` in uporabnik v besedilno polje vpiše 155, preverjanje ne sproži napake. V tem primeru sta 23 in 155 niza in se jih tako tudi primerja. Če pa `Type` nastavimo na `Integer`, se nam ob preverjanju sproži opozorilo, saj je 23 manjše od 155.
V primeru, da želimo preverjati, če je vneseni podatek pravilnega tipa, moramo lastnost `Operator` nastaviti na vrednost `DataTypeCheck`. V tem primeru je seveda nastavitev lastnosti `ControlToValidate` oziroma `ValueToCompare` brezpredmetna.
V primeru, da bi želeli preverjati, če je vpisana vrednost število manjše od 100, bi morali postaviti dva primerjalnika vrednosti vezana na isto besedilno polje. Prvi bi preverjal tip (lastnost Operator bi nastavili na `DataTypeCheck`), drugi pa, če je vrednost manjša od 100 (100 bi vpisali v `ValueToCompare`). Če pa bi želeli na primer preveriti, ali je število med denimo 18 in 100, bi lahko uporabili tri primerjalnike vrednosti. Pri prvem bi nastavili lastnost `Operator` na `DataTypeCheck`, pri drugem na `LessThen` in pri tretjem na `GreaterThen`. Pri drugem primerjalniku bi lastnost `ValueToCompare` nastavili na 18, pri tretjem pa na 100. Lahko pa bi zadnja dva primerjalnika nadomestili s posebnim primerjalnikom (`CustomValidator`), kjer bi nastavili ustrezni primerjalni izraz.
Primer:

```
<asp:Label ID="lblIdTest" runat="server" Text="ID testa*"></asp:Label>
<asp:TextBox ID="txtIdTest" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvIdTest" runat="server"
                            ControlToValidate="txtIdTest"
                            ErrorMessage="Vpisana mora biti že obstoječa
                            ID številka testa. " ValidationGroup="Excel">*
</asp:RequiredFieldValidator>
<asp:CompareValidator ID="cvIdTest" runat="server" ControlToValidate=
                      "txtIdTest" ErrorMessage="ID testa je tipa število.
                      Operator="DataTypeCheck" Type="Integer">*
</asp:CompareValidator>
<asp:Button ID="btnUvoz" runat="server" Text="Uvozi podatke" Width="127px"
            ValidationGroup="Excel" />
<asp:FileUpload ID="fuTest" runat="server"/>
<asp:Label ID="lblIzberi" runat="server" Text="Izberi datoteko (v formatu
          xls) za uvoz podatkov*" Width="305px"></asp:Label>
<asp:RequiredFieldValidator ID="rfvDatoteka" runat="server"
                            ControlToValidate ="fuTest"
                            ErrorMessage="Nobena datoteka ni izbrana"
                            ValidationGroup="Excel">*
</asp:RequiredFieldValidator>
```

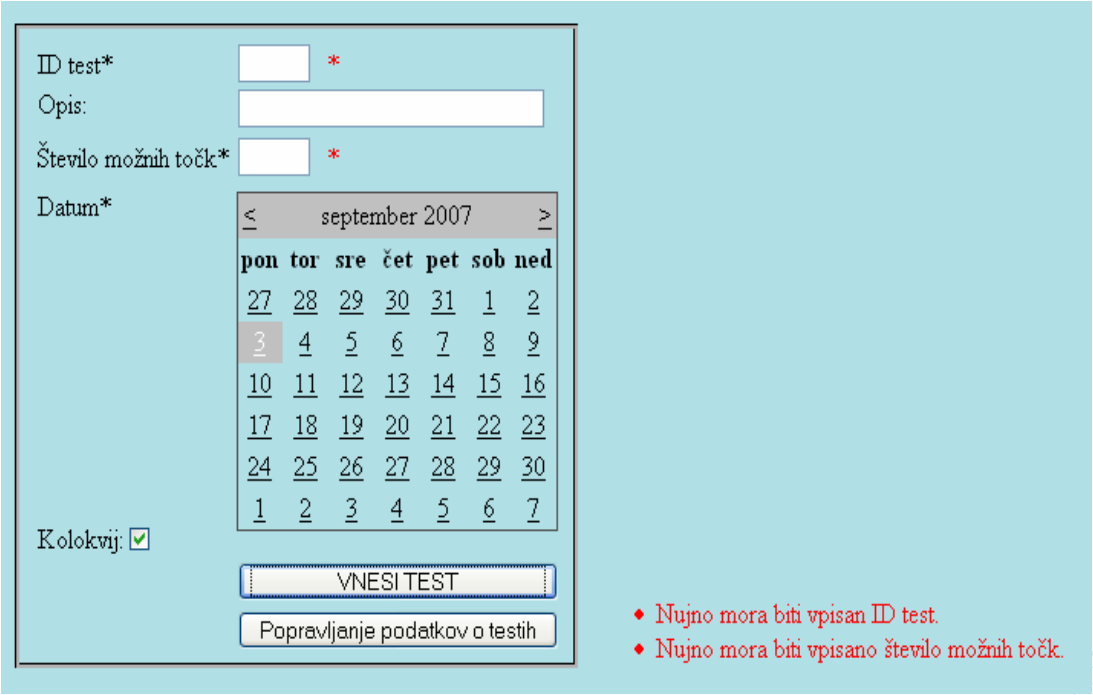


Slika 38 - Primerjalnik vrednosti

Na besedilno polje `txtIdTest` sta vezana dva gradnika za preverjanje. Prvi je gradnik za preverjanje praznega vnosa in drugi primerjalnik vrednosti, ki preverja, če je vneseni podatek tipa število.
Na gradnik za izbiranje in nalaganje datotek `fuTest` je vezan le gradnik za preverjanje praznega vnosa, kjer preverjamo, če je datoteka izbrana. Ustreznega gradnika, ki bi preveril še, če datoteka res obstaja in je ustreznega tipa, pa žal ni. Zato bi morali to preverjanje sprogramirati sami s pomočjo ustreznih metod.

Gradnik za prikaz opozoril (*ValidatonSummary*)

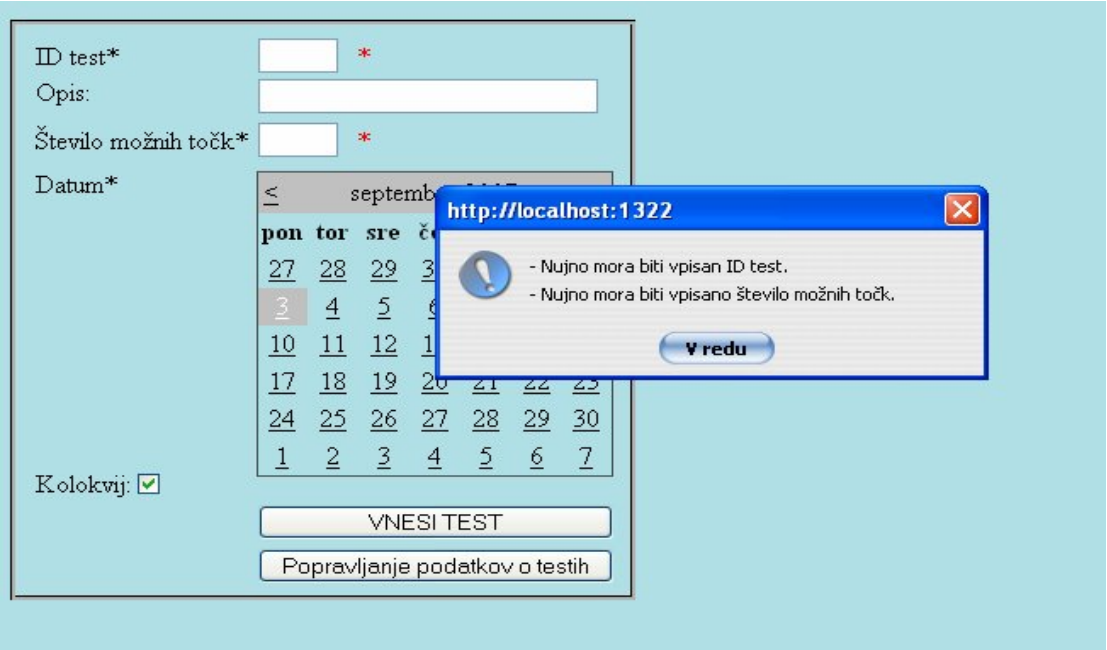
Če na spletni strani uporabljamo več gradnikov za primerjanje, lahko uporabimo poseben gradnik za prikaz opozoril. Tako uporabnika na enem mestu opozorimo na napačne oz. manjkajoče podatke na strani in zaradi preglednosti zraven vsakega napačno vnesenega podatka izpišemo le rdečo zvezdico (ali kako drugo oznako, da je z vnosom nekaj narobe). Opozorilo se prikaže takrat, ko uporabnik klikne na gradnik, ki ima lastnost `CausesValidation` nastavljeno na `True`.
V tem primeru pri vsakem gradniku za primerjanje nastavimo lastnosti `ErrorMessage` in `Text`. Besedilo, ki ga pripišemo lastnosti `ErrorMessage`, se bo avtomatično prikazalo v gradniku za prikaz opozoril. Besedilo, ki smo ga pripisali lastnosti `Text` (v našem primeru *), pa se izpiše tam, kjer smo postavili gradnik za preverjanje. Isti učinek dosežemo, če to besedilo zapišemo med začetno in končno značko gradnika za preverjanje.
Če nastavimo le lastnost `ErrorMessage`, se to besedilo izpiše na mestu, kjer je gradnik za preverjanje in še v gradniku za prikaz opozoril.
Seznam opozoril lahko prikažemo na dva načina. Lahko uporabimo navaden seznam, ki se bo izpisal na mestu, kjer smo postavili gradnik za prikaz opozoril. V tem primeru mora imeti gradnik nastavljeno lastnost `ShowSummary` na `True`.



Slika 39 - Delovanje gradnika za prikaz opozoril (Summary)

Primer na sliki prikazuje stanje po tem, ko je uporabnik kliknil na gumb VNESI TEST. Kot število možnih točk je uporabnik vpisal besedilo namesto števila, prav tako je manjkajoč podatek o zaporedni številki testa. Na oboje ga opozori gradnik za prikaz opozoril.

Če bi namesto takega izpisa želeli prikaz opozoril v dialognem oknu, moramo nastaviti lastnost `ShowSummary` na `False` in lastnost `ShowMessageBox` na `True`.
V primeru, da imamo tako lastnost `ShowSummary` kot lastnost `ShowMessageBox` nastavljeni na `True`, se izpiše le opozorilno okno.



