**Andmebaasipõhiste veebirakenduste arendamine Microsoft Visual Studio   
ja SQL Server’i baasil**

**ASP.NET**

Tallinn  
2011

Sisukord

[ASP.NET 3](#_Toc294497554)

[Põhivõimalused 4](#_Toc294497555)

[Lihtsa veebilehestiku loomine HTML keele abil 4](#_Toc294497556)

[Astmelised laadilehed (CSS) – cascading style sheets 26](#_Toc294497557)

[Programmeeritavad veebilehed 35](#_Toc294497558)

[Andmebaasipõhise veebirakenduse loomine 46](#_Toc294497559)

[Ülesandeid 50](#_Toc294497560)

[Andmetabeli näitamine veebilehel 51](#_Toc294497561)

[Ülesandeid 53](#_Toc294497562)

[Andmete lisamine 53](#_Toc294497563)

[Ülesandeid 55](#_Toc294497564)

[Lehe kohendamine 55](#_Toc294497565)

[Ülesandeid 61](#_Toc294497566)

[Järjestamine 62](#_Toc294497567)

[Ülesandeid 65](#_Toc294497568)

[Otsimine 65](#_Toc294497569)

[Ülesandeid 69](#_Toc294497570)

[Konfiguratsioonifail web.config 69](#_Toc294497571)

[Edasijõudnutele 112](#_Toc294497572)

[Veebi kopeerimine 112](#_Toc294497573)

[Programmi koodi paigutamine eraldi faili 113](#_Toc294497574)

[Seadistamine (Web.config) 114](#_Toc294497575)

[Rakenduse jälgimine (Trace) 115](#_Toc294497576)

[Vigade haldamine 120](#_Toc294497577)

[AJAX 124](#_Toc294497578)

[Lokaliseerimine 126](#_Toc294497579)

[Master Pages 131](#_Toc294497580)

[Elemendid lehel 133](#_Toc294497581)

[Veebilehtede kujundamine kasutades nägusid (Themes) 147](#_Toc294497582)

[Väärtuste tööaegne meelespidamine 149](#_Toc294497583)

[Veebisaidi turvamine 152](#_Toc294497584)

[Andmetega manipuleerimine 161](#_Toc294497585)

[WebParts 170](#_Toc294497586)

[Veebiteenused 171](#_Toc294497587)

[IIS 176](#_Toc294497588)

[Kokkuvõte 177](#_Toc294497589)

[Lisad 178](#_Toc294497590)

# ASP.NET

ASP.NET on .NET raamistiku moodul, mis võimaldab sul luua veebirakendusi, kasutades sealjuures minimaalselt koodi.

ASP.NET ei ole mitte ASP (Active Server Pages) uus versioon, vaid täiesti uus lähenemine veebirakenduste loomisele. Erinevalt ASPist ja ka PHPst, mis on peamiselt skriptimise keeled, on ASP.NET lehtede taga olev kood täielikult objektorienteeritud. Seega tuleks ASP.NETi võrrelda mitte PHP vaid JAVA rakendustega. Kasutaja saab, kuid ei pruugi täpselt mõelda HTMLi eripärade peale. Pigem määrab ta, milliseid komponente ta soovib veebilehel näha ning need näidatakse, arvestades vajadusel kasutaja veebilehitseja eripäradega – eriti kehtib see mobiilseadmete kohta.

Koodi ASP.NET lehtede tarbeks võib kirjutada ükskõik millises .NET keeles. Lisaks veebivormidele on võimalik oma rakendust veebis serveerida ka läbi veebiteenuste.

Ka ASP.NETist on olemas mitmeid versioone. Kui võrrelda ASP.NET versioone 1.1, 2.0, 3.5 ja 4.0 siis võib öelda, et palju on jäänud samaks, kuid üht teist on ka ümber tehtud ja lisatud. Hea uudis on see, et kõik vanad konstruktsioonid töötavad, kuid juurde on tulnud mitmed uued meetodid. Tänu provaideri(teenusepakkuja) põhisele lähenemisele on rakenduse loomine alates 2.0 versioonist muutunud märksa abstraktsemaks ja lihtsamaks. Palju koodi on viidud lehekülje tagustelt koodilehtedelt teenusepakkujatesse. ASP.NET versioonis 3.5 on juurde tulnud toetus AJAXile (Asyncronious Java and Xml), mõned uued serveri kontrollid ning LINQ kasutamise võimalus. AJAX võimaldab andmeid lehel reaalajas muuta ilma, et peaks kogu lehte selle tarbeks uuesti laadima.

ASP.NET lehed koosnevad tekstifailidest, mida saab serveerida läbi IIS (Internet Information Service) virtuaalkaustade. Lehtede loomiseks sobivad kõik tekstiredaktorid. Abivahendeid usaldavale inimesele on kõige parem kasutada kirjutamiseks sellised abivahendeid, mis lehtede loomisel kiirendavad koodi kirjutamist lõpetades alustatud sõnu, kontrollivad jooksvalt süntaksit ning aitavad HTMLi loomisel. Üheks selliseks abivahendiks on Visual Studio. Ehkki palja "rumala" tekstiredaktoriga kirjutamisel on eeliseks lihtsus, siis nt Visual Studio Web Developer Expressiga on lootus ka algajal nõnda läbi saada, et ta paljudesse menüüdesse ära ei upu.

## Põhivõimalused

### Lihtsa veebilehestiku loomine HTML keele abil

#### Esimene veebileht

Kõige lihtsam veebirakendus koosneb ühest lehest. Veebilehtede levinumaks keeleks on HTML, mida veebilehitsejad mõistavad lugeda ning loetud teksti põhjal kasutajale lehe ette kuvada. Mitmete veebitehnoloogiate (ASP.NET, Java servlet, PHP, Python ...) tulemusena lehitsejasse saadetav tekst on ikkagi „puhas“ HTML, nii et osava peitmise korral ei pruugi veebisaidi vaatajal kuidagi võimalik olla kindlaks teha, millise tehnoloogia abil vastav lehestik on kokku pandud. Ning nagu varemalt kombeks ning praegugi lihtsamate, pidevat muutmist mitte vajavate lehtede puhul kasutatakse, võibki veebileht olla üks harilik HTMLi reeglitele vastav tekstifail, mida veebilehitsejas näidatakse. Mitmesugused tehnoloogiad on lihtsalt leidnud võimalusi, kuidas võimalikult mugavalt lehtedel olevad andmeid määrata vastavalt kasutaja soovidele. Lihtsaim HTMLi reeglitele vastav veebileht näeb aga välja järgmine:

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

<html xmlns="http://www.w3.org/1999/xhtml" >

<head>

<title>Minu esimene leht</title>

</head>

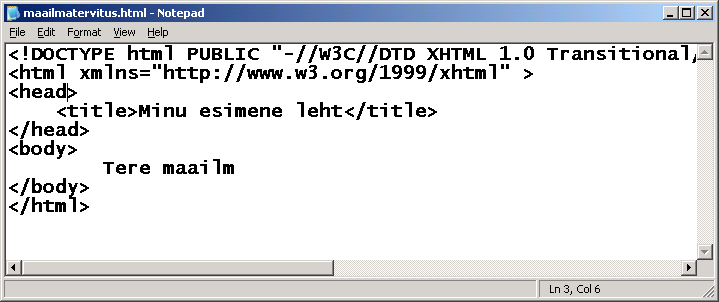
<body>

Tere maailm

</body>

</html>

Selle võib salvestada omale sobiva tekstiredaktoriga (nt. Notepad või ka vastinstalleeritud Visual Web Developer). Panna failile laiendiks .html (siis teab veebilehitseja, et vastavat teksti tuleb kujundada HTML-i reeglitele vastavalt), jätta meelde kuhu fail salvestati.



Edasi juba lehitsejas avada nagu tavalist kohaliku masina faili (failimenüüst ava). Tulemuseks leht, kus ülal pealkirjariba peal on lehe pealkiri ning allpool sisu osas lehe sisu.

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

<html xmlns="http://www.w3.org/1999/xhtml" >

<head>

<title>Minu esimene leht</title>

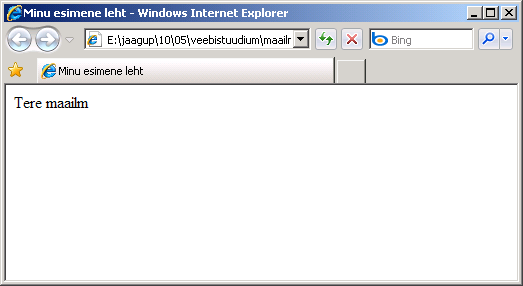
</head>

<body>

Tere maailm

</body>

</html>



Veidi lähemad seletused, et mida miski HTMLi lehe osa tähendada võiks.

Avakäsklus teatab HTMLi versiooni. Nii nagu nt. Wordi dokumentidel on versioonid 2.0, 6.0, 97, 2000, 2003, 2007 jne, nõnda ka HTML on oma arengu käigus muutunud. Kõiki eri versioone ja „murrakuid“ lugedes saaks neid kokku õite mitukümmend. Siit võib välja lugeda, et versiooniks on XHTML 1.0 Transitional – ehk siis 1999ndast ligi kümnendi püsinud kirjapanekuvorming.

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

Edasi hakkavad HTMLi käsklused. Märkide < ja > vahel olevate sõnadega antakse teada, mis nüüd tulemas on. Esimese elemendi nimeks ongi html, ehk siis lehitsejale teadmiseks, et tulemas htmli dokument. Juuresolev atribuut xmlns (xml namespace) näitab, millise nimeruumiga elemendid seotud on. XHTML 1.0 Transitionali puhul siis järgnev nimi: [www.w3.org/1999/xhtml](http://www.w3.org/1999/xhtml). Näeb välja väga hüperlingi moodi. Aga selline kuju võeti nimeruumide nimetuste puhul ette vaid selleks, et kogemata ei satuks mitmel eri firmal ette samanimelist nimeruumi. Kas vastavale lehele ka selgitav tekst andmete kohta pannakse – see on juba vajaduse ja viisakuse küsimus.

<html xmlns="http://www.w3.org/1999/xhtml" >

Veebilehe kood koosneb kahest suhteliselt iseseisvast osast. Ühe nimeks head (päis) ning teiseks body (sisu). Esimesse neist pannakse pealkiri (title) ning soovi korral lisaks igasugu muud andmed lehe kohta, mida otse näidata ei soovita. Näiteks lehe autor, märksõnad, kooditabel jm.

<head>

<title>Minu esimene leht</title>

</head>

Sisu osa lehel on kõigile nähtav. Lihtsamal juhul ongi siin paljas tervitav tekst, aga eks hiljem saab kujundust keerukamaks hakata muutma.

<body>

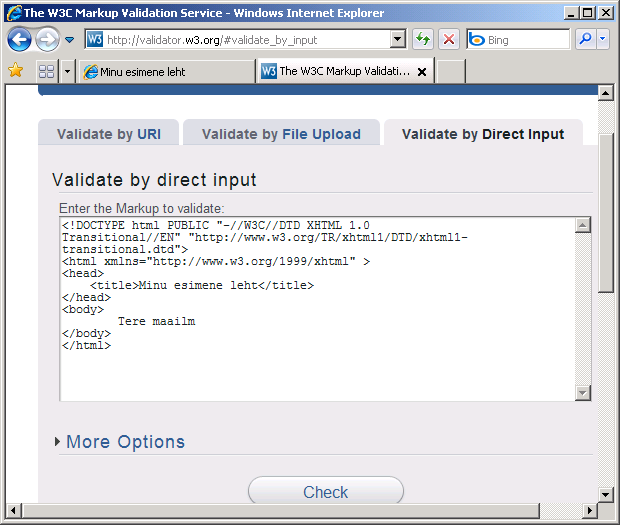
Tere maailm

</body>

Nagu näha, siis kõik alustavad elemendid peavad kusagil ka lõppema. Lehe sisuosa tähistav element body lõpeb lõpetava tähisega </body> ning dokumendi enese </html>.

</html>

Et suurem lehekülg võib ülesehituselt päris keerukaks minna, siis on vaja abivahendeid kontrollimaks, kas kõik ikka reeglitele vastab. Enamik lehitsejaid suudab küll mõningad (trüki)vead ka ise ära aimata ning siiski lehe viisakalt välja näidata. Kuid sealjuures võib hakata tekkima probleeme. Näiteks, et kas tekstid olid mõeldud üksteise kõrvale või üksteise alla. Kui veebilehe looja on oma lehe validaatoris järgi kontrollinud, siis võib loota, et lehitsejad saavad sellest enamvähem sarnaselt aru. Ametlikuks veebilehe tehnilise korrektsuse kontrolliks on loodud teenus aadressil <http://validator.w3.org>. Seal võimalik kontrollida kas juba ülesriputatud lehe korrektsust, laadida üles fail või kopeerida olemasolev HTMLi kood otse tekstiaknasse. Lihtsa väikese koodi puhul, nagu siinne alustus, on viimatine võimalus ehk mugavaim. Kui mõni viga juhtub, antakse sellest teada. Muul juhul aga teatatakse uhkesti rohelisel kirjal, et sisestatud kood on reeglipärane.

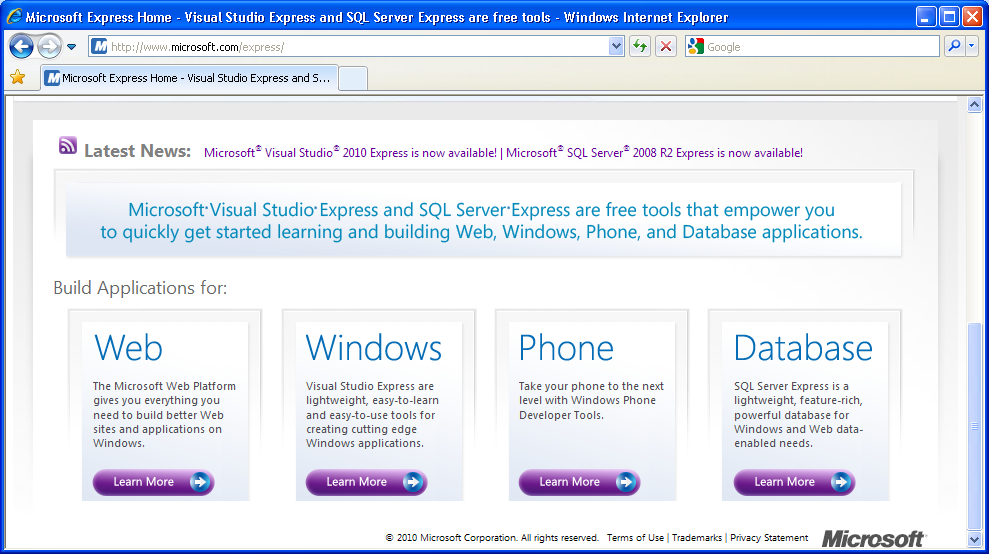


Ülesandeid

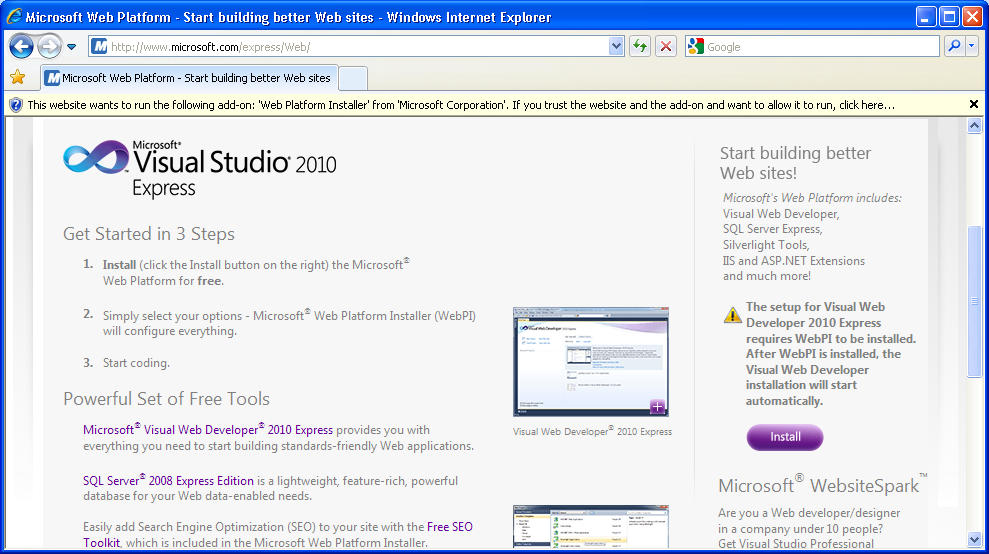
* Tee näide läbi. Vaata tulemust.
* Lisa tervitusele hüüumärk. Salvesta ja vaata veebilehitsejas tulemust.
* Veendu, et kood valideerub.
* Tekita HTML-koodi sisse viga (nt. eemalda üks < märk). Tutvu vastavate validaatori veateadetega.
* Paranda kood taas õigeks. Kopeeri validaatori antud „kvaliteedimärgi“ lõik oma koodi sisse lehe lõppu (enne </body>). Veendu, et pilt tuli nähtavale.

#### Visual Web Developer Expressi install

Tasuta kasutatavate Express-toodete jaoks tarviliku leiab aadressilt [www.microsoft.com/express](http://www.microsoft.com/express) . Veebi jaoks mõeldud toodete installimiseks on võimalik kasutada Web Platform Installerit, kus saab „linnukestega“ ära märkida soovitud komponendid.



Veebiosas Install-nupule vajutades saab vastava programmi kätte.



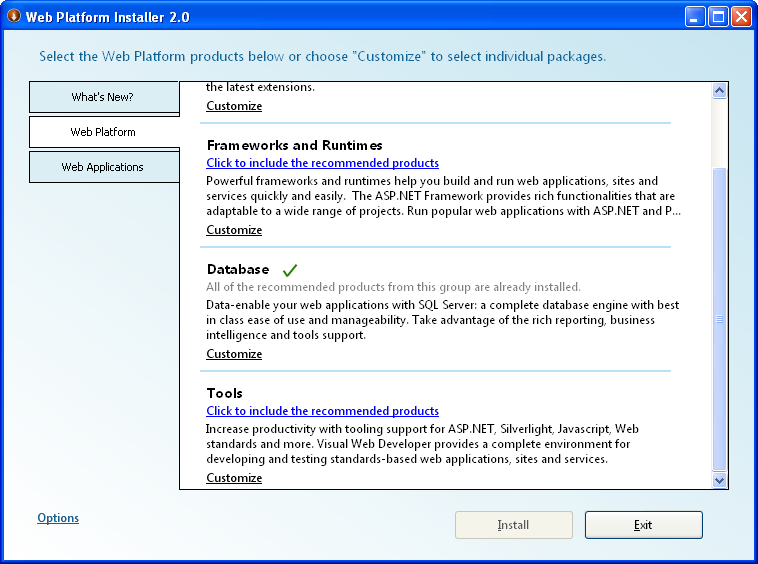
Tasub programm omale alla laadida

|  |  |
| --- | --- |
|  |  |

ja siis käima panna. Üldjuhul küsitakse, kas oled nõus laetud programmi käivitama. Vastavalt operatsioonisüsteemile võib küsimine erinev välja näha, aga administraatoriõigusi ja käivitusluba on igal pool vaja. Edasi kulub mõningane aeg vajalike täienduste laadimisele ning siis saab asuda paigaldatavaid komponente valima.

|  |  |
| --- | --- |
|  |  |

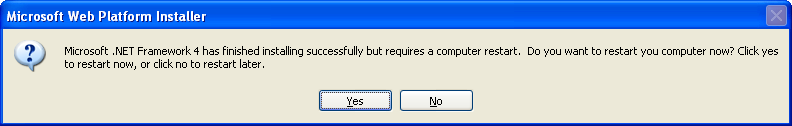
Veebilehestiku loomiseks vajalikud üksused peaksid üldjuhul olema vaikimisi juba valitud.



Tahtes aga seda üle kontrollida ja soovi järgi täiendusi lisada, saab menüüdes liikuda ja valikuid täiendada. Peab hoolitsema, et vähemasti Web Developer 2010 Express ning SQL Server peale saaksid.

|  |  |
| --- | --- |
|  |  |

Edasi on masinal hulk tegemist- kümnekonnast minutist mõne tunnini sõltuvalt veebi ühenduskiirusest ning masina enese jõudlusest. Pärast .NET 4 raamistiku installi lõppu palutakse arvuti taaskäivitada. Samuti tuleb taaskäivitus ette ülejäänud komponentide paigalduse järel. Seadistuse juures rohelistest ribadest ülemine näitab, kui palju failidest on kohale tõmmatud, kui palju veel. Alumisest näeb, mida parajasti paigaldatakse.



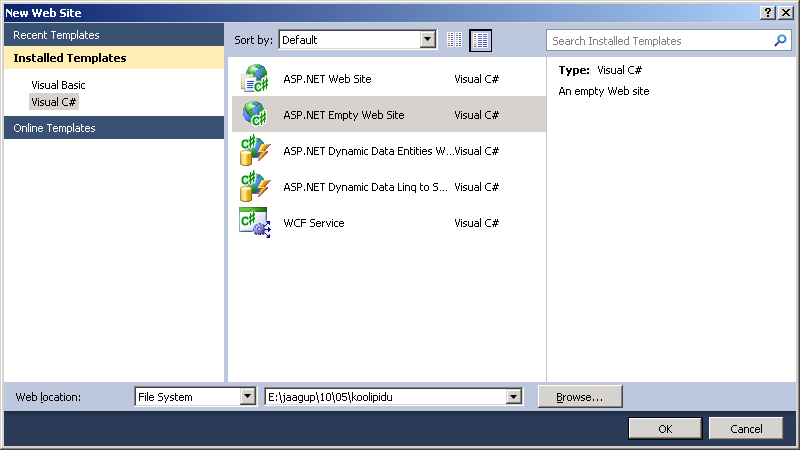
Pärast paigalduse lõppu on kasulik tarkvara registreerida – siis saab temaga töötada kauem kui prooviajaks ette nähtud 30 päeva. Help-menüüst Register Product – selle peale võimalik oma MSNi tunnusega sisse logida ning vajalik võti saada. Kui tunnust pole, saab selle samas luua.

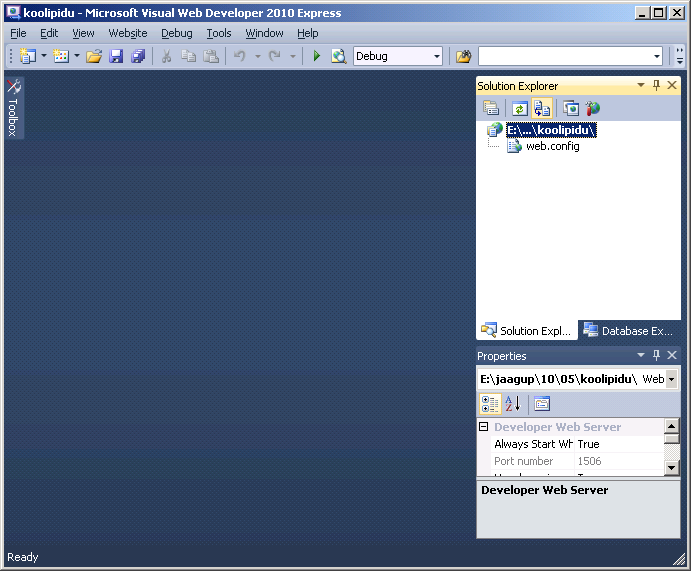
#### Esimene veebileht Visual Studio abil

Nagu näha, võib lehti luua igasuguse tekstiredaktori abil. Piisab teksti kirjutamisest, salvestamisest ning veebilehitseja mõistab lehel olevate HTMLi käskude ning nende vahel oleva sisu põhjal tulemuse välja näidata.

HTMLi käske on aga palju ning kõik neist ei pruugi sugugi kohe tuttavad olla. Samas aga enamik meist on kujundanud tekste mõne redaktori abil, kus saab valida värve, suurust jm. Seetõttu võib esmane lehtede loomine minna libedamalt mõne selleks tarbeks mõeldud vahendi abil. Et siin materjalis tutvustatakse Microsofti veebivahendeid, jääb paratamatult ette Visual Web Developer, mil ka veebilehtede koostamiseks omaette nurk olemas.

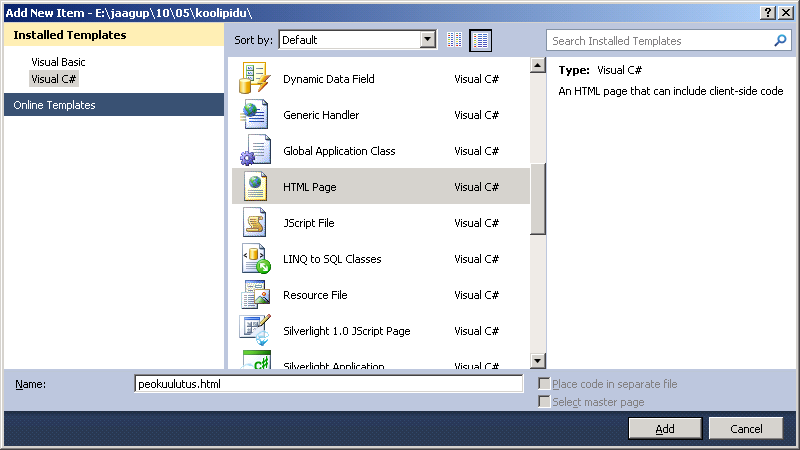
Uue veebilehestiku loomiseks tasub valida failimenüüst „New Web Site“. Ning et liialt palju lehti silmade kirjuks muutmiseks ette valmis ei genereeritaks, siis sobib valikust Empty Web Site. Allpool tasub märkida/luua kataloog, kuhu sisse loodav lehestik tuleb.



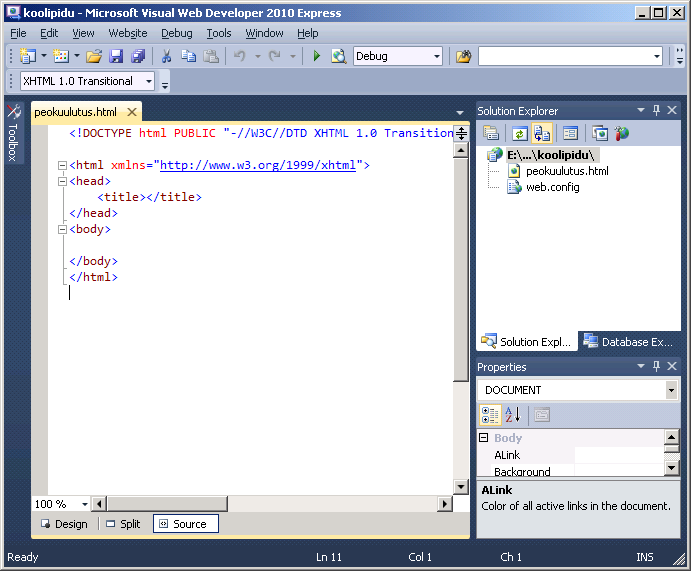


|  |  |
| --- | --- |
|  | Toimetuse tulemusena tekkis masinasse vastav kataloog vaid web.config-nimelise failiga. Sellest ei tasu end segada lasta. Web Developeri paremas servas võiks olla nähtaval Solution Explorer. Või kui seda mingil põhjusel seal ei paista, siis võiks aidata View menüüst valitud Solution Explorer. Selles parem hiireklõps rakenduse kataloogi nime peal ning võib valida enesele veebilehestiku uue elemendi. |

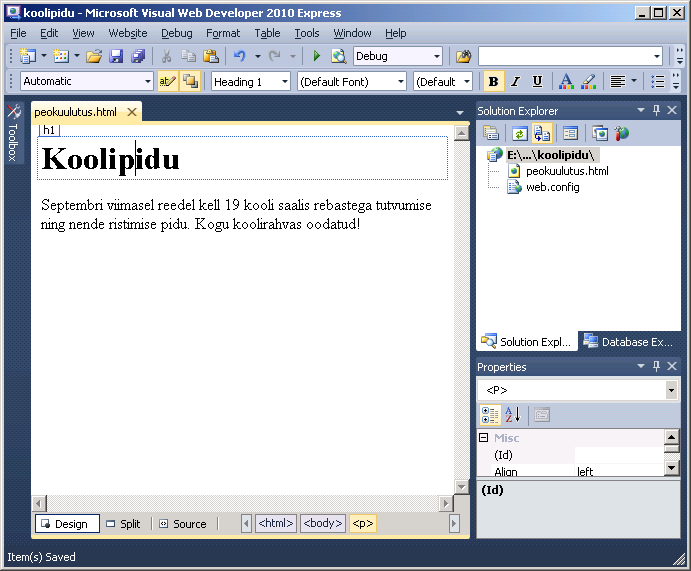
Sealt omakorda HTML-lehe ning lehe nimeks näiteks peokuulutus.html.



Tulemusena tekib HTMLi kest, mille sisse saab sobivaid teateid lisama hakata.



Üles <title> ja </title> vahele on hea siin sobiv teade panna – hiljem kipub see kergesti ununema. Edasi võib aga vaadete alt valida „Design“ ning saab lehte kujundada juba küllalt „tavalist“ redaktorit kasutades. Pealkirja tarbeks võib stiilide alt valida „Heading 1“ – selle tulemusena mõistetakse too rida juba ise nähtavamaks vormida.



Vahepeal võib huvi pärast lähtekoodi ehk Source vaatest piiluda, mis siis lehele pealkirja märkimise tulemusena tehti. Nagu näha, pandi sõna „Koolipidu“ h1-nimeliste käskluste vahele. Samuti pandi lõigumärk p (paragraph) ümber tavalisele nähtavale tekstile. Kui lõike oleks rohkem, siis saab nad lihtsalt nõnda üksteise järgi kirjutada, igaühele <p> alustuseks ning </p> lõpetuseks.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Koolipidu</title>

</head>

<body>

<h1>

Koolipidu</h1>

<p>

Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende

ristimise pidu. Kogu koolirahvas oodatud!</p>

</body>

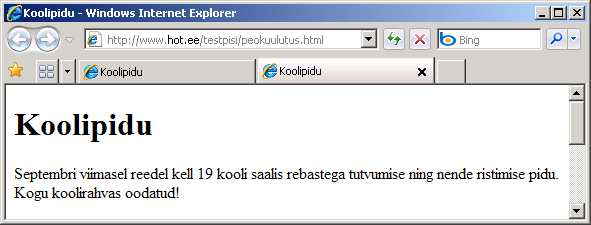
</html>

Lehe vaatamiseks tasub Web Developeris vajutada nupule, mille kirjelduseks „View in Browser“. Tulemuseks pannakse kohalikus masinas tööle veebiserver ning veebilehitseja ja loodud serveri kaudu saab nimetatud lehte vaadata. Iseenesest poleks palja HTML-lehe tarbeks serveri käivitamine hädavajalik – seda saab vaadata ka lihtsalt failina kohalikust kataloogist. Aga kuna Web Developer pigem mõeldud siiski iga kord serveris uuesti genereeritavate lehtede loomise tarbeks, siis on talle vastav võimalus sisse ehitatud. Nii võibki lehte rahumeeli imetleda aadressilt, mis hakkab tähekombinatsiooniga http – HyperText Transfer Protocol, järgneb masina nimi, kust lehte vaadatakse. Nimi localhost tähistab kohalikku masinat. Edasi pordi ehk värati number mis vajalik, et samas masinas olevad mitmed veebiühendust pakkuvad programmid omavahel tülli ei läheks. Siis juba rakenduse nimi ja sealt seest rakenduse sees oleva faili nimi.



Kui tahta loodud kuulutus teistele üle võrgu ka kättesaadavaks teha, tuleb see laadida lihtsalt kohta, kus see kõigile näha. Olgu selleks õpilasele koolivõrgus eraldatud veebiruum või ka mõnes avalikus serveris asuv koht. Eestis näiteks on levinud hot.ee nimeline veebikeskkond, kus rahumeeli igaüks saab omi HTML-faile hoida ja teistele näidata. Kel konto olemas, võib Kodulehe sektsiooni vastava faili laadida.

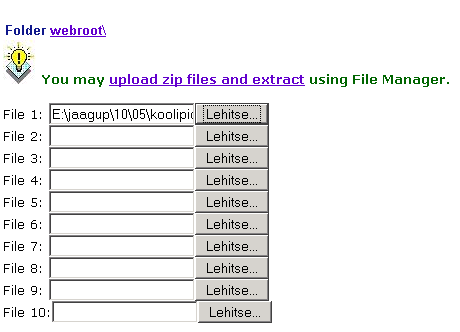
Edasi piisab juba veebilehitsejasse aadressi kirjutamisest ning igaüks võib üle võrgu vastavat teadet vaadata. Koolis saab viite panna nt. avalehele või siis huvilistele kirja teel kohale saata.



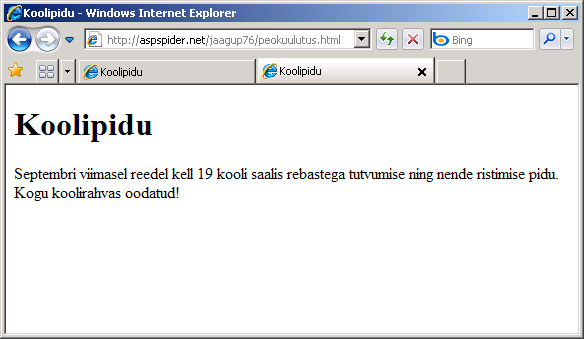
Tahtes hiljem ASP.NETi lisavõimalusi kasutada, võib püüda valida koha, kus vastavat majutusteenust pakutakse. Konspekti kirjutamise ajal jäi ühe mõistliku paigana silma aspspider.com - .NET 4.0 toega veebimajutuskeskkond. Algul tuleb end kasutajaks teha nagu ikka. Muidukatsetajate eemalhoidmiseks tuleb lehestikule koostada mõningane kirjeldus. Samuti tuleb registreerumissoov oma elektronpostiaadressilt kinnitada ning eraldi veel viis minutit oodata näitamaks, et keskkonna kasutamise huvi tõsine on.



Kui need sammud läbitud, siis lubatakse Control Panel-i ning File Manageri juurde, mille abil vajalikke faile serverisse laadida. Veebifailid tuleb panna kausta webroot. Praegusel juhul piisab vaid peokuulutus.html-ist.

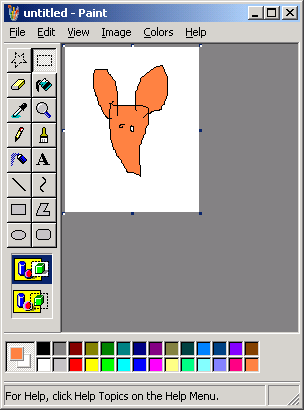
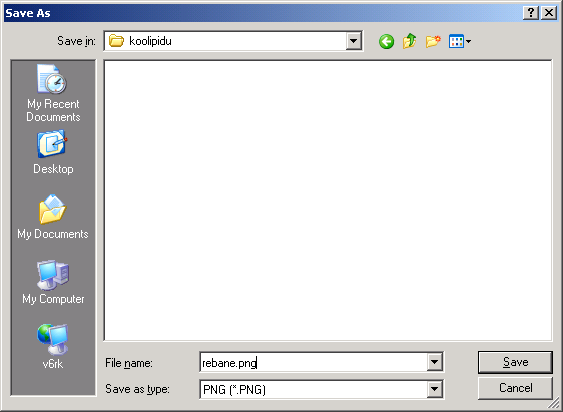


Ning tuttav leht kaugeski serveris üleval.



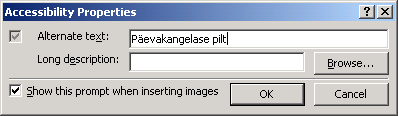
#### Piltide kasutamine veebilehel

Veebileht võib piirduda ainult tekstilise teabega. Tahtes seda aga ilusamaks muuta, on pildid igati omal kohal. Kes autoriõigustega ei karda pahuksisse minna, saab omale veebist igasuguseid vahvaid kujutisi muretseda. Kui aga joonis ise kokku pandud, siis ei saa kellelgi muret olla, et seda üles panna ei tohiks. Siin kuulutuse juures joonistati kokku rebase moodi kujutis. Ning et seda oleks mugav veebilehestiku juures pruukida, siis salvestati olemasoleva HTML-failiga samasse kausta. Veebis enamvähem kindlapeale näidatavad vormingud on gif, jpg ja png. Esimene neist jooniste tarbeks, teine fotodele ning viimatine saab enamvähem kõigega viisakalt hakkama. Igasugu muude vormingute nagu näiteks bmp näitamine sõltub rohkem konkreetse veebilehitseja oskustest ja seadistustest. Siin salvestati siis peokuulutus.html-iga samasse kausta pilt nimega rebane.png.

|  |  |
| --- | --- |
|  | Pärast pildi salvestamist ei ilmu see veel automaatselt failide juures nähtavale. Aga kui Solution Exploreris vajutada refresh-nuppu , siis peaks pildifaili nimi failide loetellu ilmuma. |

Edasi võib pildi lihtsalt lehe teksti sees sobivasse kohta – näiteks lõppu – lohistada. Keskkond võib küsida alternatiivteksti – nende jaoks, kes mingil põhjusel pilte ei saa vaadata. Näiteks, kui tegemist mobiiliekraani kaudu veebi lehitsemisega ning kiiruse ja odavuse huvides on pildid välja lülitatud.



Kui lähtekoodi piiluda, siis on näha, et tekkis uus lõik (element nimega p). Lõigu sees pilt (element nimega img) ning viimasel küljes atribuutidena mõned andmed. Tähtsam neist pildifaili nimi, siin näites rebane.png . Teiseks kohustuslikuks väljaks pildi juures on toosama alternatiivtekst, mis sisestusaknast küsituna vastavasse kohta kirjutati. Kes veebilehte koodina kirjutab, mõistab selle sobivasse alt-kohta ka ise paigutada. Ning võibki pilti vaadata.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Koolipidu</title>

<style type="text/css">

.style1 {

width: 134px;

height: 165px;

}

</style>

</head>

<body>

<h1>

Koolipidu</h1>

<p>

Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende

ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>

<img alt="Päevakangelase pilt" class="style1" src="rebane.png" /></p>

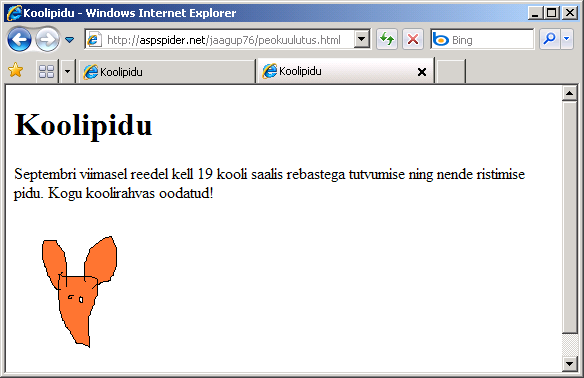
</body>

</html>

Lähtekoodi lähemal piilumisel paistab, et kasutajale on tekitatud klass nimega style1, kus kirjas pildi mõõtmed. Ülal päiseosas stiil defineeritakse ning allpool pildi juures määratakse, et selle pildi klassiks on style1. Iseenesest kannataks selle osa ka koodi seest välja võtta ning lasta pildil olla oma loomulikus suuruses.

Kui nüüd pildiga kuulutus tahta kuhugi veebiserverisse nähtavale kohale üles panna, siis tuleb veebi kopeerida mõlemad failid. Nii pildifail kus rebane peal kui ka muudetud sisuga kuulutusfail, kus öeldud, kus kohal see pilt on vajalik lehel näidata. Enne kopeerimist tasub jälgida, et Web Developeris oleksid failid salvestatud. Pildi näitamine arendusvahendi lehel ei tähenda veel andmete jõudmist kettale.

Kui nüüd lehte uuesti vaadata, võiks pilt näha olla. Mõnel pool hoitakse veel vanu andmeid meeles – seega tuleb uue tulemuse nägemiseks mõnevõrra oodata. Või siis vajutada CTRL+SHIFT+Reload.



#### Seotud lehed

Eelnevast näitest paistis, et paljaks tutvustuseks piisab ühest lehest. Rohkemate andmete põhjal pole aga põhjust kõike korraga näidata – vastavalt vajadusele saab lugeja valida omale parasjagu vajalikke lehti. Enne lehtede omavahelist sidumist tuleb aga need kõigepealt valmis teha – siis on mugavam teada anda, kust kuhu liigelda saab. Siin näites lisame peokuulutuse kõrvale lehe, kus kirjas, mille eest keegi korralduse juures vastutab. Ehk siis taas Solution Exploreris rakenduse kausta juures parem hiireklõps, sealt uus HTML-leht, mil nimeks vastutajad.html. Osa kujundust on mugav disainivaates teha. Et tulemuseks aga HTML-kood, siis ei saa disainivaates teha midagi sellist, mida otse koodi sisse kirjutada ei saa. Vastupidi aga küll. Üheks selliseks mooduseks on definitsiooniloend – sõna ja tema seletus. Ehk siin on sõna ehk definitsiooni rollis klassi number, selgituseks aga nende roll peol. Kogu selle loo ülesmärkimiseks on vaja kolme eri HTML-käsklust. Käsk dl ehk definition list teatab, et nüüd loetelu algab, nüüd lõpeb. Definition term ehk dt märgib seletatava termini, dd ehk definition määrab termini vaste. Ning nende abil saab iga klassi koos oma rolliga kirja panna.

Kood:

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Peo vastutajad</title>

</head>

<body>

<h1>Peo korralduse eest vastutajad</h1>

<dl>

<dt>10. klass</dt>

<dd>Rebased</dd>

<dt>11. klass</dt>

<dd>Võistluste ja katsumuste läbiviijad</dd>

<dt>12. klass</dt>

<dd>Muusika eest hoolitsejad</dd>

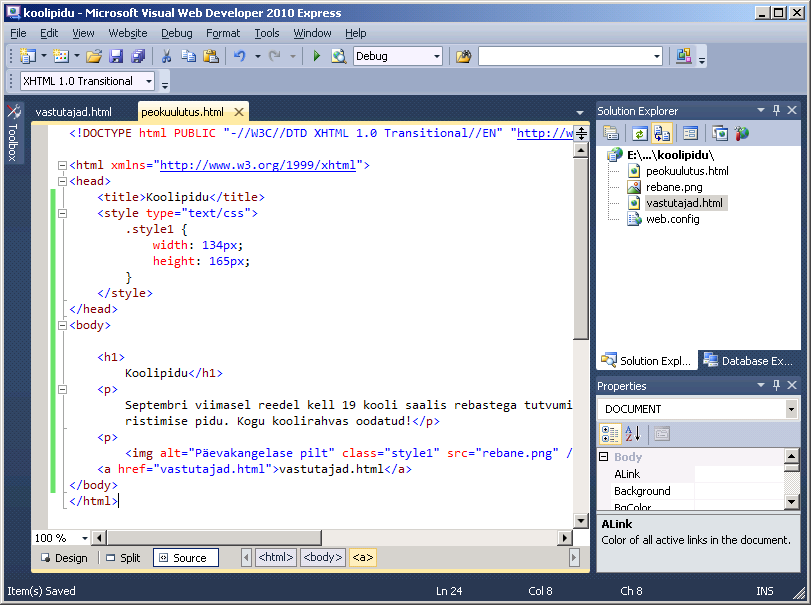
</dl>

</body>

</html>

|  |  |
| --- | --- |
|  | Disainivaates on näha, et klassid on tõmmatud vasakule, nende juurde kuuluvad seletused aga mõningasse taandesse sinna kõrvale. |

Kaks eraldi lehte olemas, nüüd siis tarvis nad omavahel siduda. Ja praegusel juhul soovitavalt mõlemat pidi – nii et avalehelt saaks vastutajate lehele ning viimasest omakorda tagasi avalehele. Viite loomiseks on vajalik teada anda kaks asja: mille peale vajutades viide käivitub ning kuhu vajutades satutakse. Viidetega ümber käimiseks on HTMLi sisse mõeldud element nimega a. Selline lihtne ühetäheline nimi, pärit sõnast anchor (ankur). Kõige lihtsam ühendusmoodus on avada avalehe lähtekood ning sinna sobivasse kohta lihtsalt vedada Solution Exploreri alt vastav fail. Ehk siis haaran hiirega kinni failinimest vastutajad.html ning sikutan ta lehe lähtekoodis kohta, kus võiks viide olla.



Tulemusena tekkis siis rida

<a href="vastutajad.html">vastutajad.html</a>

Atribuudi href (hyperlink reference) juures on kirjas avatava faili nimi. Elemendi a sisu (ehk siis <a href="vastutajad.html"> ja </a> vahelt on lehel nähtav tekst. Praegu seal lihtsalt sama faili nimi, sest lohistamise peale ei mõistnud veebiredaktor sinna paremat selgitust anda. Ise aga saab sinna mõnevõrra pikemalt kirjutada. Näiteks

<a href="vastutajad.html">Peo osaliste tööjaotus</a>

mis siis veebilehel näeb välja ilusasti viitena



Terviklik avalehe kood siis

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Koolipidu</title>

<style type="text/css">

.style1 {

width: 134px;

height: 165px;

}

</style>

</head>

<body>

<h1>

Koolipidu</h1>

<p>

Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende

ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>

<img alt="Päevakangelase pilt" class="style1" src="rebane.png" /></p>

<a href="vastutajad.html">Peo osaliste tööjaotus</a>

</body>

</html>

Sarnane viide ka vastutajate lehe koodi

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Peo vastutajad</title>

</head>

<body>

<h1>Peo korralduse eest vastutajad</h1>

<dl>

<dt>10. klass</dt>

<dd>Rebased</dd>

<dt>11. klass</dt>

<dd>Võistluste ja katsumuste läbiviijad</dd>

<dt>12. klass</dt>

<dd>Muusika eest hoolitsejad</dd>

</dl>

<a href="peokuulutus.html">Avalehele</a>

</body>

</html>

ning juba võibki viiteid pidi kahe lehe vahet liikuma hakata.



Viited aga ei pea sugugi olema vaid paari kohaliku lehe vahel. Kuna veebiaadressi kaudu saab ligi lehtedele üle Interneti, siis piisab sobiva viite lisamisest oma lehele. Lihtsa failinime puhul saab lehitseja aru, et seda otsitakse kohalikust kataloogist. Kui aga aadressil on kirjas ees http:// (või mõni muu protokolli tähis nt. ftp), siis teatakse, et vastavat faili tuleb välisvõrgust otsima hakata. Sellest ka põhjus, miks viidetele tuleb nood http-d ette kirjutada. Muul juhul kipub lehitseja otsima kohalikust kataloogist faili nimega www.ilm.ee (või näiteks faili nimega microsoft.com) ja ei leia kuidagi.

<p>

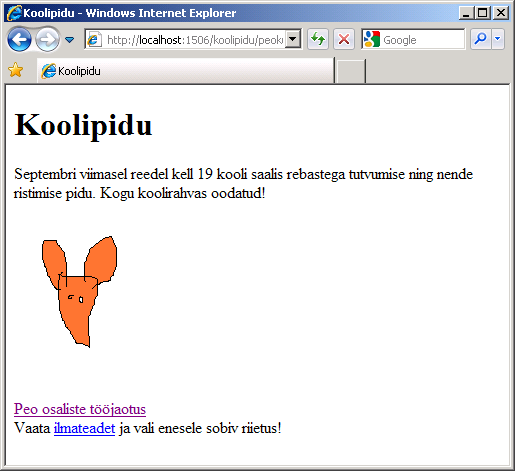
<a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />

Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv riietus!

</p>

Käsklus br tähistab reavahetust (break). Segadust kipub vahel tekitama, miks tol elemendil on kaldkriips lõpus, kui mõneski muus kohas kipub kaldkriips olema enne elemendi nime. Selgituseks, et siin on elemendi algus ja lõpp koos, st. tegemist on käsu <br></br> lühendatud kujuga. XHTML vastab XMLi reeglitele ning sealtkaudu nõutakse iga elemendi puhul algust ja lõppu. Elemendil a on näiteks sinna vahele põhjust atribuudina panna avatav aadress ning tekstilise väärtusena kasutajale nähtav jutt. Käsk br tähistab aga ainult reavahetust ning kui sinna täiendavaid andmeid (nt. rea kõrgust) ei lisata, siis piisabki vaid elemendi nimest. Mille taga kaldkriips annab teada, et sellega on vastava elemendi mõjupiirkond ühtlasi lõppenud.

Lõpetuseks siis lehitsejas töötav viidetega leht.



ja tema lähtekood

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Koolipidu</title>

<style type="text/css">

.style1 {

width: 134px;

height: 165px;

}

</style>

</head>

<body>

<h1>

Koolipidu</h1>

<p>

Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende

ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>

<img alt="Päevakangelase pilt" class="style1" src="rebane.png" /></p>

<p>

<a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />

Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv riietus!

</p>

</body>

</html>

Ülesandeid

* Loe näide mõttega läbi.
* Koosta korvpallivõistluse tarbeks kuulutusleht.
* Vaata lehte veebilehitsejas.
* Otsi või joonista lehele sobiv pilt.
* Koosta eraldi lehekülg, kus kirjas mängivate võistkondade nimed.
* Seo lehed omavahel viidetega.
* Pane avalehele viide korvpallireeglitele. Näiteks <http://www.basket.ee/uploads/docs/korvpalli_reeglid.pdf>
* Otsi korvpalliga seotud pilte. Koonda nad ühele galeriilehele. Lisa viide avalehelt galeriisse ja tagasi
* Koosta iga võistkonna kohta omaette väike tutvustav leht. Avalehelt tee viide lehele, kus asub võistkondade loetelu. Loetelust saab viite abil iga võistkonna juurde. Lisa ka tagasisuunas toimivad viited.
* Võimaluse korral pane loodud lehestik üles avalikult kättesaadavasse veebiserverisse. Veendu, et kõik on ilusti nähtav. Serveri puudumisel püüa lehestik nähtavaks saada sõbra arvutis või oma arvuti teises kataloogis.
* Tee oma lehestikus mõni täiendus (näiteks lisa üks korvpallimeeskond). Hoolitse, et uus versioon oleks nähtav ka ülespanekukohas.

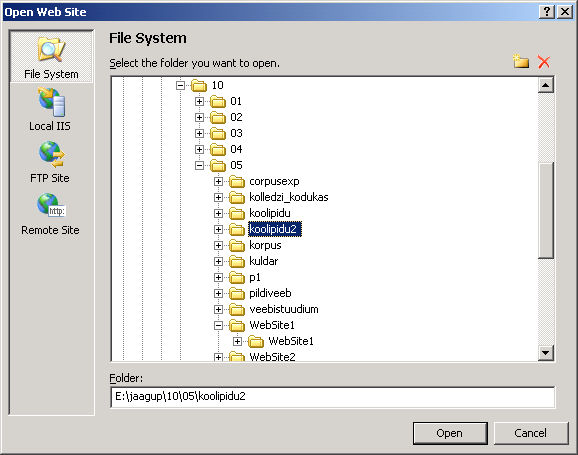
### Astmelised laadilehed (CSS) – cascading style sheets

Lehestiku kujundamisel on mitmesuguseid mooduseid. Üksiku väikese lehe puhul on tõenäoliselt lihtsaim variant graafilise redaktori abil sobivasse kohta suurused ja värvid määrata ning ongi valmis. Kui aga soovitakse, et mitmes kohas oleks sarnane kujundus – näiteks pealkirjad kindla suuruse ja paigutusega, viited üle kogu dokumendi ühtlased, siis pikema dokumendi või lehtede komplekti korral on viisakas vastavat tüüpi andmete kujundus eraldi ühes kohas määrata ning pärast seda määrangut kasutada. Liiatigi on nõnda võimalik kogu kasvõi suurema lehestiku põhikujundust hiljem muuta ilma, et peaks ükshaaval kõiki kohti läbi käima.

HTMLi varasemates versioonides oli hulk käsklusi kujundamise tarbeks. Leiti aga, et keelt tasuks lihtsamaks teha, kuna muidu on tülikas mobiilseadmetele ja muudele väiksema arvutusvõimsusega masinatele HTML-i lugejaid teha. Samas ei tahetud loobuda mitmekülgsetest kujundusvõimalustest. Nõnda koos XHTMLi tulekuga 1999ndal aastal otsustatigi enamik kujunduskäsklusi HTMLi keele seest välja visata või vähemasti ebasoovitavaks muuta ning selle asemel soovitatakse kasutada laadi- ehk stiililehtede võimalusi. Kui seade suudab sealseid kujundussoovitusi arvestada, siis ta arvestab. Kui mitte, siis mitte. Aga vähemasti põhitekst on igale kasutajale nähtav. Samuti on CSSi juures kujunduskäsklusi ja -võimalusi tunduvalt rohkem kui ennist HTMLi küljes.

#### Näite kopeerimine

Õppida ja katsetada aga on kindlam näite peal. Siis saab proovida ja vaadata, mis toimub. Võtame aluseks eelmises peatükis valminud koolipeo kuulutuse näite. Kui tahta, et vana näide tervikuna alles jääb ja uue peal rahus katsetada võib, siis on vanast hea koopia teha. Ning siinse veebirakenduse puhul tasub koopia teha tervest kataloogist. Enne tal nimeks koolipidu, uueks nimeks paneme koolipidu2. Tervikrakenduse avamiseks aitab Web Developeri failimenüü käsklus „Open Web Site“.



#### Pildi suuruse muutmine

Kui rakendust piiluda, siis lehe algusesse ongi keskkond juba ühe stiili genereerinud, paistavad olema märgitud pildi mõõdud.

.style1 {

width: 134px;

height: 165px;

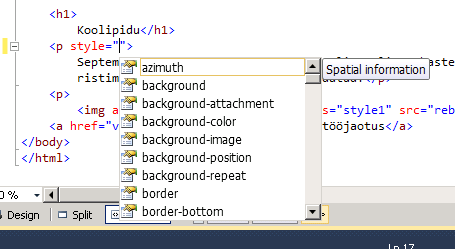
}

Tasub proovida neid muuta ja veenduda, et pilt ka rakenduse juures ekraanil muutub.

|  |  |
| --- | --- |
|  | width: 50px; height: 165px juures näiteks pressiti pilt kokku |
|  | width: 50px; juures aga määratakse laius kindalt paika, kõrgus võetakse proportsioonis vastavalt laiusele. |

#### Käskluste valik

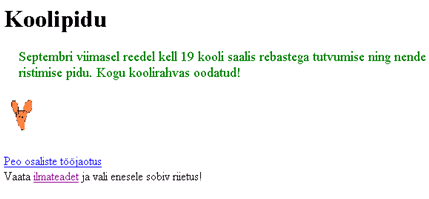
Laadikäskluse võib aga pea iga elemendi külge kirjutada. Ning keskkond on piisavalt tark, et sinna võimalikke käske juurde pakkuma hakata. Ehk siis kui esimese lõigu juurde sai kirjutatud style=““ , siis sinna juurde tekkis valikmenüü päris mitmesuguste käskudega, millest saab hakata omale sobivaid kombineerima. Nagu aimata võib – backgroundiga algavad käsud määravad tausta, borderiga jooned, color seab esiplaani värvi, font ja tekst teksti fondi, suuruse ja joondusega seotu.



Määrasin näiteks kuulutuselõigule omadused, et kaugus vasakust servast on 20 pikslit (ekraanipunkti), tekst on suur (large) ning värv roheline.

<p style="margin-left: 20px; font-size: large; color:Green">Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende ristimise pidu. Kogu koolirahvas oodatud!</p>

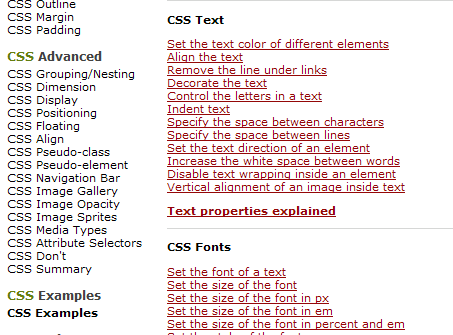
Tulemust näeb pildil:



#### Valmisnäidete kasutamine

Edasi on juba proovimise ja katsetamise asi. Hulga ilusaid näiteid leiab veebiõpetuste lehelt w3schools.com aadressilt <http://www.w3schools.com/css/css_examples.asp>. Tuleb vaid omale sobiv jupp leida ning töötavana oma lehele üle kanda.

Muude hulgas paistab teksti joondamise näide (align the text)



Sealt leiab järgneva näite

|  |  |
| --- | --- |
| <html>  <head>  <style type="text/css">  h1 {text-align: center}  h2 {text-align: left}  h3 {text-align: right}  </style>  </head>  <body>  <h1>This is header 1</h1>  <h2>This is header 2</h2>  <h3>This is header 3</h3>  </body>  </html> |  |

Edasi tuleb nuputada, kuidas näitest omale kasulik rida ära näpata. Siin paistavad laadid olema kirjas lehe päiseosas (head). Kui praegu aga katsetame vaid ühele lõigule omaduste andmist, siis on meil vaja vaid sobiv käsklus õigesse kohta tõsta. Siin on märgitud kolme taseme pealkirjad. Ning ülal on igaühele neist määratud, kus ta paikneb. Teksti paremale joondamise käsuks on siis järelikult text-align:right. Tahtes näiteks tööjaotuse ja ilmateate lõigu paremale joondada, kopeerime vastava käsu sinna lõigu juurde.

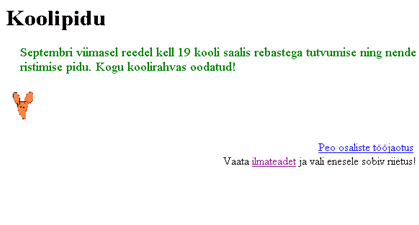
<p style="text-align: right">

<a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />

Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv riietus!

</p>

ning pärast salvestamist võibki alumist teksti nõnda paremale joondatult vaadata.



#### Elementide omaduste määramine päises

Elementide omadused aga saab määrata ka lehe päiseosas – samuti nagu w3schools.com näite juures tehti. Selleks head osa sisse element style, sinna määratava kujundusega HTMLi käskude loetelu ning iga elemendi taha looksulgudes ja semikoolonitega eraldatult need käsud, mis vastava elemendi külge tahetakse panna.

<style type="text/css">

body{background-color: Yellow}

h1{text-align: center; color: Red}

</style>

teatab siis, et lehe sisu (body) on kollase taustaga. Esimese taseme pealkiri (h1) asetseb keskel ning on punast värvi.



Ning selguse huvides ka kogu avalehe kood:

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Koolipidu</title>

<style type="text/css">

.style1 {

width: 134px;

height: 165px;

}

body{background-color: Yellow}

h1{text-align: center; color: Red}

</style>

</head>

<body>

<h1>

Koolipidu</h1>

<p style="margin-left: 20px; font-size: large; color:Green">

Septembri viimasel reedel kell 19 kooli saalis rebastega tutvumise ning nende ristimise pidu. Kogu koolirahvas oodatud!</p>

<p>

<img alt="Päevakangelase pilt" class="style1" src="rebane.png" /></p>

<p style="text-align: right">

<a href="vastutajad.html">Peo osaliste tööjaotus</a> <br />

Vaata <a href="http://www.ilm.ee">ilmateadet</a> ja vali enesele sobiv riietus!

</p>

</body>

</html>

#### Laadileht eraldi failis

Lihtsa teksti puhul piisab kujunduse määramisest lõigu või sõna juures. Pikemal lehel tasub vähemasti korduvate elementide sõnad panna päisesse. Kui aga tahta sarnast kujundust kasutada mitme lehe juures, siis on viisakas kujunduskäsud tõsta eraldi faili ning need siis igal lehel sisse lugeda. Nii mõjuvad kujundusfaili käsud kogu lehestikule ning näiteks kõigi lehtede taust on kerge asendada sobivaga. Viimatises näites on määratud lehe taust ning esimese taseme pealkirja andmed.

body{background-color: Yellow}

h1{text-align: center; color: Red}

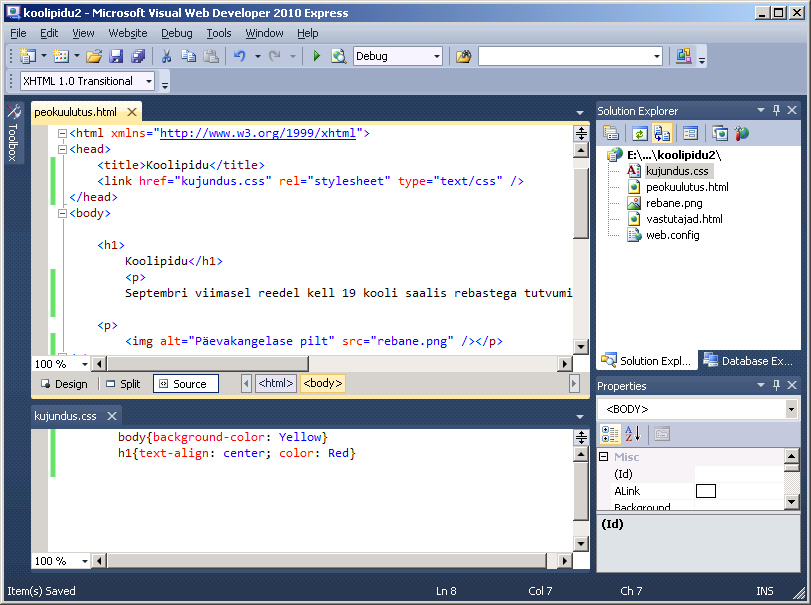
Nende jaoks tekitamegi eraldi faili. Solution Explorer -> paremklõps rakenduse nimel -> Add New Item -> Style Sheet ning nimeks näiteks kujundus.css. Viimasesse on juba „seemneks“ pandud body-elemendi nimi. Kopeerin aga julgesti kogu olemasolevad kaks kujundusrida sinna faili ning algsest kustutan ära.

|  |  |
| --- | --- |
|  | Nõnda läheb ülal määratud kujundus esimese hooga veebilehelt kaduma. Nii et veebilehitsejas paistab endine valge taustaga kuulutus, kus pealkiri on vasakul. |

Edasi aga aitab käsklus

<link href="kujundus.css" rel="stylesheet" type="text/css" />

kujunduse eraldi failist sisse lugeda.



Tulemuseks siis taas kujundatud leht.



Kui nüüd soovida sama kollase tausta ja punase pealkirjaga kujundus panna ka tööjaotuse lehele, pole muud, kui sama link-käsklus ka sinna lisada.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Peo vastutajad</title>

<link href="kujundus.css" rel="stylesheet" type="text/css" />

</head>

<body>

<h1>Peo korralduse eest vastutajad</h1>

<dl>

<dt>10. klass</dt>

<dd>Rebased</dd>

<dt>11. klass</dt>

<dd>Võistluste ja katsumuste läbiviijad</dd>

<dt>12. klass</dt>

<dd>Muusika eest hoolitsejad</dd>

</dl>

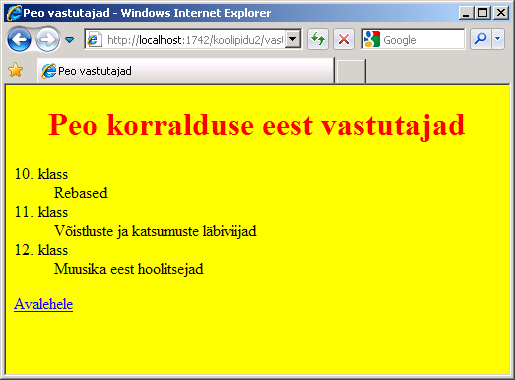
<p>

<a href="peokuulutus.html">Avalehele</a>

</p>

</body>

</html>



#### Kujundusklass

Kujundusomadusi saab määrata elemenditüüpide kaupa nagu eelnenud näitest näha: kõik esimese taseme pealkirjad paigutati keskele ja värviti punaseks. Mõningaid omadusi aga soovitakse panna ainult mõnede vastavate elementide juurde või siis sootuks erisuguste elementide külge. Näiteks soovitakse kõik uued teated lehel eraldi ära märkida sõltumata sellest, kas teated on tabeli, loetelu, lõigu või mõne muu tehnilise vahendi abil lehele kuvatud. Siinses väljamõeldud näites sobib eraldi rõhutada vastavast klassist osalejate roll. 10. klassi rahvas on rebased, 11. klassi omad peo võistluste läbiviijad. Kujundusfailis võib defineerida rolli klassi, edasi juba määrata, milline sõna või lõik selle klassi järgi kujundatakse. Ning kui juhtub, et kujundusklass tahetakse kinnitada osa külge, mis pole eraldi elemendiga piiritletud, siis piiritlemiseks sobib element nimega span. Element ise ei tee midagi, küll aga saab sinna külge kujunduskäsklusi ja –klasse kinnitada. Lisan kujundusfaili rea

.roll{text-decoration: underline;}

ning vastutajate loetelu juures märgin ära, millised kohad tähistavad rolli.

<dl>

<dt>10. klass</dt>

<dd class="roll">Rebased</dd>

<dt>11. klass</dt>

<dd>Võistluste ja katsumuste

<span class="roll">läbiviijad</span></dd>

<dt>12. klass</dt>

<dd>Muusika eest hoolitsejad</dd>

</dl>

Nagu näha, mõistab arenduskeskkond olemasoleva(d) klassi(d) ise välja pakkuda.



Ning ongi rolli tähistavad kohad alla joonitud.



Ülesandeid

* Loo oma korvpallimeeskonna lehestikust koopia või koosta ise väike lihtne lehestik.
* Värvi mõni lõik ja mõne lõigu taust, katseta teksti suuruse ja paigutusega.
* Tutvu [www.w3schools.com](http://www.w3schools.com) css-i näidetega. Katsu sealt vähemasti kümmekond enesele üle võtta (nt borderi abil jooned koos värvi ja laiusega, joondus, rasvasus (font-weight)).
* Määra lehe taust ja vaba serva laius (margin-left) päises. Katseta.
* Tõsta need määrangud eraldi laadifaili. Veendu, et algsest lehest on kujundus kadunud.
* Ühenda veebileht ja laadifaili @import käsu abil
* Ühenda sama laadifail ka teise veebilehega ning veendu, et kujundus tuli mõlemasse üle.
* Muuda laadifailis lehe tausta ja veendu, et see mõjub mõlemale lehele.
* Loo klass nt. noortemeeskondade tähistamiseks. Sealne tekst rohelise värviga. Tähista noortemeeskondade nimed vastava klassiga. Veendu, et kõikidel lehtedel on nende nimed rohelised. Muuda noortemeeskondade klassi nõnda, et roheline pole mitte tekstivärv vaid taustavärv. Veendu muutuse toimumises.
* Koosta täiesti uus lehestik näiteks kavandatava matka kohta. Märgi paberile üles teemad, mis võiksid lehestikus olla. Koosta paberile struktuur, kust lehelt kuhu võiks liikuda, mõtle lehtedele sobivad pealkirjad ning failinimed. Joonista valmis eskiisid, millised võiksid lehed välja näha – st. kas kõik lehed ühesuguse plaaniga või nt. avaleht ja menüülehed teistest erinevad. Koosta iga lehetüübi jaoks blankett. Stiilikäsklused koonda eraldi kujundusfaili, mis igasse blanketti imporditakse. Tee blankettidest iga lehe jaoks koopia, kuhu ainult märgitakse teema, näiteks „siia tuleb matka teise päeva söögikoha kirjeldus“. Seo lehed omavahel viidetega. Joonista matka marsruut kaardile ning pane ka see leht lehestikku. Palu naabril otsida üles soovitud leht ning vaata, kui kergesti ta selle menüüde kaudu leidis. Vajadusel lisa selgitusi või keerukamal juhul muuda oma lehestiku struktuuri. Mängi kujundusfailiga, et lehestik võimalikult meeldiv välja näeks. Täida lehed võimalikult meeldivalt ja usutavalt. Ning lõpetuseks – kui lehestik võimalikult hea ja nii omale kui ka sõpradele huvitav, siis tasub see matk võimaluse korral ka teoks teha :-)

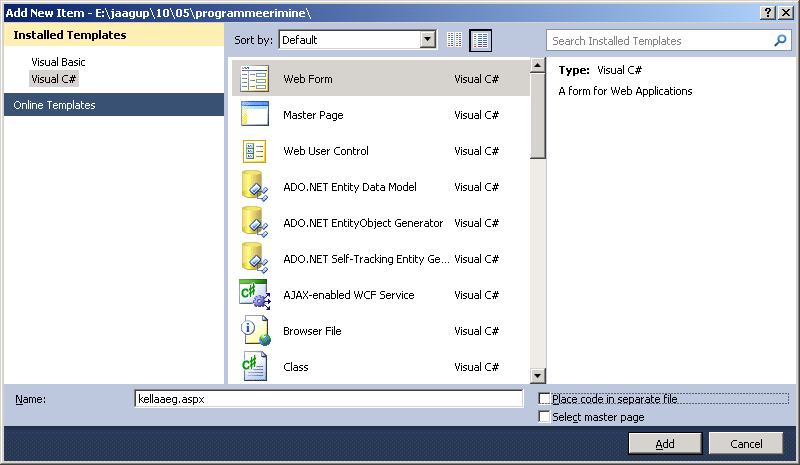
### Programmeeritavad veebilehed

Eelnevas peatükis kokku pandud veebilehti nimetatakse staatilisteks. See tähendab, et ükskõik kes või millal seda lehte vaatab – leht on ikka sama sisuga. Ning alles siis, kui autor midagi muudab võib lehe peal uut sisu nägema hakata. Nagu näha, nii peokuulutuse kui matkalehe kannatab sellisena täiesti kokku panna ning sageli polegi teabe edastamiseks rohkemat vaja. Selline veeb oli aga küllalt viisakal tasemel juba 1997ndaks aastaks olemas, edasi on jõudsalt mitmesuguseid lisavõimalusi arendatud, et just konkreetsele kasutajale meelepärane olla, või siis kasutajate sisestatud andmeid töödelda nõnda, et sellest midagi suurt ja ilusat sünniks. Kogu ASP.NET on taoliste võimaluste tarbeks loodud. Staatilisi lehti võiks südamerahus kirjutada ükskõik millega ja panna üles ükskõik kuhu. Visual Web Developeri juures on „hariliku“ lehe kujundusvahendid lihtsalt üks killuke paljude programmeerimisvahendite hulgas.

Programmeeritaval veebilehel e. aspx lehel on samuti võimalik kasutada staatilisi HTML elemente, kuid lisaks neile avaneb võimalus kasutada programmelt kontrollitavaid ja hallatavaid ASP.NET elemente e. ServerControl’e

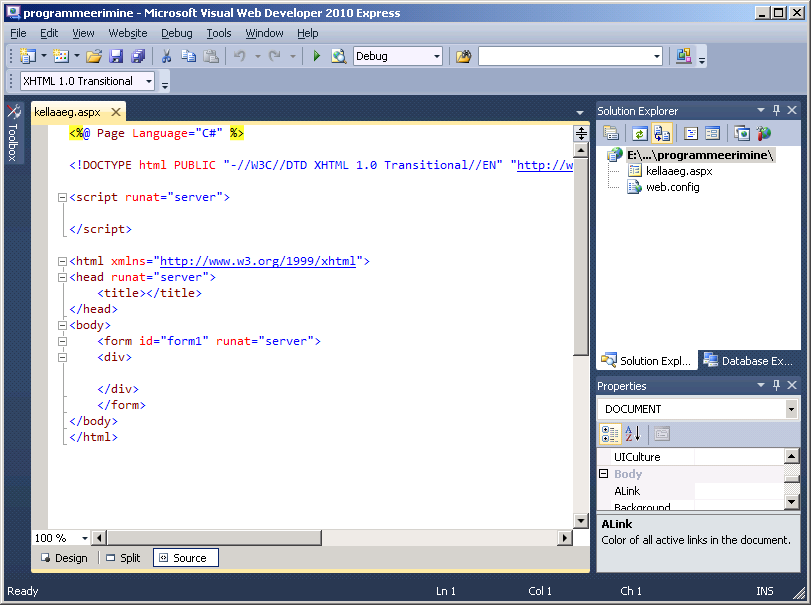
#### Kellaaeg

Programmeerimisvahendite katsetamiseks loon uue tühja veebiprojekti (failimenüü 🡪 New Web Site 🡪 Empty Web Site, nimeks programmeerimine). Solution Explorerist taas New Item, aga sedakorda Web Form. Nimeks kellaaeg.aspx. Keeleks siinse konspekti edasise järgimise huvides Visual C# ning hoolitsen, et „Place code in separate file“ juures ei oleks linnukest – sedaviisi ei teki esialgu tarbetut lisafaili.



Keeleks C#!

Tekkis ilus lehe blankett, kuhu saab tasapisi omi andmeid kirjutama hakata.



Lähemal uurimisel paistab, et sealne kood nagu oleks HTML ja nagu ei oleks ka. !DOCTYPE on täiesti olemas. Samuti <html> oma alguse ja lõpuga </html>. Ja <head> ja <body>. Aga üles on ilmunud salapärane rida

<%@ Page Language="C#" %>

Ning päris mitmel pool on juures runat=“server“. Nende järgi võib aimata, et tõenäoliselt on tegemist ASP.NETi lehega. Ehk pea kõik, mis on HTMLina, saadetakse otse veebilehitsejasse. Kus aga juures kirjas runat=“server“, selle osaga võib lehe serveerimise käigus midagi juhtuda – sinna võidakse miskit programmikoodi abil juurde kirjutada või seal midagi muuta. Samuti on võimalik otse HTMLi koodi programmi kaudu midagi juurde kirjutada. Viimast võimalust ei peeta küll kuigi viisakaks. Aga kuna vanem ASP just nõndaviisi käis ning nii on lihtsaid asju kerge näidata, siis vastav alustusnäide ka siia. DateTime.Now annab praeguse kellaaja, Kell on <%=DateTime.Now %> abil trükitakse see just sinna kohta lehele. Nii et siinne lihtne lehe kood

<%@ Page Language="C#" %>

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

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Kellaleht</title>

</head>

<body>

<form id="form1" runat="server">

<div>

Kell on <%=DateTime.Now %>

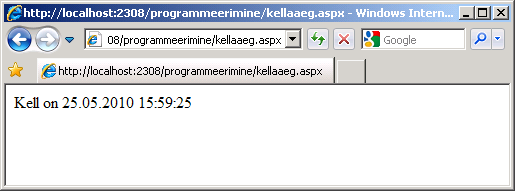
</div>

</form>

</body>

</html>

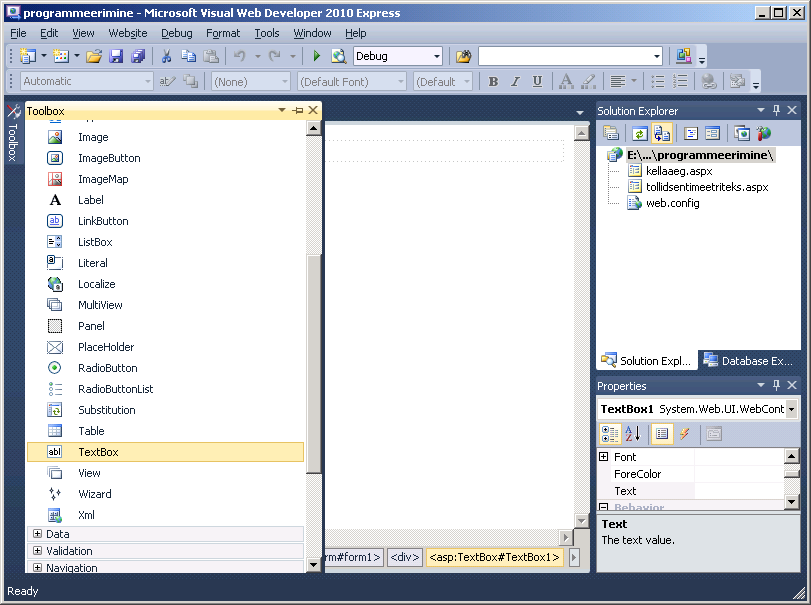
annab kokku lehe, kus nähtaval praegune kellaaeg.



Tegemist siis kõige ehtsama dünaamilise veebilehega – iga kord kui avad, on sisu uus.

#### Arvutav veebileht

Inimeselt andmete kättesaamiseks tuleb luua sisestuskoht. Üsna lihtne vahend selleks on tekstiväli. Sisestus- ja muid elemente saab valida tööriistakastist (Toolbox). Kui seda pole parajasti näha, siis View-menüüst saab selle esile kutsuda. Lohistades tekstikasti lehel sobivasse kohta, ilmub sinna tekst, mille abil ASP.NET teab veebilehe avamisel tekstikasti HTML-käskluse kuvada.



Automaatselt pannakse elementidele ka id-d. Nende järgi on hiljem programmikoodis võimalik nende poole pöörduda. Ning runat=“server“ jällegi iga elemendi külge, et server sellega midagi peale hakata tohiks. Kui unikaalne id ei taha tekkida, siis tasub tekstikast lehele lohistada disainivaates.

<form id="form1" runat="server">

<div>

Tollide arv:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

</div>

</form>

Sentimeetrites oleva tulemuse vaatamiseks piisab kohast, kuhu programmiga pääseb kirjutama. Tekstiväli lubab üldjuhul ka inimesel andmeid sisestada. Paljalt andmete näitamiseks on aga loodud element nimega Label. Selle siis lehele lohistamegi ning tulemuseks on järgnev koodilõik:

<form id="form1" runat="server">

<div>

Tollide arv:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

Sentimeetrite arv:

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

</div>

</form>

Et millegi peale oleks võimalik arvutama hakata, selleks tasub lehele lisada nupp – element tüübist Button. Ning et vastusesildile ei jääks sõna „Label“ vaid midagi selgitavamat, siis määrasin sinna teksti „Teadmata“, samuti nupule teksti „Arvuta“.

<form id="form1" runat="server">

<div>

Tollide arv:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

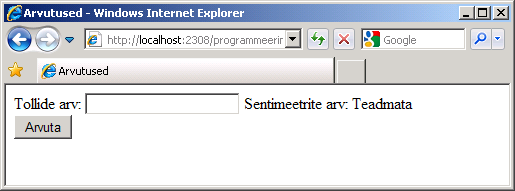
Sentimeetrite arv:

<asp:Label ID="Label1" runat="server" Text="Teadmata"></asp:Label> <br />

</div>

<asp:Button ID="Button1" runat="server" Text="Arvuta" />

</form>



Nagu aimata, ei juhtu nupuvajutuse peale veel midagi – arvutile tuleb enne kõik asjad ette öelda. Saab küll tekstivälja kirjutada ning nupule vajutada, aga vastust ekraanile ei ilmu.

Vastuse leidmiseks tuleb nupu külge programmikoodi lõik siduda. Lihtsaimaks mooduseks on minna disainivaatesse ning vajutada nupule topeltklõps.

Tulemuse peale tehti koodis mõningad automaatsed täiendused.

<%@ Page Language="C#" %>

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

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Arvutused</title>

</head>

<body>

<form id="form1" runat="server">

<div>

Tollide arv:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

Sentimeetrite arv:

<asp:Label ID="Label1" runat="server" Text="Teadmata"></asp:Label> <br />

</div>

<asp:Button ID="Button1" runat="server" Text="Arvuta" onclick="Button1\_Click" />

</form>

</body>

</html>

Kui teraselt algse versiooniga erisusi piiluda, siis paistab, et ülesse script-ossa on tekkinud alamprogramm nimega Button1\_Click. Ning veebilehe sisuosas on nupu külge tekkinud juurde sündmuse kirjeldus ="Button1\_Click" . Viimane tähendab, et kui nupule hiirega vajutatakse, siis käivitub märgitud nimega käsklus. Erinevalt mõnest muust programmeerimiskeelest ei ole siin kohustuslikku seost käskluse (funktsiooni) nime ning vajutatava nupu (või muu elemendi) nime vahel. Mõnikord võib panna ka mitmed elemendid sama käsklust välja kutsuma ning funktsiooni parameetrite (nime järel sulgudes olevate andmete) kaudu saab teada, millisele nupule tegelikult vajutati. Aga kuna praegu meil ainult üks nupp, siis pole mõtet lugu segaseks ajada.

Nupuvajutuse juurde siis programmikood, mis sammu kaupa teeb kõik vajalikud toimingud, et tähtedena (tekstina, stringina) sisestatud tollide arvust saaks vastusesse ka tähtedena sentimeetrite arv. Aga vahepealsete arvutuste juures on vaja andmetüüpi muuta, sest tekstidega ei saa C# keeles matemaatilisi tehteid ette võtta. Küll aga on arvude jaoks loodud omaette andmetüübid. Komadega hakkama saavaks arvutüübiks on double, ainult täisarve tunnistavaks tüübiks int. Levinumate andmetüüpide (nagu näiteks string ja int ja double) omavaheliseks tüübiteisenduseks sobivad klassi Convert käsklused. Rida

double TollidArvuna = Convert.ToDouble(TollidTekstina);

võtab tekstina olevad tollid, teisendab nad arvutimälus ümber reaalarvuna kirjutatavateks andmeteks ning jätab tulemuse meelde muutuja (märksõna) TollidArvuna alla. Viimase saab juba koefitsiendiga läbi korrutada, tulemuse tekstiks muundada ning vastusesildi peal näidata.

protected void Button1\_Click(object sender, EventArgs e)

{

string TollidTekstina = TextBox1.Text;

double TollidArvuna = Convert.ToDouble(TollidTekstina);

double koefitsient = 2.54;

double SentimeetridArvuna = TollidArvuna \* koefitsient;

string SentimeetridTekstina = Convert.ToString(SentimeetridArvuna);

Label1.Text = SentimeetridTekstina;

}



Kes Excelis või mujal valemeid kirjutanud, teab, et sarnased arvutused saab mõnikord ka lühemalt kirja panna. Siinsel puhul siis näiteks annaks sama tulemuse rida

Label1.Text=Convert.ToString(Convert.ToDouble(TextBox1.Text)\*2.54);

Mõnikord on ühte rida isegi kergem lugeda kui pikemat programmilõiku. Kui aga kood nagunii nõnda pikk, et ühe pilguga ei haara, siis pärastise kontrolli ja arusaamise huvides on vahel kasulik toimetused sammude kaupa välja kirjutada.

Mõningast tähelepanu võivad vajada komad ja punktid. Nagu näha, programmikoodi sees on tollide ja sentimeetrite vaheline kordaja murdosa eraldaja kirjutatud punktina – 2.54. Veebilehitseja akna ekraanil aga vastus 25,4 kirjas komaga. C# keeles tuleb koodi sees eraldaja kirjutada alati punktiga – nii nagu keele sünnimaal tavaks. Kui aga arvutil seadeteks valitud Eestimaa, siis mõistetakse andmed sisse lugeda ja väljastada kasutades murdosa eraldajaks koma.

#### Tehtevalikuga kalkulaator

Eelnev kalkulaator suudab vaid tolle sentimeetriteks teha. Kui on tegemist ühe teisendusega, siis piisab taolisest alusest täiesti. Võib kindla kursi alusel arvutada kroone eurodeks või untse grammideks. Samuti kannatab mõningase mõtlemise peale eelneva näite põhjalt ehitada võimaluse, kus kahe või enama sisendandme põhjal vastus arvutatakse. Olgu näiteks sisendiks kauba kilode hind ja kilode arv ning väljundiks makstav summa. Lihtsalt tekstikaste tuleb rohkem ning igal neist peab olema oma, teistest erinev ID, mille kaudu vastavat väärtust küsida. Arvutamiseks tuleb andmetüüp doubleks muuta ning pärast näitamise tarvis jälle tagasi.

Kui aga ei soovita, et kõik tuleks kasutajal ise sisse kirjutada, vaid osa andmeid saaks ka mugavamalt ja veakindlamalt valida – siis aitab rippmenüü, DropDownList. Üksikud valikud selle sees saab ListItemite abil ära märkida. Ning programmikoodis saab rippmenüüst valitud väärtust kasutada sarnaselt kui igast muust sisestuselemendist tulnut.

Järgnev kood siis kõigepealt lehe kujunduseks, kus andmete sisestamiseks kaks tekstivälja, nende vahele rippmenüü. Ning teise tekstivälja järele nupp arvutuse alustamiseks ja nupu taha silt tulemuse näitamiseks.

<%@ Page Language="C#" %>

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

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

}

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Tehtevalikuga kalkulaator</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:DropDownList ID="DropDownList1" runat="server">

<asp:ListItem>+</asp:ListItem>

<asp:ListItem>-</asp:ListItem>

<asp:ListItem>x</asp:ListItem>

<asp:ListItem>/</asp:ListItem>

</asp:DropDownList>

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="=" onclick="Button1\_Click" />

<asp:Label ID="Label1" runat="server" text=""></asp:Label>

</div>

</form>

</body>

</html>

Nupu peal tekstiks võrdusmärk, sildi peal esiotsa tühjus. Et arendusvahendi disainivaates sildi olemasolu arusaadav oleks, selleks näidatakse seal kantsulgude vahel selle sildi nime. Veebilehel aga on vastaval kohal tühjus. Ning nupu külge siis taas seotud disainivaate topeltklõpsu kaudu onclick-sündmus, mille abil saab enesele vajaliku funktsiooni käima panna.



Funktsioonis nagu näha võetakse kõigepealt tekstikastidest välja väärtused ning teisendatakse need arvutamiseks sobivasse tüüpi. Millise tehte on inimene valinud, selle kindlakstegemiseks on üheks võimaluseks rippmenüü omandus SelectedIndex, mis annab valitud rea järjekorranumbri alates nullist. Iseenesest oleks võimalik ka rea väärtus SelectedValue abil kätte saada, aga järjekorranumber on kindlam. Näiteks rakenduse tõlkimise puhul võib nähtav jutt keelest sõltuda, kui aga järjekord samaks jätta, sama valiku peale saab ikka sama toimetuse teha.