Slika 40 - Delovanje gradnika za prikaz sporočil (MessageBox)

Dogodki (Events)

Dogodkovno programiranje je način pisanja programov, pri katerem posamezni objekti "čakajo", da se zgodi kak dogodek. Ko pride do dogodka, se izvede koda, ki je napisana za posamezen dogodek pri posameznem objektu. Tipični primer dogodkovnega programiranja je model programiranja odjemalec/strežnik, na katerem temelji tudi ASP.NET. Strežnik je gradnik, ki čaka, da odjemalec postavi zahtevo po podatkih, nato pa se na ta dogodek odzove tako, da odjemalcu pošlje podatke.
V našem primeru se strežniški gradniki odzivajo na dogodke. Dogodek je na primer klik na gumb, izbira v kombiniranem polju, premik miške … Ko se dogodek sproži, vir dogodka pokliče ustrezno metodo v rokovalniku (*handler*) in ji kot parameter preda dogodek. Odzivanje na dogodke v kodi izvedemo z dogodkovnimi podprogrami, ki imajo posebno sintakso. Tako je v Visual Basicu na koncu napovedi podprograma dodana rezervirana beseda `Handles`, ki pove, da gre za podprogram, ki opisuje reakcijo gradnika na določen dogodek. Primer glave podprograma, ki se izvede ob kliku na gumb z `ID`-jem `btnVnesiTest`:

```
Protected Sub btnVnesiTest_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnVnesiTest.Click
    ' sledi ostala koda
```

Najbolj pogosto uporabljeni dogodki so:

- **Click**: Dogodek se sproži ob uporabnikovem kliku na gradnik.
- **Disposed**: Dogodek se sproži, ko se gradnik sprosti.
- **Load**: Dogodek se sproži, ko se gradnik naloži v strežnik.
- **CheckedChanged**: Dogodek se sproži, ko uporabnik izbere ali odkljuka izbirni gumb ali potrditveno polje oz. enega od polj v seznamu izbirnih gumbov ali potrditvenih polj.
- **SelectedIndexChanged**: Dogodek se sproži, ko uporabnik v spustnem seznamu izbere eno od naštetih možnosti.

Nekaterih dogodkov, ki smo jih vajeni iz okenskih aplikacij, kot so `OnMouseOver`, `OnChange` ... tukaj žal nimamo na voljo.

Ker gre pri programiranju spletnih strani za uporabo povezovanja med odjemalcem (brskalnikom) in strežnikom (spletni strežnik in izvajalec strani ASP), so med dogodki pomembni tudi tisti, ki se izvedejo ob nalaganju aplikacije oz zapiranju le te.

Dogodki, ki se lahko zgodijo na strani, so:

- `DataBinding`: Dogodek se sproži, ko pride do povezovanja s podatkovnimi gradniki.
- `Disposed`: Dogodek se sproži, ko se stran sprosti. To je zadnja faza življenjskega kroga strani.
- `Error`: Dogodek se sproži, ko se zgodi napaka, ki ni bila prestrežena.
- `PreInit`: Dogodek se sproži tik preden se začne začetna postavitev stran na strežniki, torej prirejanje vseh osnovnih značilnosti strani, postavljanje gradnikov na stran ... .
- `Init`: Dogodek se sproži, ko se začne začetna postavitev strani. To je prva faza življenjskega kroga strani.
- `InitComplete`: Dogodek se sproži, ko so vsi gradniki in stran nastavljeni.
- `PreLoad`: Dogodek se sproži tik pred tem, ko se začne nalaganje strani.
- `Load`: Dogodek se sproži ob samem nalaganju strani v strežnik.
- `LoadComplete`: Dogodek se sproži, ko je stran naložena.
- `PreRender`: Dogodek se sproži tik preden se pokliče metoda `Render` najprej za stran in nato še za vse strežniške gradnike. Metoda Render poskrbi za prikaz vseh gradnikov in strani same v spletnem brskalniku.
- `PreRenderComplete`: Dogodek se sproži tik preden je stran poslana strežniku.
- `UnLoad`: Dogodek se sproži, ko se sprostijo vsi strežniški gradniki. Dogodek se izvede tik pred dogodkom `Disposed`.

Zaporedje, v katerem se zgodijo posamezni dogodki, je naslednje: `PreInit`, `Init`, `InitComplete`, `PreLoad`, `Load`, `LoadComplete`, `PreRender`, `PreRenderComplete`, `Unload`. Dogodki pred dogodkom `PreLoad` se zgodijo na strežniški strani, ostali pa se zgodijo na strani odjemalca.

Osveževanja (PostBacks)

Spletne strani ASP se osvežujejo vsakič, ko se zgodi kak dogodek. Dogodek se izvede in strežnik ponovno odjemalcu posreduje spletno stran.
Za določeno kodo (npr. polnjenje spustnih seznamov, prikazovanje določenih podatkov v gradnik za prikaz podatkov, branje podatkov iz baze …) želimo, da se izvede le ob prvem nalaganju strani. Večkratno nalaganje ob vsakem osveževanju bi bilo nepotrebno, ker so podatki že naloženi. V tem primeru s pomočjo lastnosti `IsPostBack` preverimo, če se je zgodilo osveževanje. V primeru, da ne gre za osveževanje, pomeni, da je uporabnik šele odprl spletno stran.

```
If Page.IsPostBack Then ' stran se osvežuje
    ' Izvedi ukaze
Else
    ' Izvedi ukaze ob prvem odpiranju strani
End If
```

Lastnost `IsPostBack` ponovno dobi vrednost `False` takrat, ko uporabnik obišče drugo spletno stran, torej, ko zapusti stran v brskalniku.

Ko se stran osveži in se naloži na novo, se izgubijo vsi podatki, ki jih nismo tako ali drugače shranili. Za ohranjevanje podatkov, ki jih želimo med osveževanjem ohraniti, nam pridejo prav objekti v katere shranjujemo podatke. Načeloma so za ohranjevanje podatkov med osveževanjem na voljo različni tovrstni objekti. A sama sem v aplikaciji uporabljala le objekte tipa Session (seja).
Informacije, ki pripadajo posameznemu uporabniku, nadziramo z objektom `Session` (seja). Seja je definirana kot časovno obdobje, ko je uporabnik na naši spletni strani. Ta objekt se ob vsakem obisku strežnika ustvari za vsakega uporabnika posebej.
Vrednosti, ki jih želimo prenašati med stranmi, lahko programsko shranimo v spremenljivko stanja seje in so tako dostopne tudi na drugih straneh naše spletne aplikacije.
Način dodajanja spremenljivk v objekt Session je razviden iz zgleda.

```
' Spravimo tabelo v spremenljivko stanja seje
Dim dtStudentExcel As DataTable = New DataTable()
Session("dtStudentExcel") = dtStudentExcel
```

Do teh spremenljivk dostopamo tako, da naredimo nov objekt tipa `DataTable` in vanj preberemo vrednost iz spremenljivke stanja seje:

```
Dim dtStudentExcel As DataTable = New DataTable()
dtStudentExcel = CType(Session("dtStudentExcel"), DataTable)
```

Vrednost spremenljivk se v objektu ohranja na vseh straneh toliko časa, dokler uporabnik ne zapre svojega brskalnika in s tem prekine sejo oz. je določen čas neaktiven in ne klikne na nobeno povezavo. Privzeti čas trajanja seje je 20 minut. Ta čas lahko povečamo oz. zmanjšamo s pomočjo lastnosti `TimeOut`:

```
Session.Timeout = 30
```

Delo s podatki

Spletne aplikacije delimo na tiste s strukturo dvoslojne arhitekture odjemalec/strežnik in na večslojne, porazdeljene aplikacije. Slabost dvoslojnih aplikacij je ta, da skozi ves čas izvajanja aplikacije ohranjajo aktivno povezavo do podatkovnega vira. Takšen pristop gradnje aplikacije temelji na povezanem dostopu do podatkovnih virov (*connected access*). Največja slabost takega pristopa je prekomerna uporaba povezave, saj mora biti ta ves čas aktivna.
Večina sedanjih aplikacij, ki se povezujejo z določenim podatkovnim virom, uporablja strukturo večslojne, porazdeljene aplikacije. Ta temelji na odklopljenem dostopu do podatkovnih virov (*disconnected access*). To pomeni, da se povezava do podatkovnega vira ohranja le toliko časa, da se iz vira poberejo želeni podatki ali se v vir podatki zapišejo. Ko program potrebuje določene podatke, se vzpostavi povezava med programom in virom podatkov. Vir podatkov programu podatke posreduje, nato pa se povezava zapre. Program podatke obdela in če jih mora ponovno zapisati v bazo, se povezava znova vzpostavi.
Primer modela, ki uporablja odklopljen dostop, je ADO.NET. Podrobnosti glede uporabe tega modela in ustreznega nabora razredov bi presegale okvire te diplomske naloge. V naslednjem razdelku si bomo zgolj na kratko ogledali tiste osnovne pojme in prijeme, ki jih uporabljamo v vsaki spletni aplikaciji, ki za dostop do zunanjega vira podatkov uporablja ADO.NET.

ADO.NET

`ADO.NET` (*ActiveX Data Objects*) je nabor razredov, ki vsebujejo spremenljivke in metode, ki omogočajo dostop do vira podatkov. Vir podatkov je lahko baza podatkov, datoteka XML … Celoten nabor objektov `ADO.NET` je zajet v imenskemu prostoru `System.Data`. Vsebuje bogat nabor gradnikov, ki jih ločimo na gradnike za dostop do vira podatkov in na gradnike za obdelavo podatkov.
Primarni gradniki odklopljenega objektnega modela `ADO.NET` so nabor podatkov (`DataSet`) in nabor gradnikov za dostop do vira podatkov, ki so shranjeni pod oznako `.NET Data Provider`. Visual Web Developer v svojih orodjarnah že privzeto vsebuje vse najpomembnejše gradnike za dostop do podatkov, ponuja pa tudi čarovnike za vzpostavljanje povezav s podatkovnimi viri.

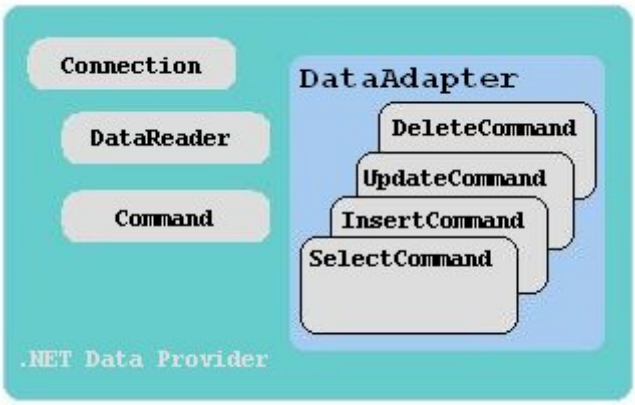
Nabor gradnikov .NET Data Provider

Osnovni gradniki nabora gradnikov `.NET DataProvider` so Connection, DataReader, Command in DataAdapter. Ti gradniki se uporabljajo za povezavo s podatkovnimi viri, za izvrševanje poizvedb v jeziku SQL ter za dostop do podatkov znotraj podatkovnih virov:

- Gradnik `Connection` skrbi za povezavo z virom podatkov (s podatkovno bazo). Njegove metode in lastnosti (`Open()`, `Close()`, `User`, `Password` …) zagotavljajo nadzor in

vzdrževanje povezave. Pomembna lastnost objekta tega tipa je `ConnectionString`, kjer je točno naveden podatkovni vir, s katerim bo objekt povezan.

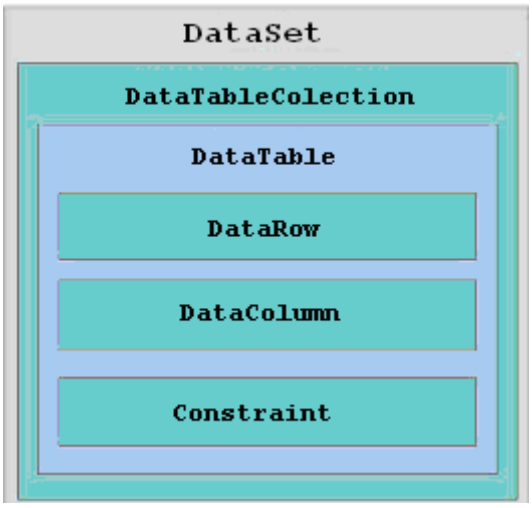
- Objekt tipa `Command` nad virom podatkov izvršuje ukaze `SELECT`, `INSERT`, `UPDATE` in `DELETE`. Ukazi so zapisani v jeziku SQL (*Structured query language*), ki je standardni jezik za dostop do podatkov.
- `DataReader` prebere podatke iz vira podatkov.
- `DataAdapter` napolni nabor podatkov ali tabelo s podatki. Je vmesni člen med virom podatkov in objektom nabor podatkov.