Vastuse leidmiseks tuleb Vastuse muutuja kõigepealt deklareerida. Ning funktsiooni sees on kohustus sellele muutujale ka miski algväärtus anda. Ehkki me praegu teame, et tegemist on ühega neljast kindlaksmääratud valikust, tahetakse programmeerimiskeeles end kaitsta ka sellise võimaluse vastu, kus mõni valik on jäetud märkimata ning näitamiseks pole Vastusele kusagilt väärtust võtta. if-valikulausete abil tehakse sobiv tehe ning lõpuks näidatakse Vastus tekstiks (stringiks) muundatuna sildile Label1.

protected void Button1\_Click(object sender, EventArgs e)

{

double Arv1 = Convert.ToDouble(TextBox1.Text);

double Arv2 = Convert.ToDouble(TextBox2.Text);

int Tehtenr = DropDownList1.SelectedIndex;

double Vastus = -1;

if (Tehtenr == 0) { Vastus = Arv1 + Arv2; }

if (Tehtenr == 1) { Vastus = Arv1 - Arv2; }

if (Tehtenr == 2) { Vastus = Arv1 \* Arv2; }

if (Tehtenr == 3) { Vastus = Arv1 / Arv2; }

Label1.Text = Convert.ToString(Vastus);

}

Edasi võibki juba imetleda, kuidas me armas väike kalkulaator töötab.

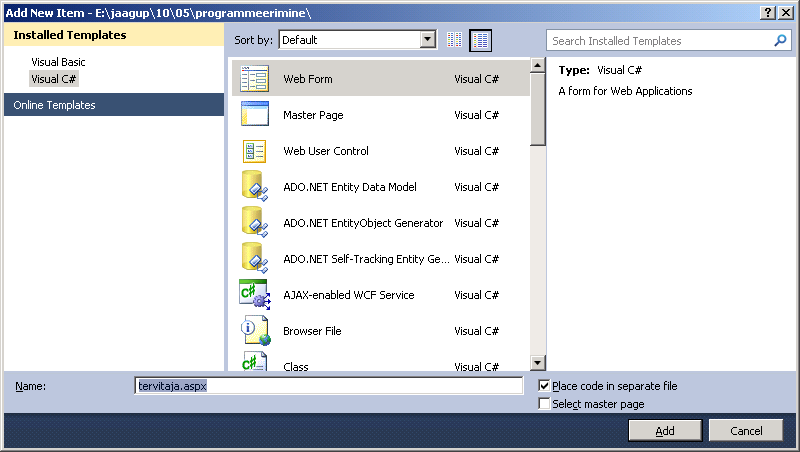


Ülesandeid

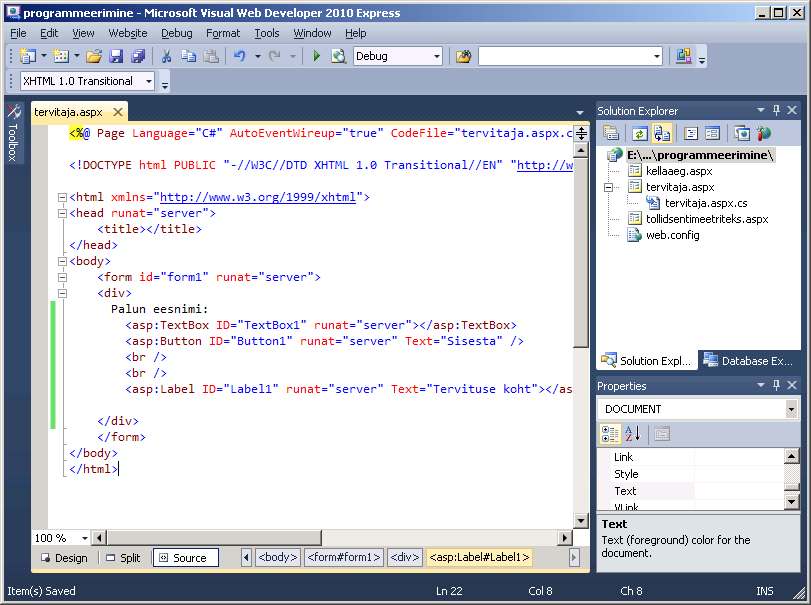
* Võta aluseks tolle sentimeetriteks teisendav näide ning koosta selle põhjal leht, kus arvutatakse tunde ümber minutiteks.
* Paiguta lehele kaks nuppu: ühe abil saab arvutada tunde minutiteks, teise kaudu minuteid tundideks.
* Vali rippmenüüst kauba nimetus, kõrvale tekstivälja kirjuta kogus. Tulemusena väljasta makstav summa.
* Selliseid rippmenüü ja koguse komplekte on lehel mitu. Rippmenüüs on esimeseks valikuks „kaup valimata“ väärtusega null krooni. Leia kokku vajalik makstav summa.
* Tekstiväljadesse kirjutatakse auto kütusekulu liitrites saja kilomeetri kohta ning läbitavate kilomeetrite arv. Rippmenüüst valitakse bensiini mark millele vastavalt on programmiskoodis kirjas selle bensiini liitrihind. Arvutuse tulemusena väljastatakse sõidu maksumus.

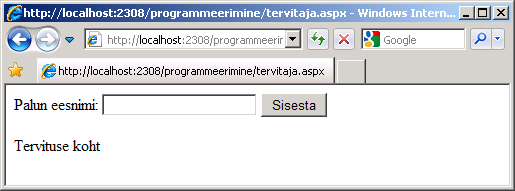
#### Kood kahes failis

Lühikeste näidete puhul on mugav kood samasse faili kirjutada – kõik korraga silma all ja ka veebi on faili mugav kopeerida. Kui aga leht läheb veidigi suuremaks, siis on hea kasutada ASP.NETi pakutud võimalust, kus programmikood paigutatakse eraldi samanimelisse cs-laiendiga faili ning aspx-leht jääb elementide paigutamise ja sündmuste välja kutsumise tarbeks. Selleks tuleb veebivormi loomisel jälgida, et oleks sees „linnuke“, mille taga kirjas „Place code in separate file“.

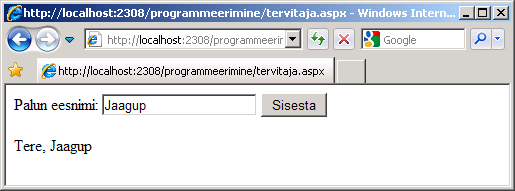


Nõnda tekkis meile fail tervitaja.aspx ning tema kõrvale abifailiks tervitaja.aspx.cs. Alustuseks lisan kujunduselehele tekstivälja, nupu ja sildi.





Kui kujundusvaates vajutada nupule topeltklõps, siis luuakse – nagu ennegi – automaatselt alamprogramm nupuvajutuse alla käiva töö kirjeldamiseks. Ning nupule lisatakse onclick-sündmus selle alamprogrammi käivitamiseks. Lihtsa tervituse jaoks võtame tekstiväljast inimese nime ning kirjutame ta koos tervitusega sildi peale.



Järgnevalt lähtekoodid.

tervitaja.aspx

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

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

Palun eesnimi:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="Sisesta"

onclick="Button1\_Click" />

<br />

<br />

<asp:Label ID="Label1" runat="server" Text="Tervituse koht"></asp:Label>

</div>i

</form>

</body>

</html>

Ning tervitaja.aspx.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

public partial class tervitaja : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void Button1\_Click(object sender, EventArgs e)

{

Label1.Text = "Tere, " + TextBox1.Text;

}

}

#### Ülesandeid

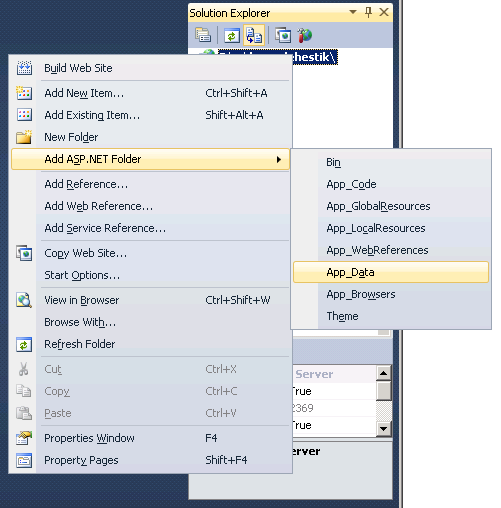
* Pane eraldi kujundus- ja koodilehe abil tööle kahe arvu korrutamine.
* Koosta valuutakalkulaator – Määra algsumma ja sihtvaluuta ning arvuta nende põhjal lõppsumma.
* Võrreldes eelmisega saab valida ka algvaluutat.

### Andmebaasipõhise veebirakenduse loomine

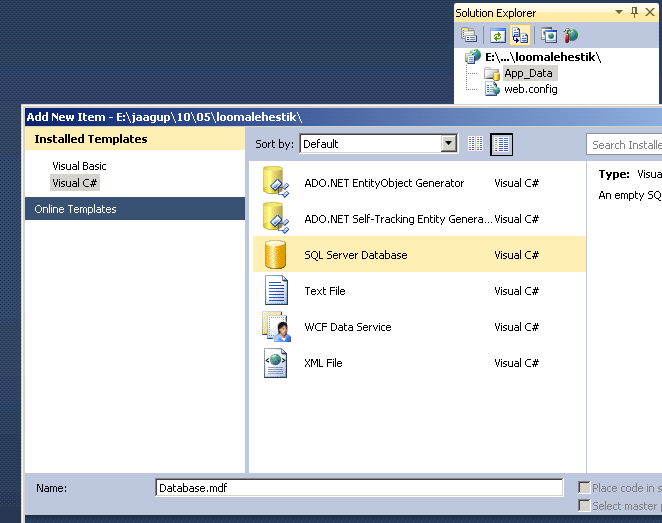
Eelnevalt näidatud kalkulaatoreid saab luua ka ilma veebipoolse programmeerimiseta – olgu siis Javaskripti, Flashi, Silverlighti või mõne muu kliendipoolse tehnoloogia abil. Kui aga soov otsida suurest andmehulgast serveris või sinna omapoolseid lisandusi teha – sellisel juhul on veebiserveripoolne programmeerimine vajalikum. Ning ASP.NETi juures on päris mitmekülgsed ja mugavad vahendid andmetabelitega ümberkäimiseks loodud.

#### Andmetabeli loomine

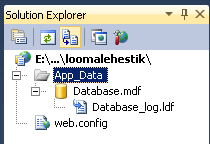
Alustame lihtsamast, ehk siis serveris tabelis olevate andmete vaatamisest Interneti kaudu. Nagu ikka, tuleb algul veebirakendus luua. Sedakorda saab talle nimeks loomalehestik. ASP.NETi rakendustele on välja mõeldud standardne struktuur. Kui andmebaasifail käib rakenduse juurde, siis ta paigutatakse kausta nimega App\_Data. Tühjal rakendusel see puudub – järelikult tuleb luua. Rakenduse nime juurest valik Add ASP.NET folder pakub sobiva nimega kausta välja.



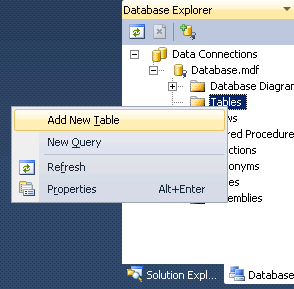
Edasi siis tasub loodud kausta sisse andmebaasifail istutada. SQL Serveri andmebaasile võib jätta vaikimisi nime – Database.mdf.



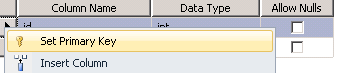
Loodud andmebaasifail paistab ilusti rakenduse puus:

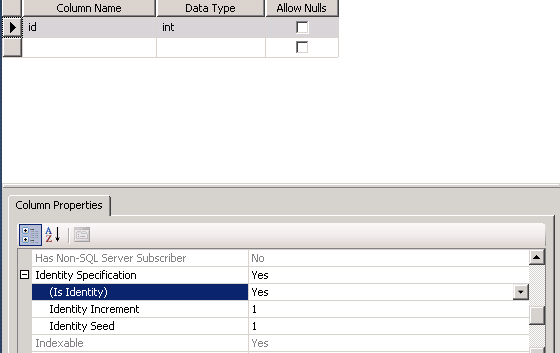


Andmebaasiga lähemaks suhtlemiseks aitab Database Exploreri nimeline abivahend. Sealtkaudu saab öelda, et soovin luua oma andmebaasi uue tabeli andmete hoidmiseks. Relatsiooniline andmebaas nagu SQL Server hoiabki kõiki andmeid tabelites.

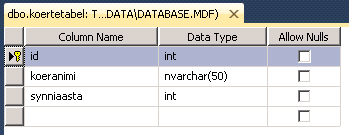


Esimeses ja lihtsamas lähenduses on andmebaas andmetabelite komplekt, igas tabelis olemas read ja veerud. Kõigepealt luuakse veerud, märkides neile ära nime ja tüübi. Siinses võimalikult lihtsas näites püüame veebilehel näidata koeri, igaühel koeranimi ja sünniaasta. Lisaks on andmebaasitabelite juures poolkohustuslikuks tulbaks identifikaatortulp, näiteks nimega id. Miks see vajalik? Kes näiteks Exceli tabeleid loob, sel tulevad reanumbrid automaatselt. Saab alati selgelt öelda, et vastav tekst on millises reas ja millises veerus. Andmebaasitabelites aga sellisel kujul reanumbreid ei hoita. Isiku identifitseerijaks võib olla näiteks isikukood. Need aga ei lähe sugugi mitte järjest. Küll aga on andmetabelite juures igati kasulik, kui on võimalik mingi kindla tunnuse järgi rida eristada. Muidu võib kergesti juhtuda, et näiteks sisestamisel tekib kaks korduvat rida ning tavavahenditega ei saagi naljalt öelda, et kustuta üks rida ära ja jäta teine alles. Siin siis on eristajaks tulp nimega „id“, tüübist int ehk täisarv. Nime ette tasub hiire parema klahviga valida primaarvõtmemärk.



Et andmebaasiprogramm taipaks automaatselt id-numbreid välja mõelda, selleks tasub alt veeru omaduste seast valida, et tulba omadus „identity“ on sisse lülitatud. Ning siis asutakse automaatselt alates ühest ühe kaupa reanumbreid panema. 

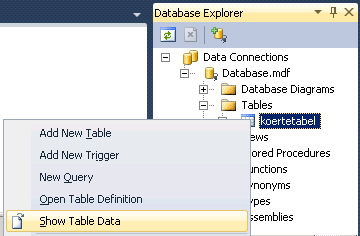
Andmetüüp int tähendab täisarve, sobib nii id kui sünniaasta puhul. Koera nimeks sai nvarchar(50) ehk siis national (lubab täpitähti) varchar ehk muutuva pikkusega, kuni 50 sümboli pikkune tekst. Omadus allow Nulls kipub vahel esmakordsel tutvusel segadust tekitama. Kõige lihtsam on sealne kast võimalusel märkimata jätta. Sel juhul tühiväärtused, ehk nullid (sõnaga) ei ole lubatud. Kui sisestame koera, peame sisestama ka tema sünniaasta. Kui aastat ei tea, saame aasta väärtuseks panna ka näiteks 0 või -1. Aga ei saa aastat panemata jätta.

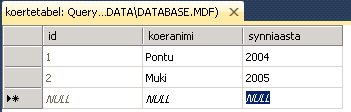


Salvestusnupule või Ctrl+S vajutades küsitakse loodavale tabelile nimi – eks see tule siis anda.



Edasi juba koerte eneste sisestamine. Selleks tuleb avada Database Explorer. Sealt valida tabelite alt koertetabel ning selle pealt parema hiireklahviga Show Table Data. Nimi ja sünniaasta tuleb kirjutada, id-number tuleb automaatselt juhul, kui Identity Specificationi juurde on ennist yes kirjutatud.



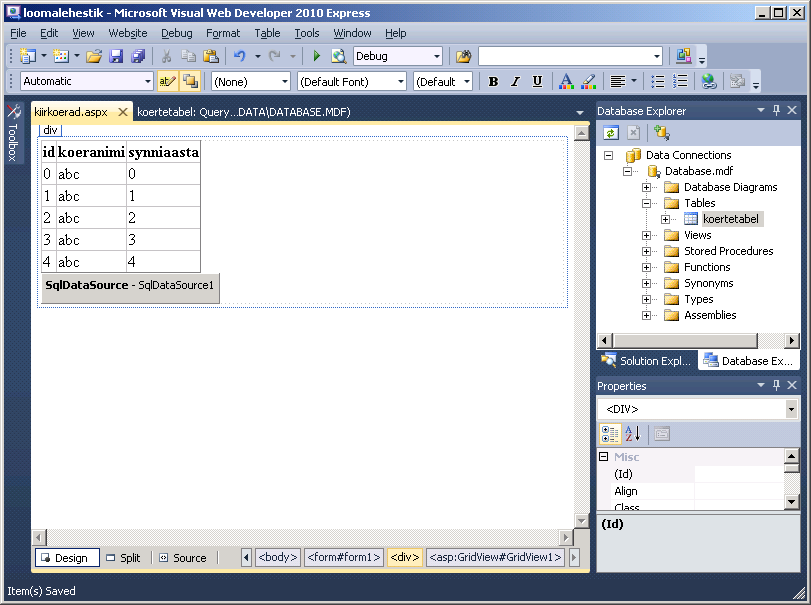


Nende toimetuste tulemusena on andmebaas ja andmetabel olemas. Edasi siis juba tasub vaadata, kuidas andmed veebilehele nähtavaks saab.

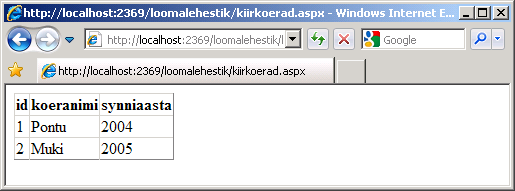
### Ülesandeid

* Tee näide läbi
* Koosta kõrvale sarnane tabel veiste tarbeks. Tulpadeks id, loomanimi, vanus ning mass
* Koosta uus veebirakendus ning sinna andmetabel autode tarbeks. Tulpadeks registrinumber (võtmeks), omanik ning aasta.

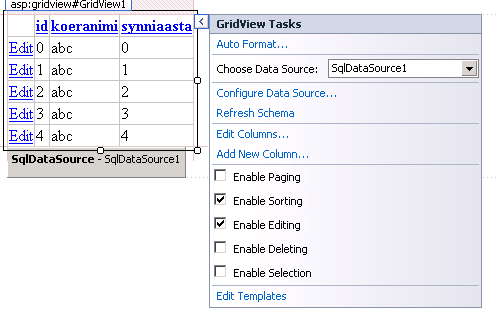
### Andmetabeli näitamine veebilehel

Nagu Microsoftil kombeks, on siiagi sisse ehitatud kiire ja lihtne võimalus. Piisab, kui andmetabelist Database Exploreri aknas hiirega kinni võtta ja lohistada olemasoleva veebilehe disainivaatesse.

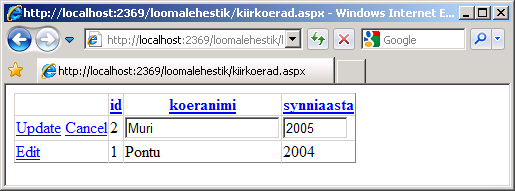
Selle peale luuakse vajalikud ühendused ning andmeid võibki veebilehel vaatama hakata.



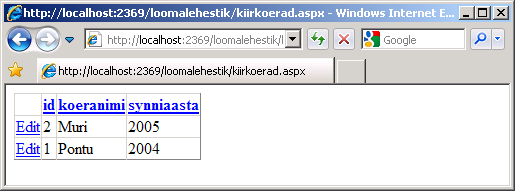
Soovite andmeid sorteerida ja parandada – palun väga. Disainivaates GridView peale minnes tekib paremasse ülanurka väike kolmnurk. Sellele vajutades hüppab välja menüü soovitud toimetuste tarbeks. Vaid „linnukesed“ sisse ja soovitu toimib. Nagu muinasjutus :-)



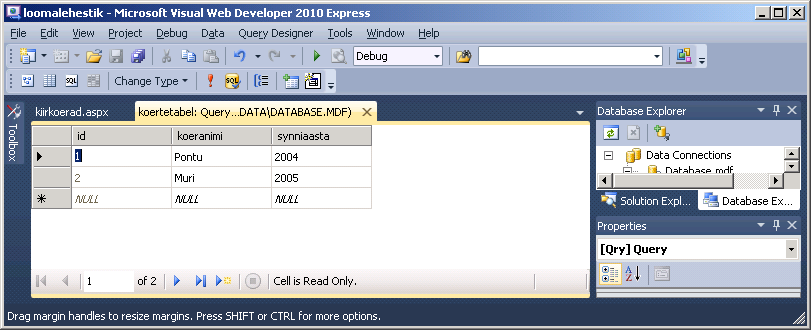
Koeranimele vajutades järjestatakse andmed kõigepealt nimede järgi tähestikulisse järjekorda. Ning vasakult edit-käskluse valides kannatab andmeid muuta ka.



Ja salvestada.



Ning hüüumärgi peale vajutades uuendatakse andmebaasivaade – võib veenduda, et ka seal on Mukist Muri saanud.

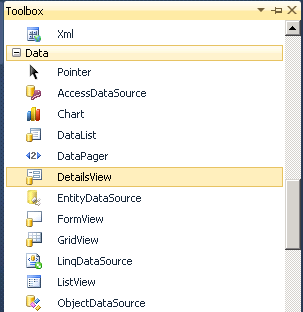


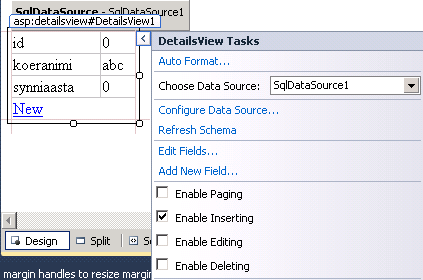
### Ülesandeid

* Tee näide läbi.
* Näita sarnaselt välja ka oma andmetabelites olevad andmed.
* Tutvu disainivaates tabeli kujundamise võimalustega (GridView ülal nurgas väike kolmnurk, sealt AutoFormat)

### Andmete lisamine

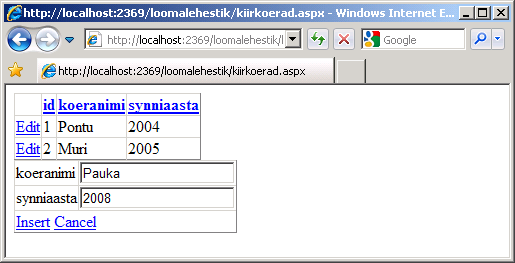
Andmetabeli lehele lohistamisega automaatselt tekkinud GridView kaudu saab küll andmeid järjestada, muuta ja kustutada, ent mitte lisada. Lisamiseks üks mugav abiline on DetailsView.



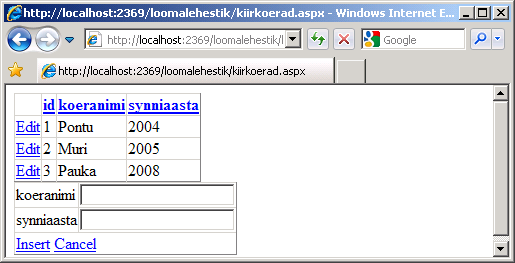
Ta tuleb lehele sobivasse kohta lohistada, andmeallikaks määrata juba lehele tekkinud SqlDataSource1 ning lubada andmete lisamine.   
  


|  |  |
| --- | --- |
| Omaduste juurest tasub määrata, et lisamine ongi selle komponendi vaikimisi omadus: DefaultMode: Insert. |  |

Nii on lehel kohe olemas lisamise jaoks vajalikud lahtrid – ainult kirjuta ja lisa.



... ja veendu, et lisatud andmed ka nähtavad on.



### Ülesandeid

* Tee näide läbi
* Loo sarnane lisamisvõimalus ka oma andmetabelite tarbeks.

### Lehe kohendamine

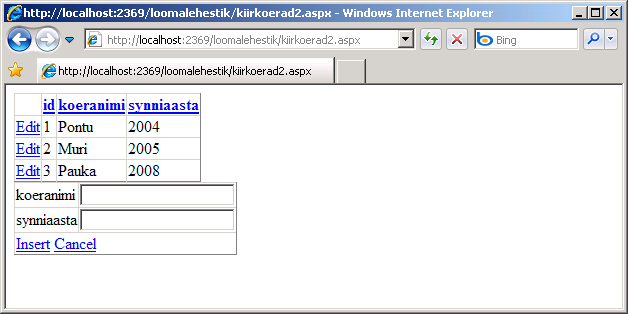
Andmeid lisada, kustutada ja muuta saab küll kergesti. Edasi kipub aga tekkima tahtmine või vajadus lehte ilusamaks või ümbritsevate lehtedega sobivamalt koos toimivamaks kujundada. Vaatame siin järgnevalt mõned võimalused.

Igasugu muutmiste tarbeks on kasulik hoida olemasolevast alles koopia – siis võimalik eelmise meeldiva seisu juurde tagasi minna muretsemata, et millestki tarvilikust ollakse ilma jäänud. ASP.NET rakendus iseenesest talletatakse tekstifailidena, ülejäänud graafiline pool pannakse nende põhjal kokku.

|  |  |
| --- | --- |
| Kui leht koosneb vaid ühest failist, sel juhul saab sellest mugavasti enesele koopia teha. Nagu ikka – hiire parema klahviga „kopeeri/copy“ ning edasi „kleebi/paste“. Kleepimiseks tuleb minna rakenduse kataloogi peale – siin näites E:\...\loomalehestik\ |  |

|  |  |
| --- | --- |
| Ning Rename abil tasub lehele Copy of ... asemel ka viisakas nimi anda. Siin lehel sai selleks nimeks kiirkoerad2.aspx. |  |

Veendumaks et kopeerimine ning ümber nimetamine õnnestus, tasub leht käivitada ja avada. Andmed peaksid samast andmetabelist ilusti nähtavale ilmuma.



Vihjeks: kui leht koosneb mitmest failist, siis ühe lehe kopeerimine nõnda lihtsalt ei õnnestu. Küll aga võiks saada kopeerida üht kataloogi rakenduses või siis keerukamal juhul tervet rakendust.

Et veebilehestik lõpuks siiski kirjas tekstifailina ning mitmeid asju saab vaid siit suhteliselt mugavamalt jälgida või muuta, siis teeme ka selle poolega tutuvust. Esiteks pilk võlurite abil kokku pandud lehe koodile.

<%@ Page Language="C#" %>

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

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:GridView ID="GridView1" runat="server" AllowSorting="True"

AutoGenerateColumns="False" DataKeyNames="id" DataSourceID="SqlDataSource1"

EmptyDataText="There are no data records to display.">

<Columns>

<asp:CommandField ShowEditButton="True" />

<asp:BoundField DataField="id" HeaderText="id" ReadOnly="True"

SortExpression="id" />

<asp:BoundField DataField="koeranimi" HeaderText="koeranimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="synniaasta"

SortExpression="synniaasta" />

</Columns>

</asp:GridView>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:DatabaseConnectionString1 %>"

DeleteCommand="DELETE FROM [koertetabel] WHERE [id] = @id"

InsertCommand="INSERT INTO [koertetabel] ([koeranimi], [synniaasta]) VALUES (@koeranimi, @synniaasta)"

ProviderName="<%$ ConnectionStrings:DatabaseConnectionString1.ProviderName %>"

SelectCommand="SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel]"

UpdateCommand="UPDATE [koertetabel] SET [koeranimi] = @koeranimi, [synniaasta] = @synniaasta WHERE [id] = @id">

<DeleteParameters>

<asp:Parameter Name="id" Type="Int32" />

</DeleteParameters>

<InsertParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

</InsertParameters>

<UpdateParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

<asp:Parameter Name="id" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

</div>

<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"

DataKeyNames="id" DataSourceID="SqlDataSource1" DefaultMode="Insert"

Height="50px" Width="125px">

<Fields>

<asp:BoundField DataField="id" HeaderText="id" InsertVisible="False"

ReadOnly="True" SortExpression="id" />

<asp:BoundField DataField="koeranimi" HeaderText="koeranimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="synniaasta"

SortExpression="synniaasta" />

<asp:CommandField ShowInsertButton="True" />

</Fields>

</asp:DetailsView>

</form>

</body>

</html>

Seletused

ASP.NET lehe päis. Lihtsamal juhul teatatakse vaid lehel kasutatav programmikoodi keel, kuid hiljem võib sinna hulgem parameetreid juurde tulla.

<%@ Page Language="C#" %>

Järgneb HTMLi päis, mis otse veebilehitsejasse näidatakse ning millise standardi järgi võiks siis ka ülejäänud leht vormindatud olla.

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

Koht skriptide jaoks. Põhjalikuma rakenduse puhul paigutatakse skriptid eraldi faili, kuid lihtsamal juhul on siin koht käskluste käivitamiseks. Nõnda koodi ühes failis hoides säilib ka lehe mugav kopeerimisvõimalus. Äramärkimist väärib runat=“server“ parameeter skripti juures. ASP.NET tegeleb vaid nende märgenditega, mil vastav atribuut juures. Muul juhul oleks teemist kliendipoolse Javaskripti lõiguga.

<script runat="server">

</script>

HTMLi osa algus. Atribuut runat head-i juures võimaldab sealset sisu(näiteks pealkirja) programmikoodi abil muuta. Samas aga jätab koha ka pealkirja otse koodi sisse kirjutamiseks.

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

ASP.NET lehel on vaid üks runat=“server“-atribuudiga vorm. Ning kogu lehe programmiline tegevus käib selle sees. Enamasti pole põhjust vormi vaikimisi id-d muuta.

<form id="form1" runat="server">

<div>

Järgnevad juba lehe töö seisukohalt vajalikud elemendid. GridView siinjuhul andmete tabeli kujul näitamiseks ning nende parandamiseks. Kõige lihtsamal juhul suudab GridView ka siis andmeid näidata, kui on loodud vastav element ning määratud, kust andmeid võtta. Siin on aga parameeter AutoGenerateColumns väärtusega „False“, mis tähendab, et veerud tuleb (ja saab) ise kirjeldada ning nende omadusi sättida.

<asp:GridView ID="GridView1" runat="server" AllowSorting="True"

AutoGenerateColumns="False" DataKeyNames="id" DataSourceID="SqlDataSource1"

EmptyDataText="There are no data records to display.">

<Columns>

<asp:CommandField ShowEditButton="True" />

<asp:BoundField DataField="id" HeaderText="id" ReadOnly="True"

SortExpression="id" />

<asp:BoundField DataField="koeranimi" HeaderText="koeranimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="synniaasta"

SortExpression="synniaasta" />

</Columns>

</asp:GridView>

Andmeallika kaudu käib suhtlus andmeelementide ning andmebaasi vahel. Siinsel juhul on näha SQL-laused põhioperatsioonide jaoks ning seade, kus tegelikku andmebaasi asukohta määrav ConnectionString võetakse konfiguratsioonifailist loetelu ConnectionStrings kaudu.

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:DatabaseConnectionString1 %>"

DeleteCommand="DELETE FROM [koertetabel] WHERE [id] = @id"

InsertCommand="INSERT INTO [koertetabel] ([koeranimi], [synniaasta]) VALUES (@koeranimi, @synniaasta)"

ProviderName="<%$ ConnectionStrings:DatabaseConnectionString1.ProviderName %>"

SelectCommand="SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel]"

UpdateCommand="UPDATE [koertetabel] SET [koeranimi] = @koeranimi, [synniaasta] = @synniaasta WHERE [id] = @id">

<DeleteParameters>

<asp:Parameter Name="id" Type="Int32" />

</DeleteParameters>

<InsertParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

</InsertParameters>

<UpdateParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

<asp:Parameter Name="id" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

</div>

Siinses rakenduses lisamiseks mõeldud DetailsView võtab tänu DefaultMode=“Insert“-ile ka vastava kuju kohe ette. Ning kasutajapoolne lisamiskäsklus kantakse üle SqlDataSource1 vastavaks Insert-käskluseks koos tekstiväljadest kaasa tulevate parameetritega.

<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"

DataKeyNames="id" DataSourceID="SqlDataSource1" DefaultMode="Insert"

Height="50px" Width="125px">

<Fields>

<asp:BoundField DataField="id" HeaderText="id" InsertVisible="False"

ReadOnly="True" SortExpression="id" />

<asp:BoundField DataField="koeranimi" HeaderText="koeranimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="synniaasta"

SortExpression="synniaasta" />

<asp:CommandField ShowInsertButton="True" />

</Fields>

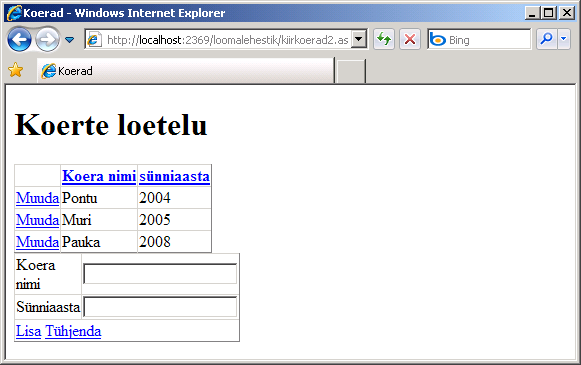
</asp:DetailsView>

</form>

</body>

</html>

Järgnevalt mõned täiendused, mida saab teha ka seadete alt valikuid otsides, kuid mis koodi sisse kirjutatuna õnnestub küllalt hõlpsalt ette võtta.



Nagu pildilt näha, muudeti pealkirjad lugemiseks mugavamaks ning tõlgiti käsklused. Samuti sai lehe pealkirjaks „Koerad“. Viimatine siis HTMLi title-elemendi sisse. Tõlkimine vastavate elementide juurde. GridView puhul EditText, UpdateText ja CancelText parameetrid, DetailsView juures InsertText ja CancelText. Veergude nimede jaoks tulpadel omadus HeaderText.

<%@ Page Language="C#" %>

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

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Koerad</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<h1>Koerte loetelu</h1>

<asp:GridView ID="GridView1" runat="server" AllowSorting="True"

AutoGenerateColumns="False" DataKeyNames="id" DataSourceID="SqlDataSource1"

EmptyDataText="Koerte andmed puuduvad.">

<Columns>

<asp:CommandField ShowEditButton="True" EditText="Muuda" UpdateText="Salvesta muutused" CancelText="Katkesta" />

<asp:BoundField DataField="id" HeaderText="id" ReadOnly="True"

SortExpression="id" Visible="false" />

<asp:BoundField DataField="koeranimi" HeaderText="Koera nimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="sünniaasta"

SortExpression="synniaasta" />

</Columns>

</asp:GridView>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:DatabaseConnectionString1 %>"

DeleteCommand="DELETE FROM [koertetabel] WHERE [id] = @id"

InsertCommand="INSERT INTO [koertetabel] ([koeranimi], [synniaasta]) VALUES (@koeranimi, @synniaasta)"

ProviderName="<%$ ConnectionStrings:DatabaseConnectionString1.ProviderName %>"

SelectCommand="SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel]"

UpdateCommand="UPDATE [koertetabel] SET [koeranimi] = @koeranimi, [synniaasta] = @synniaasta WHERE [id] = @id">

<DeleteParameters>

<asp:Parameter Name="id" Type="Int32" />

</DeleteParameters>

<InsertParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

</InsertParameters>

<UpdateParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

<asp:Parameter Name="id" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

</div>

<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"

DataKeyNames="id" DataSourceID="SqlDataSource1" DefaultMode="Insert"

Height="50px" Width="125px">

<Fields>

<asp:BoundField DataField="id" HeaderText="id" InsertVisible="False"

ReadOnly="True" SortExpression="id" />

<asp:BoundField DataField="koeranimi" HeaderText="Koera nimi"

SortExpression="koeranimi" />

<asp:BoundField DataField="synniaasta" HeaderText="Sünniaasta"

SortExpression="synniaasta" />

<asp:CommandField ShowInsertButton="True" InsertText="Lisa" CancelText="Tühjenda" />

</Fields>

</asp:DetailsView>

</form>

</body>

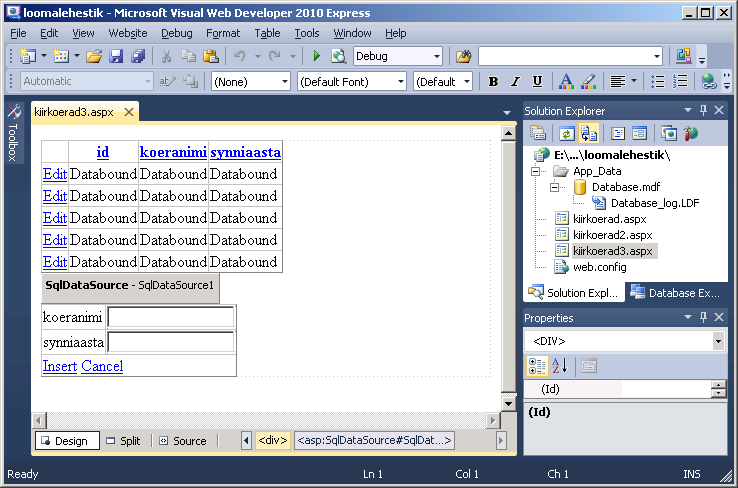
</html>

### Ülesandeid

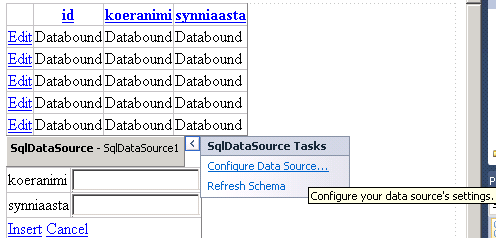
* Tee näide läbi
* Muuda oma tabelite veebiliidese juures tulpade ning käsunuppude nimesid.
* Määra lehele omapoolne pealkiri ning CSSi abil taustavärv.

### Järjestamine

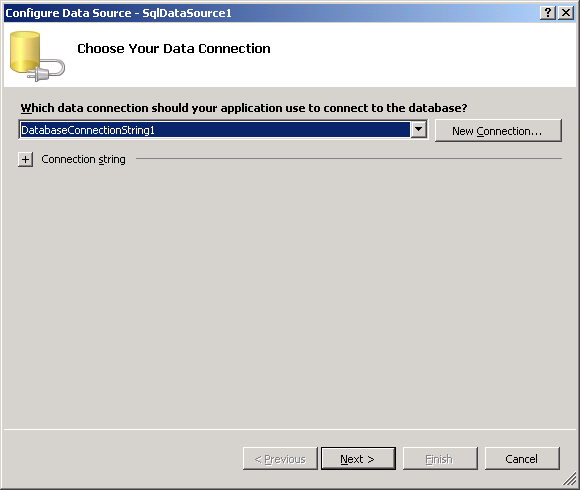
Kõigepealt taas esialgsest rakendusest koopia, et oleks koht, mille juures julgesti katsetada.



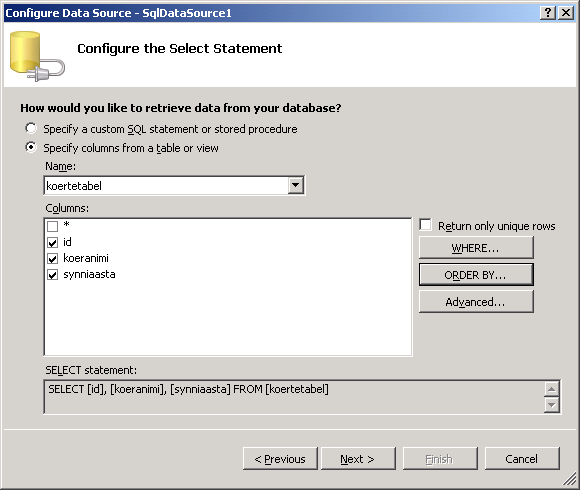
Disainivaates andmeallika peale minnes ilmub selle paremasse ülanurka väike kolmnurk. Sealt Configure Data Source võimaldab andmepäringuga seonduvat täpsemini seadistada.



Kõigepealt küsitakse üle, kust soovitakse andmeid võtta. Koht võib samaks jääda, ehk siis sobib pakutud DatabaseConnectionString1.

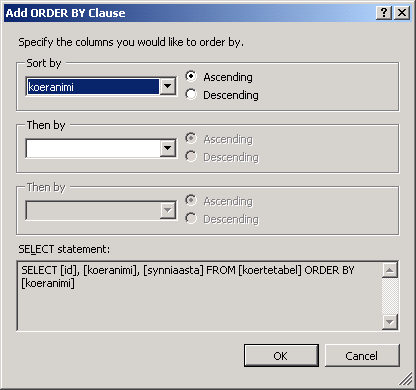


Ka koertetabel koos kõigi oma tulpadega võib samaks jääda.



Lehelt aga tasub vajutada nuppu ORDER BY..., mis võimaldab päringust tulevaid andmeid sobivalt järjestada.

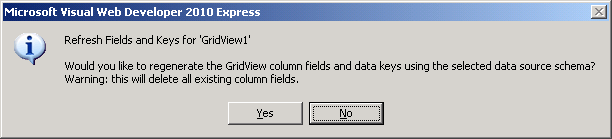
Lihtsamal juhul sorteerime ühe tulba, näiteks koera nime kaudu.

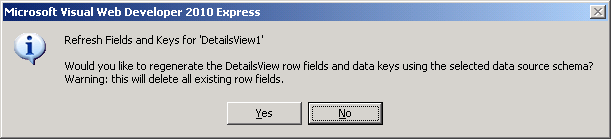


Nagu näha, tekib andmete küsimiseks siis vastav SQL päringulause, kus võrreldes eelnevaga on juures ORDER BY järjestusosa.

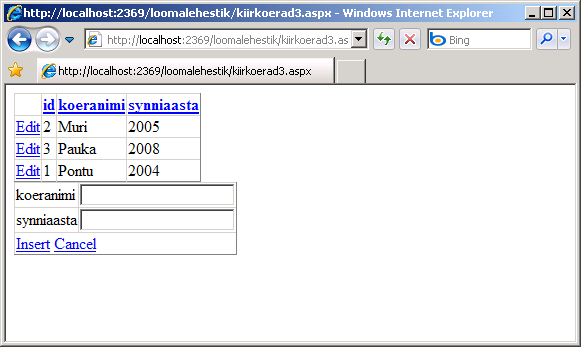
SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel] ORDER BY [koeranimi]

Pärast päringu muutmist küsitakse mõlema andmenäitamiskomponendi kohta üle, kas soovitakse nende sisu uuendada. Põhjuseks, et päringus võisid tulbad muutuda ning pakutakse võimalust, et komponendid hakkaksid näitama uut seisu. Samas – uuendamise käigus lähevad kaduma kõik vahepealsed konfigureerimistoimetused nende komponentide juures. Nii et tasub mõelda, milline valik parajasti kasulikum on. Siin näites tegelikult vahet ei ole kumb võimalus valida, sest esimesest näitest tehtud koopial pole veel elementide küljes mingeid tõlkeid ega seadistusi.





Pärast sorteerimiskäsu lisamist võib jälgida, et koerad vaikimisi näidatakse tähestikulises järjekorras.



Vastav SQL-lause siis ka andmeallika koodis nähtav.

SelectCommand="SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel] ORDER BY [koeranimi]"

### Ülesandeid

* Tee näide läbi
* Järjesta veised massi järgi
* Järjesta veised vanuse ja massi järgi

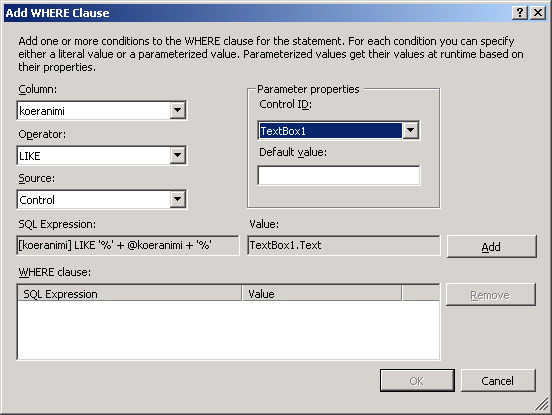
### Otsimine

Suurema andmehulga puhul aitab otsimine kasutajal sobivat leida. Lihtsamal juhul otsitakse ühe tulba järgi, kuid vajadusel saab ka keerukamaid otsinguid kokku panna. Igatahes tasub luua otsimiskast koos sisestusnupuga.

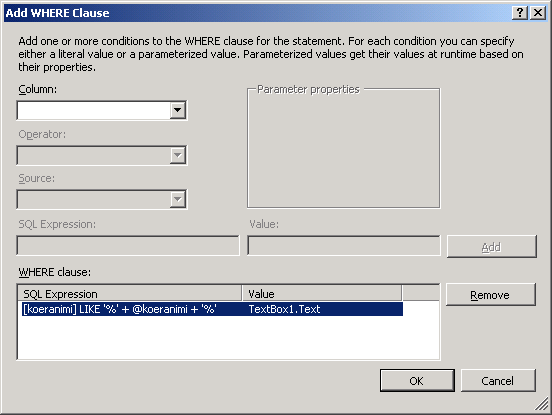


Andmeallika konfigureerimisosas saab päringule lisada WHERE-tingimuse.

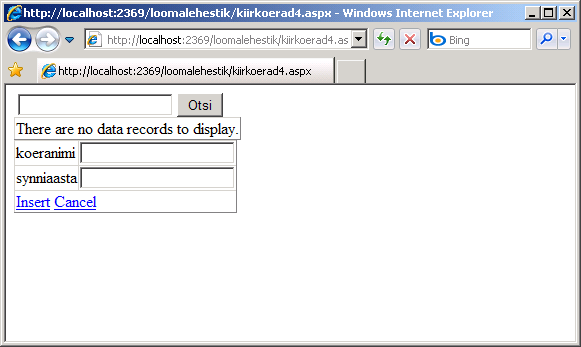
Otsimisvõimalusi on hulgem. Kui allikaks soovitakse lehel olevat sisestuselementi, siis tuleb Source-valikusse võtta Control. Edasi juba järgneb võimalus valida, millise sisestuselemendi järgi otsitakse. Kui operaatoriks valida võrdusmärgi asemel LIKE, sellisel juhul otsitakse teksti ükskõik kust vastava tulba seest – ei pea tervet nime täpselt välja kirjutama.



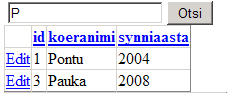
Pärast päringu valmimist tuleb vajutada Add-nupule ja seejärel OK-le.



Kui midagi pole sisestatud, siis tuleb teade, et ühtki vastet ei leitud. Selle teate saab vajadusel koodi sees ka maakeelseks tõlkida nagu ülal näites tehtud.



Kui aga kasvõi üks täht sisse kirjutatakse, siis otsitakse välja seda tähte sisaldavad nimed.



Koodis veidi uurimistööd tehes selgub, et otsimisega seonduv on pandud SelectParameters-sektsiooni. Samuti üks parameeter SQL-lause sisse.

<SelectParameters>

<asp:ControlParameter ControlID="TextBox1" Name="koeranimi" PropertyName="Text"

Type="String" />

</SelectParameters>

SelectParameters all oleva asp:ControlParameter-i juures saab määrata, et ConvertEmptyStringToNull=“false“. Sellisel juhul andmete puudumisel ei loeta seda tühiväärtuseks (null), vaid lihtsalt tühjaks tekstiks.

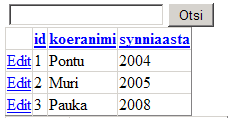
<SelectParameters>

<asp:ControlParameter ControlID="TextBox1" Name="koeranimi" PropertyName="Text"

Type="String" ConvertEmptyStringToNull="false" />

</SelectParameters>

Ning väljastatakse kõik read, sest LIKE käsklus saab aru, et tühi tekst leidub igas lauses.



Ülisuurte andmehulkade puhul peab sellise lubamisega ettevaatlik olema. Aga väiksemate koguste puhul on nii päris mugav otsida, et vaikimisi näeb kõike (või üht lehekülge sellest) ning otsingu abil saab näidatavat piirata.

Otsingulehe andmeallika kood ka eraldi, et oleks näha, mis kuhu paigutati.

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:DatabaseConnectionString1 %>"

DeleteCommand="DELETE FROM [koertetabel] WHERE [id] = @id"

InsertCommand="INSERT INTO [koertetabel] ([koeranimi], [synniaasta]) VALUES (@koeranimi, @synniaasta)"

ProviderName="<%$ ConnectionStrings:DatabaseConnectionString1.ProviderName %>"

SelectCommand="SELECT [id], [koeranimi], [synniaasta] FROM [koertetabel] WHERE ([koeranimi] LIKE '%' + @koeranimi + '%')"

UpdateCommand="UPDATE [koertetabel] SET [koeranimi] = @koeranimi, [synniaasta] = @synniaasta WHERE [id] = @id">

<DeleteParameters>

<asp:Parameter Name="id" Type="Int32" />

</DeleteParameters>

<InsertParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

</InsertParameters>

<SelectParameters>

<asp:ControlParameter ControlID="TextBox1" Name="koeranimi" PropertyName="Text"

Type="String" ConvertEmptyStringToNull="false" />

</SelectParameters>

<UpdateParameters>

<asp:Parameter Name="koeranimi" Type="String" />

<asp:Parameter Name="synniaasta" Type="Int32" />

<asp:Parameter Name="id" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

### Ülesandeid

* Tee näide läbi
* Koosta sarnane otsing veistele loomanime järgi
* Muuda otsingut nõnda, et otsitaks korraga mitme tulba andmete järgi.

### Konfiguratsioonifail web.config

Alates ASP.NET versioonist 4.0 on konfiguratsioonifail vaikimisi kujul taas suhteliselt lihtne ja kergesti hallatav. Ning tema sisu on kasulik tunda. Sest erinevalt nt. PHP-st, kus vastavale failile pääseb enamasti ligi vaid süsteemiadministraator, on siin igal programmeerijal võimalus rakenduste ja kataloogide kaupa sättida, mis ta kusagil arvab vajalik olema.

Praegusel juhul paistab, et konfiguratsioonifaili on kirjutatud andmebaasiühenduseks vajalik ConnectionString. Ning sealtkaudu määratakse, et andmebaasi asukohaks on |DataDirectory|\Database.mdf, ehk siis rakenduse alamkaustas App\_Data olev Database.mdf-nimeline fail. Rakenduse ülesseadmisel teise serverisse võib vajalik olla selle asukoha muutmine või sootuks teistmoodi andmebaasiühenduse kättesaadavaks tegemine ConnectionString-i kaudu. Kui juhtub, et ülespanekul vastav juhis antakse, siis siinne lõik on julgustamaks konfiguratsioonifaili täiendama.

Teise lõiguna on näha käsklus compilation debug=“false“. Nagu aimata võib, siis veateadete lähemaks lugemiseks võib olla põhjust see „false“ „true“-ks keerata. Esmanäitena aga mahub fail vähem kui poolele leheküljele ära. Siitkaudu konfigureerimise võimalused iseenesest päris laiad. Määrata tasub aga ainult väärtusi, mis vaikeseadetest erinevad.

<?xml version="1.0"?>

<!--

For more information on how to configure your ASP.NET application, please visit

http://go.microsoft.com/fwlink/?LinkId=169433

-->

<configuration>

<connectionStrings>

<add name="DatabaseConnectionString1" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated Security=True;User Instance=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

<system.web>

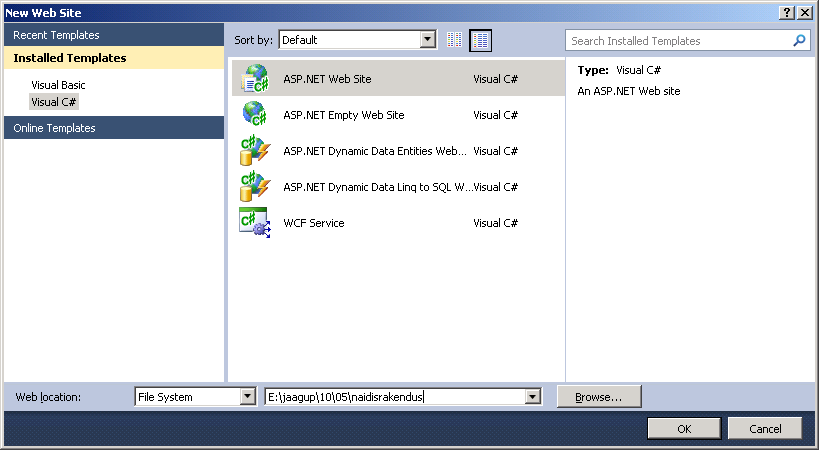
<compilation debug="false" targetFramework="4.0" />

</system.web>

</configuration>

Arenduskeskkonna näidisrakendus

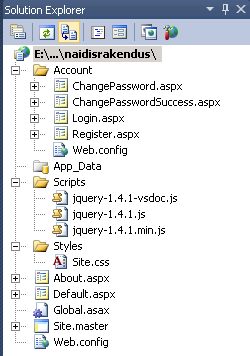
Juba aastakümneid pannakse keskkondadega kaasa näitrakendusi, mille abil võiks mugav olla pakutavate võimalustega tutvuda. Nõnda ka Visual Web Developer Express pakub kohe esimese valikuna välja ASP.NET Web Site, kus pealtnäha kõige hädapärasem on veebilehestiku loomiseks täiesti olemas. Vaja vaid kohandada oma vajadustele ja võimalustele.



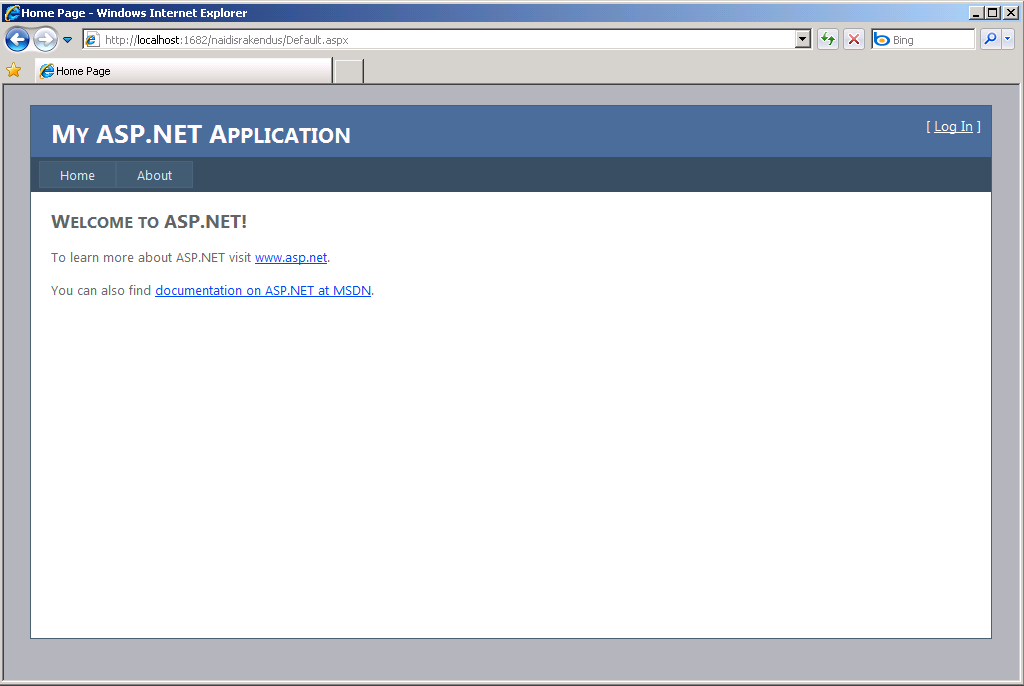
Nimeks sai talle naidisrakendus. Ning eelnevalt eelmisest tühjast näitest on siinses rakenduspuus faile hulgem. Iseenesest tegemist failidega, mis vähegi suurema veebirakenduse puhul ka ise tuleb sinna paika sättida.

Account-nimeline kataloog paistab olema kasutajate loomise ja sisenemiskontrolli jaoks. App\_Data-s hoitakse vajalikud andmed. Skriptide alla paigutatakse vajalikud Javaskripti-täiendused. Sedakorda on siia „nuusutamiseks“ pandud paar JQuery-nimelise abiraamistiku teeki. CSS-faili(de) jaoks omaette kaust. Otse nähtavaid faile on vaid kaks: Default.aspx ja About.aspx. Ning nendegi kogu sisu pole failide eneste sees: Site.master hoolitseb lehestiku üldise kujunduse eest.

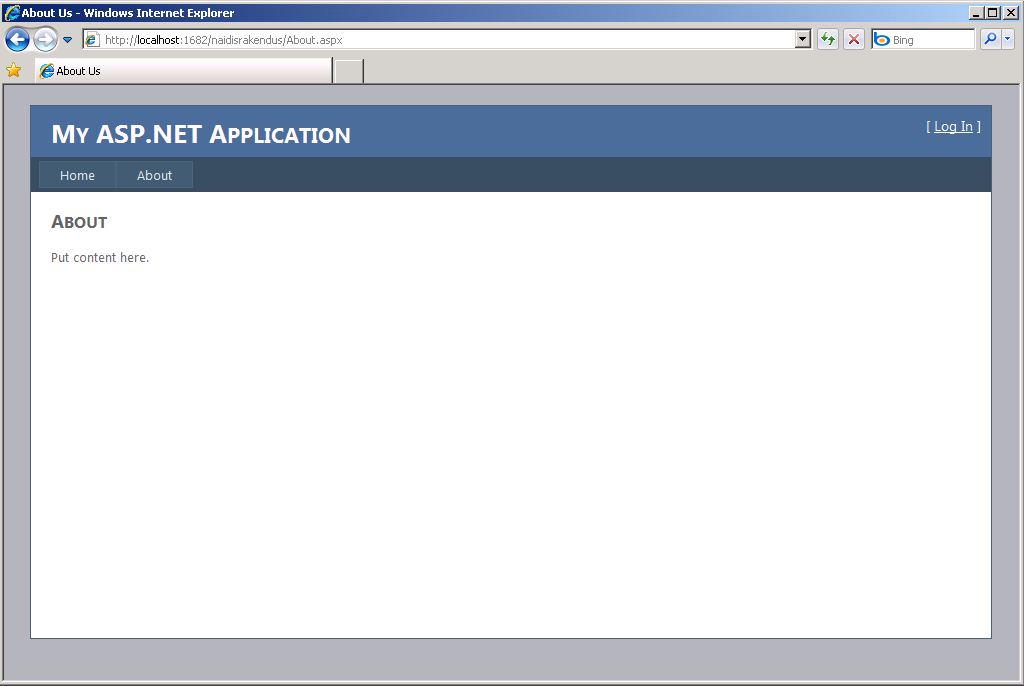
Juures veel kaks „tehnilisemat“ faili: Global.asax võimaldab mitmesuguste sündmuste peale koodijuppe käivitada. Web.config sisaldab rakenduse sätteid.



Iga rakenduse puhul tasub proovida, kas ta ikka käima ka läheb. Siin paistab avaleht käivitamisel selgesti avanema.



Ja sarnase kujundusega avaneb ka tutvustav About-leht. Tuleviku huvides tasub uurida, kuidas need kujundused nõnda sarnased on. Ning kust ja kuidas saab omale tarvilikke asju sättida.



About.aspx-lehe kood paistab üllatavalt lühike olema – puuduvad tavapärased HTMLi osad. Vaid päiserida, HeaderContent ja BodyContent, millest veel esimene on tühi.

<%@ Page Title="About Us" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"

CodeFile="About.aspx.cs" Inherits="About" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">

</asp:Content>

<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">

<h2>

About

</h2>

<p>

Put content here.

</p>

</asp:Content>

Katseks muudan sisuteksti ja vaatan tulemust lehel.

<%@ Page Title="About Us" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"

CodeFile="About.aspx.cs" Inherits="About" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">

</asp:Content>

<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">

<h2>

Koerte leht

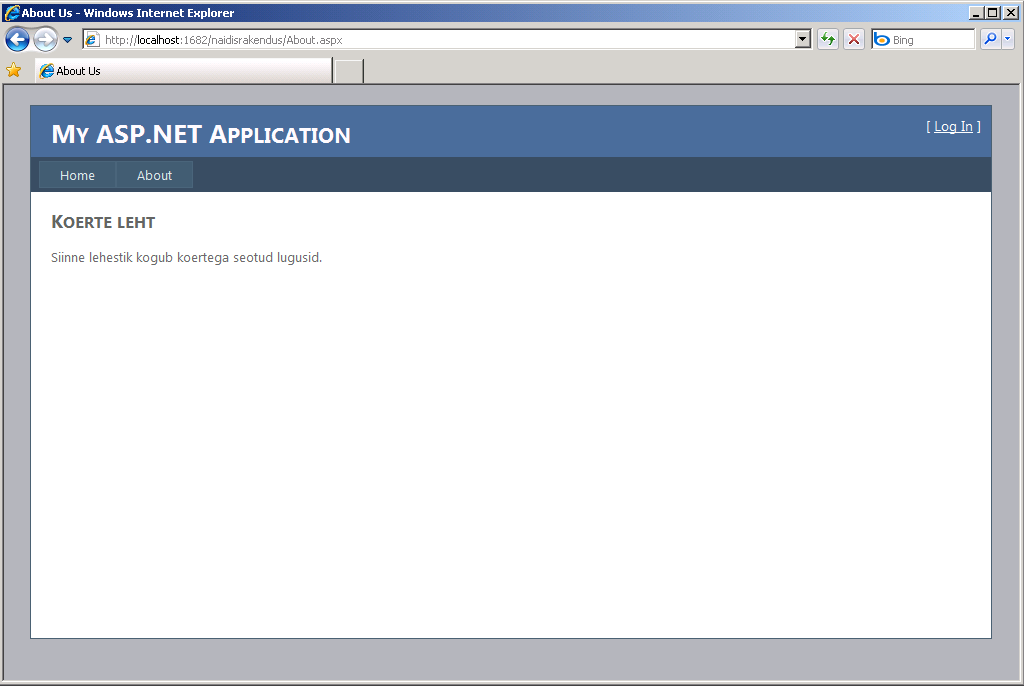
</h2>

<p>

Siinne lehestik kogub koertega seotud lugusid.

</p>

</asp:Content>



Nagu näha võib, siis maakeelne tekst ilmus täiesti nähtavale. Samas osa lehest tuleb hoopis kaugemalt. Lehe päist uurides leiab sealt atribuudi MasterPageFile="~/Site.master" Selle järgi teab sisu otsida Site.master-nimelisest failist, mille küljest About.aspx omale kujunduse külge haagib.

Master lehel tasub tähelepanu pöörata reale

<asp:ContentPlaceHolder ID="MainContent" runat="server"/>

Sinna tuleb konkreetse lehe sisu nagu About.aspx-i puhul näha oli.

<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">

...

</asp:Content>

Muu master-lehe sisu võetakse igale lehele kaasa. Selline lähenemine annab võrreldes nt. PHP-keeles tavaks olevale päise ja jaluse sisselugemisele võimalus kujundus ühe tervikuna külge võtta.

Veel mõned lõigud, millele tasub master-failis tähelepanu pöörata. Kui head-plokile on määratud atribuut runat=“server“, siis saab selle elementide väärtusi serveris koodi abil muuta. Näiteks võimaldab master-lehte kasutaval alamlehel panna näidatavale lehele pealkirja – ehkki otsest ligipääsu vastavale HTML-koodi jupile pole.

Päiseosas asuv ContentPlaceHolder võimaldab alamlehtedel sinna vajalikke täiendusi saata – enamasti Javaskripti koodi või CSSi käsklusi.

<head runat="server">

<title></title>

<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />

<asp:ContentPlaceHolder ID="HeadContent" runat="server">

</asp:ContentPlaceHolder>

</head>

Lehel olev LoginView näitab sisu vastavalt sellele, kas kasutaja on sisse logitud või mitte. Nagu siit näha – anonüümse kasutaja jaoks on viide logimislehele ~/Account/Login.aspx, sisenenud kasutajat tervitatakse tema kasutajanimega (element asp:LoginName). Kapsaraud/tilde aadressi ees tähistab vastava veebirakenduse juurkataloogi.

<asp:LoginView ID="HeadLoginView" runat="server" EnableViewState="false">

<AnonymousTemplate>

[ <a href="~/Account/Login.aspx" ID="HeadLoginStatus" runat="server">Log In</a> ]

</AnonymousTemplate>

<LoggedInTemplate>

Welcome <span class="bold"><asp:LoginName ID="HeadLoginName" runat="server" /></span>!

[ <asp:LoginStatus ID="HeadLoginStatus" runat="server" LogoutAction="Redirect" LogoutText="Log Out" LogoutPageUrl="~/"/> ]

</LoggedInTemplate>

</asp:LoginView>

Menüüvalikud kirjeldatakse ASP.NETi menüüelemendiga. Hilisemat kujundust saab parameetrite ja CSSiga seadistada.

<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">

<Items>

<asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>

<asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>

</Items>

</asp:Menu>

Järgnevalt näidisrakenduse kujundust määrav master-leht tervikuna.

<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Site.master.cs" Inherits="SiteMaster" %>

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

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head runat="server">

<title></title>

<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />

<asp:ContentPlaceHolder ID="HeadContent" runat="server">

</asp:ContentPlaceHolder>

</head>

<body>

<form runat="server">

<div class="page">

<div class="header">

<div class="title">

<h1>

My ASP.NET Application

</h1>

</div>

<div class="loginDisplay">

<asp:LoginView ID="HeadLoginView" runat="server" EnableViewState="false">

<AnonymousTemplate>

[ <a href="~/Account/Login.aspx" ID="HeadLoginStatus" runat="server">Log In</a> ]

</AnonymousTemplate>

<LoggedInTemplate>

Welcome <span class="bold"><asp:LoginName ID="HeadLoginName" runat="server" /></span>!

[ <asp:LoginStatus ID="HeadLoginStatus" runat="server" LogoutAction="Redirect" LogoutText="Log Out" LogoutPageUrl="~/"/> ]

</LoggedInTemplate>

</asp:LoginView>

</div>

<div class="clear hideSkiplink">

<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">

<Items>

<asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>

<asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>

</Items>

</asp:Menu>

</div>

</div>

<div class="main">

<asp:ContentPlaceHolder ID="MainContent" runat="server"/>

</div>

<div class="clear">

</div>

</div>

<div class="footer">

</div>

</form>

</body>

</html>

Võõrad lehed saab ikka tuttavaks neid enese järgi kohandades – siis näha millised muutused kuhu mõjuvad. Ülal koostasime koerte lehe, siin siis täiendused, et ka ülejäänu lehest temaatiline välja paistaks.

<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Site.master.cs" Inherits="SiteMaster" %>

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

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head runat="server">

<title></title>

<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />

<asp:ContentPlaceHolder ID="HeadContent" runat="server">

</asp:ContentPlaceHolder>

</head>

<body>

<form runat="server">

<div class="page">

<div class="header">

<div class="title">

<h1>

Koertelehestik

</h1>

</div>

<div class="loginDisplay">

<asp:LoginView ID="HeadLoginView" runat="server" EnableViewState="false">

<AnonymousTemplate>

[ <a href="~/Account/Login.aspx" ID="HeadLoginStatus" runat="server">Sisene</a> ]

</AnonymousTemplate>

<LoggedInTemplate>

Tere tulemast, <span class="bold"><asp:LoginName ID="HeadLoginName" runat="server" /></span>!

[ <asp:LoginStatus ID="HeadLoginStatus" runat="server" LogoutAction="Redirect" LogoutText="Log Out" LogoutPageUrl="~/"/> ]

</LoggedInTemplate>

</asp:LoginView>

</div>

<div class="clear hideSkiplink">

<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">

<Items>

<asp:MenuItem NavigateUrl="~/Default.aspx" Text="Avaleht"/>

<asp:MenuItem NavigateUrl="~/About.aspx" Text="Tutvustus"/>

</Items>

</asp:Menu>

</div>

</div>

<div class="main">

<asp:ContentPlaceHolder ID="MainContent" runat="server"/>

</div>

<div class="clear">

</div>

</div>

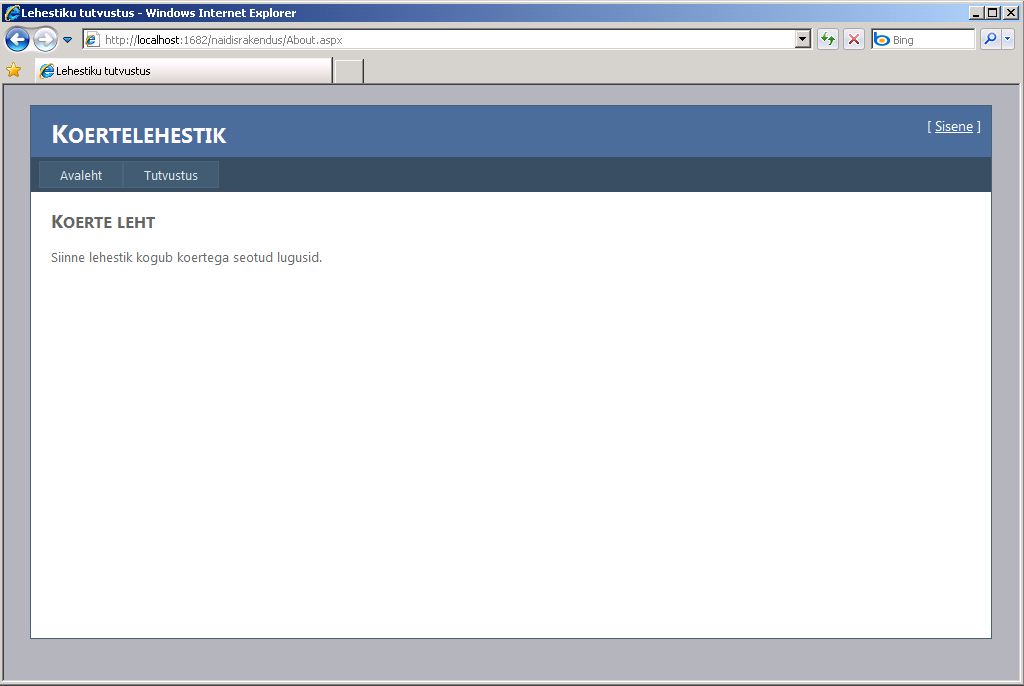
<div class="footer">

</div>

</form>

</body>

</html>

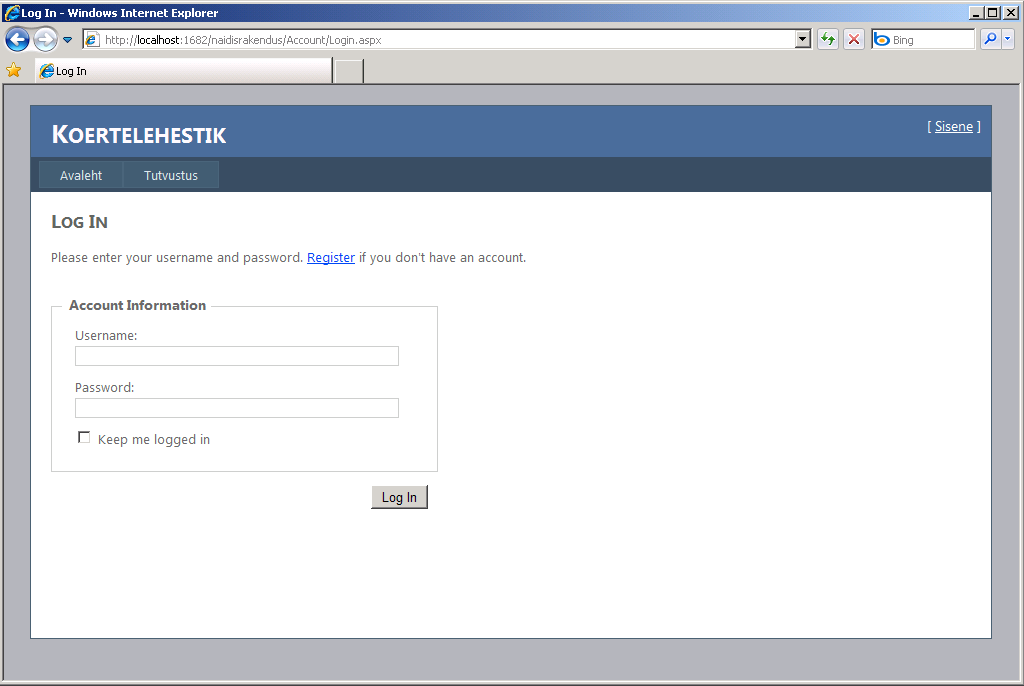


Et lehe päis maakeelseks saaks, selleks tuleb vastav täendus ka About.aspx koodi sisse panna, lehe Title-omadusele uus väärtus anda.

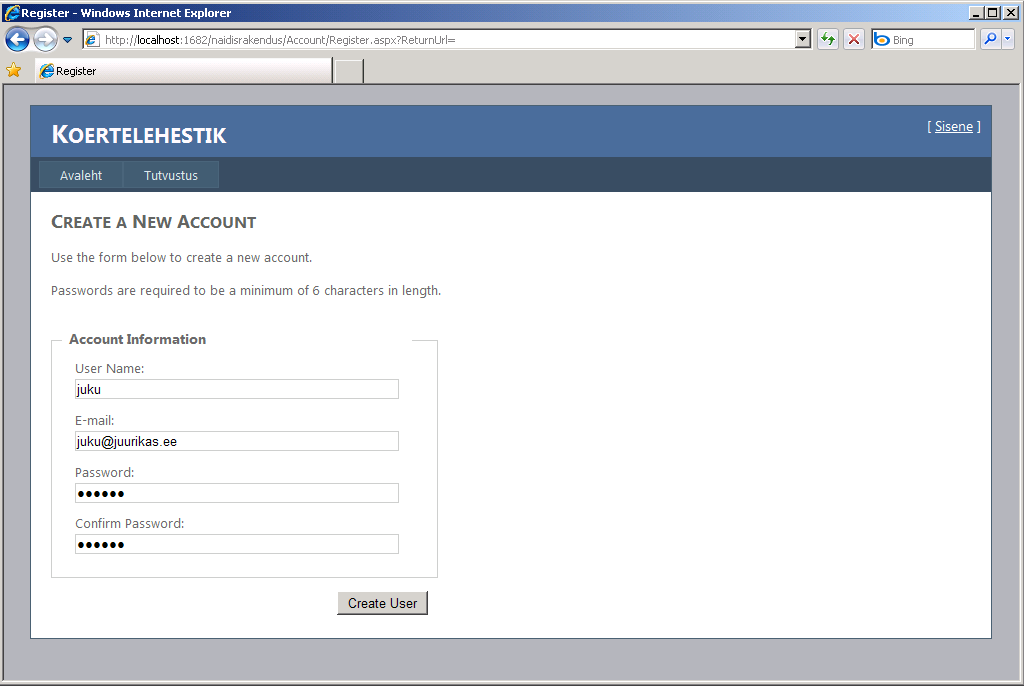
<%@ Page Title="Lehestiku tutvustus" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"

CodeFile="About.aspx.cs" Inherits="About" %>

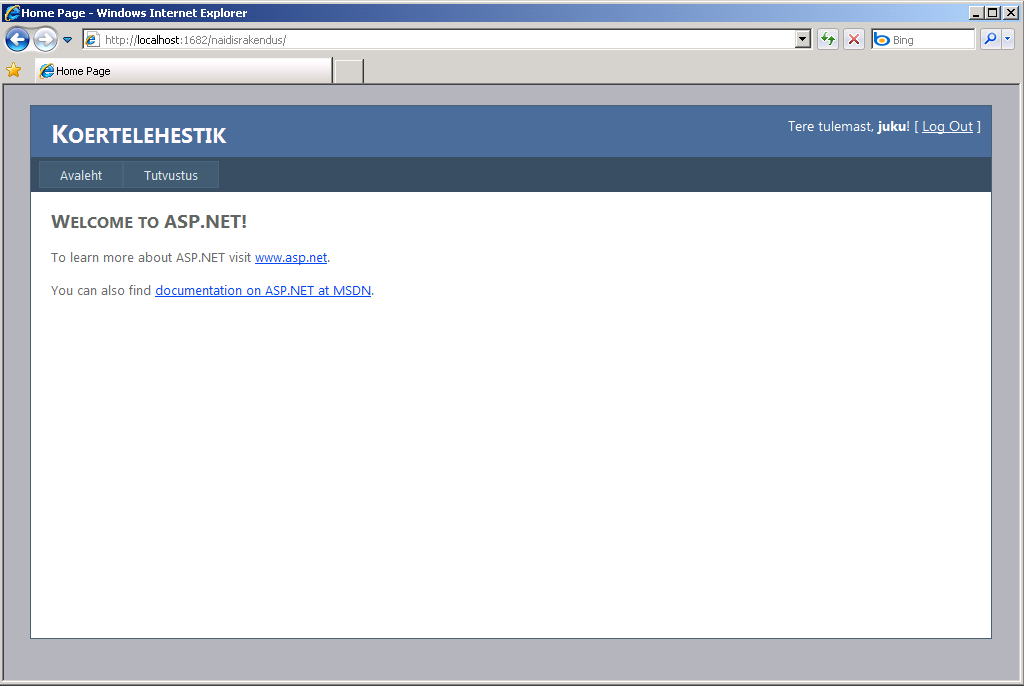
Näitlehestikus pakutakse lahkesti sisselogimisvõimalust. Uurimiseks proovime järele.



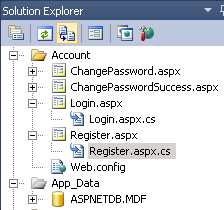
Et omal kasutajat veel loodud pole, siis tuleb see registreerida nagu viite alt pakutakse.



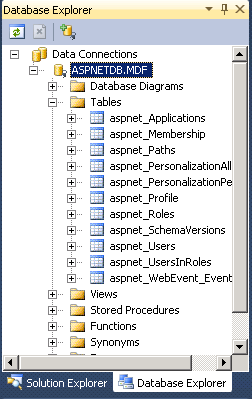
Pakkusime nimeks juku, valisime parooli ning koos kasutaja loomisega on ta ka sisse logitud.



Edasi tasub uurida, et mis rakenduses siis tegelikult selle peale juhtus. Solution Exploreris kahe rohelise noolega uuesti laadmise nuppu vajutase ilmus App\_Data kausta nähtavale andmebaasifail ASPNETDB.MDF.

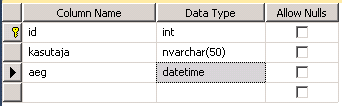


Tegemist failiga, kuhu ASP:NET suudab kasutajahalduse automaatselt genereeritud tabelid panna. Database Exploreri kaudu vaadates näeb, et tabeleid selleks tarbeks päris hulgem tehtud.

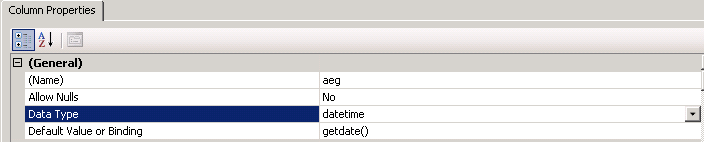


Andmebaas ei pruugi vaid nende tabelitega piirduda. Soovides omapoolseid andmeid salvestada, teeme selle jaoks eraldi tabeli. Üheks lihtsaks mooduseks on kirja panna näiteks veebilehe külastused – kes ja millal käis. Põhimõtteliselt võimalik administraatoril neid ka logifailist vaadata, aga tavainimesel mugavam veebist tulemusi uurida.

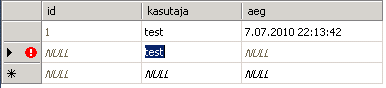
Tabeli nimeks sai kylastused ning tulpadeks id, kasutaja ja aeg.



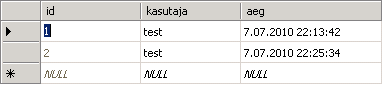
Et aeg automaatselt sisse pandaks, saab sellele seada “Default Value”. Hetke aja annab funktsioon getdate()



Katsetuseks lisame uue kirje. (Database Explorer – Show table data) Kasutajanimeks test. Esialgu jäävad id ning aja kohale tühiväärtused (NULL).

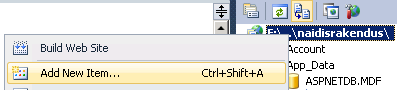


Hüüumärgile  vajutades aga luuakse baasiühendus ning kohale paigutatakse väljade automaatselt genereeritud väärtused.

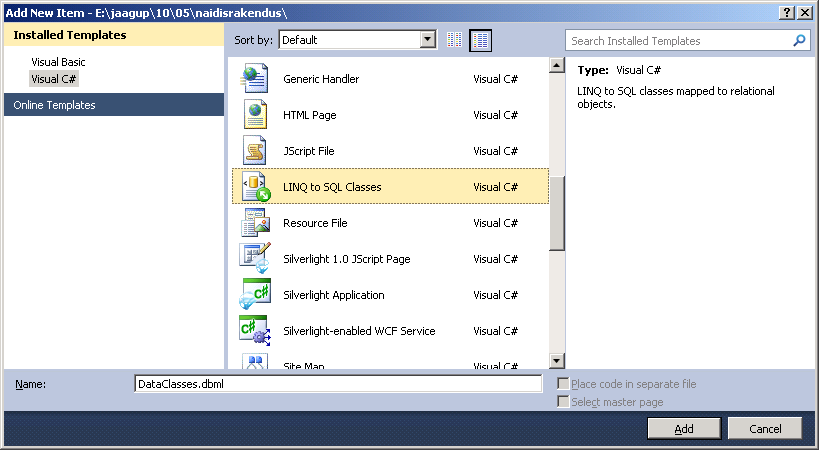


Andmete salvestus veebilehelt LINQ abil

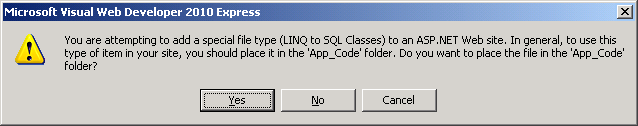
Uus LINQ-nimeline tehnoloogia annab muuhulgas mugava võimaluse programmikoodi abil andmebaasi andmeid lisada ja muidki toimetusi teha. Selleks ettevalmistuseks tuleb aga eelnevalt luua uus element



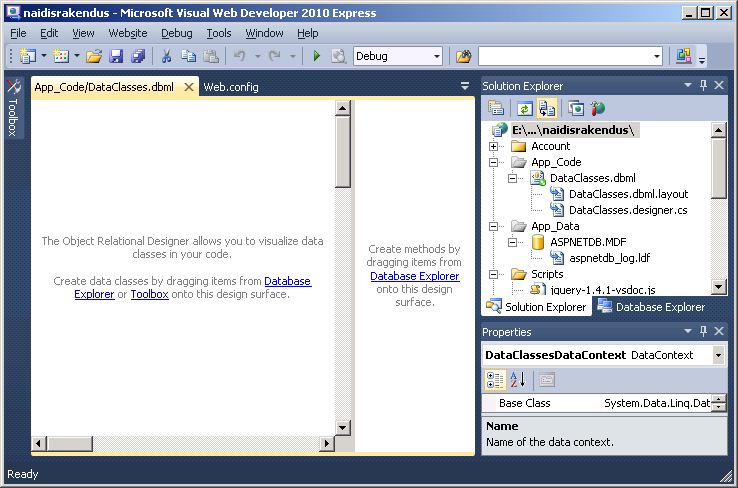
tüübist LINQ to SQL Classes. Vaikimisi pakutakse selle nimeks DataClasses.dbml, mis võib ka jääda.



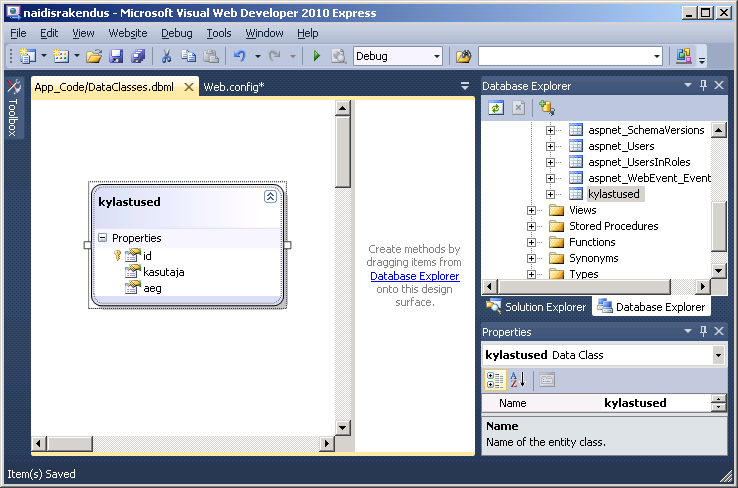
Lisaks küsitakse kas loodav klass panna App\_Code kausta – ka sellega tasub nõustuda.



Ilmub valge ala, kuhu saab need andmetabelid lohistada, milledega LINQ kaudu suhelda tahetakse.



Et meil vaid üks külastuste tabel, siis tasub ta andmetabelite loetelust suurde aknasse kohale lohistada.



Edasiseks toimetuseks vaja projekt ära kompileerida. Siin osutus vajalikuks ka korra sulgeda ja uuesti avada. LINQ to SQL tekitab automaatselt tabelite nimelised klassid. Koodi uurides: DataClassesDataContext tüüpi dc on objekt, mille kaudu andmebaasiga suhelda. Külastuste klassist luuakse uus eksemplar nimega k. Sinna kasutajanimeks pannakse sisseloginud kasutaja nimi, ehk siis HttpContext.Current.User.Identity.Name. Ajaks saab praegune aeg.