Slika 41 - Nabor gradnikov .NET Data Provider

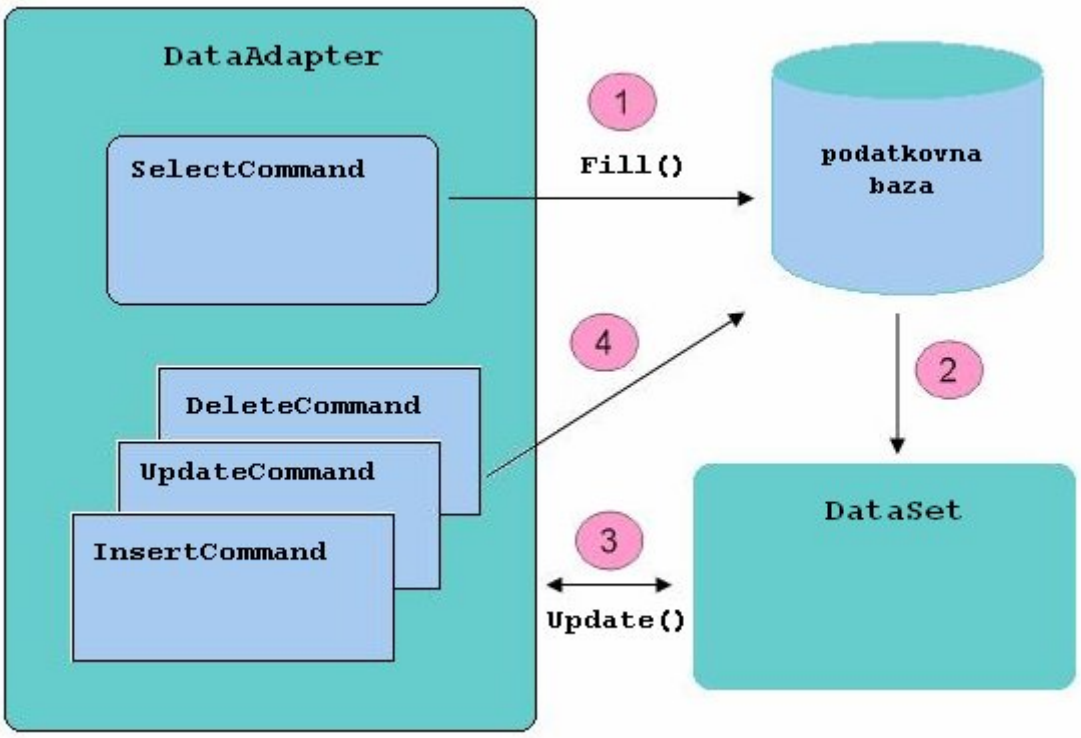
Nabor podatkov (DataSet)

Gradnik DataSet skrbi za hranjene podatkov v lokalnem pomnilniku računalnika, pri tem pa je popolnoma neodvisen od podatkovnega vira.
Podatki se shranjujejo s pomočjo hierarhično urejenega nabora gradnikov.



Slika 42 - Zgradba nabora gradnikov

Podatki so shranjeni v objektu tipa podatkovna tabela (*DataTable*). Ta se sestoji iz vrstic (*DataRow*) in stolpcev (*DataColumn*). Nad vsakim stolpcem v tabeli je mogoče določiti primarne (*Primary keys*) in tuje ključe (*Foreign keys*).
S pomočjo nabora podatkov polnimo gradnike, kot so tekstovna polja, tabele, gradniki za prikaz podatkov …



Slika 43 – Povezava predmetov DataAdapter in nabora podatkov

Slika 43 prikazuje povezavo med objekti tipa DataAdapter in objekti tipa DataSet. Najprej je potrebno zgraditi objekt `SelectCommand`. Po klicu metode `Fill` na tem objektu se s podatki napolni DataSet. Ob klicu metode `Update` se DataSet poveže z ustreznimi lastnostmi vrste Command, te pa poskrbijo za dejansko spremembo podatkov v podatkovni bazi.

`DataAdapter` omogoča neposredno povezavo med podatkovno bazo in naborom podatkov. Povezava je zgrajena s pomočjo štirih lastnosti tipa `Command`, njihova funkcionalnost pa je določena že z njihovimi oznakami:

- `SelectCommand` omogoča prebiranje podatkov iz podatkovne baze.
- `InsertCommand` omogoča dodajanje novih vrednosti v podatkovno bazo
- `UpdateCommand` omogoča spreminjanje že obstoječih vrednosti v podatkovni bazi
- `DeleteCommand` omogoča brisanje ustreznih podatkov iz podatkovne baze

DataAdapter pri polnjenju nabora podatkov uporablja metodi `Fill` in `Update`.

Dodajanje študentov v podatkovno bazo

Kot primer povezovanja med programom in bazo si oglejmo, kako v naši aplikaciji v podatkovno bazo dodamo študente:

```
Protected Sub btnUvoz_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnUvoz.Click
   Try
      'Deklaracija
      Dim sql As String
      Dim strConn As String = System.Configuration.
         ConfigurationManager.ConnectionStrings("Student").
            ConnectionString
      Dim conn As OleDbConnection = New OleDbConnection(strConn)

      Dim i As Integer
      Dim stVrstic As Integer
      Dim stStolpcev As Integer
      Dim IdNaloge As Integer = 1
      Dim dr As DataRow
```

```
' Prikazovalnik podatkov gvStudent ne prikažemo
gvStudent.Visible = False

' Preberemo podatke o študentih iz Excel datoteke
Dim napaka As String = BranjeExcelStudenti()
stVrstic = dtStudentExcel.Rows.Count
stStolpcev = dtStudentExcel.Columns.Count

If napaka <> "" Then
   ' V primeru napake prenehamo z izvajanjem kode in sporočimo
   ' uporabniku napako
   MsgBox(napaka, MsgBoxStyle.Exclamation, "Napaka")
   Exit Sub
End If

Dim dtStudent As New DataTable
With dtStudent.Columns
   .Add("ID_STUDENT", GetType(Integer))
   .Add("IME", GetType(String))
   .Add("PRIIMEK", GetType(String))
   .Add("LETNIK", GetType(Integer))
End With
' Določimo primarni ključ, da bomo lahko uporabljali metodo Find
      dtStudent.PrimaryKey = New DataColumn()
                             {dtStudent.Columns("ID_STUDENT")}

' Preberemo študente, ki so že v bazi
sql = "SELECT ID_STUDENT, IME, PRIIMEK, LETNIK FROM STUDENT "
Dim daStudent As OleDbDataAdapter = New OleDbDataAdapter(sql, conn)
lblOpozorilo.Text = "Naslednje vpisne številke so že v bazi
                     študentov, zato podatki teh študentov ne bodo
                     dodani: "
For i = 0 To stVrstic - 1
   ' Preverimo, če je vpisna številka študenta v tabeli
   ' študentov, če je že, podatka ne vpišemo
   dr = dtStudent.Rows.Find(dtStudentExcel.Rows(i)(0))
   If dr IsNot Nothing Then
      ' Študent s tem ID-jem je že v bazi
      lblOpozorilo.Text &= dtStudentExcel.Rows(i)(0) & " "
   Else
      ' napolnimo vrstico z ustreznimi podatki
      dr = dtStudent.NewRow
      dr("ID_STUDENT") = Convert.ToInt16(dtStudentExcel.Rows(i)(0))
      dr("IME") = dtStudentExcel.Rows(i)(1)
      dr("PRIIMEK") = dtStudentExcel.Rows(i)(2)
      dr("LETNIK") = Convert.ToInt16(dtStudentExcel.Rows(i)(3))
      ' dodamo vrstico tabeli dtStudent
      dtStudent.Rows.Add(dr)
   End If
Next

'Shranimo tabelo dtStudent v sejo, zato, da bo dostopna tudi
' znotraj naslednje procedure
Session("studentShranjevanje") = dtStudent

'Shranjevanje v bazo podatkov
napaka = ShranjevanjeVBazoStudent()
If napaka <> "" Then
   MsgBox(napaka, MsgBoxStyle.Exclamation, "Napaka")
   Exit Sub
End If
```

```
      If lblOpozorilo.Text = "Naslednje vpisne številke so že v bazi " & _
         študentov, zato podatki teh študentov ne bodo  dodani: " Then
         lblOpozorilo.Text = "Vsi podatki so bili uspešno uvoženi v " & _
                             bazo podatkov."
      End If

      gvUvoz.Visible = True
   Catch ex As Exception
      MsgBox("Prišlo je do napake pri vnašanju rezultatov v " & _
             podatkovno tabelo.", MsgBoxStyle.Exclamation, "Napaka")
   End Try

End Sub
```

Najprej smo v tabelo `dtStudent` dodali novo vrstico s podatki študenta, za katerega smo preverili, da še ni v bazi podatkov. V objekt `Session` smo dodali tabelo `dtStudent`, da je bomo lahko uporabili tudi v funkciji `ShranjevanjeVBazoStudent`:

```
Private Function ShranjevanjeVBazoStudent() As String
   Try
      ' Deklaracija
      Dim conn As OleDbConnection = New OleDbConnection(System & _
      .Configuration.ConfigurationManager.ConnectionStrings("Student") & _
      .ConnectionString)
      Dim tr As OleDbTransaction = Nothing
      ' Preberemo tabelo dtStudent iz seje
      Dim dtStudent As DataTable =
      CType(Session("studentShranjevanje"), DataTable)
      ' Odpremo povezavo z bazo
      If conn.State <> ConnectionState.Open Then
         conn.Open()
      End If
      ' postavimo INSERT stavek
      Dim insStudent As String = "INSERT INTO [Student]
      ([ID_student], [IME], [PRIIMEK], [LETNIK]) " & _
      "VALUES (@ID_STUDENT, @IME, @PRIIMEK, @LETNIK)"
      Dim cmdInsStudent As New OleDbCommand(insStudent, conn)
     ' Določimo, kakšnih tipov morajo biti podatki
     ' V primeru napačnih vržemo napako
     With cmdInsStudent
        .Parameters.Add("@ID_STUDENT", OleDbType.Integer, 0, "ID_STUDENT")
        .Parameters.Add("@IME", OleDbType.VarChar, 0, "IME")
        .Parameters.Add("@PRIIMEK", OleDbType.VarChar, 0, "PRIIMEK")
        .Parameters.Add("@LETNIK", OleDbType.Integer, 0, "LETNIK")
     End With
     ' Izgradnja objekta DataAdapter
     Dim daInsTocke As New OleDbDataAdapter
     daInsTocke.InsertCommand = cmdInsStude

      ' Vzpostavitev transakcije
      tr = conn.BeginTransaction
      ' Transakcijo pripišemo objektu tipa Command, ki izvaja operacije nad
      ' podatkovno bazo preko ustreznega objekta Connection
      daInsTocke.InsertCommand.Transaction = tr
      ' dodamo vse vrstice, ki imajo oznako Added
      daInsTocke.Update(dtStudent.Select(Nothing, Nothing,
      DataViewRowState.Added))
      ' Potrdimo vse spremembe v bazi
      tr.Commit()
```

```
        ' Prekinemo povezavo s podatkovno bazo
    conn.Close()
    Catch ex As Exception
        Return "Prišlo je do napake pri vnašanju rezultatov v bazo podatkov."
    End Try

    Return ""
End Function
```

Pogosto je pomembno, da so podatki, ki jih uporabnik zahteva, ažurni. Če dva uporabnika zahtevata iste podatke in jih nato eden od njiju spremeni, bi bilo dobro, da bi drugi uporabnik takoj videl popravljene podatke. Ena od možnosti, ki sem jo tudi sama uporabila pri svojem primeru, je, da program vsakič znova bere podatke iz podatkovne baze. Ta rešitev pa je nepraktična in, v primeru velike količine podatkov, tudi počasna. Obstaja boljša rešitev z uporabo gradnikov SqlCacheDependency. Tu se vsi podatki preberejo na začetku v vmesni pomnilnik in ko pride do kake spremembe, se podatki avtomatično obnovijo. Vendar mora tak način delovanja podpirati tudi podatkovna baza. Sama je nisem uporabila zato, ker ta rešitev ni združljiva z bazo Access.