Inglise grammatika järgi paistab tabelisse lisamine lisamisena kollektsiooni „kylastuseds“. Lõpuks „submit“ ning andmed ongi kohal.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

public partial class About : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

DataClassesDataContext dc = new DataClassesDataContext();

kylastused k = new kylastused();

k.kasutaja = HttpContext.Current.User.Identity.Name;

k.aeg = DateTime.Now;

dc.kylastuseds.InsertOnSubmit(k);

dc.SubmitChanges();

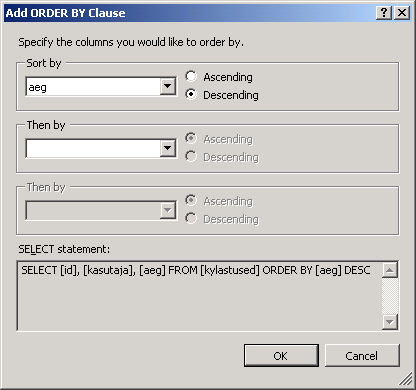
}

}

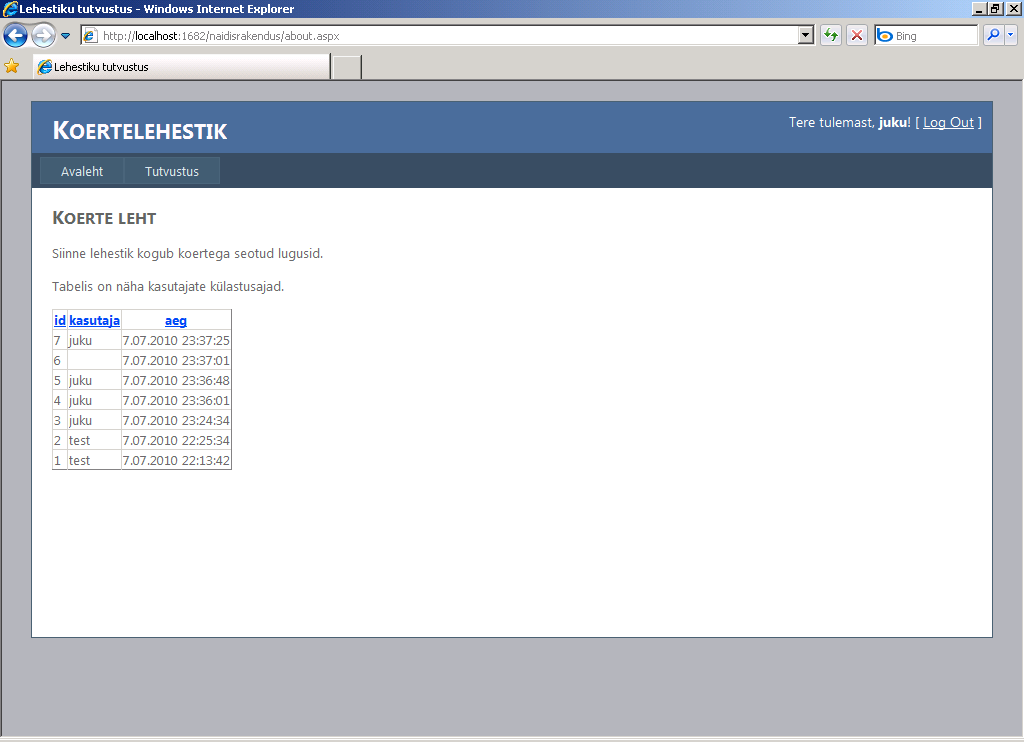
Piisab vaid sisselogitud jukuna lehe avamisest ning tema andmed ongi tabelis kirjas.



Andmebaasi administraator saab otse vaadata, teistele on avalikuks nägemiseks parem veebiliides teha. Lihtsamal juhul taas hiirega andmetabelist kinni ja võib selle disainivaatesse lohistada nagu eelnenud näites. Et viimaseid külastusi eest mugavam vaadata oleks, selleks tasub andmeallika juures teated aja järgi kahanevas järjekorras sortida – siis uuemad külastused kõigepealt.



Lehte piiludes võibki näha, kes ja millal seda külastanud on. Kui tegemist registreeritud kasutajaga, siis näha kasutajanimi. Kui anonüümse kasutajaga, siis siinsel puhul näidatakse tühja lahtrit. Aga eks rakenduse kirjutaja saab seda vabalt oma soovide järgi muuta.



Rakenduses tehtud ja tekkinud konfiguratsioonist saab aimu web.config-i nimelise faili kaudu. Veidi pikem kui tühja rakenduse juures, aga ei veel midagi lootusetult keerulist. Mõned tähelepanu väärivad lõigud.

Andmebaasi ühendavate ConnectionString’ide juures paistab, et tegelikult ühendatakse mõlema ühendusteksti kaudu sama baasiga. Lihtsalt LINQ ühenduse loomise automaatika arvas, et tal on vajalik omale lisakirje luua. Eks vajadusel saa need ka ühe nime alla kokku tõsta, aga siis rohkem koodis sättimist.

Märgend assemblies näitab, et Linq tarbeks on vastav teek juurde sisse võetud. Autentimise juures on moodiks määratud Forms. See võimaldab veebirakenduse siseselt kasutajaid hoida ja tuvastada.

Märgendi membership sees määratakse provaider (teenusepakkuja) uute kasutajate loomise tingimuste sättimiseks. ASP.NET võimaldab päris palju omadusi sättida just olemasolevate klasside parameetrite või omaloodud teenusepakkujaklasside abil.

<?xml version="1.0"?>

<!--

For more information on how to configure your ASP.NET application, please visit

http://go.microsoft.com/fwlink/?LinkId=169433

-->

<configuration>

<connectionStrings>

<add name="ApplicationServices" connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\aspnetdb.mdf;User Instance=true"

providerName="System.Data.SqlClient" />

<add name="ASPNETDBConnectionString" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\ASPNETDB.MDF;Integrated Security=True;User Instance=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

<system.web>

<compilation debug="false" targetFramework="4.0">

<assemblies>

<add assembly="System.Data.Linq, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"/></assemblies></compilation>

<authentication mode="Forms">

<forms loginUrl="~/Account/Login.aspx" timeout="2880"/>

</authentication>

<membership>

<providers>

<clear/>

<add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider" connectionStringName="ApplicationServices" enablePasswordRetrieval="false" enablePasswordReset="true" requiresQuestionAndAnswer="false" requiresUniqueEmail="false" maxInvalidPasswordAttempts="5" minRequiredPasswordLength="6" minRequiredNonalphanumericCharacters="0" passwordAttemptWindow="10" applicationName="/"/>

</providers>

</membership>

<profile>

<providers>

<clear/>

<add name="AspNetSqlProfileProvider" type="System.Web.Profile.SqlProfileProvider" connectionStringName="ApplicationServices" applicationName="/"/>

</providers>

</profile>

<roleManager enabled="false">

<providers>

<clear/>

<add name="AspNetSqlRoleProvider" type="System.Web.Security.SqlRoleProvider" connectionStringName="ApplicationServices" applicationName="/"/>

<add name="AspNetWindowsTokenRoleProvider" type="System.Web.Security.WindowsTokenRoleProvider" applicationName="/"/>

</providers>

</roleManager>

</system.web>

<system.webServer>

<modules runAllManagedModulesForAllRequests="true"/>

</system.webServer>

</configuration>

Edasine rakenduse sättimine on juba suurelt jaolt kasutaja oma teha – arenduskeskkonnaga kaasa tulnud näite põhjal saab täiesti viisaka lehestiku kokku panna.

Esmapilgul võõrkehana võib tunduda skriptikaustas peituv jQuery-failide kogum. Uurimisjulguse tõstmiseks toome siia väikese näite, mida vastav teek Javaskripti abil teha aitab. Kui väike lõik oma „valgete näppudega“ tööle saadud, siis ehk julgem ka juba pikemaid ja keerulisemaid õpetusi lugeda.

Katsetamiseks lihtne HTML-fail – ehkki vajadusel saab seda ka aspx-lehe osana kasutada. Skripti omaduste kasutamiseks tuleb too kõigepealt sisse lugeda – sedakorda Scripts-nimelisest kaustast.

Lehe elementide peale võimaldab jQuery pöörduda dollarimärgi ja sellele järgnevates sulgudes oleva määratlusega. $(document).ready()-käsklusele antakse ette kohapeal loodud funktsioon, mis käivitatakse pärast kogu dokumendi kohale jõudmist. $(„li[class=’suur’]“) märgib kõiki li-elemente, millel küljes class-atribuut väärtusega suur – sarnaselt nagu XMLi XPathis elementide poole pöördutakse, kes sellega kokku puutunud. Sinna külge sündmus click paneb testiks käima Auh-tervituse.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Skriptiproov</title>

<script src="Scripts/jquery-1.4.1.js" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function () {

$("li[class='suur']").click(function () {

alert("Auh!");

});

});

</script>

<style type="text/css">

.suur {font-weight: bold;}

</style>

</head>

<body>

<ul>

<li class="suur">Muri</li>

<li>Pauka</li>

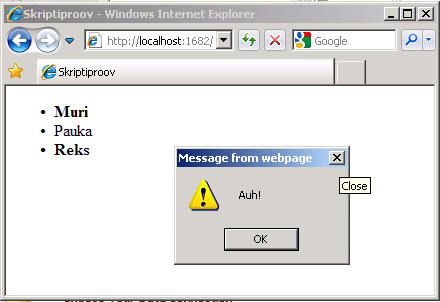
<li class="suur">Reks</li>

</ul>

</body>

</html>

Vaid suurte koerte peale vajutades tuleb nüüd ette Auh!



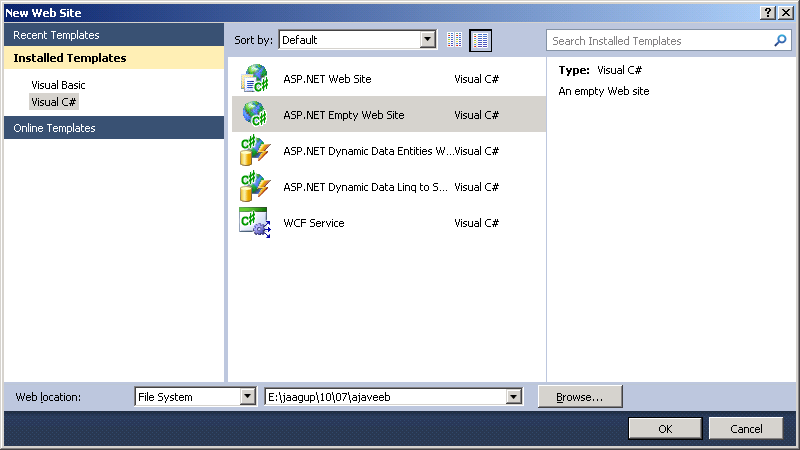
Ülesandeid

* Tee näide läbi
* Lisa andmetabelisse tulp, kus kirjas vaadatava lehe aadress. Salvesta see lehe külastamisel.
* Lisa teisele veebilehele ka külastuste logimine.
* Näita igal lehel ainult selle lehe külastusi.
* Eraldi lehel on võimalik näha kõiki külastusi ning sealt kasutaja või lehe järgi otsida.

Ajaveeb

... ehk peaaegu et reaalelus kasutatav näitrakendus. Kasutaja saab ise omale kandeid päevikusse kirja panna ning muu rahvas vaid olemasolevat vaadata.

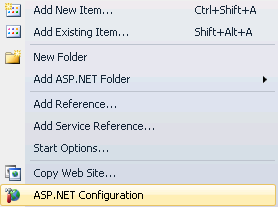
Toimuva täpsemaks jälgimiseks tasub taas tega tühi veebilehestik.



Haldamisega seonduva koodndame selle jaoks loodud eraldi kataloogi.



Rakenduse õiguse graafiliseks seadistamiseks sobib Website-menüüst avanev ASP.NET Configuration.



|  |  |
| --- | --- |
|  | Sealsete valikute alt tasub valida Security – seal kasutajate haldamisega seonduvad valikud. |

|  |  |
| --- | --- |
|  | Select Authentication Type alt võimalik valida, et ei soovi mitte Windowsi sisevõrgu kasutajatunnustega toimetada, vaid üle veebi suhtlemiseks on eraldi kasutajad ja paroolid. |

|  |  |
| --- | --- |
|  | Edasi võib juba luua rolle, õigusi ja kasutajaid. |

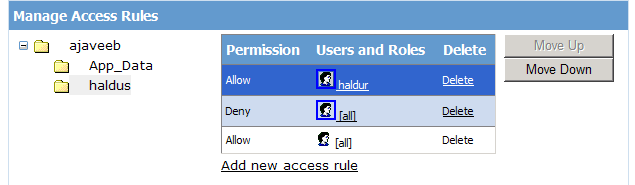
|  |  |
| --- | --- |
|  | Uueks rolliks haldur |

|  |  |
| --- | --- |
|  | Ning õiguste juurde kirja, et halduskausta muud ühendusevõtjad ei pääse |

|  |  |
| --- | --- |
|  | Haldur aga pääseb |

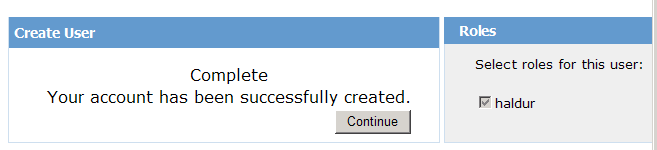
|  |  |
| --- | --- |
|  | Tähtsamad rollid tuleb liigutada ülespoole, sest vaatama hakatakse algusest ning esimese leitud vastuse peale tehakse otsus. |

Kui halduri lubav õigus kõigepealt ning ülejäänute keelav õigus selle järel, siis ongi soovitud olukord saavutatud.

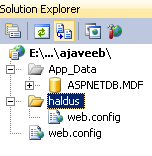


|  |  |
| --- | --- |
|  | Paljalt rollidest ei piisa – sisenemiseks peab olema ka vastava rolliga kasutaja. Siin näites juhan parooliga juhan1! – sest nõutakse ka numbrilist ja märgilist sümbolit parooli sisse. Nende piirangute muutmise näide on ülalpool kirjeldatud Visual Web Developeri enese näitrakenduse juures. |

Kui väljad sobivalt täidetud, võib lugeda rõõmusõnumit kasutaja registreerimise kohta.



Koodi poolt paistab, et juurde tekkis App\_Data kaust seal sees oleva ASPNETDB.MDF-failiga. Täiendati juurkataloogis olevat konfiguratsioonifaili ning halduskataloogi loodi oma ja uus.



web.config

Rollihaldur sees ning autentimimoodus lülitatud vaikimisi „Windows“-i pealt „Forms“ peale.

<?xml version="1.0"?>

<!--

For more information on how to configure your ASP.NET application, please visit

http://go.microsoft.com/fwlink/?LinkId=169433

-->

<configuration>

<system.web>

<roleManager enabled="true" />

<authentication mode="Forms" />

<compilation debug="false" targetFramework="4.0" />

</system.web>

</configuration>

haldus/web.config

Halduskaustas olevas konfiguratsioonifailis on kirjas, et halduri rolliga kasutajad pääsevad sisse, teised mitte.

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<system.web>

<authorization>

<allow roles="haldur" />

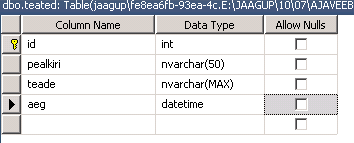
<deny users="\*" />

</authorization>

</system.web>

</configuration>

Esialgsed turvatoimetused tehtud, võib edasi andmete juurde minna. Aluseks ajaveebile andmetabel tulpadega id, pealkiri, teade ja aeg. Viimasele võib taas vaikeväärtuseks määrata getdate(). Ning id-le omaduseks identity, et loendur automaatselt töötaks.



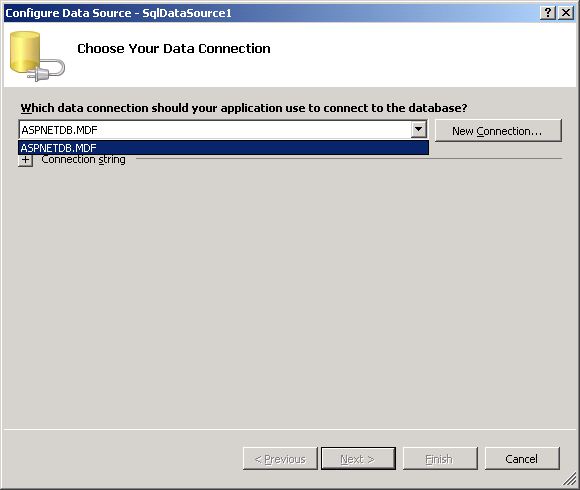
|  |  |
| --- | --- |
|  | Sedakorda ei lohista me andmetabelit otse lehele, vaid käime etapid sammude kaupa läbi ning kõigepealt toome andmekomponentide juurest lehele andmeallika. SQL Serveri jaoks on loodud SqlDataSource. |

. Alustuseks tuleb Toolboxist andmeallikas oma lehele vedada. Tulemusena ilmub ta Visual Web Developeri lehele omaette elemendina. Veebilehe tarvis on aga tegemist nähtamatu elemendiga, st. lehitsejas avades andmeallikat eraldi näha pole.

j

|  |  |
| --- | --- |
|  | Graafilise osa juurest saab allikat ka edasi konfigureerida. Allika peale hiirega minnes ilmub nähtavale väike kolmnurk. Sealt tasub valida „Configure Data Source“. |

Praegu ainukeseks andmefailiks ongi App\_Data kaustas olevas andmebaasifailis ASPNETDB.MDF. Selle valimegi.



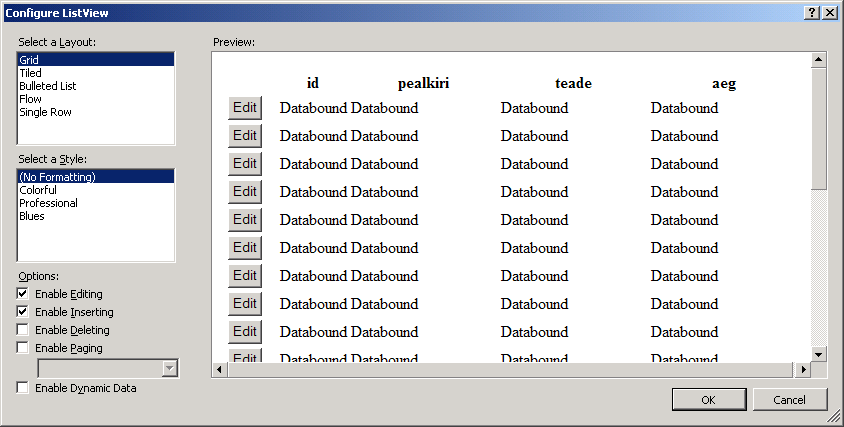
Edasi pakutakse andmeühenduse seadistuste salvestuse võimalust. Et oleks võimalik edasi rakendust hiirevalikute abil koostada, tuleb nad salvestada. Samuti on võimalik nõnda sama ühendust tulevikus mujal kasutada ilma, et peaks taas valima ja sättima hakkama. Siin näites on tegemist ühe failiga rakenduse kataloogis ning seetõttu ligipääs lihtne.

|  |  |  |
| --- | --- | --- |
|  | Mõnikord võib aga andmetele ligipääsuks olla vaja läbida õiguste ja võrgupõhine rägastik. Sel juhul ühenduse andmete korduvkasutus eriti vajalik. Samuti annab ühenduse andmete salvestamine võimaluse hiljem muuta andmebaasi asukohta ilma, et peaks selleks kõiki rakenduses leiduvaid andmebaasiga suhtlevaid lehti eraldi ümber tegema. Nimeks pakutakse praegu lihtsalt ConnectionString. No las ta olla – ehkki keerulisemate rakenduste puhul on vahel kasulik nimi määrata selle järgi, kuhu parajasti ühenduda tahetakse. | |
|  | | Järgmise sammuna uuritakse, et milliseid andmeid soovitakse vaadata. Et baasis on esialgu ainult üks tabel, pole valida kuigi paljude kohtade vahel. Ning kui tulbanimede alt valida tärn, siis tähendab see, et soovime näha kõikide tulpade andmeid. |

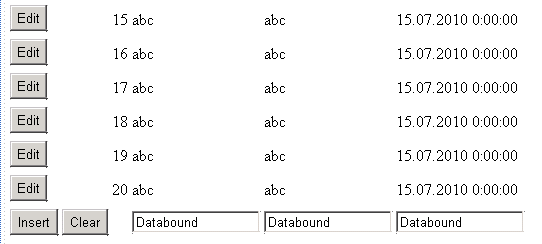
|  |  |
| --- | --- |
|  | Tahtes andmeid lisada, muuta ja kustutada, tuleb vastav linnuke advanced-sektsioonist sisse lülitada. |

Andmetega ümber käimiseks kasutame siin lehel ListView-d. See võimaldab täpselt HTMLi järgi paika sättida, et mida ja kuidas näidatakse. Saab ta nagu ikka tööriistade juurest Data-sektsioonist.

Ülalt avanevast kolmnurkast saab omadusi sättida. Kõigepealt ListView juurde kuuluvaks andmeallikaks toosama SqlDataSource1.



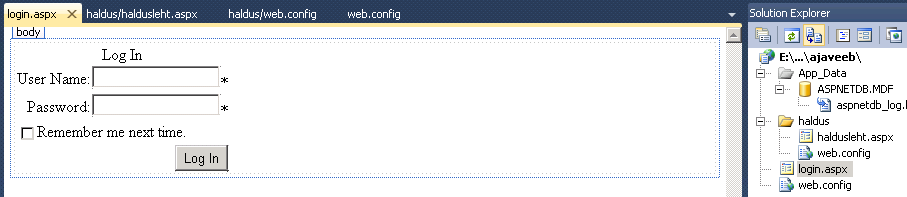
Configure alt tuleb määrata, et millisel kujul andmeid näha soovitakse – vastavalt sellele koostatakse HTMLi põhi. Kasutame sedakorda tabelit ehk Grid-i. Nii genereeritakse kood andmete nägemiseks ja sisestamiseks.

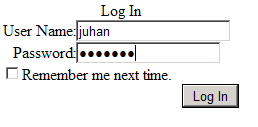


Lehe vaatamisel tuleb hoiatus, et puudub järgnev fail:

**Requested URL:** /ajaveeb/login.aspx

Kuna halduskaust sai parooliga kaitstud, siis ligipääsu puudumisel saadetakse automaatselt juurkausta login.aspx-nimelist faili otsima. See tasub siis tekitada. Ning sinna sisse element nimega Login. Üheskoos eelnevalt genereeritud ASPNETDB.MDF-iga suudab see element automaatselt kasutajaid tuvastada.

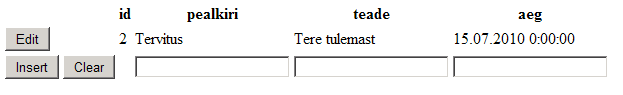




Kui nüüd andmete sisestusleht käivitada, siis küsitakse automaatselt kasutajanime ja parooli ning alles edasi suunatakse halduslehele. Seal nõutakse esialgu pealkirja, teate ning aja sisestamist.

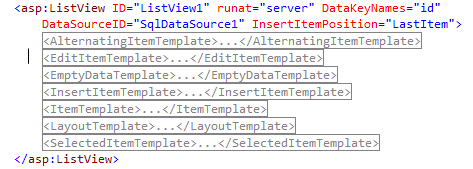


Nupule vajutamise järel võib veenduda, et andmed ka kohale jõudsid.



Et aga aeg võiks automaatselt kohale jõuda nagu andmetabeli loomisel tellitud, siis tasub lehe ülesehitust lähemalt uurida.

ListView puhul saab üksikute mallide ehk Templatedega ära määrata, kuidas midagi veebilehel paistab. Vaatamiseks mõeldud ItemTemplate. Üle rea värvimiseks AlternatingItemTemplate. Lisamiseks InsertItemTemplate. LayoutTemplate annab andmestiku näitamise üldise kuju – näiteks selle, et tabel algab ja lõpeb. Ka muudest elementidest võiks nime järgi aru saada, millega tegemist. Edit seondub muutmisega, Empty andmete puudumisega. SelectedItemTemplate võimaldab ühe kirje andmeid lähemalt vaadata.



Andmeallika juures on näha, et SQL-lauses sisestatakse andmed tabelisse teated tulpadesse pealkiri, teade ning aeg. Viimast aga vaikeväärtuse tõttu poleks vaja sisestada.

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:ConnectionString %>"

InsertCommand="INSERT INTO [teated] ([pealkiri], [teade], [aeg]) VALUES (@pealkiri, @teade, @aeg)"

SelectCommand="SELECT \* FROM [teated]"

Sama lugu InsertParams-loendi juures. Aeg on tegelikult üleliigne.

<InsertParameters>

<asp:Parameter Name="pealkiri" Type="String" />

<asp:Parameter Name="teade" Type="String" />

<asp:Parameter Name="aeg" Type="DateTime" />

</InsertParameters>

Elementide omadused seotakse andmebaasiparameetritega Bind-käskluse abil.

<InsertItemTemplate>

<tr style="">

<td>

<asp:Button ID="InsertButton" runat="server" CommandName="Insert"

Text="Insert" />

<asp:Button ID="CancelButton" runat="server" CommandName="Cancel"

Text="Clear" />

</td>

<td>

&nbsp;</td>

<td>

<asp:TextBox ID="pealkiriTextBox" runat="server"

Text='<%# Bind("pealkiri") %>' />

</td>

<td>

<asp:TextBox ID="teadeTextBox" runat="server" Text='<%# Bind("teade") %>' />

</td>

<td>

<asp:TextBox ID="aegTextBox" runat="server" Text='<%# Bind("aeg") %>' />

</td>

</tr>

</InsertItemTemplate>

Kuna aega pole enam parameetrina vaja, võib selle lahtri andmed sidumata jätta.

<td>

Praegune aeg

</td>

Samuti saab aja välja lükata SQL-lausest.

InsertCommand="INSERT INTO [teated] ([pealkiri], [teade]) VALUES (@pealkiri, @teade)"

ning parameetritest.

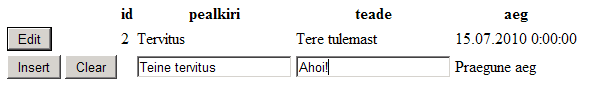
<InsertParameters>

<asp:Parameter Name="pealkiri" Type="String" />

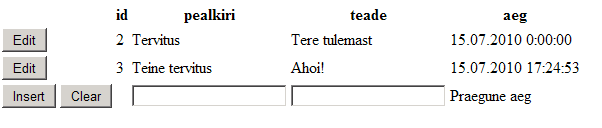
<asp:Parameter Name="teade" Type="String" />

</InsertParameters>

Sellise muutuse peale kaob sisestusvormist aja lahter.



Ning pärast sisestamist võib näha kellaaega salvestununa.



Andmete vaatamine

ListView võimaldab ka vaatamise oma soovide järgi kohandada. Kui muid toimetusi ei soovita, piisab andmete näitamisest ItemTemplate-st. Siinsel juhul pealkiri <h2></h2> vahele ning teate sisu div-elemendi sisse.

<%@ Page Language="C#" %>

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

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Teadete loetelu</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:ConnectionString %>"

SelectCommand="SELECT \* FROM [teated]"></asp:SqlDataSource>

<asp:ListView ID="ListView1" runat="server" DataSourceID="SqlDataSource1">

<ItemTemplate>

<h2><asp:Label ID="pealkirjasilt" runat="server" Text='<%# Bind("pealkiri") %>'></asp:Label></h2>

<div><asp:Label ID="Label1" runat="server" Text='<%# Bind("teade") %>'></asp:Label></div>

</ItemTemplate>

</asp:ListView>

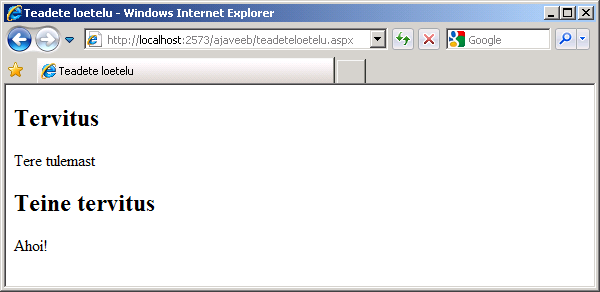
</div>

</form>

</body>

</html>

Võib imetleda tekkinud tulemust.



Ka HTML on meeldivalt loetav. Vaid ViewState on ASP.NETi siseseks andmete ülekandeks, muu koodi sisu pääseb täiesti vabalt sättima ja kujundama.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head><title>

Teadete loetelu

</title></head>

<body>

<form method="post" action="teadeteloetelu.aspx" id="form1">

<div class="aspNetHidden">

<input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE" value="/wEPDwUKMTg2NzU0NzA0OA9kFgICBA9kFgICAw8UKwACDxYEHgtfIURhdGFCb3VuZGceC18hSXRlbUNvdW50AgJkZBYEZg9kFgQCAQ8PFgIeBFRleHQFCFRlcnZpdHVzZGQCAw8PFgIfAgUNVGVyZSB0dWxlbWFzdGRkAgEPZBYEAgEPDxYCHwIFDlRlaW5lIHRlcnZpdHVzZGQCAw8PFgIfAgUFQWhvaSFkZBgBBQlMaXN0VmlldzEPFCsADmRkZGRkZGQUKwACZGQCAmRkZGYC/////w9kVLKSoRqn1MrZC+35ky9yWWWfyRqUOc6cFj/K4Vg3B/Y=" />

</div>

<div>

<h2><span id="ListView1\_pealkirjasilt\_0">Tervitus</span></h2>

<div><span id="ListView1\_Label1\_0">Tere tulemast</span></div>

<h2><span id="ListView1\_pealkirjasilt\_1">Teine tervitus</span></h2>

<div><span id="ListView1\_Label1\_1">Ahoi!</span></div>

</div>

</form>

</body>

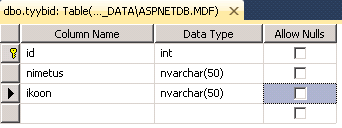
</html>

Andmed mitmes tabelis.

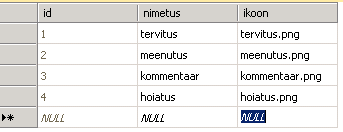
Andmebaasipõhise lehe koostamisel tasub eri tüüpi andmed eraldi tabelites hoida. Kui tabelid pole omavahel seotud, siis saab ka andmeid rahumeeli eraldi lisada-näidata-muuta. Võib toimetada nagu eelnenud näidetes, vaid erisuguseid lehti ja tabeleid tuleb juurde.

Põhjalikumates kohtades hakatakse aga tabeleid siduma. Lihtsamal juhul kaks tabelit omavahel kokku, kuid juba küllalt tavalise igapäevaelus ette tuleva andmestiku puhul võib parvena koos olevaid tabeleid olla mituteist või mitukümmend. Siin aga teeme kahe tabeliga tutvust – eks hiljem saab neid vajadusel juurde lisada.

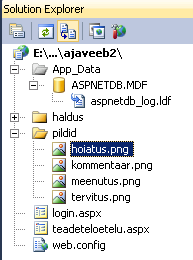
Lisame ajaveebi teadetele tüübid. Tabel tyybid, tulpadeks id, nimetus ja ikoon ehk vastava tüübi pildifaili nimi.



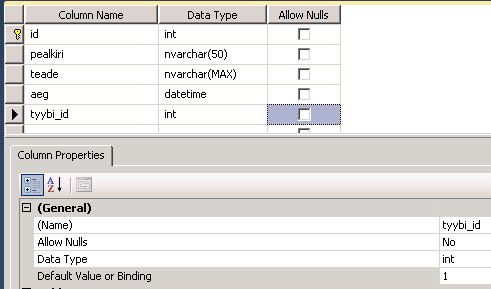
Mõned andmed sisse ka.



Pildid ise omaette kausta, et mugavam rakenduse sees asju korras hoida oleks.



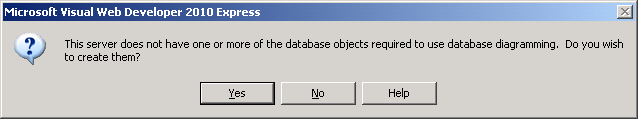
Tabelite sidumiseks peab neil olema ühine väli. Siinsel juhul lisame teadete tabelisse tulba nimega tyybi\_id milles oleva numbri järgi saab vaadata vastava teate tüüpi tüüpide tabelist. Vaikimisi väärtuseks sai siin arv 1, mis tähendab, et teade on tervitus juhul, kui midagi muud määratud ei ole.



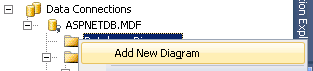
Nõnda on olemasolevatel teadetel kohe tüübid küljes.



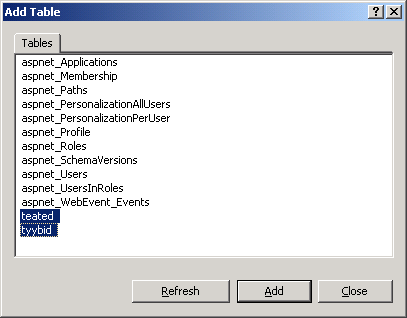
Andmetabelite vahelised seosed on viisakas kirja ka panna. Nõnda, et poleks võimalik vigaseid seoseid tekitada, vaid, et andmebaasimootor kontrollib, kas viidatav rida ka tabelis kindlasti olemas on. Siin keskkonnas on seoseid mugav luua diagrammide abil. Esmakordsel juhul küsitakse, kas vastav tükk sisse lülitada.



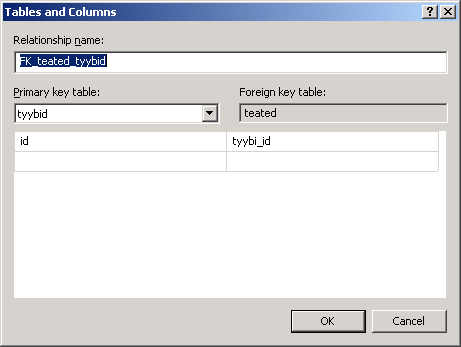
Edasi luua diagramm



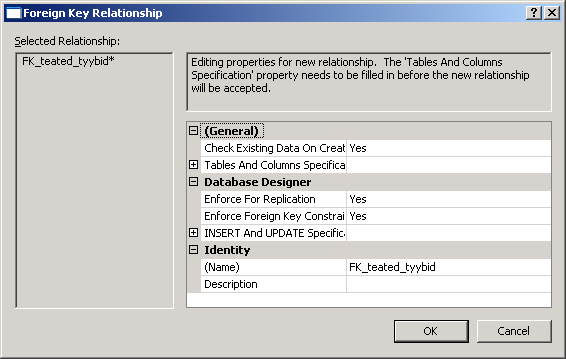
Valida sobivad tabelid



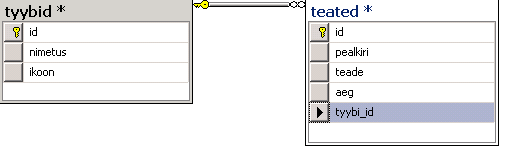
Kui tabelid nähtaval, siis lohistame teadete tabeli tyybi\_id tüüpide tabeli id peale.



Selle peale küsitakse, et kas soovime luua seost, kus teadete tabeli tyybi\_id on võõrvõtmeks ning tüüpide tabeli id primaarvõtmeks. Paistab, et mõistlik on nõustuda.



Küsitakse veel üle, et kas seoste vastavust kontrollitakse – ka sellega tasub nõustuda. Tulemusena võiks näha olla üks-mitmele seos, kus tüübile võib vastata määramata arv teateid. Igal teatel on aga üks kindel tüüp.



Teate lisamisel oleks nüüd tüüp kasulik olemasolevate hulgast valida. Et need tüübid kätte saada, tasub luua uus anmeallikas vastava päringuga.

<asp:SqlDataSource runat="server" ID="SqlDataSource2" ConnectionString="<%$ ConnectionStrings:ConnectionString %>"

SelectCommand="SELECT id, nimetus FROM tyybid">

</asp:SqlDataSource>

ListView InsertItemTemplate’t tasub sobivalt kohandada. Näitest paistab, et pealkirja ja teate sisestamiseks on tekstikastid, mille Text-nimeline omadus on Bind-käsuga sobiva SQLi parameetri külge seatud. Kuna aja saab vaikimisi väärtuse abil automaatselt lisatavaks, siis võib selle parameetrite hulgast maha võtta. Ning tüübi valimiseks sobib DropDownList. Andmeallikana kasutatakse selleks otstarbeks loodud SqlDataSource2-elementi. Kasutajale näidatakse tüüpide tabelist nimetust, tegelikult serverisse saadetakse aga id-tulba väärtus, mis teadete tabelisse tyybi\_id alla kirja pannakse.

<InsertItemTemplate>

<tr style="">

<td>

<asp:Button ID="InsertButton" runat="server" CommandName="Insert"

Text="Lisa" />

</td>

<td>

&nbsp;</td>

<td>

<asp:TextBox ID="pealkiriTextBox" runat="server"

Text='<%# Bind("pealkiri") %>' />

</td>

<td>

<asp:TextBox ID="teadeTextBox" runat="server" Text='<%# Bind("teade") %>' />

</td>

<td>

Praegune aeg

</td>

<td>

<asp:DropDownList id="tyybivalik" runat="server" DataSourceID="SqlDataSource2" DataTextField="nimetus" DataValueField="id"

SelectedValue='<%# Bind("tyybi\_id") %>'></asp:DropDownList>

</td>

</tr>

</InsertItemTemplate>

Et vastav väärtus ka tabelisse jõuaks, peab see olema SqlDataSource lisamisparameetrite hulgas.

<InsertParameters>

<asp:Parameter Name="pealkiri" Type="String" />

<asp:Parameter Name="teade" Type="String" />

<asp:Parameter Name="tyybi\_id" Type="Int32" />

</InsertParameters>

Ning loomulikult ka Insert-lause enese sees.

InsertCommand="INSERT INTO [teated] ([pealkiri], [teade], [tyybi\_id]) VALUES (@pealkiri, @teade, @tyybi\_id)"

Muutmiseks tuleb hoolitseda, et vanad väärtused lehe vaatajale näha oleksid. Tekstiväljadesse kirjutatakse need otse Bind-i abil. Kuid õnneks suudab ka rippmenüü SelectedValue kaudu sobiva väärtuse enesele ette kerida.

Teate tüübile vastava pildi ekraanile saamiseks tuleb viimase failinimi tüüpide tabelist küsida ning pildi näitamiseks kataloogitee sinna ette panna.

<asp:Image ID="pilt1" runat="server" ImageUrl='<%# "~/pildid/"+Eval("ikoon") %>' />

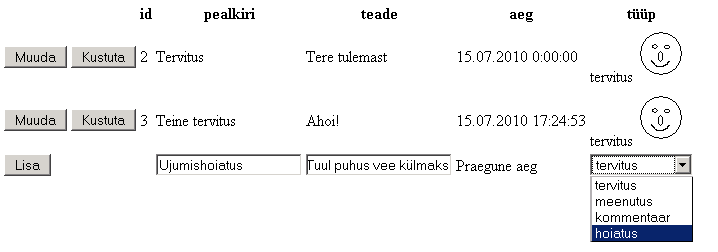
Ikooni failinime näitamiseks tuleb see ka Select-lausega küsida.

SelectCommand="SELECT teated.id, pealkiri, teade, aeg, tyybi\_id, nimetus, ikoon FROM teated, tyybid

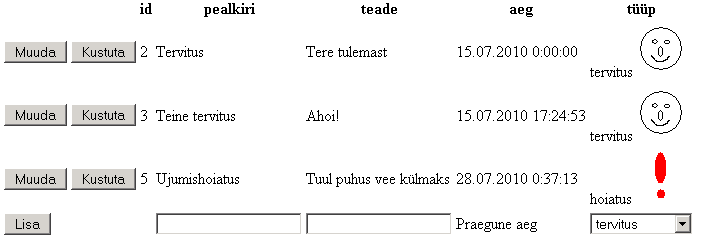
WHERE teated.tyybi\_id=tyybid.id"

Edasi halduslehe toimimine piltkirjas.

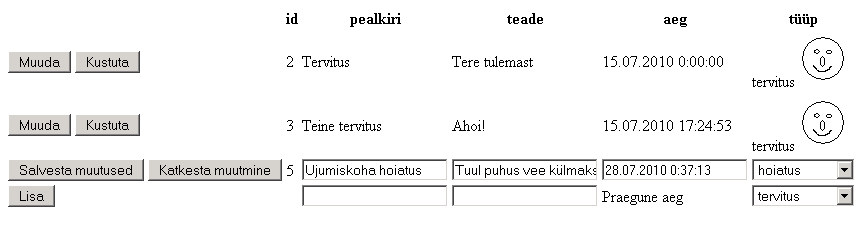
Andmete sisestus



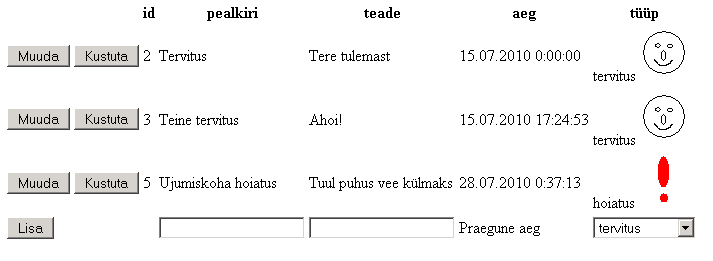
Kohale jõudnud andmed



Muutmine



Muutmise salvestus.



Ning lehe kood tervikuna

<%@ Page Language="C#" %>

<!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>Teadete haldus</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:ConnectionString %>"

DeleteCommand="DELETE FROM [teated] WHERE [id] = @id"

InsertCommand="INSERT INTO [teated] ([pealkiri], [teade], [tyybi\_id]) VALUES (@pealkiri, @teade, @tyybi\_id)"

SelectCommand="SELECT teated.id, pealkiri, teade, aeg, tyybi\_id, nimetus, ikoon FROM teated, tyybid

WHERE teated.tyybi\_id=tyybid.id"

UpdateCommand="UPDATE [teated] SET [pealkiri] = @pealkiri, [teade] = @teade, [aeg] = @aeg, tyybi\_id=@tyybi\_id WHERE [id] = @id">

<DeleteParameters>

<asp:Parameter Name="id" Type="Int32" />

</DeleteParameters>

<InsertParameters>

<asp:Parameter Name="pealkiri" Type="String" />

<asp:Parameter Name="teade" Type="String" />

<asp:Parameter Name="tyybi\_id" Type="Int32" />

</InsertParameters>

<UpdateParameters>

<asp:Parameter Name="pealkiri" Type="String" />

<asp:Parameter Name="teade" Type="String" />

<asp:Parameter Name="aeg" Type="DateTime" />

<asp:Parameter Name="id" Type="Int32" />

<asp:Parameter Name="tyybi\_id" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

<asp:SqlDataSource runat="server" ID="SqlDataSource2" ConnectionString="<%$ ConnectionStrings:ConnectionString %>"

SelectCommand="SELECT id, nimetus FROM tyybid">

</asp:SqlDataSource>

</div>

<asp:ListView ID="ListView1" runat="server" DataKeyNames="id"

DataSourceID="SqlDataSource1" InsertItemPosition="LastItem">

<EditItemTemplate>

<tr style="">

<td>

<asp:Button ID="UpdateButton" runat="server" CommandName="Update"

Text="Salvesta muutused" />

<asp:Button ID="CancelButton" runat="server" CommandName="Cancel"

Text="Katkesta muutmine" />

</td>

<td>

<asp:Label ID="idLabel1" runat="server" Text='<%# Eval("id") %>' />

</td>

<td>

<asp:TextBox ID="pealkiriTextBox" runat="server"

Text='<%# Bind("pealkiri") %>' />

</td>

<td>

<asp:TextBox ID="teadeTextBox" runat="server" Text='<%# Bind("teade") %>' />

</td>

<td>

<asp:TextBox ID="aegTextBox" runat="server" Text='<%# Bind("aeg") %>' />

</td>

<td>

<asp:DropDownList id="tyybivalik" runat="server" DataSourceID="SqlDataSource2" DataTextField="nimetus" DataValueField="id"

SelectedValue='<%# Bind("tyybi\_id") %>'></asp:DropDownList>

</td>

</tr>

</EditItemTemplate>

<EmptyDataTemplate>

<table runat="server" style="">

<tr>

<td>

Andmed puuduvad</td>

</tr>

</table>

</EmptyDataTemplate>

<InsertItemTemplate>

<tr style="">

<td>

<asp:Button ID="InsertButton" runat="server" CommandName="Insert"

Text="Lisa" />

</td>

<td>

&nbsp;</td>

<td>

<asp:TextBox ID="pealkiriTextBox" runat="server"

Text='<%# Bind("pealkiri") %>' />

</td>

<td>

<asp:TextBox ID="teadeTextBox" runat="server" Text='<%# Bind("teade") %>' />

</td>

<td>

Praegune aeg

</td>

<td>

<asp:DropDownList id="tyybivalik" runat="server" DataSourceID="SqlDataSource2" DataTextField="nimetus" DataValueField="id"

SelectedValue='<%# Bind("tyybi\_id") %>'></asp:DropDownList>

</td>

</tr>

</InsertItemTemplate>

<ItemTemplate>

<tr style="">

<td>

<asp:Button ID="EditButton" runat="server" CommandName="Edit" Text="Muuda" />

<asp:Button ID="DeleteButton" runat="server" CommandName="Delete" Text="Kustuta" />

</td>

<td>

<asp:Label ID="idLabel" runat="server" Text='<%# Eval("id") %>' />

</td>

<td>

<asp:Label ID="pealkiriLabel" runat="server" Text='<%# Eval("pealkiri") %>' />

</td>

<td>

<asp:Label ID="teadeLabel" runat="server" Text='<%# Eval("teade") %>' />

</td>

<td>

<asp:Label ID="aegLabel" runat="server" Text='<%# Eval("aeg") %>' />

</td>

<td>

<asp:Label ID="Label1" runat="server" Text='<%# Eval("nimetus") %>' />

<asp:Image ID="pilt1" runat="server" ImageUrl='<%# "~/pildid/"+Eval("ikoon") %>' />

</td>

</tr>

</ItemTemplate>

<LayoutTemplate>

<table ID="itemPlaceholderContainer" runat="server" border="0" style="">

<tr runat="server" style="">

<th runat="server">

</th>

<th runat="server">

id</th>

<th runat="server">

pealkiri</th>

<th runat="server">

teade</th>

<th runat="server">

aeg</th>

<th runat="server">tüüp</th>

</tr>

<tr ID="itemPlaceholder" runat="server">

</tr>

</table>

</LayoutTemplate>

</asp:ListView>

</form>

</body>

</html>

Ülesandeid

* Tee näide läbi
* Lisa juurkataloogi kasutaja jaoks teadete vaatamise leht

## Edasijõudnutele

### Veebi kopeerimine

ASP.NET veebirakendused on lihtsalt teisaldatavad. Nende liigutamiseks uude serverisse on vaja kopeerida kõik veebi jaoks vajalikud failid uude asukohta ning ongi veeb uues kohas.

Selle lihtsa tegevuse veelgi lihtsamaks muutmiseks on Visual Sudiosse lisatud veebi kopeerimise vahend, mille leiate menüüst „Website/Copy Web Site…“

Veebi kopeerimise vahend võimaldab teil veebi kopeerida nelja erinevat tüüpi sihtkohta. Ühenduse loomiseks oma veebi sihtkohaga vajutage nupule Connect ning valige veebi sihtkoht:

* File System – veebi asukoht on ligipääsetav failisüsteemi kaudu (nagu teeks windowsis copy/paste teise kausta)
* Local IIS – sama arvuti IIS Serveris[[1]](#footnote-1) paiknev veebisait
* FTP Site – FTP[[2]](#footnote-2) ligipääsuga veebisait.
* Remote Site – HTTP[[3]](#footnote-3) või HTTPS[[4]](#footnote-4) abil ligipääsetav veeb. Ligipääs toimub üle tavalise veebikanali nagu ka veebi vaatamine, kuid lisaks peab teil olema veebi kirjutamise õigus. Üheks heaks näiteks selles vallas on oma veebi kopeerimine SharePoint[[5]](#footnote-5) portaali.

### Programmi koodi paigutamine eraldi faili

Kuigi veebirakendus lubab kirjutada koodi ja lehekülje märgistuse samasse faili, on enamasti mõistlik need kaks eraldada. Põhjus seisneb selles, et üldjuhul ei ole programmeerijad head disainerid ja ka vastupidi: hea disainer ei ole hea programmeerija. Jagades rakenduses lehekülje disaini ja koodi eraldi, saavad mõlemad tegelda just sellega, mis kõige paremini välja tuleb.

Seega kui teeme lihtsat veebirakendust, tuleks luua tegelikult kaks faili: default.aspx ja default.aspx.cs.

Faili default.aspx sisu võiks olla järgmine:

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

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

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title>Minu esimene leht</title>

</head>

<body>

<form id="form1" runat="server">

Tere maailm

</form>

</body>

</html>

Võrreldes esimese näitega on Page elemendile lisandunud kaks atribuuti.

* CodeFile atribuut – ütleb kompilaatorile, millisesse faili on salvestatud lehega seotud kood.
* Inherits atribuut – ütleb, millisesse klassi koodifailis on kood paigutatud

Lisaks .aspx failile tuleb nüüd luua ka page elemendis kirjeldatud koodifail (kuna kirjutame C# keeles peaks see olema .cs laiendiga), koos vastava klassiga. Kuna hetkel, meie lehel mingit erilist dünaamikat ei ole sisaldab loodav default.cs fail vaid klassi kirjeldust:

public partial class \_Default : System.Web.UI.Page {}

Nagu ülal olevast näites näha pärinevad kõik ASP.NET lehed System.Web.UI.Page klassist, kus on realiseeritud raamistik, mis vajalik lehe genereerimiseks ning kontrollide kasutamiseks.

Tegelikult on üsna uskumatu, kui palju on võimalik ära teha deklaratiivselt ilma koodi e. .cs faili poole pöördumata. Samas on enam kui selge, et kui funktsionaalsus läheb natukenegi keerukamaks, tuleb ka programselt üht-teist korda saata.

Näiteks sellel samal lihtsal lehel on võimalik kirjutada kogu tekst dünaamiliselt koodi abil. Selleks eemaldame teksti .aspx failist:

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

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

<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

<title />

</head>

<body>

<form id="form1" runat="server">

<asp:Label ID="lblTere" runat="server" />

</form>

</body>

</html>

Ning lisame selle koodi abil:

using System;

public partial class \_Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

Page.Title = "Minu esimene leht";

lblTere.Text = "Tere maailm!";

}

}

Page\_Load on sündmus, mis tekib iga kord lehekülje laadimisel. Selleks, et uurida genereeritava asp.net lehe omadusi saame ära kasutada Page objekti.

Staatilise teksti kirjutamiseks on kõige mugavam kasutada lipikuid (asp:Label). See võimaldab hiljem lihtsa vaevaga teksti kujundada. Iga objekt, mille poole soovite koodis pöörduda peab omama unikaalset ID’ d! Teksti saab lipikule lisada läbi Text atribuudi. Lipikud genereeruvad enamasti SPAN html tagideks.

Läbi kontrollide teksti kirjutamisel on üks oluline eelis võrreldes staatilise tekstiga – kontrollidel olevat teksti on lihtne koodis kasutada ning seda on võimalik lihtsa vaevaga lokaliseerida! Seega on alati soovitav kirjutada teksti veebilehele kas asp:label või mõne muu serveri kontrolli kaudu.

Ülesandeid

\* Installeeri Visual Web Developer või leia enesele ligipääs IIS serverile

\* Koosta ja käivita tervitav leht

\* Määra lehel olev tervitus programmijupi abil.

\* Küsi inimeselt nimi ja nupuvajutuse peale tervita teda, kopeerides tekst tekstiväljast sildile

### Seadistamine (Web.config)

Igal veebirakendusel peaks olema ka oma seadistuste fail. ASP.NET rakenduse seadistust on võimalik seadistada üsna mitmest kohast. Alustades raamistiku üldseadistustest machine.config (paikneb .NET raamistiku programmifailide hulgas), seejärel veebisaidi seadistus e. web.config fail juurveebis ning lõpetades veebsaidi juurkataloogis oleva web.config failiga. Lisaks on võimalik erinevatele alamkaustadele eriseadistuse andmiseks võimalik lisada igasse kausta oma web.config fail.

Lõpuks jäävad kehtima need määrangud, mis on leheküljele kõige lähemal! Kuna masina seadistusele ja veebisaidi seadistusele meil enamasti ligipääsu ei ole siis püüame kogu vajaliku seadistuse ära teha veebirakenduse juurkataloogis oleva web.config faili abil.

.config failid on XML failid, milles antakse lisainformatsiooni nii rakendusele endale kui ka .NET raamistikule ja kompilaatorile.

Kõik väärtused/muutujad, mida soovite, et programmi kasutaja/administraator saaks hiljem lihtsalt muuta tuleks sisestada web.config faili!

Esialgu võiks web.config fail näha välja järgmine:

<?xml version="1.0"?>

<configuration>

<system.web>

<compilation debug="true" />

</system.web>

</configuration>

Selle seadistusega ütleme, et meil on kõik nii nagu serveris ikka, kuid me soovime oma rakenduse juures näha lisainfot silumiseks. Kui programm saab valmis, peaks panema debug atribuudi väärtuseks ”false”. Silumisinfo olemasolul kuvatakse teile vigade tekkimisel konkreetsed koodiread.

ASP.NET veebi seadistus on mitmekihiline ning esmased seadistused teie veebisaidile võetakse masina (veebiserveri) seadistustest. Seejärel lisandub juurveebi seadistus ning alles seejärel jõutakse teie veebi seadistusteni.

Lisaks oma veebi juurkataloogis olevale seadistusele võite teatud seadeid muuta ka alamkaustades. Selleks on teil kaks võimalust:

* Lisada alamkasuta eraldi web.confog fail, kus on vastavad seaded ümber tehtud
* Lisada veebi juurkataloogis olevasse web.config faili location element, ning näidata ära, mida soovite ümber teha. Sellekohast näidet saad vaadata leheküljel 470.

### Rakenduse jälgimine (Trace)

Nii ASPi kui ka PHP suur puudus on see, et programmi silumiseks või vea leidmiseks vajaliku metainfo kuvamine on väga keeruline. Enamasti lõppeb see metainfo kirjutamisega otse lehele, rikkudes sellega ära lehekülje üldise väljanägemise.

ASP.NETil on selle tarbeks märksa mugavam vahend – Trace. Selleks, et jälgida ühel lehel toimuvat on kõige lihtsam Page elemendis sisse lülitada Trace atribuut.

<%@ Page Language="C#"

CodeFile="Default.aspx.cs"

Inherits="\_Default" ‚

Trace ="true" %>

Selle ühe väikse atribuudi lisamisega lisatakse lehekülje lõppu terve hulk informatsiooni alustades sellest, mis moodi seda lehekülge küsiti ning kirjeldades ära kogu lehekülje loomise protseduuri (koos lehekülje struktuuriga) ning lõpetades loeteludega erinevatest muutujatest (sessioon, programm, server jne).



Loomulikult on võimalik sinna abiinfosse lisada ka oma märkuseid. Jälitusinfo kuvamiseks ja uurimiseks on Trace klass. Jälitus infot kirjutada saab kahe meetodiga:

* Trace.Write – lihtsalt jutt jälitusinfosse
* Trace.Warn – hoiatus (kuvatakse jälitusinfot vaadates punasena)

Lisame näiteks pisut jälitusinfot oma default.cs faili:

using System;

public partial class \_Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

Trace.Write("Alustan Page\_Load meetodiga ...");

Page.Title = "Minu esimene leht";

lblTere.Text = "Tere maailm!";

Trace.Warn("Meetod lõpetatud");

}

}

|  |  |
| --- | --- |
| Tulemuseks on kaks uut rida Traces, Begin Load ja End Load sündmuste vahel: |  |

Kui soovite mingi sündmuse ilmnemisel natukene põhjalikumat analüüsi teha siis on kasulik enne analüüsi alustamist kontrollida, kas jälitus on üldse sisse lülitatud. Selleks saame uurida Trace objekti IsEnabled omadust. Väga kasulik nt Try ... Catch konstruktsioonides:

protected void Page\_Load(object sender, EventArgs e)

{

try

{

object o = null;

Trace.Write("Alustan Page\_Load meetodiga ...");

Trace.Warn("Tekitan vea", o.ToString());

Page.Title = "Minu esimene leht";

lblTere.Text = "Tere maailm!";

}

catch (Exception ex)

{

if (Trace.IsEnabled)

{

Trace.Warn("error", "Tekkis tõrge", ex);

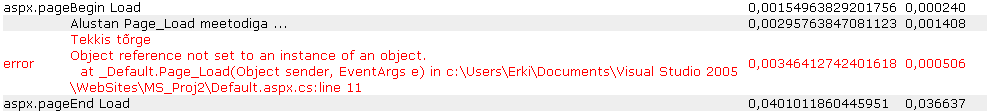
// TODO: Siia tuleks teha analüüs, miks viga tekkis

}

}

}

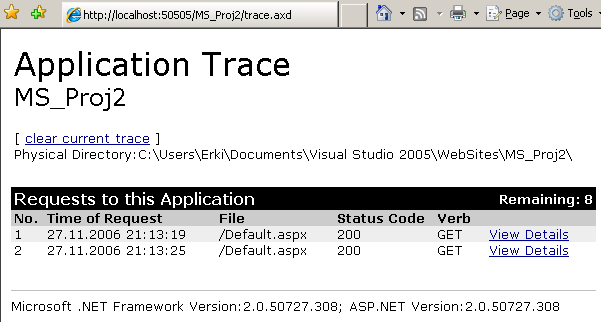
Tulemuseks trükitakse detailne vea kirjeldus jälitusinfo sisse:



Lisaks lehekülje põhisele sisse-välja lülitamisele on võimalik jälitusinfot kontrollida ka keskselt. Selleks tuleb vastav viide panna web.config faili system.web elemendi alamelemendiga trace:

<trace enabled="true" />

Nüüd ei kuvata enam infot mitte iga lehekülje lõpus vaid virtuaalse veebilehe trce.axd abil veebirakenduse juurkataloogis:



Klõpsates iga päringu järel oleval View Details lingil kuvatakse meile info, mis varem oli lisatud lehekülje lõppu.

Trace elemendil on lisaks enable atribuudile veel teisigi mis võimaldavad selle info kogumist paremini korraldada. Näiteks võime öelda, et

* jälitusinfo on nähtav ainult konsoolil (localhost) ja üle veebi seda ei serveerita localOnly = „true” (default)
* nähtavale jäävad kõige viimased päringud mostRecent = „true”. Vaikimisi näidatakse esimest x päringut!
* Sorteerime päringud ajalisse järjekorda traceMode=”SortByTime”
* Jätame meelde 100 päringut (10 asemel) requestLimit=”100”
* Korjame jälitusinfo trace.axd sisse mitte ei pane lehekülgede lõppu pageOutput=”false”

<trace enabled="true"

localOnly="true"

mostRecent="true"

traceMode="SortByTime"

requestLimit="100"

pageOutput="false"/>

PS! Jälitusinfot hoitakse mälus ning see kustub koos rakenduse mälust eemaldamisega.

Ülesandeid

\* Lülita lehel sisse Trace, tutvu saada oleva teabega

\* Trüki lehe koodis trace-teateid, loe neid.

\* Vaata trace-infot nii lehe alt kui eraldi lehelt.

### Vigade haldamine

Ükskõik kui väga me ka ei püüaks ikkagi esineb programmis vigu, olgu nad siis tingitud asjaolude kokkusattumusest, programmeerijapoolsest lohakusest või kasutaja pahatahtlikkusest.

.NET raamistik koos ASP.NETiga pakub vigade haldamiseks üsna mitmekihilist süsteemi. Loomulikult tuleks vead kõrvaldada nii vara kui võimalik, kuid alati tuleb arvestada asjaoludega, et kõike pole võimalik ette ennustada ja midagi võib jääda kahe silma vahele! Seega tuleks igal kihil pakkuda mingit lahendust vigade haldamiseks.

Kui Te jätate mõne vea kinni püüdmata siis teeb seda lõpuks ASP.NET ise, kuid nagu arvata võite, ei pruugi sealt tulevad veateated kasutajatele meeldida.

#### Rakenduse veebist eemaldamine

Kui rakenduses ilmnevad suuremad tõrked või on vaja teha hooldustöid on kasulik veebirakendus ajutiselt „maha võtta” e. muuta kasutajatele kättesaamatuks. ASPi ajal oli ainukeseks arvestatavaks meetodiks, kas rakenduse ümbernimetamine või veebiteenuse peatamine. ASP.NET pakub aga väga mugavat alternatiivi – nimelt tuleb rakenduse maha võtmiseks lisada veebirakenduse juurkataloogi fail app\_offline.htm. Fail peab kindlasti olema htm mitte html laiendiga! Faili sisse kirjutage teade, mida soovite saidi külastajatele anda.

Näiteks võiks kirjutada midagi järgmist:

<!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>

<meta content="text/html; charset=utf-8" http-equiv="content-type" />

<title>Teenus ajutiselt suletud</title>

</head>

<body>

<h1>Teenus ajutiselt suletud!</h1>

<p>

Vabandame, seoses hooldustöödega ei ole veebirakendus hetkel kättesaadav.<br />

Püüame probleemi kõrvaldada nii kiiresti kui võimalik.</p>

<p>Ootame teid peatselt tagasi!</p>

<!-- Selleks, et kõik ka IEs korrektselt toimiks peab lehekülje maht olema vähemalt 512B -->

</body>

</html>

Kui soovite rakenduse taas käima panna siis võib selle app\_offline.htm faili ära kustutada või ümber nimetada.

Failide veebist väljaarvamiseks võib neile kirjutada .exclude laiendi. Sellise laiendiga faile veebi ei serveerita. Tulemuseks on HTTP error 403 (Page not served).

#### Globaalne veakontroll

Globaalne veakontroll on viimane kontrollliin enne raamistikku ning võimaldab reageerida vigadele vastavalt HTTP veakoodidele. Veakontrolli sisselülitamiseks tuleb web.configi system.web sektsiooni lisada customErrors element. Kõige lihtsama lahenduse jaoks on vaja moodustada üks lehekülg, mis hakkab kas viga programselt haldama või kuvab kasutajale mingi viisaka teate.

customErrors elemendil on kaks atribuuti:

* mode näita, kuidas vea puhul käituda (On – näidata kõigile oma veateadet, Off – näidata kõigile süsteemset veateadet, RemoteOnly - näidata konsoolilt pöördujatele süsteemset veateadet ja teistele kohandatud veateadet[[6]](#footnote-6).
* defaultRedirect näitab, kuhu kasutaja suunata, kui midagi konkreetsemat pole öeldud

Nt järgnev rida web.configis ütleb, et kui tekib mingi tõrge tuleb kasutaja suunata lehele error.html.

<customErrors mode="On" defaultRedirect="~/vead/error.html" />

Kui soovite teatud HTTP vigade puhul anda mingi konkreetsema teate siis saate lisada customErrors elemendi alla täiendavad error elemendid.

Näiteks järgmise seadistusega ütleme, et kui tuleb HTTP viga 404 suuname kasutaja lehele error404.html, kui tuleb HTTP error 500 suuname kasutaja lehele error500.html, kui tuleb mingi muu viga suuname kasutaja lehele error.html.

<customErrors mode="On" defaultRedirect="~/vead/error.html">

<error statusCode="404" redirect="~/vead/error404.html"/>

<error statusCode="500" redirect="~/vead/error500.html"/>

</customErrors>

#### Keskne veakontroll

Globaalsetele sündmustele saab reageerida globaalses koodifailis global.asax. Veakontrolli tarbeks on seal meetod Application\_Error, mille sisse võite kirjutada oma vea haldamise protseduuri.

Application\_Error meetod käivitatakse, kui programmis tekib viga, mida leheküljel ära ei lahendata. Tegemist on viimase programse võimalusega tekkinud vea viisakaks lahendamiseks.

Vea haldamiseks tuleb ära kasutada Server objekt. Server.GetLastError() annab tagasi Exception tüüpi objekti viimase veaga. Sealt saate teada missuguse veaga oli tegemist.

Peale seda kui olete pakkunud veale mingi viisaka lahenduse saate öelda, et viga on lahendatud kutsudes välja Server.ClearError() meetodi.

Nt moodustame järgmise global.asax faili:

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

<script runat="server">

void Application\_Error(object sender, EventArgs e)

{

Response.Write("Tekkis viga<br /><b>");

Response.Write(Server.GetLastError().Message);

Server.ClearError();

Response.Write("</b><br />Lahendasin ära ...");

}

</script>

Kui nüüd mõni leht tekitab vea, mida kohapeal ära ei lahendata (nt all olev protseduur) siis käivitub keskne veakontroll ning ASP.NETi süsteemset veateadet kasutajale ei näidata.

using System;

public partial class Default6 : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

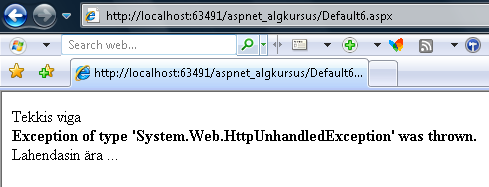
object o = null;

Page.Title = "Globaalse veakontrolli test";

lblTere.Text = o.ToString();

}

}



#### Veakontroll lehel

ASP.NET leheküljele leheküljekeskse veakontrolli paigaldamiseks tuleb leheküljele lisada Page\_Error meetod. Page\_Error meetod on mõeldud olukordade jaoks kus viga ei ole võimalik meetodite siseselt kinni püüda ja lahendada või kui soovite pakkuda lehekülje lõikes keskset veakontrolli.

Vea uurimiseks on võimalik kasutada taas (nagu ka veakontrollis global.asax failis) Server objekti.

protected void Page\_Error(object sender, EventArgs e) {

Response.Write("Tekkis viga<br /><b>");

Response.Write(Server.GetLastError().Message);

Server.ClearError();

Response.Write("</b><br />Lahendasin lokaalselt ära ...");

}

Page\_Error meetod on ainult System.Web.UI.Page klassist pärinevatel lehtedel. Pealehel (master page vt lk ) sellist meetodit ei ole! Kui soovite sellist leheküljekeskset veakontrolli kasutada kõigil lehtedel on võimalik aspx lehe tarbeks moodustada oma baasklass, mis pärineb System.Web.UI.Page klassist ning lahendada Page\_Error meetod selle sees.

Konstruktsioon näeks siis välja järgmine:

* lisame uue klassifaili LehePohi.cs. See fail oleks kasulik paigutada veebi juurikas olevasse App\_Code kausta, siis on ta kõikjalt ühte moodi hästi kasutatav. Sinna faili tekitame oma lehtede tarbeks baasklassi:

using System;

public abstract class LehePohi : System.Web.UI.Page

{

protected void Page\_Error(object sender, EventArgs e)

{

Response.Write("Tekkis viga<br /><b>");

Response.Write(Server.GetLastError().Message);

Server.ClearError();

Response.Write("</b><br />Lahendasin lokaalselt ära ...");

}

}

* Kui seejärel pärime oma lehe LehePohi klassist saame sellega kaasa ka lahenduse Page\_Error meetodile.:

using System;

public partial class Default7 : LehePohi

{

protected void Page\_Load(object sender, EventArgs e)

{

object o = null;

Page.Title = "Globaalse veakontrolli test";

lblTere.Text = o.ToString();

}

}

* Kui mõnel lehel on vaja personaalset lähenemist veakontrollile on võimalik Page\_Error meetod kas üle kirjutada või pärida see leht otse System.Web.UI.Page klassist

#### Veakontroll koodis

Kuigi ASP.NET pakub mitmeid võimalusi nii globaalseks kui ka lokaalseks veakontrolliks on kõige parem kui suudate vigu ennetada või siis lahendada nad kohe tekkimise hetkel. Meetodi sees vigadele reageerimiseks saab kasutada Try ... Catch konstruktsiooni, mis võimaldab väga mugavalt vigu hallata.

Ülesandeid

\* Lisa rakenduse juurkataloogi fail app\_offiline.htm. Veendu, et rakendus pole veebist kättesaadav. Eemalda faili ja kontrolli, et kõik taas toimiks.

\* Loo vea puhul näitamiseks eraldi leht. Tekita lehel veaolukord. Veendu, et kasutaja suunataks sinna.

\* Koosta eri vealehed lehe puudumise (404) ning muude vigade kohta. Katseta toimimist

### AJAX

AJAX - Asyncronious Java And Xml on raamistik, mis võimaldab asünkroonset andmevahetust kliendi e. veebisirvija ja serveri vahel. Kui AJAX-it ei kasuta siis iga serveri poole pöördumisega renderdatakse terve lehekülg uuesti ning laetakse terve lehekülg uuesti ka veebisirvijasse. Seda protsessi on veebilehte kasutades hästi näha sest iga klõpsu peale veebivormil lehekülg korraks vilksatab ning laetakse uuesti. Kasutades ajaksit värskendatakse lehekülge osaliselt. See saavutatakse tänu ASP.NET ja JavaScripti koostööle, kus andmevahetus käib XML vormingus. Java võimaldab lehekülje osalist muutmist kasutades DHTMLi ning ASP.NETi on lisatud võimalus lehekülje osaliseks renderdamiseks.

AJAX on sisse ehitatud .NET raamistik 3.5e ning saadaval ka eraldi .NET raamistik 2.0 tarbeks.

#### AJAXi kasutamine

Kõik AJAXiga seotud vahendid paiknevad Visual Studio Tööriistades (Toolbox) AJAX Extension kategoorias.

AJAXi kasutamiseks tuleb igale lehele lisada üks ja ainult üks ScriptManager. ScriptManageri ülesandeks on kasutaja browserisse vajalike JavasCriptide üleslaadimine ning andmevahetuse korraldamine veebisirvija ja serveri vahel.

<asp:ScriptManager ID="sm" runat="server" />

Kõik osad, veebivormist, mis vajavad osalist uuendamist tuleb paigutada UpdatePanel plokkidesse. Selliseid elemente võib lehel olla mitu ning iga UpdatePanel plokki on võimalik värskendada vastavalt vajadusele.

Värskendamise ajal on võimalik kuvada informatiivseid teated UpdateProgress elemendi abil.

Timer element võimaldab sooritada ajastatud tegevusi.

Järgnevalt väike näide veebilehest, mis kasutab AJAX funktsionaalsust GridViews olevate andmete filtreerimiseks.

<asp:ScriptManager ID="sm" runat="server" />

<p>Staatus:

<asp:UpdatePanel ID="u1" runat="server">

<ContentTemplate>

<asp:DropDownList ID="ddl1" runat="server"

AutoPostBack="True">

<asp:ListItem Value="True">Tehtud</asp:ListItem>

<asp:ListItem Selected="True" Value="False">

Tegemata</asp:ListItem>

</asp:DropDownList>

</ContentTemplate>

</asp:UpdatePanel></p>

<asp:UpdateProgress ID="UpdateProgress1" runat="server">

<ProgressTemplate>Uuendan andmeid .....</ProgressTemplate>

</asp:UpdateProgress>

<asp:UpdatePanel ID="u2" runat="server">

<ContentTemplate>

<asp:GridView ID="GridView1" runat="server"  
 DataKeyNames="Kood" DataSourceID="sds1"

AllowSorting="True" >

<Columns>

<asp:CommandField ShowEditButton="True" />

</Columns>

</asp:GridView>

<asp:SqlDataSource ID="sds1" runat="server"

ConnectionString="<%$ ConnectionStrings:toodConn %>"

SelectCommand="SELECT [Kood], [Nimi], [Tehtud] FROM

[tooderegister] WHERE Tehtud = @Tehtud"

UpdateCommand="UPDATE [tooderegister]

SET [Nimi] = @Nimi, [Tehtud] = @Tehtud WHERE [Kood] = @Kood">

<SelectParameters>

<asp:ControlParameter ControlID="ddl1" Name="Tehtud"

PropertyName="SelectedValue" Type="Boolean" />

</SelectParameters>

<UpdateParameters>

<asp:Parameter Name="Nimi" Type="String" />

<asp:Parameter Name="Tehtud" Type="Boolean" />

<asp:Parameter Name="Kood" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

</ContentTemplate>

</asp:UpdatePanel>

#### AJAX Control Toolkit

Kui standardsetest vahenditest jääb väheseks siis saate lisaks laadida endale AJAX Control Toolkit’i. <http://www.asp.net/ajax/ajaxcontroltoolkit>

Juhendi Control Toolkiti paigaldamiseks leiate aadressilt <http://www.asp.net/AJAX/AjaxControlToolkit/Samples/Walkthrough/Setup.aspx>

Samal leheküljel on ka näiteid kõigi Toolkiti koosseisu kuuluvate elementide kasutamise kohta.

### Lokaliseerimine

Tänapäeva globaliseeruvas keskkonnas ei ole mõeldav, et veebirakenduse kasutajaliides on loodud vaid ühes keeles. On iseenesest mõistetav, et kasutaja saab ise valida endale sobiva keele ning veelgi enam võiks rakendus ka esmase keelevaliku teha vastavalt kasutaja eelnevatele seadistustele.

Seega, kui hakkame looma oma veebirakendust, püüame selle teha kohe nii, et tegemist oleks mitmekeelse rakendusega.

Rakenduse lokaliseerimiseks kasutatakse ASP.NETis ressursifaile (.resx), mis tuleb koondada leheküljega samal tasemel olevatesse alamkaustadesse:

* App\_GlobalResources
* App\_LocalResources

App\_GlobalResources kaustas paiknevad rakenduse globaalsed ressursid e. väärtused, mis rakenduse piires ei muutu.

App\_LocalResources kaustas paiknevad lehekülje jaoks lokaalsed ressursid e. väärtused, mis on seotud vaid ühe leheküljega. Sinna kausta tuleb iga lehe ning keele jaoks teha eraldi ressursi .resx fail. Kõige lihtsam oleks ressursi fail lisada läbi Visual Studio - siis kirjutatakse kogu päise info ära automaatselt.

Loomulikult on võimalik lokaliseeritud ressursse hoida ka kusagil mujal, kuid selle jaoks tuleb ümber kirjutada lokaliseerimise teenusepakkuja (Provider). Kuidas seda korraldada saate vaadata materjali lõpus olevast lisast lk .

#### Lokaalsed ressursid

Samas võib ressursifaile ka käsitsi teha. Näiteks lisame faili Default.aspx.resx. .resx fail on XML fail, mille juurelemendiks on <root>. Järgnevalt tuleks kirjeldada faili versioon:

<resheader name="resmimetype">

<value>text/microsoft-resx</value>

</resheader>

<resheader name="version">

<value>2.0</value>

</resheader>

<resheader name="reader">

<value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

<resheader name="writer">

<value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

Kui päis moodustatud, lisame kõik vajalikud väärtused. Lisaks tekstilistele väärtustele on tegelikult võimalik lokaliseerida ka kõiki teisi väärtuseid näiteks suurus, värv jne

<data name="Label1.Text">

<value>Esimese labeli tekst Eesti keeles</value>

</data>

<data name="Label1.ForeColor">

<value>Blue</value>

</data>

<data name="Label2.Text">

<value>Teise labeli tekst Eesti keeles</value>

</data>

<data name="Page.Title" >

<value>Esimene lehekülg</value>

</data>

Kui eestikeelse tekstiga ressursifail on valmis, tuleks korrata sama toimingut ka kõigi teiste vajalike keelte jaoks. Kõigi järgnevate keelte ressursifailide nimedes peab sisalduma ka keel, mille kohta ressursse lisate. Kirjutades failide nimed sellisel kujul, vaatab .NET raamistik kõigepealt vajaliku tunnusega keelefaili, kui seda ei leia (või ei leia sellest otsitavat ressurssi), pöördutakse ilma keele laiendita ressursifaili poole. Seega ilma laiendita ressursifailis PEAVAD olema kõik kasutatavad ressursid kirjeldatud! Muul juhul ei pruugita neid ka olemasolevatest keelefailidest leida.

Näiteks lisame inglise keele tarbeks faili Default.aspx.en.resx

<root>

<resheader name="resmimetype">

<value>text/microsoft-resx</value>

</resheader>

<resheader name="version">

<value>2.0</value>

</resheader>

<resheader name="reader">

<value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

<resheader name="writer">

<value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

<data name="Label1.Text">

<value>First label in English</value>

</data>

<data name="Label1.ForeColor">

<value>Red</value>

</data>

<data name="Label2.Text">

<value>Second label in English</value>

</data>

<data name="Page.Title" >

<value>First Page</value>

</data>

</root>

Loomulikult tuleb ka Default.aspx failis mainida, et ta võib ressursse otsima hakata. Selleks tuleb päisesse (Page tag) lisada atribuudid Culture="Auto" UICulture="Auto".

Kui on soov lokaliseerida terve veebirakenduse, võib selle info kirjutada ka konfiguratsioonifaili lisades system.web elemendi alla alamelemendi globalization:

<globalization culture="auto" uiCulture="auto"/>

Culture ja UICulture vahe seisneb selles, et Culture määrab ära kuupäevade ja numbrite kujunduse ning UICulture määrab ära ressursifailide kasutuse.

PS! Jälgige tähesuuruseid: aspx failis algavad Culture ja UICulture suurte tähtedega, web.config failis aga väikeste tähtedega!

Lisaks tuleb ära märgistada kõik kontrollid, mis on plaanis lokaliseerida. Selleks tuleb kontrolli alustava tagi sisse lisada atribuut meta:resourcekey, mille väärtus peab vastama ressursifailis kirjeldatule. Antud juhul soovime lokaliseerida lehekülje pealkirja, seega peaks lisama meta atribuudi Page elemendile.

Page element näeb nüüdseks välja järgmine:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" meta:resourcekey="Page" Culture="Auto" UICulture="Auto" %>

Lisaks mõnele lehekülje määrangule soovime lokaliseerida kahe labeli Label1 ja Label2 omadusi. Nimetatud lipikud näevad välja järgnevad:

<asp:Label ID="Label1" runat="server" meta:resourcekey="Label1" /><br />

<asp:Label ID="Label2" runat="server" meta:resourcekey="Label2" />

Nüüd jääb muidugi küsimus, kuidas raamistik aru saab, millist keelt kasutada? Selleks on kaks võimalust. Esmalt on võimalik käsitsi muuta Culture ja UICulture atribuute. Samas kui kultuuri atribuudid on seatud automaatse valiku peale, valib raamistik keele vastavalt sellele, mis määratud veebisirvijas. Internet Exploreris on see info kirjas Tools/Internet Options/General/Languages vormil.

Lokaalseid ressursse on võimalik kasutada ka programselt. Lokaliseerimisega seotud klasside kasutamiseks on kasulik öelda, et soovite kasutada System.Globalization nimeruumi.

using System.Globalization;

Ressursside lugemiseks on kõige mugavam kasutada meetodit GetLocalResourceObject, mis vajab parameetrina ressursi nime. Näiteks muudame programselt Label1 lipiku sisu:

Label1.Text = GetLocalResourceObject("Label1.Text").ToString();

#### Globaalsed ressursid

Globaalseteks nimetatakse ressursse, mis on samad terves rakenduses. Globaalse ressursi lisamiseks tuleb lisada sobiva nimega ressursifail kausta App\_GlobalResources. Lisame näiteks ressursifaili asjad.resx ning iga vajaliku keele jaoks keele tähisega ressursifail asjad.en.resx, asjad.ru.resx jne.

<root>

<resheader name="resmimetype">

<value>text/microsoft-resx</value>

</resheader>

<resheader name="version">

<value>2.0</value>

</resheader>

<resheader name="reader">

<value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

<resheader name="writer">

<value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>

</resheader>

<data name="jutt">

<value>Mingi jutt</value>

</data>

</root>

Globaalse ressursi kasutamiseks tuleb teada nii ressurssi sisaldava faili nime kui ka ressursi nime. Kui see info teada, võime ressurssi kasutada ükskõik millise serveri kontrolli atribuudina. Näiteks lipiku teksti võime globaalsest ressursist lugeda järgmiselt:

<asp:Label ID="Lbl1" runat="server" Text="<%$ Resources:asjad, jutt %>" />

Kui on soov globaalset ressurssi lugeda programselt, saate kasutada GetGlobalResourceObject meetodit:

Lbl1.Text = GetGlobalResourceObject("asjad", "jutt").ToString();

Ressursse, nii lokaalseid kui ka globaalseid, on võimalik lisaks .resx failidele hoida ka andmebaasis või ükskõik millises teises andmehoidlas. Selleks tuleb ümber kirjutada lokaliseerimist pakkuv teenusepakkuja e. provider. Keda see võimalus huvitab, võib otsida sellekohaseid materjale nii otsingumootoritest kui ka erinevatest foorumitest. Ühe võimaliku lahenduse leiate ka lisast lk 341 „Ressursside hoidmine SQL Serveris”.

#### Programselt keele muutmine

Selleks, et muuta keelt programselt, oleks esmalt vajalik tekitada mingi mehhanism keele valimiseks. Üks variant on kasutada ripploendit, kõige parem koht sellele ripploendile oleks pealeht (master page), kuid võime selle lisada ka igale .aspx lehele. Siinses näites kirjutame ripploendisse elemendid staatiliselt, kuid neid võiks lugeda ka andmebaasist.

<asp:DropDownList runat="server" ID="ddlKeel" AutoPostBack="true"

OnSelectedIndexChanged="ddlKeel\_SelectedIndexChanged"

OnPreRender="ddlKeel\_PreRender" >

<asp:ListItem Text="Eesti" Value="et" />

<asp:ListItem Text="English" Value="en" />

</asp:DropDownList>

Et ripploend hakkaks keelt vahetama, tuleb ära kasutada SelectedIndexChanged sündmus. Valitud keelt oleks hetkel kõige parem hoida sessiooni muutujas. Selleks, et raamistik saaks aru, et midagi muutus, palume lehekülje uuesti laadida:

protected void ddlKeel\_SelectedIndexChanged(object sender, EventArgs e)

{

Session["keel"] = ddlKeel.SelectedValue;

Server.Transfer(Request.Url.PathAndQuery, false);

}

Server.Transfer teeb ümbersuunamise serveri tasemel e. kasutajale märkamatult.

Kuna lehe laadimine katkestatakse üsna varases faasis, ei õnnestu raamistikul kasutaja tehtud valikut salvestada. Seega tuleb kasutaja valik ise taastada. Selle koha peal aitab meid PreRender sündmus:

protected void ddlKeel\_PreRender(object sender, EventArgs e)

{

object oKeel = Session["keel"];

if (oKeel != null)

ddlKeel.SelectedValue = (string)oKeel;

}

Hetkel oleme loomulikult tegelenud vaid kasutaja soovide meelde jätmisega. Nüüd tuleks kirjutada ka meetod nende soovide realiseerimiseks. Keele muudatus on selline, mis tuleb paika panna lehe renderdamise varases faasis, sest sisuliselt kõik sõltub keelest. Selle tarbeks pakub raamistik 2.0 meile leheküljel meetodit InitializeCulture:

protected override void InitializeCulture()

{

object oKeel = Session["keel"];

if (oKeel != null)

{ // kui sessioonis on määratud, millist keelt kasutada

string sKeel = oKeel.ToString().ToUpper();

if (sKeel !=

Thread.CurrentThread.CurrentUICulture.ToString().ToUpper())

{ // kui kehtiv keel ei ole sama mis valitud keel

Thread.CurrentThread.CurrentUICulture =

CultureInfo.CreateSpecificCulture(sKeel);

}

}

else { // kui sessioonis keeleinfot veel ei ole

Session["keel"] = Request.UserLanguages[0];

Thread.CurrentThread.CurrentUICulture =

new CultureInfo(Request.UserLanguages[0]);

}

Thread.CurrentThread.CurrentCulture =

CultureInfo.CreateSpecificCulture(

Thread.CurrentThread.CurrentUICulture.Name);

}

Kuna meetod on kirjeldatud ka System.Web.UI.Page klassis, peame selle üle kirjutama sellest ka märksõna override meetodi kirjelduses. Edasi kontrollime, kas sessioonimuutujasse on kasutaja soov salvestatud. Kui ei, siis kasutame veebisirvija määranguid.

Ülesandeid

\* Loo tervitusega leht. Tervituse tekst võetakse keelefailist

\* Määra ressursifailis lisaks tervituse tekstile ka tervituse taustavärv

\* Küsi andmeid programmi abil globaalsete ja lokaalsete ressursside failidest

### Master Pages

Pealehed (Master Pages) on mõeldud rakenduses ühtse kujunduse või programsete elementide paigutuse hoidmiseks. Pealehed võimaldavad vältida või vähendada koodi dubleerimist ja lihtsustavad oluliselt rakenduse disaini ühtsustamist ning ka hilisemat haldamist. Lõppkasutajale on pealehed täiesti nähtamatud st leht, kus olete kasutanud pealehti, näeb veebisirvijas välja täpselt samasugune, kui leht, kus neid pealehti kasutatud ei ole. Tegemist on vahendiga, mis on mõeldud arendajatele produktiivsuse tõstmiseks.

Pealeht erineb tavalisest ASP.NET lehest selle poolest, et tema laiendiks on .master, lisaks muudele elementidele kirjeldatakse temas, kuhu võib panna programseid elemente ning kood pärineb klassist System.Web.UI.MasterPage.

Järgnevalt loomegi ühe lihtsa pealehe tavaline.master, kus näitame, kuhu tekivad lehekülje päis, menüü ning üks piirkond, kuhu aspx lehed saavad oma funktsionaalsuse paigutada.

<%@ Master Language="C#" AutoEventWireup="true" CodeFile="tavaline.master.cs" Inherits="tavaline" %>

<!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>Minu esimene pealeht</title>

</head>

<body>

<form id="form1" runat="server">

<table>

<tr>

<td colspan = "2" >

Siia tuleb lehekülje päis

</td>

</tr>

<tr >

<td>

Siia ehitame menüü

</td>

<td>

<asp:contentplaceholder id="sisu" runat="server" />

</td>

</tr>

</table>

</form>

</body>

</html>

Kuna hetkel sellel lehel mingit programset funktsionaalsust ei ole, siis on koodifail tavaline.master.cs sisuliselt tühi.

using System;

public partial class tavaline : System.Web.UI.MasterPage

{

protected void Page\_Load(object sender, EventArgs e) {}

}

Kui soovime, et meie default.aspx kasutaks seda pealehte, tuleb teha paar muudatust: lisada Page elemendi atribuutidesse MasterPageFile atribuut, nii et Page element näeb välja järgmine:

<%@ Page Language="C#" MasterPageFile="~/tavaline.master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" Title="Minu teine leht" %>

Viidates ASP.NET rakendustes teistele failidele, kasutatakse tihtipeale faili absoluutset aadressi rakenduse kausta suhtes. Rakenduse juurkaustale viitamiseks kasutatakse tildet „~” e. kui me vaatame ülal olevat näidet, siis paikneb fail tavaline.master rakenduse juurkataloogis.

Lisaks muudatusele lehekülje Page elemendi atribuutides, tuleb ära kustutada kõik, mis asub form elemendist väljaspool, kaasaarvatud form elemendi alustavad ja lõpetavad tagid. Selle asemele tulevad piirkonnad pealehelt. Seega näeb ülejäänud default.aspx välja järgmine:

<asp:Content ID="Content1" ContentPlaceHolderID="sisu" Runat="Server">

<asp:Label ID="Label1" runat="server" />

</asp:Content>

Lehekülje koodis mingeid muudatusi teha ei tule.