Prikazovalnik podatkov (GridView)

Prikazovalnik podatkov je gradnik, s katerim na enostaven način prikazujemo podatke, ki jih pridobimo iz podatkovne baze,. Podatki so v naboru podatkov v podatkovni tabeli, ki jo vežemo na prikazovalnik podatkov.
Denimo, da imamo tabelo `dtDatumi` in jo želimo prikazati. Najprej iz orodjarne na obrazec povlečemo prikazovalnik podatkov.

```
<asp:GridView ID="gvDatumi" runat="server" BorderStyle="Solid"
              AutoGenerateColumns="False">
   <HeaderStyle BackColor="PaleTurquoise" />
   <AlternatingRowStyle BackColor="PaleTurquoise" />
</asp:GridView>
```

Nastavljena je tudi lastnost `AlternatingRowStyle`, ki določa, da ima vsaka druga vrstica v tabeli drugačno ozadje. To je še posebej uporabno, če tabela vsebuje veliko vrstic, saj je tako bolj pregledna. Lastnosti `DataSource`, ki prikazovalniku podatkov pove, podatke katerega podatkovnega vira mora pokazati, pripišemo vrednost spremenljivke `dtDatumi` (torej naslov tabele z datumi). S tem smo povedali, od kje bo prikazovalnik črpal podatke. Za tem je treba nujno poklicati metodo `DataBind`, ki poskrbi, da se podatki iz tabele prikažejo na prikazovalniku podatkov:

```
gvDatumi.DataSource = dtDatumi
gvDatumi.DataBind()
```

Kakor hitro se ta koda izvede, v prikazovalniku podatkov vidimo ustrezne zapise.

Datumi kolokvijev in izpitov, ki sledijo v letošnjem letu:

Datum testa	Opis testa	Vrsta testa	Letnik
13.9.2007	3. izpit	izpit	1
21.9.2007	4. izpit	izpit	1

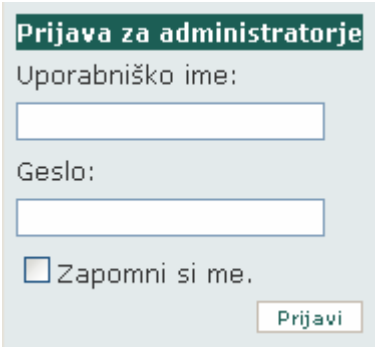
Slika 44 - Prikazovalnik podatkov

VARNOST

V ASP.NET imamo na voljo različne gradnike, ki skrbijo za prijavo in avtorizacijo uporabnikov. Z avtorizacijo uporabnikov določimo, do katerih strani lahko posamezen uporabnik dostopa. S pomočjo teh gradnikov običajno zgradimo obrazec za prijavo uporabnika. Tega uporabimo, kadar želimo nadzirati dostop do določenih strani v naši aplikaciji.

Prikazovalnik za prijavo uporabnikov (Login)

Gradnik za prijavo uporabnikov najdemo v orodjarni v razdelku `Login`:



Slika 45 - Gradnik za prijavo uporabnikov

Videz gradnika spremenimo tako, da kliknemo na trikotnik desno zgoraj. Tam kliknemo na Auto format in izberemo stil, ki ustreza naši aplikaciji. S pomočjo lastnosti `UserNameRequiredErrorMessage` in `PasswordRequiredErrorMessage` določimo besedilo, ki se izpiše, če uporabnik ne vnese uporabniškega imena oz. gesla. Z lastnostjo `FailureText` določimo besedilo, ki se izpiše ob napačnem vnosu uporabniškega imena in/ali gesla. Pomembna lastnost je tudi `DestinationPageUrl`, s katero določimo naslov spletne strani, kamor usmerimo uporabnika ob pravilni prijavi .

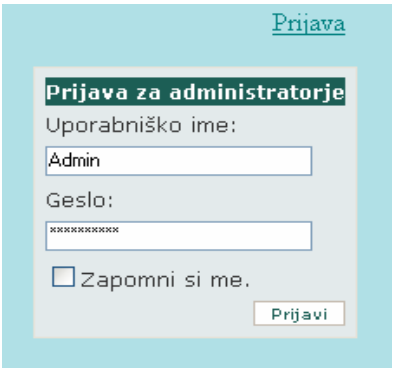
```
<asp:Login
   ID="Login1" runat="server"
   DestinationPageUrl="~/PregledRezultatov.aspx"
   FailureText="Prijava ni bila uspešna.Poskusi znova."
   PasswordLabelText="Geslo:"
   PasswordRequiredErrorMessage="Geslo ni vpisano."
   RememberMeText="Zapomni si me."
   TitleText="Prijava za administratorje"
   UserNameLabelText="Uporabniško ime:"
   UserNameRequiredErrorMessage="Uporabniško ime ni vpisano."
   BackColor="#E3EAEB" BorderColor="#E6E2D8" BorderPadding="4"
   BorderStyle="Solid" BorderWidth="1px"
   Font-Names="Verdana" Font-Size="0.8em"
   ForeColor="#333333" Height="164px" LoginButtonText="Prijavi"
   TextLayout="TextOnTop" Width="177px">
   <TitleTextStyle BackColor="#1C5E55" Font-Bold="True" Font-Size="0.9em"
                   ForeColor="White" />
   <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
   <TextBoxStyle Font-Size="0.8em" Width="150px" />
   <LoginButtonStyle BackColor="White" BorderColor="#C5BBAF"
                     BorderStyle="Solid" BorderWidth="1px"
                     Font-Names="Verdana" Font-Size="0.8em"
                     ForeColor="#1C5E55" />
</asp:Login>
```

Ko uporabnik vpiše uporabniško ime in geslo, se v primeru, da je vnos pravilen in da je uporabniku dovoljen dostop do te aplikacije, odpre spletna stran, navedena z lastnostjo `DestinationPageUrl`. Če prijava ne uspe, se prikaže opozorilo o napačni prijavi.

Indikator prijave (LoginStatus)

Ta gradnik prikazuje, ali je uporabnik prijavljen ali ne in poskrbi za to, da se uporabnik lahko prijavi oziroma odjavi. V primeru, da uporabnik klikne na ta gradnik, ko še ni prijavljen, bo avtomatsko preusmerjen ustrezno stran, kjer bo imel možnost prijave. Če pa je uporabnik že prijavljen, se ob kliku na ta gradnik lahko odjavi iz spletne aplikacije. Ob tem uporabnika preusmerimo na določeno strani, katere naslov določimo z lastnostjo `LogoutPageUrl`. Preko lastnosti `LoginText` in `LogoutText` določimo besedilo, ki bo izpisano na samem gradniku, ko uporabnik še ni, oziroma je že prijavljen. V zgledu, ki sledi, smo poskrbeli, da je na gradniku neprijavljenim uporabnikom pisalo `Prijava`, prijavljenim pa `Odjava`.

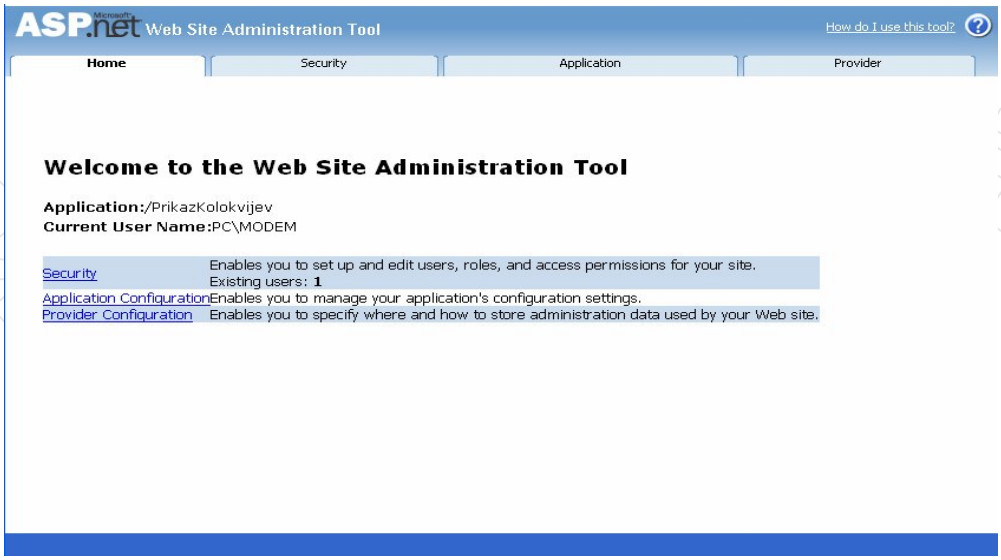
```
<asp:LoginStatus ID="lsPrijava" runat="server" LoginText="Prijava"
                 LogoutText="Odjava" Width="33px" ForeColor="Teal" />
```



Slika 46 - Indikator prijave (zgoraj desno)

Administratorski vmesnik

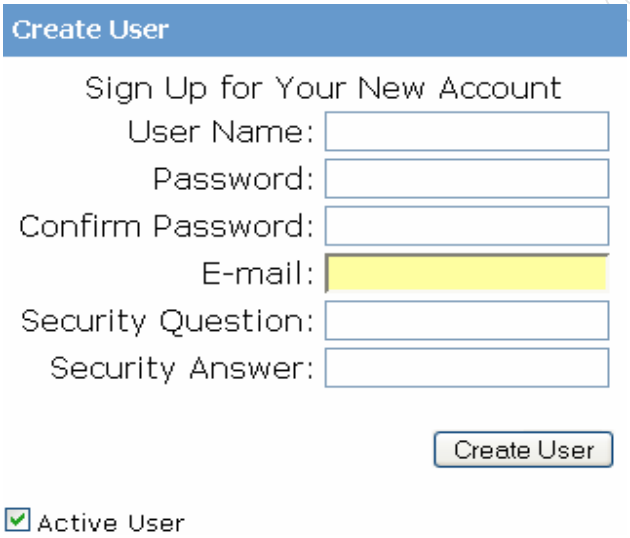
Same uporabnike spletne aplikacije določamo s pomočjo vgrajenega administratorskega vmesnika. V orodni vrstici okolja Visual Web Developer izberemo Website in nato ASP.NET Configuration. Odpre se naslednja stran:



Slika 47 - Prva stran administratorskega vmesnika

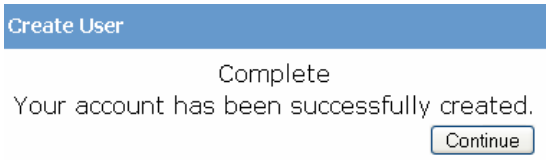
Ustvarjanje novega uporabnika

Novega uporabnika ustvarimo tako, da izberemo zavihek Security (Varnost) in kliknemo na hiperpovezavo Create user. Odpre se obrazec, v katerega vpišemo uporabniško ime in geslo uporabnika, potrditev gesla, e-mail, varnostno vprašanje in odgovor:



Slika 48 - Obrazec za ustvarjanje nove uporabnika

Ko vse podatke vnesemo in kliknemo na gumb Create User, se, če so bili vsi podatki pravilno vneseni, prikaže sporočilo:

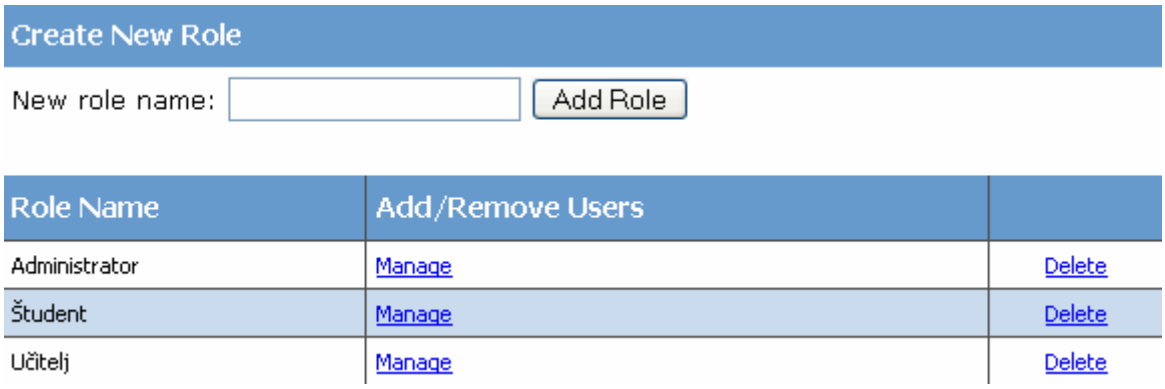


Slika 49 - Sporočilo ob pravilnem vnosu uporabnika

Ko kliknemo na gumb Continue, se ponovno prikaže obrazec za dodajanje novega uporabnika. Dodamo toliko uporabnikov, kot jih potrebujemo. Seveda nove uporabnike lahko dodajamo tudi naknadno.

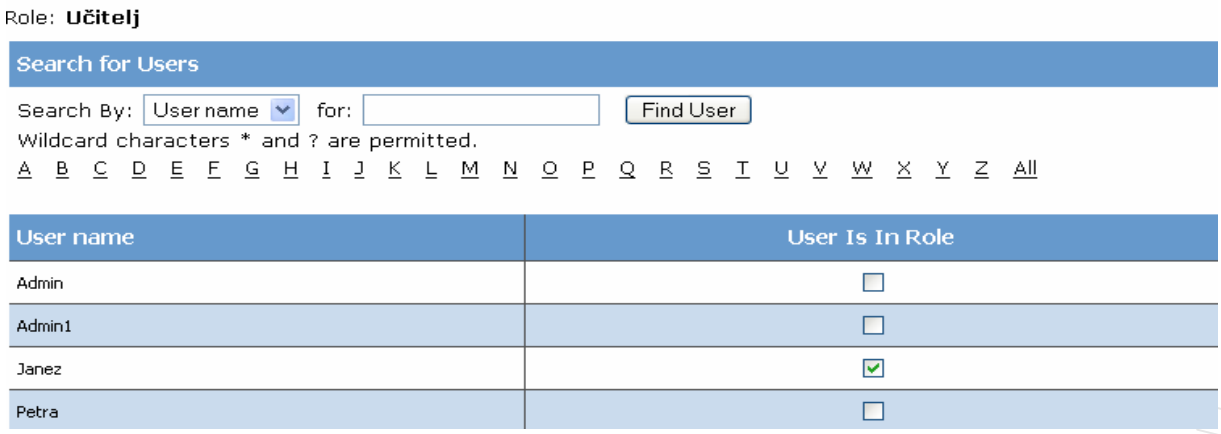
Ustvarjanje vlog

Za vsakega uporabnika posebej lahko določimo, do katerih strani ima oz. do katerih nima dostopa. Vendar je običajno bolj smiselno, če uporabnike razvrstimo v skupine in skupini kot celoti določimo, kakšne pravice ima. Skupine pogosto imenujemo tudi vloge, saj določajo v kakšni vlogi nek uporabnik nastopa. Vsaki vlogi torej pripišemo, kaj lahko počne, oziroma do katerih strani lahko dostopa.
V primeru moje aplikacije sem uporabila dve vlogi: Študent in Učitelj. Študent ima pravico dostopa do strani, kjer je možno pregledovanje rezultatov in datumov preverjanj znanja. Učitelj ima dostop do vseh strani. Vsem ostalim uporabnikom dostop do te aplikacije ni dovoljen.
Najprej ustvarimo novo vlogo. To storimo v zavihku Security. Ko kliknemo na Create or Manage roles se odpre obrazec, kjer lahko ustvarimo novo vlogo:



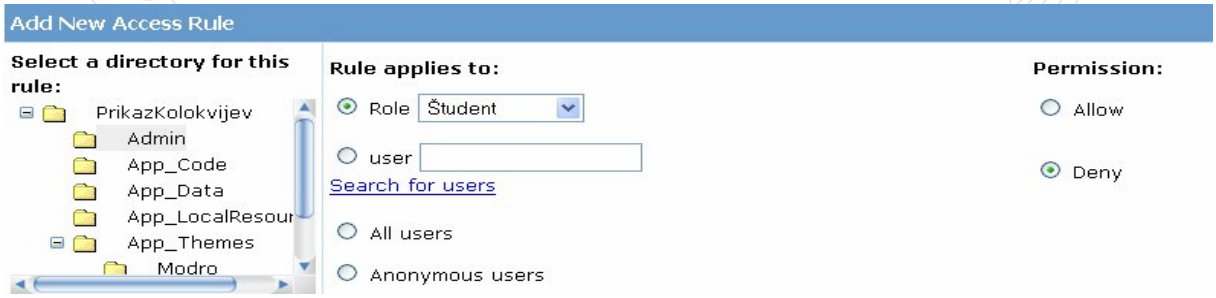
Slika 50 - Obrazec za ustvarjanje nove vloge

Ko vpišemo ime vloge in kliknemo na gumb Add Role, se nova vloga doda v spisek vseh že obstoječih vlog.
Za urejanje vlog kliknemo na hiperpovezavo Manage. Znajdemo se na strani, kjer izbrani vlogi lahko dodajamo uporabnike. Odkljukamo tiste uporabnike, za katere želimo, da pripadajo izbrani vlogi. V našem primeru je vlogi Učitelji dodan le uporabnik Janez.



Slika 51 - Obrazec za dodeljevanje uporabnikov posameznim vlogam

Vloge imajo pomen, če v njih določimo, do katerih strani lahko uporabniki v tej vlogi dostopajo in do katerih ne. To storimo z dostopnimi pravili. Enak postopek uporabimo tudi takrat, ko želimo omogočiti oziroma onemogočiti dostop do posameznih spletnih strani posameznemu uporabniku.
V primeru, da uporabnika ne uvrstimo v nobeno vlogo oz. mu ne določimo dostopnih pravil, lahko dostopa do vseh spletnih strani v tej aplikaciji.
Naslednja slika prikazuje pravilo, ko uporabniki, dodani vlogi Študent, ne smejo dostopati do strani, ki so v mapi Admin (sivo označena mapa levo). V našem primeru so to strani za dodajanje študentov, testov in rezultatov testov:



Slika 52 - Obrazec za dodajanje pravil za dostop vlogi Študent

Poglejmo si, kako nastavimo tako pravilo. V levem meniju imamo prikazane vse mape, ki pripadajo naši aplikaciji. Izberemo mapo, za katero želimo določiti novo pravilo za dostop. V srednjem delu izberemo za koga dodajamo pravilo za dostop. To je lahko vloga, posamezni uporabnik, vsi uporabniki ali pa anonimni (neprijavljeni) uporabniki. Na desni strani pa določimo, ali izbranim dovoljujemo ali preprečujemo dostop do izbrane mape.

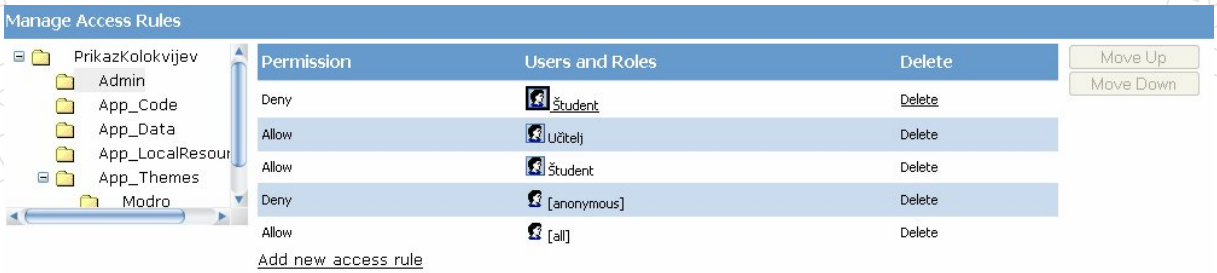
Uporabniki, dodani vlogi Učitelj, lahko dostopajo vsem stranem v aplikaciji:

Slika 53 - Obrazec za dodajanje pravil za dostop vlogi Učitelji

Ostali, neprijavljeni uporabniki, pa imajo dostop le do prve strani:

Slika 54 - Obrazec za dodajanje pravil za dostop za anonimnega uporabnika

V zavihku Security si s klikom na hiperpovezavo Manage access rules lahko ogledamo seznam vseh pravil:



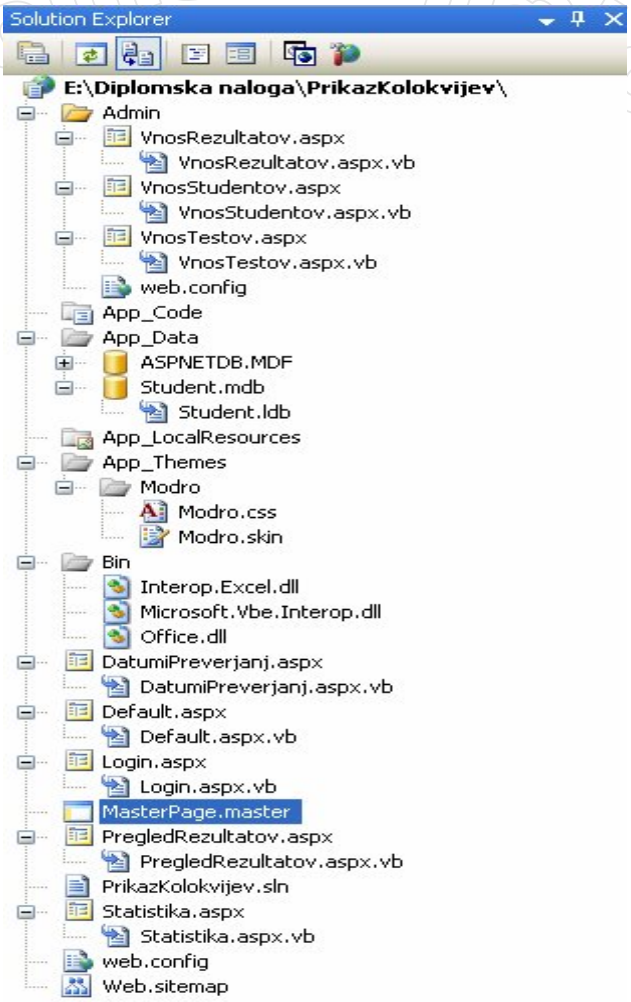
Slika 55 - Pregled vseh pravil

Vrstni red pravil je pomemben. Če sta si dve pravili v navskrižju, se upošteva prvo (višje v seznamu).