Kui soovite kasutada mõnda pealehel olevat elementi või omadust aspx lehel, siis leiate lehe küljes oleva pealehe Page.Master omaduse alt. Samas pole see omadus otse kasutatav. See tähendab, kui näiteks teie pealehel on omadus jutt, siis ei ole võimalik seda muuta näiteks sellise lausega Page.Master.jutt = „mingi jutt”. Selleks, et saada ligi pealehe omadustele ja sealolevatele elementidele, tuleb Master all olev objekt teisendada õigesse tüüpi. Seega kui teie pealehe taga oleva klassi nimi on MinuPealeht, siis tuleks seal oleva omaduse jutt muutmiseks kirjutada järgnevalt:

((MinuPealeht)Master).jutt = ”mingi jutt”;

### Elemendid lehel

Lehekülje loomisel on meil võimalus kogu töö teha ära kasutades ASPist tuntud Response.Write meetodit, genereerides rea kaupa kliendile mineva HTMLi, kuid saame kasutada ka erinevaid kontrolle.

Elemendid võimaldavad läheneda leheküljele objekt-orienteeritud põhimõtteid kasutades. Elemente, mida kasutada, on lugematul hulgal. Suur hulk neist on kaasas koos .NET raamistikuga, väga palju saab tõmmata Internetist (osad on tasuta, osad tasulised) ning loomulikult on neid võimalik ka ise kokku panna.

Loomulikult ei jõua me siinkohal vaadata kõiki võimalikke kontrolle väga detailselt, kuid püüame anda ülevaate olulisemast ning sellest, mida kus ja kuidas kasutada.

#### Standardsed serveri elemendid

Alustame standardsete ASP.NET elementidega. ASP.NET Server elemendid on põhilised abivahendid veebirakenduste loomisel. Serveri elementidel on loogiline ja lihtsalt kasutatav objektmudel ning lisaks sellele oskab raamistik neid sõltuvalt kliendi veebisirvija võimekusest renderdada erinevateks HTML tagideks.

Standardsed ASP.NET kontrollid tunnete ära sellejärgi, et nende nime eesliide on asp:. Kõigil kontrollidel peab kindlasti olema täidetud:

* runat=”server” atribuut, mis näitab, et kontroll tuleb renderdada serveris
* ID atribuut, mis on selle kontrolli unikaalne tunnus. ID kasutatakse kontrolli poole pöördumiseks nii märgistuses kui ka koodis. Kõik ID peavad lehekülje piires olema erinevad!

Järgnevalt standardsete serverielementidele lühikirjeldused koos näidetega:

|  |  |
| --- | --- |
| AdRotator | <asp:AdRotator runat="server" ID="adrLogod" AdvertisementFile="~/App\_Data/sisu.xml" />  Näitab ühe kaupa, juhuslikus järjekorras andmeballikast leitud pilte. Kasulik nt banneri loomisel |
| BulletedList | <asp:BulletedList ID="BulletedList1" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" />  Täppidega loend andmebaasis olevatest andmetest |
| Button | <asp:Button ID="bNupp" runat="server" OnClick="bNupp\_Click" Text="Vajuta mind" />  Nupud on mingite kasutaja poolt algatatavate tegevuste tegemiseks. Nupuvajutusele reageerimiseks tuleb realiseerida OnClick sündmus. |
| Calendar | <asp:Calendar ID="cKalender" runat="server" />  Tekitab kalendri, mille pealt saab kasutaja valida kuupäeva. Kuupäeva kasutamiseks tuleb pöörduda kalendri SelectedDate atribuudi juurde |
| CheckBox | <asp:CheckBox ID="cbYx" runat="server" Text="Vali mind" />  Tekitab märkeruudu, mille väärtust saate edaspidi kasutada läbi Checked atribuudi. |
| CheckBoxList | <asp:CheckBoxList ID="cblValikud" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood" />  Tekitab märkeloetelu ruutude loetelu, kust on kasutajal võimalik valida mitmeid. Esimese valiku leidmiseks saate kasutada SelectedIndex, SelectedItem või SelectedValue atribuute. Kõigi valikute leidmiseks tuleb läbi käia Items kollektsioon ning kontrollida iga valiku Checked atribuuti. |
| DropDownList | <asp:DropDownList ID="ddlValikud" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood" />  Ripploend andmeallikast leitud andmetest. Valitud väärtuse leidmiseks saab kasutada SelectedIndex, SelectedItem või SelectedValue atribuute. |
| FileUpload | <asp:FileUpload ID="fuFail" runat="server" />  Võimaldab faile ülesse laadida. Kasutaja poolt valitud faili leiate PostedFile atribuudi alt. Faili saab kätte andmevoona PostedFile.InputStream atribuudist. |
| HiddenField | <asp:HiddenField ID="hfPeidetud" runat="server" />  Peidetud väli, kus on võimalik hoida teksti, mida kasutajale näidata ei soovi. |
| HyperLink | <asp:HyperLink ID="hlLink" runat="server" NavigateUrl="http://www.savisaar.ee" Text="koduleht" />  Hüperlink, kus lingina võib toimida nii tekst kui ka pilt. |
| Image | <asp:Image ID="iPilt" runat="server" ImageUrl="~/images/Sample.jpg" />  Pilt, mis renderdub HTMLi IMG tagiks. |
| ImageButton | <asp:ImageButton ID="ibNupp" runat="server" ImageUrl="~/images/Sample.jpg" OnClick="bNupp\_Click" />  Pilt, mis reageerib hiire klikile e. töötab nagu tavaline nupp. |
| ImageMap | <asp:ImageMap ID="imJooksja" runat="server"  ImageUrl="~/images/Sample.jpg">  <asp:CircleHotSpot X="184" Y="38" Radius="20"  PostBackValue="pea" HotSpotMode="PostBack" />  <asp:RectangleHotSpot Left="160" Top="90" Right="283"  Bottom="212" PostBackValue="parempool"  HotSpotMode="PostBack" />  <asp:PolygonHotSpot PostBackValue="jooksja"  Coordinates="0,0,105,0,108,68,150,102,150,  111,135,116,85,112,64,18,49,41,61,139,107,175,103,  194,51,202,0,85" HotSpotMode="PostBack" />  </asp:ImageMap>  ImageMap tekitab pildi, millel on erinevaid piirkondi, kuhu kasutaja saab klõpsata. Klõpsatavate piirkondade tekitamiseks tuleb ImageMap elemendi sisse panna HotSpot alamelemendid. CircleHotSpot tekitab ringi kujulise piirkonna, RectangleHotSpot tekitab ristkülikukujulise piirkonna, PolygonHotSpot tekitab hulknurgakujulise piirkonna.  Kui keegi klikib mingil piirkonnal siis on võimalik, kas suunata nii uuele aadressile kui ka programselt klikki hallata. Programse haldamise jaoks tuleb ära realiseerida Click sündmus, kus on võimalik vaadelda parameetriga kaasa antud PostBackValue omadust |
| Label | <asp:Label ID="lblJutt" runat="server" Text="Mingi tekst" />  Label on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.  Labeli peale saab panna staatilist teksti ja ka lihtsamaid HTMLi tage kuid pole võimalik lisada dünaamiliselt serveri kontrolle.  Label renderdub veebisirvijas SPAN tagiks. |
| LinkButton | <asp:LinkButton ID="lbNupp" runat="server" Text="Vajuta mind" OnClick="bNupp\_Click" />  Hüperlink, mis töötab nupuna e. kui hiirega klõpsata ei minda mitte uuele lehel vaid tekitatakse Click sündmus. |
| ListBox | <asp:ListBox ID="lbLoend" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst"  DataValueField="kood"/>  Loend väärtustest, kust sõltuvalt seadistusest on võimalik valida, kas üks või mitu väärtust e. sõltuvalt seadistusest käitub analoogselt RadioButtonList’iga või CheckBoxList’iga. Kõige olulisem erinevus seisneb selles, et pikk loetelu on võimalik kokku suruda väga väiksele alale. |
| Literal | <asp:Literal ID="lKoht" runat="server" />  Literal on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.  Literal erineb Label elemendist selle poolest, et ta ei lisa dünaamilise sisu ümber mingit lisamärgistust. Sellest tulenevalt ei ole võimalik sisu stiilidega kujundada. |
| Localize | <asp:Localize ID="lzKoht" runat="server" />  Erinevalt Literal kontrollist võimaldab Localize kontroll reserveerida koha lokaliseeritud sisu tarbeks. Muus osas on need kaks kontrolli täiesti ühesugused.  Erinevalt Label kontrollist ei lisa Localize sisu ümber mingit märgistust ega võimalda ka sisu kujundamist. |
| MultiView | <asp:MultiView ID="mvVaade" runat="Server"  ActiveViewIndex="0">  <asp:View ID="v1" runat="server">  Mingid asjad, mida näidata esimeses vaates  </asp:View>  <asp:View ID="v2" runat="server">  Mingid teistsugused asjad teises vaates näitamiseks  </asp:View>  </asp:MultiView>  MultiView võimaldab teha leheküljel oleva info vaatamiseks erinevaid vaateid. Vaateid saab vahetada saab valida nii märgistuses kui ka programselt ActiveViewIndex atribuudi abil. Sisuliselt võimaldab MultiView integreerida ühele ASP.NET lehele mitmeid analoogseid veebilehti. |
| Panel | <asp:Panel ID="pKoht" runat="server" />  Panel on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.  Erinevalt Label ja Literal kontrollidest on võimalik sinna sisse paigutada ka teisi serveri kontrolle.  Panel kontroll paneb kogu dünaamilise sisu kas div või table elemendi sisse. |
| PlaceHolder | <asp:PlaceHolder ID="phKoht" runat="server" />  PlaceHolder on konteiner lehele dünaamilise sisu tekitamiseks. Selliseid konteinereid on kokku neli: Label, Literal, Panel, PlaceHolder.  PlaceHolder võimaldab analoogselt Panel kontrolliga lisada dünaamilisi teisi serveri kontrolle kuid erinevalt Panel elemendist mingit lisamärgistust nende ümber ei lisata. |
| RadioButton | <asp:RadioButton ID="rb1" runat="server" GroupName="g1" Text="Valik1" />  Raadionupp võimaldab teha ühe valiku ette antud loetelust. Kui lehel on mitmeid loetelusid mitu on võimalik raadionupud jagada gruppidesse. |
| RadioButtonList | <asp:RadioButtonList ID="rblValikud" runat="server" DataSourceID="xmlDSLoend" DataTextField="tekst" DataValueField="kood" />  RadioButtonList võimaldab andmebaasis olevatest andmetest tekitada raadionuppudega loendi, kust saab valida ühe valiku. |
| Substitution | <asp:Substitution ID="sDynKoht" runat="server" MethodName="DynSisu" />  Substitution kontroll märgistab ära koha, mida puhverdatud (cached) lehel dünaamiliselt uuendatakse. Dünaamilise sisu peab tekitama staatiline, meetod, mis tagastab stringi. |
| Table | <asp:Table ID="t1" runat="Server" >  <asp:TableHeaderRow ID="th" runat="server">  <asp:TableHeaderCell id="th1" runat="server">  A veerg  </asp:TableHeaderCell>  <asp:TableHeaderCell id="th2" runat="server">  B veerg  </asp:TableHeaderCell>  </asp:TableHeaderRow>  <asp:TableRow ID="tr1" runat="server">  <asp:TableCell ID="a1" runat="server">  Lahter A1  </asp:TableCell>  <asp:TableCell ID="b1" runat="server">  Lahter B1  </asp:TableCell>  </asp:TableRow>  <asp:TableRow ID="tr2" runat="server">  <asp:TableCell ID="a2" runat="server">  Lahter A2  </asp:TableCell>  <asp:TableCell ID="b2" runat="server">  Lahter B2  </asp:TableCell>  </asp:TableRow>  </asp:Table>  Table kontroll võimaldab tekitada tabeli, mida on võimalik väga lihtsalt programselt hallata. Eriti kasulik on see struktuur juhul kui soovite tabelit moodustada koodi abil, mitte lehekülje märgistuses. |
| TextBox | <asp:TextBox ID="txtInf" runat="server" />  TextBox on kast, kuhu kasutaja saab sisestada mingit teksti. Tekstikaste on võimalik teha nii ühe kui ka mitmerealisi ning peidetud tekstiga paroolikaste. Sisestatud teksti saab hiljem kätte Text atribuudi alt. |
| View | <asp:View ID="v1" runat="server">  Mingid asjad, mida näidata esimeses vaates  </asp:View>  View kontrolle saab lisada ainult MultiView kontrolli sisse. Tegemist on ühega paljudest vaadetest MultiView sees. |
| Wizard | <asp:Wizard ID="wVolur" runat="server" ActiveStepIndex="1">  <WizardSteps>  <asp:WizardStep runat="server" Title="Samm 1">  Mingi esimese sammu sisu  </asp:WizardStep>  <asp:WizardStep runat="server" Title="Samm 2">  mingi teise sammu sisu  </asp:WizardStep>  </WizardSteps>  </asp:Wizard>  Võlur on sarnane MultiView kontrolliga võimaldades luua samadest andmetest mitmeid vaateid. Erinevalt MultiView’st on võluris ette määratud liikumise loogika ning info paigutus.  Võlureid on võimalik kasutada nii töövoogude (WorkFlow) kui ka pikemate vormide juures, kus on võimalik infot jagada mitmele lehele.  Võlur on mõeldud suuremahuliste sisestuste haldamiseks, näiteks küsitlused. Selle asemel, et kõik väljad koondada ühele lehele, võimaldab võlur tekitada mitu lehekülge andmete sisestamiseks, mille vahel võib kasutaja liikuda vastavalt vajadusele, kas lehtede järjekorras või vabal valikul. |
| Xml | <asp:Xml ID="Xml1" runat="server" DocumentSource="~/App\_Data/sisu.xml" />  Xml kontroll võimaldab veebilehel kuvada XML andmeid ja seda nii toorel kujul kui ka teisendatud kujul.  Loe lisaks ka XMLi peatükki. |

Dünaamiliselt sisu tekitamisel on soovitus eelistada Literal või Localize kontrolli Labelile ning PlaceHolder kontrolli Panel kontrollile.

#### Programselt hallatavad HTML elemendid

Kui HTML elemendile lisada atribuut runat=“server“ siis on HTML elemendid programmis kasutatavad analoogsed Serveri elementidega. Erinevus võrreldes serveri elementidega seisneb selles, et nende objektmudel on moodustatud DHTMLi baasil ning nende haldamine ei ole nii lihtne kui serveri elementide puhul. Samuti ei oska .NET neile pakkuda alternatiivseid renderdusi sõltuvalt veebisirvija võimekusest.

#### Sisendi kontrollimise elemendid (Validators)

Kõige rohkem tööajal tekkivatest vigadest ja ka rakenduses olevatest turvaaukudest on tingitud kasutajate poolt tehtavatest ootamatutest sisestustest. Validation kontrollid on mõeldud kasutaja poolt sisestatud info kontrollimiseks ning on väga olulised vormide ehitamisel.

Kontrollimine käib enamasti kahes etapis:

* Kontrollitakse, kliendi poolel ehk veebisirvijas, javascripti abil, kas andmed on korrektsed. See võimaldab anda kasutajale märksa kiiremaid vastuseid kui tehes sama kontrolli serveris.
* Kontrollitakse serveris, kas kõik on korrektne. Kaitseb teie veebirakendust vigaste sisestuste eest. Kui kasutaja väga tahab, siis võib ta JavaScripti välja lülitada ning sellisel juhul kliendipoolne kontroll ei toimi. Teile on aga oluline, et andmed oleksid korrektsed ennem kui Te hakkate neid edasi töötlema. Seega on serveri poole kontroll väga oluline. Enamgi veel - kliendi poolsest kontrollist on võimalik loobuda aga serveris tuleb alati kontrollida, muidu muutub loodav veebrakendus väga ebastabiilseks ning ohtlikuks.

Kõigil validaatoritel on mõned olulised omadused, mis peaks olema alati määratud:

* Display – kas kontrolli positsioonil reserveeritakse veateate jaoks ruum või mitte
* ControlToValidate – kontroll, mida validaatoriga valideerida
* ErrorMessage – täispikk veateade. Näidatakse juhul kui kontroll ebaõnnestub
* SetFocusOnError – kas vea puhul muudetakse vea põhjustanud kontroll aktiivseks
* Text – lühike veakirjeldus. Näidatakse juhul kui kontroll ebaõnnestub
* ValidationGrupp – juhul kui samal lehel on mitu iseseisvat andmekogumit on võimalik ka kontrollid jagada gruppidesse ning kontrollida iga gruppi eraldi.

|  |  |
| --- | --- |
| RequiredFieldValidator | Kontrollib, et väli oleks täidetud. |
| RangeValidator | Tunneb erinevaid andmetüüpe ning kontrollib, et sisestatud väärtus oleks etteantud vahemikus. |
| RegularExpressionValidator | Kontrollib sisestuse sobivust regulaaravalisega |
| CompareValidator | Võrdleb sisestust teise kontrolli sisuga. Kasulik nt paroolivahetuse vormi juures, kus tuleb uut parooli sisestada kaks korda |
| CustomValidator | Ise tehtud validaator. Kontrollib täpselt nii nagu ise soovid.  Validaatorit ehitades tuleb kindlasti realiseerida ServerValidate sündmus ning kui soovite ka veebisirvijas kontrolli rakendada siis JavaScripti abil ka kliendi poolne kontroll. JavaScripti funktsiooni nime saate sisestada ClientValidationFunction atribuudile. |
| ValidationSummary | Kokkuvõte vigadest. Kui lehel on ValidationSummary kontroll siis validaatori positsioonil näidatakse lühikest veakirjeldust (Text atribuut) ning ValidationSummary näitab pikka vea kirjeldust (ErrorMessage atribuut). |

Selgitavaks näiteks siia juurde ülalpool andmebaasipõhise rakendusena kasside lisamislehe juurde kuuluv kontroll, et kassinimi oleks olemas ning kassi mass kassile sobilikus vahemikus.

<InsertItemTemplate>

kassinimi:

<asp:TextBox ID="kassinimiTextBox" runat="server"

Text='<%# Bind("kassinimi") %>' />

<asp:RequiredFieldValidator   
 ControlToValidate="kassinimiTextBox"   
 ErrorMessage="Kassi nimi puudub"   
 runat="server"   
 ID="nimeolemasolukontrollija"/>

<br />

kassimass:

<asp:TextBox ID="kassimassTextBox" runat="server"

Text='<%# Bind("kassimass") %>' />

<asp:RangeValidator ID="RangeValidator1" runat="server"  
 ControlToValidate="kassimassTextBox" MinimumValue="100"  
 MaximumValue="10000"   
 ErrorMessage="Pole mõistliku suurusega kass"/>

<br />

<asp:LinkButton ID="InsertButton"

runat="server" CausesValidation="True"

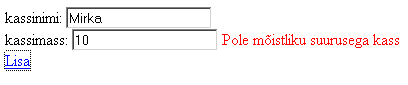
CommandName="Insert" Text="Lisa" />

</InsertItemTemplate>

Nagu näha, ei saa lisamata jätta kassi nime.



Samuti peab olema kassi mass mõistlikus vahemikus nagu ülalt koodist MinimumValue ja MaximumValue vahelised piirid määravad.



Ülesandeid

* Katseta mitmesuguseid kontrolle.
* Püüa nende väärtusi programmiga lugeda ja kätte saada.
* Kontrolli validaatori abil, et tekstiväli oleks täidetud

#### Navigeerimiselemendid

Navigeerimiseks mõeldud serveri kontrollid pakuvad mugavaid veebirakenduses navigeerimise võimalusi. Navigaatoreid on kokku kolm.

|  |  |
| --- | --- |
| SiteMapPath | <asp:SiteMapPath ID="smpKoht" runat="server"/>  SiteMapPath näitab kasutajale loogilist paiknemist veebirakenduse struktuuris. Paiknemist näidatakse analoogselt URLile või failisüsteemis kataloogiteekonnale.  SiteMap kasutamiseks tuleb veebi juurkataloogi lisada web.sitemap fail, milles kirjeldate ära veebi loogilise struktuuri. Web.sitemap on XML fail, mis vastab SiteMap-File skeemile (schema). Faili sisu võiks siis välja näha midagi sellist:  <?xml version="1.0" encoding="utf-8" ?>  <siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >  <siteMapNode url="~/Default.aspx" title="Avaleht">  <siteMapNode url="~/Default1.aspx" title="1. katse" />  <siteMapNode url="~/Default2.aspx" title="2. katse" >  <siteMapNode url="~/Default3.aspx" title="variant 1" />  <siteMapNode url="~/Default4.aspx" title="variant 2" />  </siteMapNode>  </siteMapNode>  </siteMap>  Sitemap failis peab ära kirjeldama kõikide lehekülgede loogilise positsiooni, kuhu kasutaja võib sattuda. Vajadusel saab seal pakkuda isegi erinevaid lahendusi vastavalt URLis nähtavatele parameetritele.  Kuna veebisirvija Back nupu programne haldamine on väga keeruline siis kasutatakse tagasi liikumise võimaldamiseks enamasti SiteMapPath kontrollil baseeruvat struktuuri. Mõnikord kutsutakse sellist navigeerimisvahendit ka leivapuruks (kunstmuinasjutu järgi, kus leivapuru järgi püüti koju tagasi minna). |
| Menu | <asp:Menu ID="menyy" runat="server">  <Items>  <asp:MenuItem Text="Avaleht" />  <asp:MenuItem Text="Mingid asjad" >  <asp:MenuItem Text="Variant 1" />  <asp:MenuItem Text="Variant 2" />  </asp:MenuItem>  <asp:MenuItem Text="Veel midagi" />  </Items>  </asp:Menu>  Menüü võimaldab kasutajal lihtsalt leida veebirakenduses olevat informatsiooni. Menüü sisu on võimalik sisestada nii staatiliselt (nagu on näha ülalolevast näitest), kui ka lugeda andmebaasist. |
| TreeView | <asp:TreeView ID="TreeView1" runat="server">  <Nodes>  <asp:TreeNode Text="Avaleht" />  <asp:TreeNode Text="Mingid asjad" >  <asp:TreeNode Text="Variant 1" />  <asp:TreeNode Text="Variant 2" />  </asp:TreeNode>  <asp:TreeNode Text="Veel midagi" />  </Nodes>  </asp:TreeView>  TreeView võimaldab vaadelda hierarhilisi andmeid puu kujul. Puud on võimalik moodustada nii staatiliselt (nagu näha ülal olevast näitest) kui ka moodustada dünaamiliselt andmebaasis olevate andmete järgi. |

Ülesandeid

\* Koosta menüüga pealeht (master) ning sealt viited mitmele alamlehele

\* Lisa lehestikus asukohta näitav SiteMap

\* Paiguta lehestiku lehed mõttes puukujulisse struktuuri. Aita navigeerimisel kasutajat TreeView-ga

#### Omaloodud elemendid (UserControl)

Lisaks pealehtedele saab programmiloogikat jagada ka läbi isetehtud graafilise kasutajaliidese elementide (User Controls) e. kasutaja kontrollide (KK). KK on ASP.NET leht ASP.NET lehe sees. KK abil saab kapseldada mingite graafilise liidese elementide kooslust ning funktsionaalsust.

Näiteks lisame oma veebilehele lisaks sildile veel tekstikasti ja nupu ning ütleme, et me soovime reageerida sündmusele, kui keegi klikib sellel nupul. Peale seda väikest täiendust näeb default.aspx content elemendi sisu välja järgmine:

<asp:TextBox ID="txtKast" runat="server" /><br />

<asp:Button ID="bOK" runat="server" Text="Klõpsa siia" OnClick="bOK\_Click" /><br />

<asp:Label ID="lblInf" runat="server" />

Nüüd tuleb teha ka väikesed täiendused lehekülje koodis default.aspx.cs. Esmalt muudame ära Page\_Load sündmuse, kus ütleme, et kui sellele leheküljele tullakse 1. korda, siis tuleb labeli sisu ära kustutada. Kui sama sessiooni ajal tullakse lehele 2st, 3ndat jne korda, siis ei tehta midagi. Lehekülje objekti küljes olev omadus IsPostBack on true, kui ei olda lehel esimest korda.

protected void Page\_Load(object sender, EventArgs e)

{

if (!Page.IsPostBack)

lblInf.Text = "";

}

Lisaks peame tekitama protseduuri nupu klikile reageerimiseks. Kui keegi klikib nupule, siis muudame ära labeli teksti vastavalt tekstikastis olevale jutule.

protected void bOK\_Click(object sender, EventArgs e)

{

lblInf.Text = "Sa kirjutasid tekstikast: " + txtKast.Text;

}

Kui me nüüd avastame, et sellist tekstikasti, labeli ja nupu interaktiivset kooslust läheb meil ka mujal vaja, siis on mõistlik see funktsionaalsus realiseerida KK abil. KK lehed e. .ascx lehed on oma ülesehituselt väga sarnased .aspx lehtedega seega on üsna tavaline, et esmalt realiseeritakse mingi funktsionaalsus .aspx lehel ning seejärel teisendatakse see leht KK-ks.

KK-l on võrreldes .aspx lehega 4 erinevust:

* faililaiend on .ascx
* Page element on asendatud Control elemendiga, millel on peaaegu samad atribuudid.
* kuna tegemist on osaga lehest, siis ei saa seal kasutada ei form elemente ega content elemente
* KK taga olev klass päritakse System.Web.UI.UserControl klassist

Seega võiks meie kolme komponendiga kk.ascx näha välja järgmine:

<%@ Control Language="C#" AutoEventWireup="true" CodeFile="kk.ascx.cs" Inherits="kk" %>

<asp:TextBox ID="txtKast" runat="server" /><br />

<asp:Button ID="bOK" runat="server" Text="Klõpsa siia" OnClick="bOK\_Click" /><br />

<asp:Label ID="lblInf" runat="server" />

Ning kood selle KK juures (failis kk.ascx.cs) näeks välja selline:

using System;

using System.Web.UI;

public partial class kk : System.Web.UI.UserControl

{

protected void Page\_Load(object sender, EventArgs e)

{

if (!Page.IsPostBack)

lblInf.Text = "";

}

protected void bOK\_Click(object sender, EventArgs e)

{

lblInf.Text = "Sa kirjutasid tekstikast: " + txtKast.Text;

}

}

Selleks, et oma värskelt loodud KK’d kasutada, peame selle paigutama mõnele aspx lehel. Kuna tegemist ei ole raamistikku kuuluva elemendiga, tuleb see esmalt registreerida. Selleks lisame lehekülje algusesse peale Page elementi ning enne igasuguseid teisi elemente Register elemendi, milles ütleme, millises failis meie KK paikneb, ning kuidas me soovime temale viidata.

<%@ Register Src="kk.ascx" TagName="kk" TagPrefix="kool" %>

Ning kohas, kus see KK peab paiknema, lisame vastava kirjeldusega elemendi:

<kool:kk ID="Kk1" runat="server" />

Antud juhul võiks selleks kohaks olla Content element default.aspx lehel. Kuna kogu eelnev funktsionaalsus sai KK’sse üle viidud, siis võiks kogu ülejäänud sisu Content elemendi seest ära kustutada. Ühtlasi tuleks kustutada ka kogu kood default.aspx.cs seest e. alles jääb vaid tühi klass.

public partial class \_Default : System.Web.UI.Page{}

Sama KK’d võib ühel ja samal lehel kasutada mitmes eksemplaris ning samuti võime selle KK registreerida nii mitmele lehele kui soovime.

Seega võiks kokkuvõtteks öelda, et läbi pealehtede MasterPage ühtlustame üldise elementide paigutuse aspx lehtedel. KK abil kapseldame kasutajaliidese elementide kooslust.

Ülesandeid

\* Kasuta näites loodud kasutajakontrolli oma lehel

\* Loo kontroll kahe arvu korrutamiseks. Katseta.

### Veebilehtede kujundamine kasutades nägusid (Themes)

Üldiselt on tuntud tõde, et hea programmeerija ei ole tavaliselt hea disainer ning hea disainer ei ole enamasti hea programmeerija. Luues aga veebirakendust, on vaja, et hästi oleks tehtud nii disain kui ka funktsionaalsus. See tähendab aga seda, et ühe veebilehega tegelevad korraga kaks inimest. Juba ASP.NET esimeses versioonis oli võimalik kood ja kujundus üksteisest eraldada (aspx fail kujunduse ja paigutuse jaoks, .cs fail koodi jaoks), kuid sellel lähenemisel oli kaks nõrka kohta:

* Läbi aspx faili ei saa kujundada dünaamiliselt tekitatud objekte
* Ühtse kujunduse hoidmine on raske, kuna kujundust tuleb igas aspx failis iga kontrolli juures uuesti korrata.

Üks lahendus on kasutada vana head stiilifaili .css, kuid raamistiku poolt genereeritud HTML objektide stiiliga kujundamine on üsna raske. Raskeks teeb selle asjaolu, et genereeritud struktuurid võivad tulla väga keerulised ning aeg-ajalt on väga raske ennustada, milliseks HTML objektiks üks või teine kontroll renderdub.

Lahendusena neile probleemidele pakub ASP.NET 2.0 välja näod e. Theme’d. Ühe näo alla koondatakse hulk kujundusi, mida on mugav tervele lehele peale panna. Nägu koosneb kahte liiki kujundustest:

* nahad (skin), on mõeldud serveri kontrollide kujundamiseks
* stiilid (css), on mõeldud HTML tagide kujundamiseks

Kõik näod tuleb paigutada veebi juurikas asuvasse App\_Themes kausta. Sinna tuleb teha alamkaust iga vajaliku näo jaoks. Loodud alamkaustadesse saate hakata paigutama .skin ja .css faile.

Loome nt näo nimega DefKujundus ning selle alla naha nimega nahk.skin. Naha sisse saate kopeerida .aspx failist kogu kujunduse. Näiteks on meil vaja kujundust tabelivaatele ja kahele erinevale kalendrile. Sellisel juhul võiks nt nahk.skin näha välja järgmine:

<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >

<AlternatingRowStyle BackColor="Blue" />

</asp:GridView>

<asp:Calendar runat="server" BackColor="White" BorderColor="Black"

BorderStyle="Solid" CellSpacing="1" Font-Names="Verdana"

Font-Size="9pt" ForeColor="Black" Height="250px"

NextPrevFormat="ShortMonth" Width="330px">

<SelectedDayStyle BackColor="#333399" ForeColor="White" />

<TodayDayStyle BackColor="#999999" ForeColor="White" />

<DayStyle BackColor="#CCCCCC" />

<OtherMonthDayStyle ForeColor="#999999" />

<NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="White" />

<DayHeaderStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"

Height="8pt" />

<TitleStyle BackColor="#333399" BorderStyle="Solid" Font-Bold="True"

Font-Size="12pt" ForeColor="White" Height="12pt" />

</asp:Calendar>

<asp:Calendar runat="server" SkinId="punaneKalender" BackColor="White"

BorderColor="Black" BorderStyle="Solid" CellSpacing="1"

Font-Names="Verdana" Font-Size="9pt" ForeColor="Black" Height="250px"

NextPrevFormat="ShortMonth" Width="330px">

<SelectedDayStyle BackColor="#333399" ForeColor="White" />

<TodayDayStyle BackColor="#999999" ForeColor="White" />

<DayStyle BackColor="#CCCCCC" />

<OtherMonthDayStyle ForeColor="#999999" />

<NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="White" />

<DayHeaderStyle Font-Bold="True" Font-Size="8pt" ForeColor="red"

Height="8pt" />

<TitleStyle BackColor="#333399" BorderStyle="Solid" Font-Bold="True"

Font-Size="12pt" ForeColor="White" Height="12pt" />

</asp:Calendar>

Nagu näha, on nahas olevad serveri elemendid peaaegu samasugused kui .aspx lehel. Ainukeseks erinevuseks on see, et kogu funktsionaalne osa on eemaldatud, so igasugused meetodid sündmustele reageerimiseks jne. Kujundusi saame teha kahte moodi:

* peakujundus – serveri kontroll, millel puudub kujunduse ID
* spetsiaalkujundus – serveri kontroll, millele on lisatud atribuut SkinID. Vastavad SkinIDd peavad olema samad nii aspx failis kui ka .skin failis.

Kui me soovime seda nahka kasutada oma .aspx lehe kujundamisel, siis tuleb meil:

* lisada lehekülje päisesse (Page element) atribuut Theme=”DefKujundus”
* kustutada kõigilt serveri kontrollidelt kõik kujundusega seonduv
* kontrollidele, mis vajavad spetsiaalkujundust, lisame atribuudi SkinID

Tulemusena on ilus ja lihtne aspx fail, kus on ainult funktsionaalsust puudutavad määrangud.

<%@ Page Language="C#" MasterPageFile="~/tavaline.master"

AutoEventWireup="true" CodeFile="Default.aspx.cs"

Inherits="\_Default" Title="Minu teine leht" Theme="DefKujundus"%>

<asp:Content ID="Content3" ContentPlaceHolderID="cphSisu" Runat="Server">

<asp:TextBox ID="txtAasta" runat="server" AutoPostBack="True"

OnTextChanged="txtAasta\_TextChanged"></asp:TextBox><br />

<asp:Calendar ID="Cal1" runat="server" />

<asp:Calendar ID="Cal2" SkinID="punaneKalender" runat="server" />

<asp:SqlDataSource ID="andmeallikas" runat="server"

ConnectionString="<%$ ConnectionStrings:yhendusTxt %>"

ProviderName="<%$ ConnectionStrings:yhendusTxt.ProviderName %>"

SelectCommand="SELECT TOP 100 ToodeID, Nimi, Hind FROM Toode" />

<asp:GridView ID="gwToode" runat="server" AutoGenerateColumns="true"

DataSourceID="andmeallikas" />

</asp:Content>

Ülesandeid

* Lisa Themes kausta kujundus SinineKujundus, milles määratakse nupu (asp:Button) värv siniseks.
* Katseta seda oma lehel, määrates näo (Theme) lehe algul
* Lisa teine kujunduskaust PunaneKujundus, kus nupu värv määratakse punaseks
* Veendu, et kujunduse vahetamisel muutub nupu värv.
* Püüa kujundust muuta programmi abil (Page.Theme="PunaneKujundus"). Seda on võimalik teha lehe preInit meetodi juures.

### Väärtuste tööaegne meelespidamine

Muutuvate väärtuste meeles pidamiseks on lugematul hulgal erinevaid võimalusi alustades lokaalsetest muutujatest ning lõpetades andmebaasiga. Järgnevalt vaatleme vahendeid, mida pakub ASP.NET

#### Veebi globaalsed muutujad (Application variables)

Läbi Application klassi on võimalik jagada infot kõikjale terves veebirakenduses: see info on saadaval kõigil lehtedel terves rakenduses sõltumata sellest, milline kasutaja neid vaatab.

Application klassi väärtuse salvestamine käib järgmiselt:

Application[”ProgrammiMuutujaNimi”] = ”Mingi väärtus”;

Lugemine käib loomulikult vastupidises omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole seda muutujat veel tekitatud või ei ole sinna veel ühtegi väärtust salvestatud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

Object oAppVar = Application[”ProgrammiMuutujaNimi”];

if (oAppVar != null)

Label1.Text = (string)oAppVar;

Sellised globaalsed muutujad on kasulikud jooksva statistika nt aktiivsete külastajate arvu jms tarbeks.

PS! Kõiki globaalseid muutujaid hoitakse arvuti mälus e. nendes muutujates ei tohiks hoida suuri andmehulki nagu nt tabelid jms

#### Andmete puhverdamine (Cache variable)

Application klass ei ole hea koht suurte andmehulkade hoidmiseks, kuna see raiskab palju arvuti mälu.

Cache klassi kaudu on samuti võimalik jagada infot kõikjale terves veebirakenduses: see info on saadaval kõigil lehtedel terves rakenduses sõltumata sellest, milline kasutaja neid vaatab. Info säilib Caches kuni arvutil jätkub ressurssi info hoidmiseks või kuni teie poolt määratud ajani.

Kõige lihtsam meetod Cache väärtuse salvestamiseks on järgmine:

Cache[”CacheMuutujaNimi”] = ”Mingi väärtus”;

Sellisel kujul salvestatud info säilib mälus kuni ressurssi jätkub, st kui arvuti vaba mälu hulk muutub kriitiliseks, visatakse vähem kasutatud andmed mälust välja. Kui soovite neid andmeid veel kasutada, tuleb andmed uuesti mällu laadida.

Mälust kustutamise aega saame ka ette määrata. Selleks on kaks võimalust: absoluutne aegumise aeg ja dünaamiliselt muutuv aeg.

Absoluutse aegumise aja kasutamiseks tuleb väärtuse Cache salvestamiseks kasutada Cache.Add või Cache.Insert meetodit.

Järgnevalt lisame info cache' i täpselt kümneks minutiks:

Cache.Insert("CacheMuutujaNimi", "Mingi väärtus", null,

DateTime.Now.AddMinutes(10),

System.Web.Caching.Cache.NoSlidingExpiration);

Selles näites salvestame infot cache' i nii, et info säilib 10 minutit viimasest kasutamisest: esmalt säilib info 10 minutiks salvestamisest, kui keegi selle aja jooksul infot kasutab, säilib see veel 10 min jne. Kui keegi pole 10 minuti jooksul infot kasutanud, info kustub.

Cache.Insert("CacheMuutujaNimi", "Mingi väärtus", null,

System.Web.Caching.Cache.NoAbsoluteExpiration,

TimeSpan.FromMinutes(10));

Lugemisel tuleb arvestada, et cache’ is olev info võib kaduda ning tuleks alati mõelda välja kaks sündmuste käiku: juhuks kui info on cache' is ja juhuks kui ei ole.

Object oCacheVar = Cache[”CacheMuutujaNimi”];

if (oCacheVar != null){

// info on Caches ning kasutame seda

Label1.Text = (string)oCacheVar;

} else {

// infot ei ole Caches seega otsime info kusagilt mujalt

// lisaks kasutamisele salvestame selle ka Cachesse

String jutt = ”Mingi väärtus”;

Cache[”CacheMuutujaNimi”] = jutt;

Label1.Text = jutt;

}

#### Veebilehtede puhverdamine (OutputCache atribute)

Lisaks oma info puhverdamisele on võimalik puhvrisse salvestada ka renderdatud veebilehti. Veebilehtede puhverdamine tõstab oluliselt veebirakenduse jõudlust, kuna raamistik ei pea neid lehti uuesti genereerima, vaid saab kasutada varem loodut. Samas tuleb selle meetodiga olla äärmiselt ettevaatlik, kuna puhverdamine tähendab, et uut infot lehele ei tule. Seega ei tohiks seda kasutada kiiresti muutuvat infot sisaldavate lehtede peal.

Puhverdamiseks tuleb veebilehe .aspx faili või ka kasutaja kontrolli ascx algusesse lisada OutputCache element, kus näitate ära, kui kaua tuleb infot Caches hoida ning kuidas seda hoitakse

Näide 1: Hoiame infot puhvris 100 sekundit ning info on sama sõltumata parameetritest, millega lehe poole pöördutakse.

<%@ OutputCache Duration="100" VaryByParam="none" %>

Näide 2: Hoiame infot puhvris 100 sekundit ning infost on üks eksemplar iga pöördunud veebisirvija ja vormil olevate väärtuste koha st. kui üks kasutaja kasutab IE’d ja teine Mozillat, siis salvestatakse puhvrisse üks eksemplar mõlemale sirvijale saadetud lehest. Kui nt IE kasutaja muudab vormil infot või lisab päringuteksti ?id=2, siis salvestatakse lehest uus koopia vastavalt kasutatud parameetritele.

<%@ OutputCache Duration="100" VaryByParam="\*" VaryByCustom="Browser" %>

#### Seansi muutujad (Session variables)

Läbi Session klassi on võimalik jagada infot kõikjale ühe seansi (sessiooni) piires. See info on saadaval kõigil lehtedel, kuhu kasutaja antud külastuse vältel satub. Seega on Session klass hea kasutaja staatust puudutava info hoidmiseks.

Session klassi väärtuse salvestamine käib järgmiselt:

Session[”SessiooniMuutujaNimi”] = ”Mingi väärtus”;

Lugemine käib loomulikult vastupidises järjestus omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole te seda muutujat veel tekitanud või ei ole sinna veel ühtegi väärtust salvestanud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

Object oSessVar = Session[”SessiooniMuutujaNimi”];

if (oSessVar != null)

Label1.Text = (string)oSessVar;

#### Lehelkülje seisund (ViewState variables)

Läbi ViewState klassi on võimalik säilitada infot ühe lehekülje piires. ViewState klassi on mõtet salvestada infot, mida on vaja programmis pruukida, kuid mida ei soovi veebilehel kasutajale näidata. Sellist asja on võimalik korraldada kahte moodi:

* Kasutada hidden välju (input type=”hidden”)
* Kasutada ViewState klassi

ViewState klassil on peidetud välja ees kaks eelist:

* ise ei pea mingit välja tegema ja haldama
* Info on kergelt krüpteeritud, st kui kasutaja vaatab lehe lähtekoodi (View\Source), siis ta ei loe sealt midagi välja.

ViewState klassi väärtuse salvestamine käib järgmiselt:

ViewState[”LeheMuutujaNimi”] = ”Mingi väärtus”;

Lugemine käib loomulikult vastupidises järjestus omistamisega, kuid lugemisel tuleks alati arvestada asjaolu, et võib-olla ei ole te seda muutujat veel tekitanud või ei ole sinna veel ühtegi väärtust salvestanud. Seega tuleks lugemisel esmalt kontrollida, kas üldse on midagi lugeda:

Object oVwVar = ViewState[”LeheMuutujaNimi”];

if (oVwVar != null)

Label1.Text = (string)oVwVar;

Ülesandeid

\* Pane iga salvestusmeetodi (Application, Cache, Session, ViewState) juurde muutuja, mille väärtust suurendatakse igal pöördumisel. Vaata tulemusi ning püüa kindlaks teha sarnasused ja erinevused.

### Veebisaidi turvamine

Kuna maailm on muutunud väga vaenulikuks, oleme sunnitud ka kõik veebilahendused ehitama üles nii, et ilma ennast tutvustamata seal palju teha ei saaks.

Üldjuhul on kõigil veebisaitidel mitu turvataset. Näiteks on olemas mingi avalik vaade ning kohad, kuhu saavad ligi ainult valitud kasutajad. Lisaks sellele võivad ka tuvastatud kasutajad olla erinevate õigustega nt Admin, Müügimees, Ostja. Kõigi nende elementaarsete kuid keeruliste protseduuride jaoks pakub ASP.NET alates versioonist 2.0 välja terve hulga serverikontrolle ning rollihalduri.

Kuna kogu kasutajatehalduse ja ning tuvastamise teema on üsna mahukas ja keeruline siis ei maksa ise jalgratast leiutada ning tasub maksimaalselt ära kasutada ASP.NETi poolt pakutavaid lihtsustavaid vahendeid.

#### Üldised seadistused

Alustuseks tuleks konfiguratsioonifailis öelda, et me soovime kasutajaid autentida ning autentimata kasutajad sisse ei saa. Selleks lisame system.web elemendi alla kaks uut elementi:

* authentication – ütleb, kuidas plaanime kasutajaid tuvastada
* authorization – ütleb, kes saab sisse

Autentimiseks kasutame vormipõhist autentimist, mis võimaldab meil kirjutada oma metoodika kasutajate autentimiseks:

<authentication mode="Forms">

<forms name="Projekt" loginUrl="~/logon.aspx" />

</authentication>

Järgmiseks tuleb anda ligipääs kõigile kasutajatele/kasutajagruppidele keda soovite oma veebile ligi lasta ning keelata ära ligipääs ülejäänutele. Ligipääsu andmiseks või keelamiseks tuleb autorization elemendi sisse lisada allow ja deny alamelemendid vastavalt ligipääsu andmiseks ja keelamiseks.