Aplikacija za dodajanje in pregledovanje rezultatov

Aplikacija vsebuje strani za pregledovanje kolokvijev oz. izpitov, pregledovanje datumov prihajajočih kolokvijev oz. izpitov, za dodajanje podatkov o študentih, testih in za dodajanje rezultatov testov. Uporabniki v vlogi študenta do strani za vnašanje podatkov nimajo dostopa, ampak imajo dostop le do strani, namenjenih pregledovanju. Učitelj ima dostop do vseh strani. Neprijavljen uporabnik ne more niti pregledovati niti vnašati podatkov.

V nadaljevanju si bomo ogledali kodo posameznih datotek, krajši opis dela aplikacije s slikami in nekatere zanimivejše dele kode.



Slika 56 - Vse datoteke in viri, ki pripadajo aplikaciji

web.config (mapa PrikazKolokvijev)

Kot smo že povedali v razdelku o pregledovalniku rešitev (stran 14), je to datoteka, v kateri so različne nastavitve spletnega strežnika, ki veljajo pri tem projektu. Te nastavitve veljajo pri dostopu do vseh datotek v mapi PrikazKolokvijev.

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <appSettings>
   </appSettings>
   <connectionStrings>
      <!-- Povezovalni niz, ki ga potrebujemo za povezavo s podatkovno bazo. -->
      <add name="Student"connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
          Data Source= E:\Diplomska naloga\PrikazKolokvijev\App_Data\Student.mdb" />
   </connectionStrings>
   <system.web>
      <authorization>
         <!-- Avtentikacija uporabnikov: -->
         <!-- Neprijavljeni uporabnik nima dostopa do nobene od strani
              v mapi PrikazKolokvijev. -->
         <deny users="?" />
         <!-- Uporabniki, ki pripadajo vlogi Učitelj in Študent, lahko
```

```
            dostopajo do strani v mapi PrikazKolokvijev. -->
        <allow roles="Učitelj" />
        <allow roles="Študent" />
      </authorization>
      <roleManager enabled="true" />
      <trace enabled="true"  />
      <compilation debug="true" strict="false" explicit="true"/>
      <pages>
        <namespaces>
          <clear/>
          <!-- Dodani imenski prostori: -->
          <add namespace="System"/>
          <add namespace="System.Collections"/>
          <add namespace="System.Collections.Specialized"/>
          <add namespace="System.Configuration"/>
          <add namespace="System.Text"/>
          <add namespace="System.Text.RegularExpressions"/>
          <add namespace="System.Web"/>
          <add namespace="System.Web.SessionState"/>
          <add namespace="System.Web.Security"/>
          <add namespace="System.Web.Profile"/>
          <add namespace="System.Web.UI"/>
          <add namespace="System.Web.UI.WebControls"/>
          <add namespace="System.Web.UI.WebControls.WebParts"/>
          <add namespace="System.Web.UI.HtmlControls"/>
        </namespaces>
      </pages>
    <authentication mode="Forms" />
  </system.web>
</configuration>
```

V zgornji kodi vidimo, da datoteka `web.config` vsebuje tudi del o avtorizaciji uporabnikov (`authorization`). Znotraj značke `deny` je določeno, katerim uporabnikom (`users`) je prepovedan dostop do strani, ki so v mapi PrikazKolokvijev, kateri pripada tudi datoteka `web.config`. Znak `?` nam pove, da je dostop prepovedan vsem neprijavljenim uporabnikom. Tu bi lahko našteli tudi le posamezne vloge (`roles`), kot to vidimo pri znački `allow`. Z značko `allow` torej povemo, katerim vlogam oz posameznim uporabnikom je dovoljen dostop do strani v tej mapi.

MasterPage.master

V tej datoteki je zapisano, kateri so skupni deli spletnih strani v aplikaciji. V mojem primeru je v datoteki MasterPage.master opisano, kakšno osnovno postavitev ima vsaka spletna stran aplikacije. Več o samih predlogah strani je razloženo v poglavju o spletnih gradnikih, razdelek Vsebovalnik strani (stran 20).

```
<%@ Master Language="VB" %>

<!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>Sistem za prikaz kolokvijev in izpitov</title>
</head>
<body alink="#33ffff" style="background-color: powderblue">
<left style="background-color: lightblue;">
    <form id="form1" runat="server" style="width: 1000px;  background-
          color: teal">
    <div >
    <table border="0" cellpadding="0" cellspacing="0" style="width:
          1000px; z-index: 100; position:absolute"  >
```

```
        <tr>
           <td colspan="4" style="font-weight: bold; font-size: 28px;
               border-bottom-color: teal; vertical-align: middle;
               text-transform: uppercase; color: teal; font-family:
               'Book Antiqua'; height: 103px; background-color:
               lightblue; text-align: center; border-bottom-style: solid">
               Računalništvo
            </td>
        </tr>
       <tr>
          <td style="width: 135px; height: 705px; background-color:
              cornflowerblue;" valign="top">
              <asp:TreeView ID="tvMeni" runat="server"
                            DataSourceID="SiteMapDataSource1"
                            NodeIndent="10"
                            style="width: 200px">
              </asp:TreeView>
                   
          </td>
          <td style="height: 1024px; width: 203%; color: black;
              background-color: powderblue;" valign="top">
              <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                                      runat="server" >
              </asp:ContentPlaceHolder>
                  
          </td>
       </tr>
    </table>
    <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
    </div>
    </form>
</left>
              
   <td style="width: 213px; height: 705px; background-color: cornflowerblue"
       valign="top">
   </td>
</body>
</html>
```

Kot smo povedali že v poglavju o spletnih gradnikih, ta predloga določa, da je stran razdeljena v tri dele. V osrednjem delu se izvaja sama aplikacija (gradnik `ContentPlaceHolder`), zgoraj je pasica z imenom predmeta. Levo pa smo s pomočjo gradnika `SiteMapDataSource` postavili drevesni pregled ustreznih strani, ki nam služi kot menujski sistem.

Login.aspx

`Login.aspx` je začetna stran aplikacije. Do nje lahko dostopa vsak uporabnik, ki ve naslov spletne strani s to datoteko. Vendar brez uporabniškega imena in gesla v aplikacijo ne more vstopiti in podatke pregledovati. Ko uporabnik vpiše pravilno uporabniško ime in geslo, se odpre stran za pregled rezultatov.

Na levi strani je meni, kjer se lahko uporabnik sprehaja po straneh. Študentje imajo možnost dostopa do strani pod napisom Ogledovalec, Učitelji pa lahko s svojim geslom dostopajo do vseh strani. Če neprijavljen uporabnik klikne v meniju na eno od strani pod napisom Ogledovalec oz. Vnašalec ali če uporabnik, ki pripada vlogi Študent, klikne na eno od strani pod napisom Vnašalec, se prikaže začetna stran, ki omogoča prijavo uporabnika.

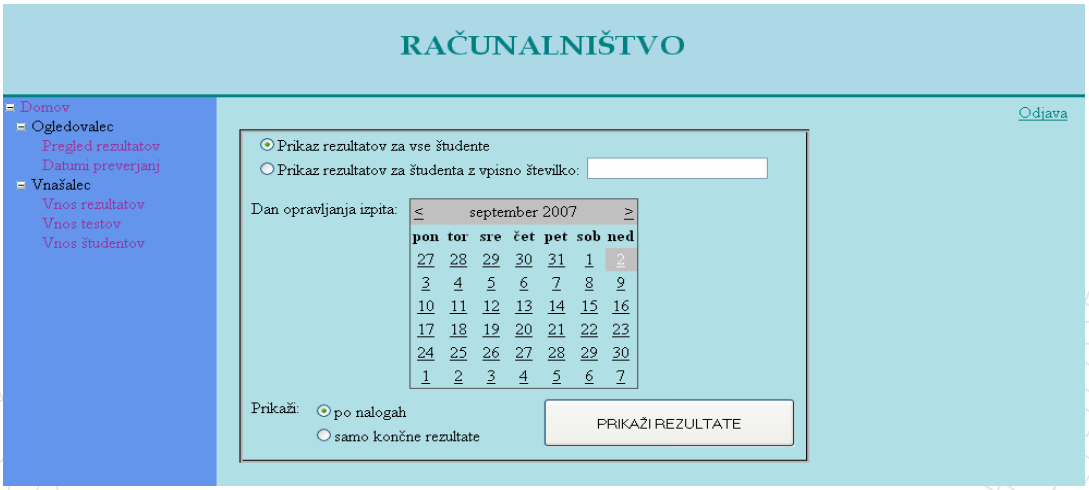
Pod urnikom so tudi hiperpovezave do treh zanimivih strani. Na te povezave lahko klikne vsak, tudi neprijavljeni obiskovalec.

Slika 57 - Začetna stran

Osrednji del te strani je gradnik Login, katerega uporabo smo opisali v razdelku o varnosti, v razdelku o prikazovalniku za prijavo uporabnika (stran 43). Urnik je sestavljen s pomočjo tabele, ki je primer hipertekstnega gradnika.

PregledRezultatov.aspx

Ta stran je namenjena pregledovanju rezultatov. Uporabnik se lahko odloči, ali želi pregledovati rezultate za vse študente, ali le za enega. V slednjem primeru mora vnesti še vpisno številko študenta. Na koledarju mora izbrati datum opravljanja testa. Potem je treba označiti še želen izpis. Ta je bodisi tak, da prikaže rezultate vsake posamezne naloge ali pa tak, da so izpisani le končni rezultati testa.



Slika 58 - Pregledovanje rezultatov

Kot vidimo, smo na strani dvakrat uporabili seznam izbirnih gumbov, kjer imamo možnost izbire ene od možnosti, besedilno polje je namenjeno za vnos vpisne številke., na koledarju se izbere želen datum. Vsi ti gradniki so opisani v poglavju o spletnih gradnikih (stran 20). Prav tako je na strani še gumb, ki najprej sproži preverjanje. Več o preverjanjih je opisano v poglavju o gradnikih za preverjanje na strani 31.
Ko uporabnik klikne na gumb, se najprej preveri, če sploh obstaja kombinacija tega datuma in vrste testa v bazi podatkov:

```
'Datoteka PregledRezultatov.aspx.vb
Protected Sub btnRezultati_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnRezultati.Click
```

```
Try
    'Deklaracija
    Dim strConn As String = System.Configuration & _
    .ConfigurationManager.ConnectionStrings("Student") & _
    .ConnectionString
    Dim conn As OleDbConnection = New OleDbConnection(strConn)
    conn.Open()

    Dim i, j, stMoznihTock, stNalog As Integer
    Dim sqlRezultati As String = ""
    Dim sql As String = ""
    Dim skupnoTock As Double = 0

    Dim dr As DataRow
    Dim dt, dtPodatki As New DataTable
    Dim daStMoznihTock, daNaloge As OleDbDataAdapter

    Dim izbranDatum As Date = cldDan.SelectedDate
    ' Datum mora biti oblike dd.mm.llll zaradi podatkovne baze Access
    Dim datum As String = izbranDatum.Day & "." &
                          izbranDatum.Month & "." & izbranDatum.Year

    sql = "SELECT ST_MOZNIH_TOCK " & _
          "FROM TEST " & _
          "WHERE DATUM LIKE '" & datum & "' "

    ' Postavimo dataAdapter in ga napolnimo s podatki, ki jih
    ' dobimo s pomočjo stavka SELECT v spremenljivki sql
    daStMoznihTock = New OleDbDataAdapter(sql, conn)
    ' Napolnimo tabelo s podatki iz dataAdapterja
    daStMoznihTock.Fill(dt)

    ' Če je tabela prazna, uporabnika opozorimo, da za ta datum ni
    ' nobenih rezultatov, v nasprotnem primeru pa pogledamo
    ' kolikšno je število možnih točk pri tem testu
    If dt.Rows.Count = 0 Then
       MsgBox("Za ta datum ni nobenih rezultatov " & aliKolokvij
              & ".", MsgBoxStyle.ApplicationModal, "Opozorilo")
       Exit Sub
    Else
       stMoznihTock = dt.Rows(0)("ST_MOZNIH_TOCK")
    End If
```

Če je na označeni datum bil izveden test, najprej preberemo, kolikšno število nalog je bilo na tem testu. To število dobimo s pomočjo stavka SELECT.

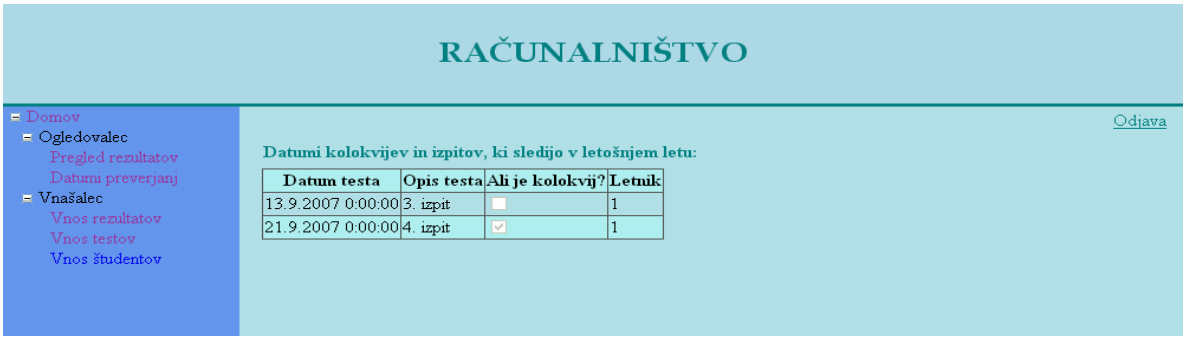
```
    ' Preberemo število nalog na tem izpitu / kolokviju
    sql = "SELECT DISTINCT T.ID_NALOGE " & _
          "FROM TEST TE " & _
          "INNER JOIN TOCKE T ON T.ID_TEST = TE.ID_TEST " & _
          "WHERE TE.DATUM LIKE '" & datum & "' "

    ' Postavimo dataAdapter in ga napolnimo s podatki, ki jih
    ' dobimo s pomočjo SELECT stavka sql
    daNaloge = New OleDbDataAdapter(sql, conn)
    ' Napolnimo tabelo s podatki iz dataAdapterja
    daNaloge.Fill(dt)
    ' Preštejemo število vrstic, ki predstavljajo ravno število
    ' nalog pri tem izpitu / kolokviju
    stNalog = dt.Rows.Count
```

```
        ' manjka koda, ki iz baze pobere podatke glede na želeno izbiro

        ' Nastavimo podatkovni vir prikazovalnika podatkov na tabelo
        gvRezultati.DataSource = dtPodakti
        ' Nastavimo tekst prve vrstice v tabeli rezultatov
        gvRezultati.Caption = "Rezultati z dne " & datum & Chr(13) &
                              "  (št. možnih točk: " & stMoznihTock & ")"
        ' Vežemo tabelo na prikazovalnik podatkov (gridView)
        gvRezultati.DataBind()
        ' Zapremo povezavo z bazo
        conn.Close()
        ' V primeru napake se pojavi opozorilo.
    Catch ex As Exception
        MsgBox("Prišlo je do napake pri pregledu rezultatov."
               & Chr(13) & ex.Message, MsgBoxStyle.Exclamation, "Napaka")
    End Try
End Sub
```

DatumiPreverjanj.aspx



Slika 59 - Datumi preverjanj

Ko neprijavljeni uporabnik klikne na hiperpovezavo za pregled datumov preverjanj, program v datoteki web.config najprej preveri, če obstaja pravilo za neprijavljenega uporabnika. Ker je neprijavljenemu uporabniku prepovedan dostop do spletne strani v mapi PregledKolokvijev,

```
<deny users="?" />
```

se stran preusmeri na začetno stran, ki omogoča prijavo.

web.config (mapa Admin)

Datoteka `Web.config` velja le za strani aspx v podmapi Admin. Nastavitve v tej datoteki `web.config` »prekrijejo« nastavitve iz nadmape PrikazKolokvijev.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
    <system.web>
        <authorization>
            <!-- Študentom je prepovedan dostop do strani v mapi Admin. -->
            <deny roles="Študent" />
        </authorization>
    </system.web>
</configuration>
```

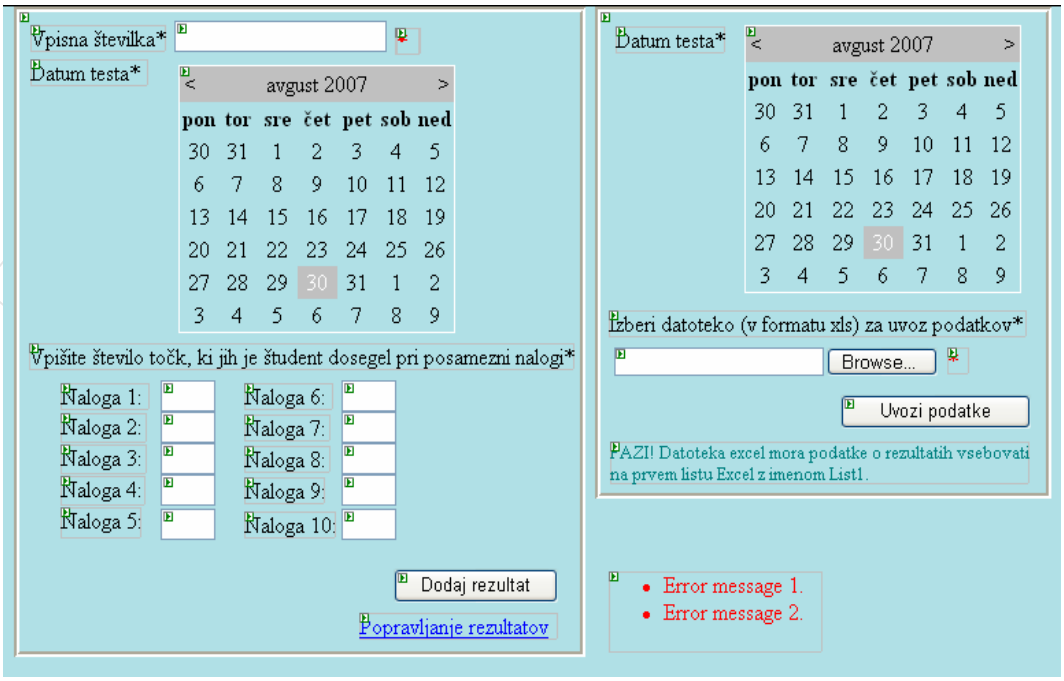
Ko se sestavljajo pravila v administratorskem vmesniku (stran 44), se le te avtomatsko vpišejo v datoteko `web.config`, kot vidimo v zgornji kodi, ki nam pove, da je uporabnikom, ki pripadajo vlogi Študent, prepovedan dostop do strani v mapi Admin.

VnosRezultatov.aspx

Uporabnik ima možnost vnašanja rezultatov za posameznega uporabnika ali za večje število študentov. Pri prvi možnosti mora vpisati vpisno številko študenta, datum testa in rezultate nalog. Če test z vpisanim datumom ni v bazi podatkov, se pokaže opozorilo.
Druga možnost je, da uporabnik uvozi podatke iz datoteke Excel. Predpostavlja se, da so podatki vpisani v datoteko v pravilnem vrstnem redu: v prvi vrstici so našteta imena stolpcev ID_STUDENT, Naloga1, Naloga2 … Imena samih stolpcev niso pomembna, saj se v dataAdapter shranijo le podatki od druge vrstice naprej, pomemben pa je njihov vrstni red. Pomembno je tudi, da so vsi podatki na datoteki Excel na listu z imenom List1.
Uporabili smo gradnik za izbiranje in preverjanje datotek, ki smo ga spoznali v poglavju o spletnih gradnikih (stran 20). Na sliki 55, ki prikazuje obrazec za vnos rezultatov v načrtovalnem načinu, lahko vidimo tudi gradnike za preverjanje. Pri vpisni številki sta postavljena dva gradnika za preverjanje, gradnik za preverjanje obveznega vnosa in primerjalnik vrednosti, s katerim s pomočjo lastnosti `DataTypeCheck`, preverimo, če je vpisan niz res število. Pri gradniku za izbiranje in preverjanje datotek je prav tako postavljen gradnik za preverjanje obveznega vnosa. Na obrazcu je postavljen tudi gradnik za prikaz opozoril, ki ob napačnem oz. pomanjkljivem vnosu, prikaže vsa opozorila. Več o teh gradnikih je napisanih v poglavju o gradnikih za preverjanje na strani 31.

Oglejmo si metodo, ki prebere rezultate iz datoteke Excel:

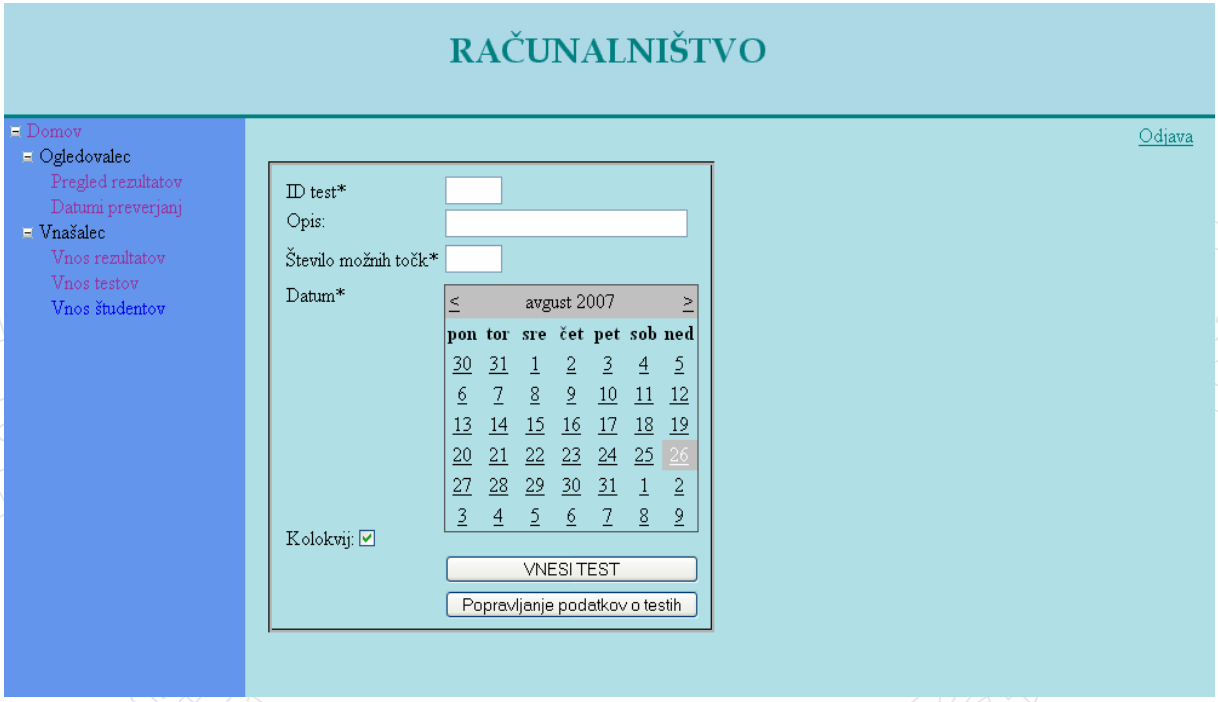
```
Private Function BranjeExcelRezultati() As String
   Try
      ' Ustvarimo začasno mapo, kamor bomo shranili Excel datoteko,
      ' ki jo je uporabnik naložil
      Directory.CreateDirectory("C:\Zacasna")
      ' Shranimo datoteko Excel v to novoustvarjeno mapo
      If fuTest.HasFile Then
         fuTest.SaveAs("C:\\Zacasna\\" & fuTest.FileName)
      End If
      ' Naredimo povezovalni niz za povezavo z datoteko Excel
      Dim strConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;"&_
      "Data Source=C:\Zacasna\" & fuTest.FileName & ";" & _
      "Extended Properties=""Excel 8.0;"""
      ' Postavimo dataAdapter in ga napolnimo s podatki, ki jih
      ' dobimo s pomočjo stavka SELECT
      Dim da As New OleDbDataAdapter("SELECT * FROM [List1$]", strConn)
      ' Napolnimo tabelo s podatki iz dataAdapterja
      da.Fill(dtTockeExcel)
      ' Nastavimo tabelo kot podatkovni vir za prikza na prikazovalniku podatkov
      gvUvoz.DataSource = dtTockeExcel
      ' Vežemo tabelo na prikazovalnik podatkov (gridView)
      gvUvoz.DataBind()
     ' Izbrišemo najprej datoteko, ki jo je uporabnik naložil
     ' (metoda Delete briše le prazne mape)
     File.Delete("C:\Zacasna\" & fuTest.FileName)
     ' Izbrišemo začasno mapo, ki smo jo ustvarili
     Directory.Delete("C:\Zacasna")
     lblExcel.Visible = True
   Catch ex As Exception
      ' V primeru napake izbrišemo datoteko in mapo, ki smo jo ustvarili
      File.Delete("C:\Zacasna\" & fuTest.FileName)
      Directory.Delete("C:\Zacasna")
      Return "Napaka pri branju iz datoteke Excel."
   End Try
   Return ""
End Function
```



Slika 60 - Vnos rezultatov (pogled v načrtovalnem načinu)

VnosTestov.aspx

Ta stran je namenjena vnašanju testov. Na obrazcu so podobni gradniki kot na strani za pregled rezultatov, le da imamo tu namesto izbirnih gumbov potrditveno polje. Če ga uporabnik odkljuka, je dodan test vrste kolokvij, v nasprotnem primeru pa vrste izpit.



Slika 61 - Vnos testov

Obrazec vsebuje tudi gumb za možnost popravljanja podatkov o testih. Ko uporabnik klikne na gumb Popravljanje podatkov o testih, se prikaže tabela vseh testov:

	ID_test	Opis	Datum	St_moznih_tock	Ali_kolokvij
Popravi Izbriši	7	3. izpit	13.9.2007 0:00:00	112	☐
Popravi Izbriši	8	4. izpit	21.9.2007 0:00:00	100	☑
Popravi Izbriši	9	test	4.7.2007 0:00:00	33	☐
Popravi Izbriši	11		13.7.2007 0:00:00	34	☑
Popravi Izbriši	111		5.8.2007 0:00:00	34	☑
Popravi Izbriši	10	frege	23.8.2007 0:00:00	117	☐
Shrani Prekliči	85	767	24.8.2007 0:00:00	76	☑
Popravi Izbriši	1	Osnove HTML	11.11.2006 0:00:00	100	☑
Popravi Izbriši	2	Osnove Jave	10.1.2007 0:00:00	90	☑
Popravi Izbriši	3	Derive	17.4.2007 0:00:00	110	☑
Popravi Izbriši	4	Celoletni pregled	28.5.2007 0:00:00	110	☑
Popravi Izbriši	5	1. izpit	6.6.2007 0:00:00	90	☐
Popravi Izbriši	6	2. izpit	25.6.2007 0:00:00	115	☐

Slika 62 - Popravljanje podatkov o testih

S klikom na povezavni gumb Popravi se prikaže besedilno okno, v katerem lahko popravimo podatke. Nato popravljene podatke shranimo ali pa prekličemo popravljanje. Podatke o testih lahko tudi brišemo s klikom na povezavni gumb Izbriši.
Naslednja koda prikazuje prikazovalnik podatkov in podatkovni vir AccessDataSource v urejevalniku kode:

```
<asp:GridView ID="gvTest" runat="server"
              Style="z-index: 106; left: 229px; position: absolute; top: 558px"
              DataSourceID="adsTest" DataKeyNames="ID_TEST" Visible="False">
   <Columns>
      <asp:CommandField CancelText="Prekliči"
                        CausesValidation="False" DeleteText="Izbriši"
                        EditText="Popravi" ShowDeleteButton="True"
                        ShowEditButton="True" UpdateText="Shrani" >
         <ItemStyle ForeColor="Teal" />
         <ControlStyle ForeColor="Teal" />
      </asp:CommandField>
   </Columns>
   <HeaderStyle BackColor="PaleTurquoise" ForeColor="Teal" />
   <AlternatingRowStyle BackColor="PaleTurquoise" />
</asp:GridView>
<asp:AccessDataSource ID="adsTest" runat="server"
                      DataFile="~/App_Data/Student.mdb"
                      DeleteCommand="DELETE FROM [Test] WHERE [ID_test] = ?"
                      UpdateCommand="UPDATE [Test] SET [Opis] = ?,
                                     [Datum] = ?,[St_moznih_tock] = ?,
                                     [Ali_kolokvij] = ? WHERE [ID_test] = ?"
                      SelectCommand="SELECT [ID_test], [Opis], [Datum],
                                     [St_moznih_tock], [Ali_kolokvij]
                                     FROM [Test]">
   <DeleteParameters>
      <asp:Parameter Name="ID_test" Type="Int16" />
   </DeleteParameters>
   <UpdateParameters>
      <asp:Parameter Name="Opis" Type="String" />
      <asp:Parameter Name="Datum" Type="DateTime" />
      <asp:Parameter Name="St_moznih_tock" Type="Decimal" />
      <asp:Parameter Name="Ali_kolokvij" Type="Boolean" />
      <asp:Parameter Name="ID_test" Type="Int16" />
   </UpdateParameters>
</asp:AccessDataSource>
```

VnosStudentov.aspx

Ta stran je namenjena vnašanju študentov. Kot pri vnosu rezultatov imamo tudi tu možnost vnosa posameznega študenta in uvoza študentov preko datoteke Excel. Pri izbiranju letnika si uporabnik pomaga s spustnim seznamom, možnost pa ima klikniti tudi na povezavni gumb, ki nam prikaže seznam vseh študentov, in možnost popravljanja njihovih podatkov.
Ta povezavni gumb ima lastnost `CausesValidation` nastavljeno na `False`, saj ne želimo, da klik nanj sproži preverjanje pri gradnikih za preverjanje. Ti so postavljeni zraven vsakega besedilnega polja.V tem primeru, bi dobivali sporočila o pomankljivem vnosu toliko časa, dokler ne bi bili izpolnjeni vsi pogogu pri preverjanju..
Če dodajamo posameznega študenta, lahko opazimo, da se ob pravilno vpisanih podatkih in ob kliku na gumb za dodajanje, ne sproži opozorilo pri praznem besedilnem oknu za uvoz podatkov, kljub temu, da ob njem stoji gradnik za preverjanje obveznega vnosa. To se ne zgodi zato, ker ima gumb za dodajanje študenta nastavljeno lastnost `ValidationGroup` na student, enako lastnost in vrednost pa imajo nastavljena tudi vsa besedila polja pri dodajanju posameznega študenta, tako da se ob kliku na gumb sproži preverjanje le za gradnike, ki imajo `ValidationGroup` nastavljeno na student.

Slika 63 - Vnos študentov

ZAKLJUČEK

V svoji diplomski nalogi sem obravnavala tehnologijo za razvoj spletnih aplikacij ASP.NET. Pred tem sem o ASP.NET-u, o strežnikih in ostali spletni tehnologiji vedela bolj malo. V času študija sem spoznala le jezik HTML in pripravo statičnih spletnih strani. Prav tako sem pisala le aplikacije, ki se izvajajo v okolju Windows. Te se v kar nekaj pogledih razlikujejo od spletnih aplikacij. Zaradi tega sem imela v začetku pisanja diplomske naloge nemalo težav. Tehnologija, ki omogoča nastajanje Windows aplikacij, je že veliko bolj razvita in se zaradi tega da narediti veliko več uporabniku, na izgled, bolj uporabnih strani. Pogrešala sem predvsem različne kazalce (*Cursors*), gradnike, kot npr. numerični števec (*NumericUpDown*), indikator napredka (*ProgressBar*) …
Z načrtovanjem svoje spletne aplikacije, branjem različnih knjig s to tematiko in predvsem s poskušanjem, sem pridobila veliko znanja s področja načrtovanja spletnih strani s pomočjo tehnologije ASP.NET. Aplikacijo, ki sem jo uporabila kot primer, bi se dalo še razviti, optimizirati in vizualno popraviti. Prav tako bilo verjetno smiselno pri razvoju uporabiti tudi tehnologijo AJAX. Na voljo je brezplačno ogrodje ASP.NET AJAX, ki je združljivo s tehnologijo ASP.NET. Ta vsebuje veliko dodatni knjižnic, gradnikov, ki so bolj prijazni uporabniku, zanimiva pa je predvsem novost – asinhrono osveževanje, ki se izogne nenehnemu osveževanju strani. Več o tem si lahko preberete na [8].

LITERATURA

1. D. Esposito, Programming Microsoft ASP.NET 2.0 Core Reference, Microsoft Press, 2006
2. J. Buyens, Microsoft Visual Web Developer 2005 Express Edition: Build a Web Site Now!, Microsoft Press, 2006
3. B. A. DePetrillo, Razumeti Microsoft .NET, Pasadena, 2002
4. P.Mrhar, Active Server Pages, Založba Flamingo, 2002

Spletni viri

5. Uradna spletna stran ASP.NET, Dostopno na naslovu http://www.asp.net/, (zadnji obisk: 2. 9. 2007)
6. Knjižnica MSDN (online). Dostopno na naslovu http://msdn2.microsoft.com/en-us/library/default.aspx, (zadnji obisk: 2. 9. 2007)
7. Full Web Building Tutorials W3 Schools (online). Dostopno na naslovu http://www.w3schools.com/, (zadnji obisk: 2. 9. 2007)
8. ASP.NET in Ajax (online). Dostopno na http://asp.net/ajax/, (zadnji obisk: 2. 9. 2007)