Kasutajale ligipääsu andmiseks või keelamiseks tuleb lisada allow elemendi sisse atribuut users. Näiteks:

<allow users="mati"/>

Annab ligipääsu kasutajale mati. Lisaks sellele on võimalik kasutada ka kahte erimärki \* (tärn) ning ? (küsimärk). Tärniga tähistatakse kõiki kasutajaid ning küsimärgiga tähistatakse anonüümseid ehk autentimata kasutajaid. Kuni te autoriseerimisreegleid ei ole kehtestanud kehtivad vaikimise määratud autoriseerimisreeglid, mis annavad ligipääsu kõigile kasutajatele:

<allow users="\*"/>

Kõige lihtsamaks muudatuseks autoriseerimisreeglites oleks keelata ligipääs kõigile anonüümsetele kasutajatele lubades kõik ülejäänud (kes on oma kasutajanime ja parooli sisestanud) veebile ligi:

<authorization >

<allow users="\*"/>

<deny users="?"/>

</authorization>

See tekitab muidugi olukorra, kus ilma paroolita ei saa külastaja mitte ühelegi lehele ligi. Kui soovite osaliselt lubada ka anonüümset ligipääsu, tuleks kas:

* kõik turvamist vajavad lehed tõsta teise kausta ja lubada juurkausta kõik ning seada täiendavad piirangud alamkaustadele
* või öelda, et nt default.aspx’i me ei kaitse

Mõlemal juhul tuleb lisada konfiguratsiooni eraldi location element, millega anname teatud asukohale erineva seadistuse. Hetkel läheme seda teed, et lubame kõigil vaadata default.aspx faili.

<location path="default.aspx">

<system.web>

<authorization >

<allow users="\*"/>

<deny users="?"/>

</authorization>

</system.web>

</location>

Nüüd tuleks valmis teha kasutajate autentimise vorm logon.aspx. ASP.NET alates versioonist 2.0 on tore selle poolest, et kasutaja tuvastamise vormi ei pea ise moodustama, selle jaoks on spetsiaalne server element nimega logon. Seega näeb meie login.aspx välja järgmine:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="logon.aspx.cs"

Inherits="logon" %>

<!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>Kasutaja tuvastamine</title>

</head>

<body>

<form id="form1" runat="server">

<asp:Login ID="Login1" runat="server"

OnAuthenticate="TuvastaKasutaja" />

</form>

</body>

</html>

Koodifailis peame realiseerima kasutajate tuvastamise protseduuri. Hetkel lahendame selle ülilihtsalt. Tegelikkuses peaks olema tegemist mingi andmebaasipõhise kasutaja tuvastamisega.

public partial class logon : System.Web.UI.Page

{

protected void TuvastaKasutaja(object sender, AuthenticateEventArgs e)

{

e.Authenticated = Login1.UserName.ToUpper() == "MATI" &&

Login1.Password == "pwd";

}

}

Ülesandeid

* Kaitse parooliga terve kataloog
* Küsi lubatud kasutajanimed ja paroolid andmebaasist

#### Kasutajate haldus

Kasutajatunnuste haldamiseks pakub ASP.NET välja mitmeid põnevaid elemente, mis kõik töötavad MembershipProvider’i peal. Vaikimisi kasutab kõnealune teenusepakkuja ühte konkreetset SQL serveri andmebaasi, kuid seda funktsionaalsust on võimalik ümber ehitada.

Kui teil on veebiserveris piisavalt õigusi siis võite rahus kasutada Microsofti poolset lahendust kasutajateinfo hoidmiseks. Selle info tekitamiseks ja haldamiseks on teil võimalik kasutada WebSite menüüst käsklust „ASP.NET Configuration“, mille abil saate muuta ära nii kasutajate tuvastamise mehhanismi kui ka teha uusi kasutajakontosid ja hallata olemasolevaid kasutajakontosid. Kõik need määrangud paigutatakse teie veebi App\_Data kasuta ASPNETDB nimelisse andmebaasi.

Selle info asukoht on määratud masina seadistustes e. kaustas %windir%\Microsoft.NET\Framework\v2.0.50727\CONFIG asuvas failis machine.config. Asukoht on määratud SQL serveri asukohaga, ühendustekstis LocalSqlServer

<connectionStrings>

<add name="LocalSqlServer"   
 connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"   
 providerName="System.Data.SqlClient" />

</connectionStrings>

Kui te ei saa oma veebis sellist lähenemist andmebaasile kasutada siis võite kogu selle ASP.NET tarkuse ümber suunata oma andmebaasi. Selleks tuleb teil:

* Oma veebi seadistustes muuta ära vaikimisi määratud andmebaasi asukoht.
* Lisada oma andmebaasi vajalikud tabelid, vaated ja protseduurid

Vaikimisi määratud andmebaasi asukoha muutmiseks peate eemaldama süsteemse ühendusteksti ning lisama sama nimega oma ühendusteksti.

<connectionStrings>

<remove name="LocalSqlServer" />

<add name="LocalSqlServer"   
 connectionString="Data Source=localhost\sqlexpress;Initial Catalog=omabaas;Integrated Security=True"

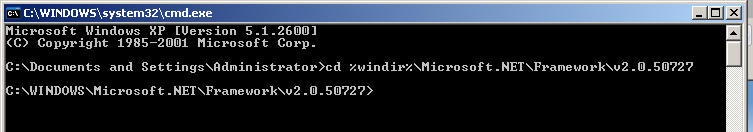
providerName="System.Data.SqlClient" />

</connectionStrings>

Loomulikult eeldab ASP.NET, et see andmebaas on ka vastavalt ette valmistatud – nii, et seal oleksid olemas kõik vajalikud tabelid, vaated ja protseduurid. Nende andmebaasiobjektide tekitamiseks on .NET raamistikuga kaasas üks programm, mis paikneb kaustas %windir%\Microsoft.NET\Framework\v2.0.50727 ning kannab nime aspnet\_regsql.exe. Sellel programmil on olemas nii graafiline kui ka käsurea liides. Soovitan kasutada käsurea liidest, sest see võimaldab anda programmile rohkem infot selle kohta, milliseid tabeleid on vaja tekitada. Graafiline liines (avaneb topeltklõpsuga) tekitab teie andmebaasi kõik tabelid kõigi võimalike teenuste tarbeks.

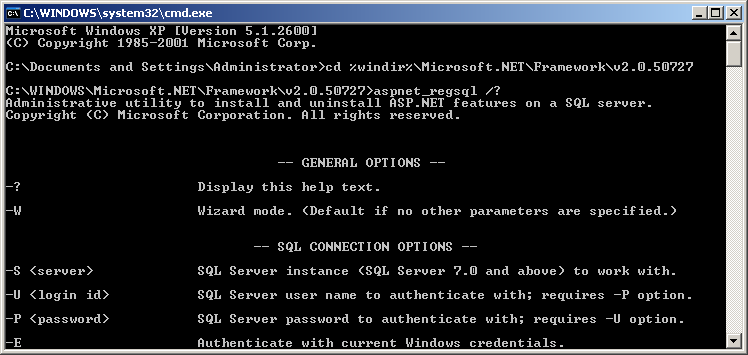
Programmi käivitamiseks käsurealt avage Windowsi Command Prompt. See paikneb start menüüs Programs\Accessories\Command Promt.

Seejärel navigeerige õigesse kausta kasutades käsku cd <kaustanimi>



Nüüd saate programmi käivitada, kuid enne oleks mõistlik lugeda kuidas seda programmi kasutada. Selleks kirjutage käsureale <programminimi> /?

See käsk kuvab teile lühiõpetuse, käsu kasutamiseks.



Sellest õpetusest tuleks leida, kuidas saab ühendust andmebaasiga ning kuidas määrata paigaldatavaid komponente. Järgnevalt on toodud väike tõlgitud väljavõte sellest õpetusest:

-S <server> SQL Server, kus paikneb teie andmebaas

-E Kasutame SQLiga suhtlemisel Windowsi autentimist

-A all|m|r|p|c|w Milliseid funktsioone soovime paigaldada

all: Kõik funktsioonid

m: Kasutajate haldus

r: Rollide haldus

p: Profiilid

c: Isikupärastamine

w: veebisündmused

-R all|m|r|p|c|w Milliseid funktsioone soovime eemaldada.

-d <database> Millisesse andmebaasi funktsioonide jaoks vajalik paigaldada

Ülaltoodust tulenevalt saaksime oma andmebaasi paigaldada kasutajatehalduse järgmise käsureaga:

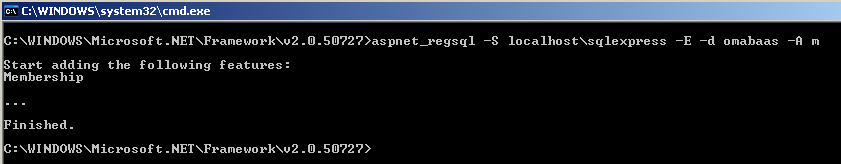
aspnet\_regsql -S localhost\sqlexpress -E -d omabaas -A m

Kui soovite paigaldada rollihaldust kirjutate vastavalt

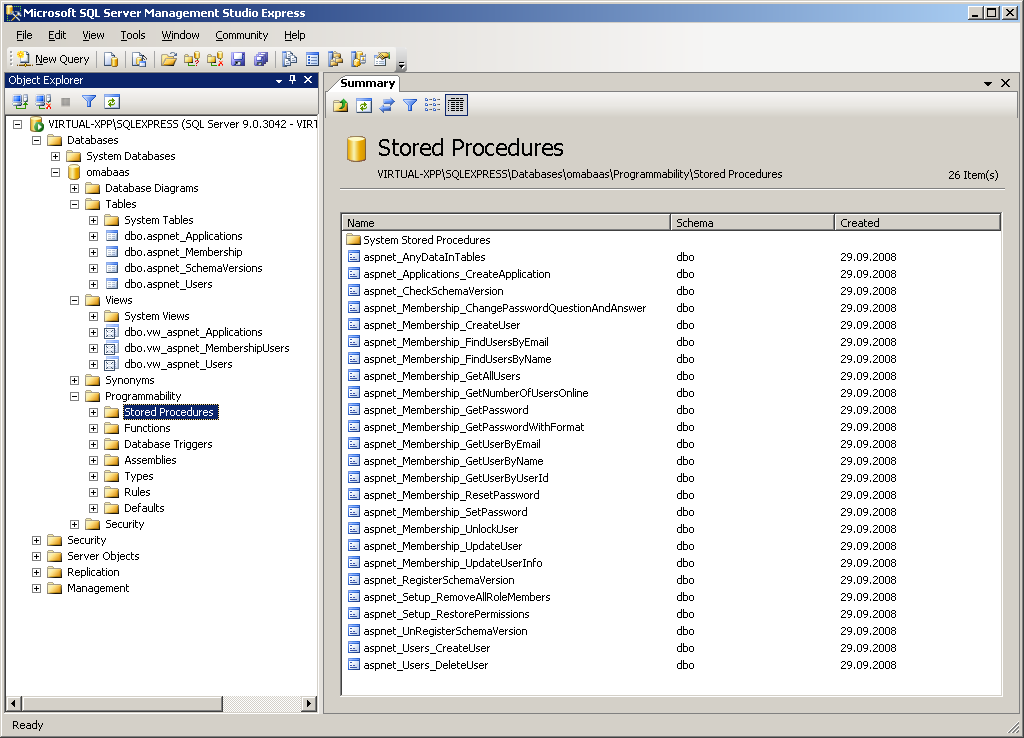
aspnet\_regsql -S localhost\sqlexpress -E -d omabaas -A r

Kui soovite paigaldada nii kasutajate kui ka rollihaldust siis saate seda teha

aspnet\_regsql -S localhost\sqlexpress -E -d omabaas -A mr



Kui kõik õnnestub, siis loetletakse ülesse, millised funktsioonid said paigaldatud. Ning võite vaadata oma andmebaasi, kuhu tekkis terve hulk uusi tabeleid vaateid ja protseduure.



Tekitatud objektide hulk on päris suur. Vaataks neist üle olulisemad:

Tabel: dbo.aspnet\_applications – teie programmid/veebid, mis seda teenust kasutavad

Tabel: dbo.aspnet\_membership – kasutajate andmed

Tabel: dbo.aspnet\_users – kasutajate nimed

Vaade: dbo.vw\_aspnet\_membershipusers – kasutajate andmed koos nimedega

Üldiselt on nii, et ei ole soovitav nende tabelite sisu otse muuta, sest selle tulemusena võib muutuda see info ASP.NETile kasutamiskõlbmatuks. Selle asemel saate infot vaadata ja muuta kasutades ASP.NET poolt pakutavat membership klassi.

Hea uudis on aga see, et kõik ASP.NETiga kaasas olevad, kasutajate haldusega tegelevad serveri elemendid töötavad ☺ ning nende kasutamiseks ei pea te enam ühtegi koodirida kirjutama. Lihtsalt lohistate teid huvitava elemendi oma veebilehele sobivasse kohta ning see töötab.

|  |  |
| --- | --- |
| Elemendi nimi | Kirjeldus |
| Login | Vorm kasutaja autentimiseks |
| LoginView | Vaade vastavalt kasutaja õigustele |
| PasswordRecovery | Vorm kasutajale oma parooli meeldetuletuse saamiseks  Kuna parool saadetakse meiliga siis selle elemendi töölepanek eeldab, et teie kasutada on mõni meiliserver (SMTP teenus) |
| LoginStatus | Lipik, mis näitab, kas kasutaja on autenditud või mitte |
| LoginName | Lipik, mis näitab kasutajanime |
| CreateUserWizard | Võlur uute kasutajate loomiseks |
| ChangePassword | Vorm parooli vahetamiseks |

Lisaks sellele saate kasutajaid hallata läbi VisualStudios asuva configureerimislehel: WebSite\ASP.NET Configuration

Loomulikult surub selline lähenemine peale Microsofti nägemuse kasutajate haldamise funktsionaalsusest ning andmete paigutamisest. Kui soovite päris oma lähenemist siis tuleb teil luua oma teenusepakkuja, mis realiseerib kõik vajalikud funktsioonid just teile vajalikul moel. Näide selle kohta on leheküljel 423.

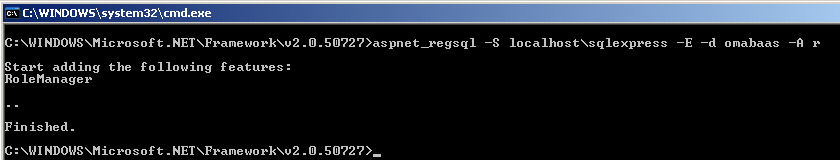
#### Rollide kasutamine ligipääsu kontrollimisel

Kuni veebil on kasutajaid vähe on kasutajanime järgi õiguste andmine mõeldav. Kuid kui kasutajaid tekib palju või kasutajad vahelduvad tihti muutub sellise süsteemi haldamine äärmiselt vaevarikkaks ning märksa lihtsam on ligipääsu anda vastavalt sellele, milliseid rolle antud kasutaja peab teie veebis täitma.

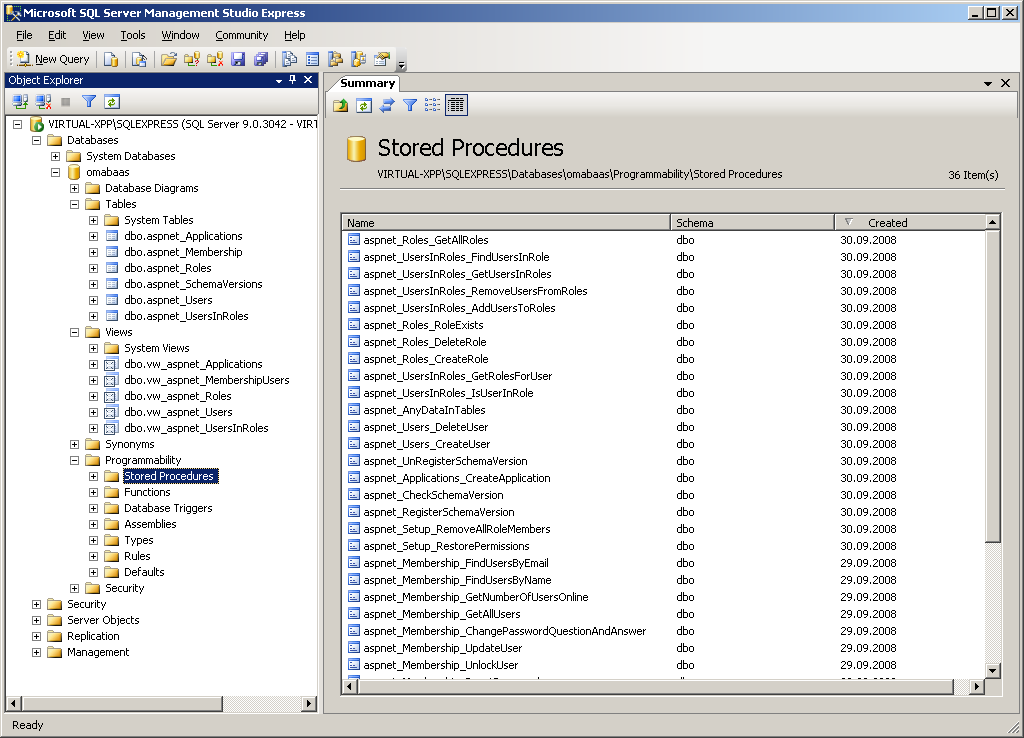
Selleks, et kasutada rollipõhist ligipääsu tuleb iga kasutaja juures määrata, milliseid rolle kasutaja täidab. Lisaks sellele, kui hoiate kõiki andmeid oma andmebaasis peate täiendama ka oma andmebaasi rollihalduseks vajalike tabelite, vaadete ja protseduuridega või siis kirjutama oma teenusepakkuja rollihalduseks (vaata lk 425).

Rollihalduseks vajalike andmebaasiobjektide lisamiseks saate kasutada taas aspnet\_regsql.exe programmi.

aspnet\_regsql -S localhost\sqlexpress -E -d omabaas -A r



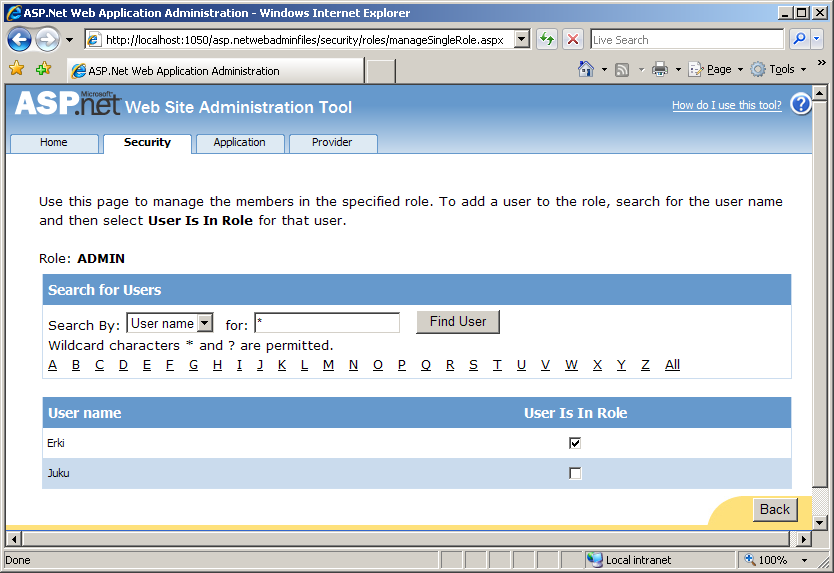
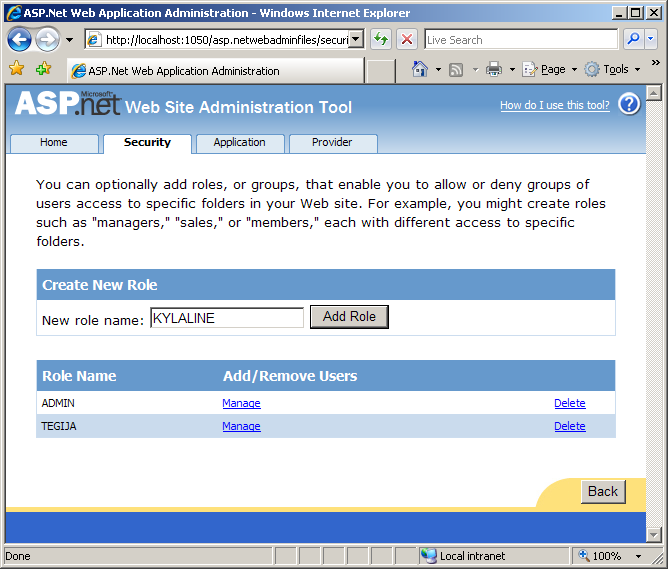
Selle tulemusena lisatakse andmebaasi mõned uued tabelid, vaated ja protseduurid.



Pale struktuuri loomist on vaja ASP.NETile kuulutada, et kasutate rollihaldust e. teha väike muudatus web.config failis lisades system.web elemendi alla elemendi rolemanager:

<roleManager enabled="true" />

Peale seda saate tekitada kõik vajalikud rollid ning määrata kasutajate rollikuuluvuse. Nende tegevuste tarbeks saate luua oma halduslehe, kasutades Roles klassi poolt pakutavaid meetodeid või kasutada veebi konfigureerimisutiliiti WebSite\ASP.NET Configuration



Õiguste jagamiseks saate kasutada web.config faili autorization elemendi alamelemente allow ja deny, määrates seekord users atribuudi asemel roles atribuudi. Näiteks anname ligipääsu kõigile kasutajatele, kes kuuluvad ADMIN rolli ning keelame ligipääsu kõigile teistele.

<authorization >

<allow roles="ADMIN"/>

<deny users="\*"/>

</authorization>

### Andmetega manipuleerimine

Andmetega manipuleerimiseks ASP.NET keskkonnas on alates versioonist 3.5 kaks võimalust:

* Kasutada vanamoelist lähenemist ADO.NET vahendite abil
* Kasutada uut ja põnevat lähenemist LINQ abil

Nende kahe meetodi erinevus seisneb andmebaasiga suhtlemise metoodikas. Kui kõik suhtlusreeglid on kokkulepitud siis selle andmeallika kasutamine andmeid kuvavate elementide peal käib mõlemal juhul ühte moodi.

#### Andmete kasutamine ADO.NET abil

Selleks, et andmetele ligi pääseda, tuleb luua ühendus andmebaasiga. Ühenduse loomiseks on vaja andmebaasi. Andmebaasile viitamiseks kasutatakse ühendusteksti (ConnectionString). Kuna peale rakenduse valmimist võib administraator tõsta andmebaasi mõnda teise serverisse või anda andmebaasile teise nime või muuta turvameetmeid, peab ühendustekst olema administraatorile ehk rakenduse hooldajale lihtsasti kättesaadav. Seega ei saa ühendusteksti kirjutada tekstina otse koodi. Parim koht selle hoidmiseks on konfiguratsioonifail web.config. ASP.NET konfiguratsioonis on loodud isegi eraldi konteinerelement ühendustekstide hoidmiseks. configuration elemendi alamelemendi nimeks on connectionStrings.

Ühendustekst nimega yhendusTxt võiks välja näha midagi sarnast järgnevaga:

<connectionStrings>

<add name="yhendusTxt" connectionString="packet size=4096;uid=KASUTAJANIMI; pwd=PAROOL;data source=dotnet;persist security info=False;initial catalog=KASUTAJANIMI" providerName="System.Data.SqlClient" />

</connectionStrings>

Kui on kasutada rohkem kui üks andmebaas, võib siia ühendustekste juurde lisada. Suurema turvalisuse saavutamiseks on võimalik ka teatud osade sh connectionStrings krüpteerimine konfiguratsiooni failis.

Ühendusteksti kasutamiseks tuleb koodis sellele viidata. Selleks on kaks võimalust.

Esiteks võime seda teha aspx lehel andmeallika kirjelduses.

<asp:SqlDataSource id="andmeallikas" runat="server" ProviderName="System.Data.SqlClient" ConnectionString=’<%#

ConfigurationManager.ConnectionStrings("yhendusTxt").ConnectionString %>’

SelectCommand="SELECT TOP 100 ToodeID, Nimi, Hind FROM toode"/>

Sellest pikast lausest on olemas ka lühendatud kuju:

<asp:sqldatasource id="andmeallikas " runat="server" ProviderName="System.Data.SqlClient" ConnectionString="<%$ ConnectionStrings:yhendusTxt %>" SelectCommand="SELECT TOP 100 ToodeID, Nimi, Hind FROM toode"/>

Saadud andmete näitamiseks võimalik kasutada nt tabeli vaadet (GridView).

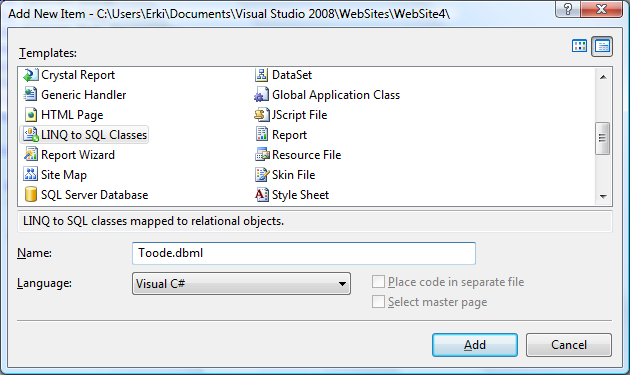
<asp:GridView ID="gw" runat="server" AutoGenerateColumns="True" DataSourceID="andmeallikas" EmptyDataText="Pole midagi näidata" />

Nagu näete, on võimalik lugeda andmebaasist andmeid tabelivaatesse ilma ridagi koodi kirjutamata. Kui kasutate veebilehe loomiseks graafilisi abivahendeid nagu VisualStudio, genereeritakse teile ka eelpool kirjeldatud read aspx failidesse.

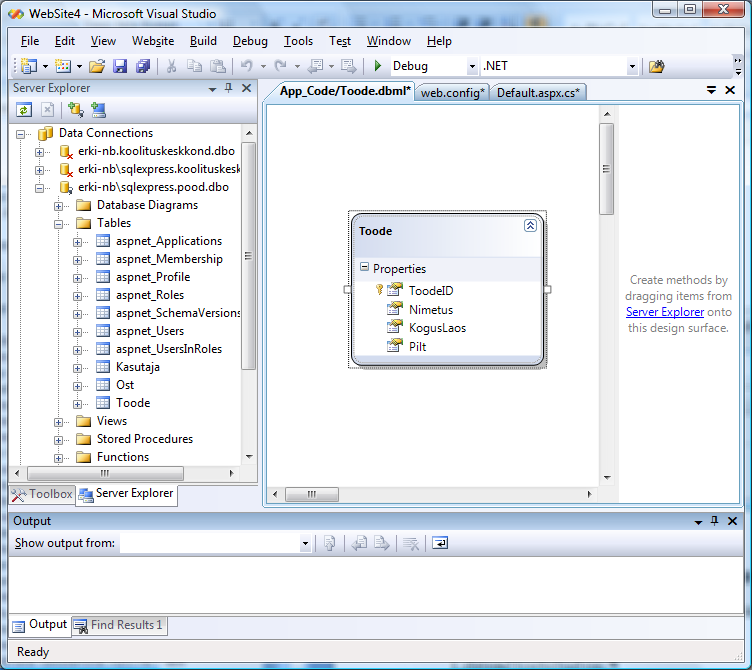
#### Andmete kasutamine LINQ abil

Andmete leidmiseks LINQ abil tuleb esmalt luua App\_Code kausta „LINQ to SQL“ klass, mis hakkab korraldama andmevahetust veebirakenduse ja andmebaasi vahel.

Selleks lisage oma veebi App\_Code kausta uus item, mis on „LINQ to SQL Classes“ tüüpi.



Avaneb LINQ to SQL klassi disainerivaade, kus saate lohistada kõik teile olulised tabelid, koos nendevaheliste seostega LINQ to SQL klassi. Näiteks haarame ServerExplorerist oma andmebaasis paikneva toodete tabeli ning lohistame disainerisse:

Vajadusel saate lohistada sellele skeemile tabeleid juurde ning määrata keerukamaid meetodeid nii andmete leidmiseks kui muutmiseks. Samas kui teile piisab sellisest lihtsustatud ligipääsust siis võite selle klassi ära salvestada ja hakata teda kasutama andmetega manipuleerimisel.

|  |  |
| --- | --- |
| Selles disainerivaates näete te selle klassi graafilist kuju. Klassi tegelik kood paikneb selle disainifaili taba olevas koodifailis <klassinimi>.designer.cs  Lisaks on seal ka fail <klassinimi>.dbml.layout, milles on XML kujul see sama graafiline vaade teie loodud klassile.  Visates kiire pilgu loodud koodile võib selles välja lugeda, et tegemist on maskeeringuga, kus klass Toode maskeerib ära andebaasis oleva tabeli Toode, koos kõigi selle tabeli omaduste ja muutmisvõimalustega ning klass ToodeDataContext on andmeallikas LINQ päringute tegemiseks, mis kasutab seda maskeeritud Toode tabelit. |  |

#pragma warning disable 1591

//-------------------------------------------------------------------------

// <auto-generated>

// This code was generated by a tool.

// Runtime Version:2.0.50727.1433

//

// Changes to this file may cause incorrect behavior and will be lost

// if the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.Linq;

using System.Data.Linq.Mapping;

using System.Linq;

using System.Linq.Expressions;

using System.Reflection;

[System.Data.Linq.Mapping.DatabaseAttribute(Name="pood")]

public partial class ToodeDataContext : System.Data.Linq.DataContext

{

private static System.Data.Linq.Mapping.MappingSource mappingSource = new AttributeMappingSource();

#region Extensibility Method Definitions

partial void OnCreated();

partial void InsertToode(Toode instance);

partial void UpdateToode(Toode instance);

partial void DeleteToode(Toode instance);

#endregion

public ToodeDataContext() :

base(global::System.Configuration.ConfigurationManager.ConnectionStrings["poodConnectionString"].ConnectionString, mappingSource)

{

OnCreated();

}

public ToodeDataContext(string connection) :

base(connection, mappingSource)

{

OnCreated();

}

public ToodeDataContext(System.Data.IDbConnection connection) :

base(connection, mappingSource)

{

OnCreated();

}

public ToodeDataContext(string connection, System.Data.Linq.Mapping.MappingSource mappingSource) :

base(connection, mappingSource)

{

OnCreated();

}

public ToodeDataContext(System.Data.IDbConnection connection, System.Data.Linq.Mapping.MappingSource mappingSource) :

base(connection, mappingSource)

{

OnCreated();

}

public System.Data.Linq.Table<Toode> Toodes

{

get

{

return this.GetTable<Toode>();

}

}

}

[Table(Name="dbo.Toode")]

public partial class Toode : INotifyPropertyChanging, INotifyPropertyChanged

{

private static PropertyChangingEventArgs emptyChangingEventArgs = new PropertyChangingEventArgs(String.Empty);

private int \_ToodeID;

private string \_Nimetus;

private int \_KogusLaos;

private System.Data.Linq.Binary \_Pilt;

#region Extensibility Method Definitions

partial void OnLoaded();

partial void OnValidate(System.Data.Linq.ChangeAction action);

partial void OnCreated();

partial void OnToodeIDChanging(int value);

partial void OnToodeIDChanged();

partial void OnNimetusChanging(string value);

partial void OnNimetusChanged();

partial void OnKogusLaosChanging(int value);

partial void OnKogusLaosChanged();

partial void OnPiltChanging(System.Data.Linq.Binary value);

partial void OnPiltChanged();

#endregion

public Toode()

{

OnCreated();

}

[Column(Storage="\_ToodeID", AutoSync=AutoSync.OnInsert, DbType="Int NOT NULL IDENTITY", IsPrimaryKey=true, IsDbGenerated=true)]

public int ToodeID

{

get

{

return this.\_ToodeID;

}

set

{

if ((this.\_ToodeID != value))

{

this.OnToodeIDChanging(value);

this.SendPropertyChanging();

this.\_ToodeID = value;

this.SendPropertyChanged("ToodeID");

this.OnToodeIDChanged();

}

}

}

[Column(Storage="\_Nimetus", DbType="NVarChar(50) NOT NULL", CanBeNull=false)]

public string Nimetus

{

get

{

return this.\_Nimetus;

}

set

{

if ((this.\_Nimetus != value))

{

this.OnNimetusChanging(value);

this.SendPropertyChanging();

this.\_Nimetus = value;

this.SendPropertyChanged("Nimetus");

this.OnNimetusChanged();

}

}

}

[Column(Storage="\_KogusLaos", DbType="Int NOT NULL")]

public int KogusLaos

{

get

{

return this.\_KogusLaos;

}

set

{

if ((this.\_KogusLaos != value))

{

this.OnKogusLaosChanging(value);

this.SendPropertyChanging();

this.\_KogusLaos = value;

this.SendPropertyChanged("KogusLaos");

this.OnKogusLaosChanged();

}

}

}

[Column(Storage="\_Pilt", DbType="VarBinary(MAX)", UpdateCheck=UpdateCheck.Never)]

public System.Data.Linq.Binary Pilt

{

get

{

return this.\_Pilt;

}

set

{

if ((this.\_Pilt != value))

{

this.OnPiltChanging(value);

this.SendPropertyChanging();

this.\_Pilt = value;

this.SendPropertyChanged("Pilt");

this.OnPiltChanged();

}

}

}

public event PropertyChangingEventHandler PropertyChanging;

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void SendPropertyChanging()

{

if ((this.PropertyChanging != null))

{

this.PropertyChanging(this, emptyChangingEventArgs);

}

}

protected virtual void SendPropertyChanged(String propertyName)

{

if ((this.PropertyChanged != null))

{

this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

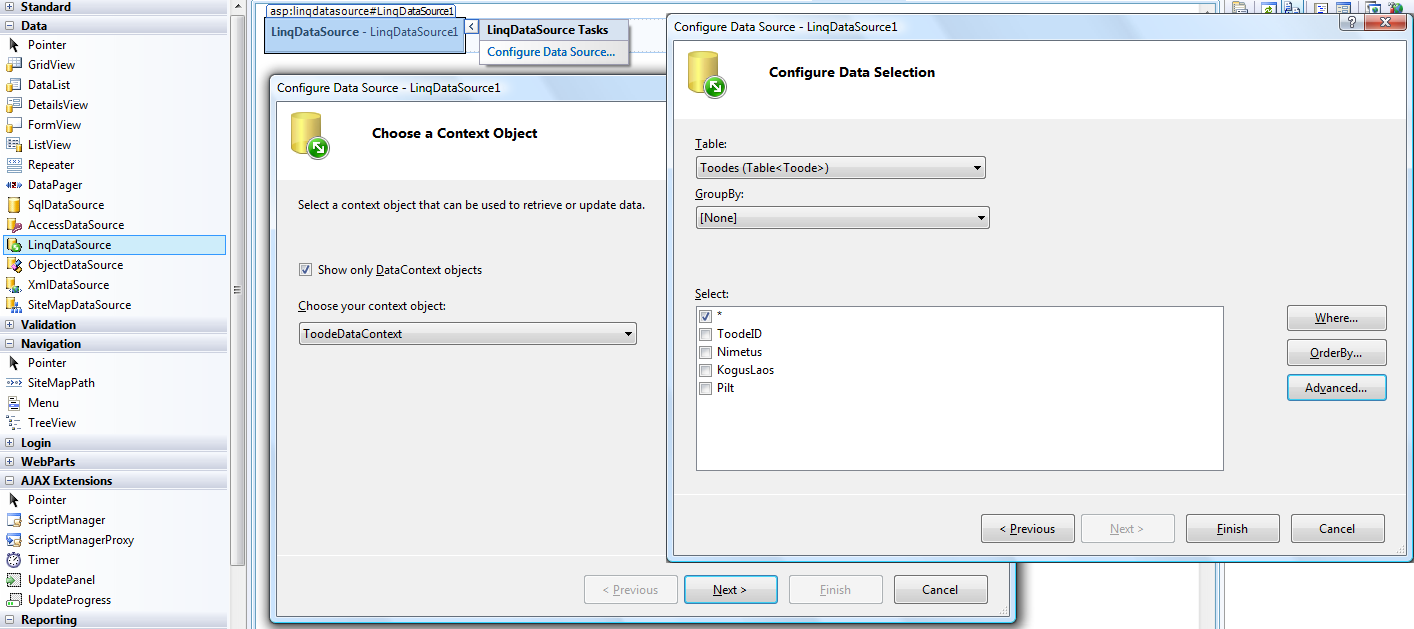
}

}

}

#pragma warning restore 1591

Loodud klassi kasutamiseks andmeallikana tuleb lisada veebilehele LinqDatasource. Lohistades vastava elemendi Toolbox’ilt oma veebilehele, ning seadistades teda läbi seadistusvõluri, mis paikneb Configure Data Source käsu all.



Kui peale seda protsessi vaatate oma veebilehe koodi siis leiate, et tulemuseks element asp:LinqDataSource, mis seab ContextTypeName atribuudi abil kokku loodud andmeallika ning LINQ to SQL klassi:

<asp:LinqDataSource ID="LinqDataSource1" runat="server"

ContextTypeName="ToodeDataContext" TableName="Toodes">

</asp:LinqDataSource>

Andmeallik kasutamiseks GridView peal tuleb vaid viidata loodud andmeallikale:

<asp:GridView ID="GridView1" runat="server"   
 AutoGenerateColumns="True"

DataKeyNames="ToodeID"   
 DataSourceID="LinqDataSource1" />

Kokkuvõtteks võiks öelda, et LINQ kasutamine vajab esmalt pisut rohkem ettevalmistust, kui traditsiooniline ADO.NET, kuid kui ettevalmistused tehtud (LINQ to SQL klassi loomise teel) on selle kasutamine oluliselt lihtsam kui seda on ADO.NET kasutamine. Lisaks sellele on loodud ToodeDataContext kasutatav ka programmelt e. C# koodi abil. See võimaldab teil luua nt programme, mis vaatavad selle toodete tabeli rea kaupa üle ning teevad vastavalt vajadusele seal muudatusi jne.

Samuti on hiljem selgub, et andmete poole pöördumist või nendega manipuleerimist on vaja täiustada siis saate seda teha ühes keskses kohas (LINQ to SQL klassis) ning tehtud muudatustega arvestatakse kõigil veebilehtedel, mis neid andmeid kasutasid. Kasutades traditsioonilist ADO.NET lähenemist peate te muudatusi tegema eraldi igal lehel, mis andmetega tegeleb!

#### Andmete kuvamine lihtsate loetelude abil

Lihtsateks loeteludeks nimetatakse kasutajaliidese elemente, mis võimaldavad vaid ühe väärtuse loetelu.

Lihtsad loetelud on:

* AdRotator – võimaldab kuvada bannereid. Iga kord kui kasutaja tuleb lehele, valitakse juhuslikult uus pilt kuvamiseks
* BulletedList – täppidega loetelu
* DropDownList – ripploend
* ListBox – kerimisribadega loetelu kastis
* RadioButtonList – raadionuppudega loetelu, kust saab valida vaid ühe väärtuse
* CheckBoxList – valikukastidega loetelu, kust saab valida mitmeid väärtuseid

Selleks, et lihtsaid loetelusid kasutada, tuleb määrata ära DataSource või DataSourceID omadus, millega näitate, kust tulevad andmed.

Lisaks on soovitav täita väljad DataTextField, mis näitab milline väli andmetest kuvatakse kasutajale ning DataValueField e milline väli on objekti väärtuseks, kui kasutaja midagi valib.

Kõik loetelus olevad elemendid on programselt saadaval läbi Items kollektsiooni ning kasutaja poolt tehtud valiku saate teada läbi SelectedIndex omaduse. Kui soovite igale kasutaja valikule reageerida, võite teha protseduuri sündmuse SelectedIndexChanged sündmuse tarbeks.

Järgnevalt tekitame toodetest täppidega loetelu:

<asp:BulletedList ID="loetelu " runat="server" DataSourceID="andmeallikas" DataTextField="Nimi" />

#### Andmete kuvamine keerukate loetelude abil

Keerukateks loeteludeks nimetatakse kasutajaliidese elemente, mis andmete kuvamiseks tekitavad hulga alamelemente.

Keerulised loetelud on:

* GridView – võimaldab näidata andmeid tabeli kujul, pakkudes lihtsaid vahendeid andmete sorteerimiseks, filtreerimiseks ja muutmiseks.
* DetailsView – võimaldab andmeid kirje kaupa vaadata, muuta ja juurde lisada. Andmed kuvatakse tabeli kujul, väljad üksteise all.
* FormView – sisuliselt nagu DetailsView, kuid andmete kuvamise loogika tuleb läbi põhjade (Template) ise paika panna.
* Repeater – Kõige abstraktsem loetelu, võimaldab põhjade abil leida andmetele just sellise kuju nagu teil vaja.

#### Hierarhiliste andmete kuvamine

Hierarhilised loetelud on mõeldud hierarhiliste andmete näitamiseks nagu XML failis olevad andmed ja vanema – lapse suhted. Hierarhilised loetelud on:

* TreeView – andmete esitus puu kujul
* Menu – menüü rakenduses navigeerimiseks. Võib olla nii puu kujul kui ka puu kujul.

### WebParts

Siiani oleme kogu veebi kujunduse ise valmis teinud, kuid lisaks sellele on võimalik veebileht ehitada üles ka nii, et erinevate osade paigutuse saab valida kasutaja ise. Me võime seda funktsionaalsust lubada kõigile, või piirata nt administraatorite grupile. Oluline on see, et me saame lihtsate vahenditega anda kasutajale võimaluse veebi väljanägemise muutmiseks. WebPart’id on kontrollide kogumik, mis seda funktsionaalsust võimaldab.

WebPart’ide struktuur on kolme kihiline: isikupärastamine, kasutajaliidese struktuuri komponendid ja kasutajaliidese komponendid ise.

Isikupärastamine on WebPart’ide struktuuri alustalaks. See võimaldab kasutajatel muuta – isikupärastada – WebPartide paigutust, väljanägemist ja käitumist. Tehtud muudatused on püsivad st nad jäävad kehtima ka siis kui kasutaja käib vahepeal teistel lehtedel või paneb üldse veebisirvija kinni.

Struktuuri elemendid baseeruvad isikupärastamisel ning pakuvad baasstruktuuri ja teenused kõigile WebPartidele. Üks struktuuri element, mis peab olema igal WebParte kasutaval leheküljel on WebPartManager. Kuigi see kontroll ei ole kunagi nähtaval on tal üks kriitiline ülesanne – koordineerida kõigi WebPartide olekut leheküljel – ta haldab kõiki tsoone, kus on võimalik WebParte hoida ning seda, millised WebPart’id ühes või teises on olemas ning milline on nende olek. Lisaks sellele koordineerib ta ka WebPartide vahelist suhtlemist.

Teine struktuurielement, mis peab alati olemas olema on tsoon e. koht kuhu on võimalik WebParte paigutada.

Järgnevalt ülevaade WebPart’idega seotud kontrollidest:

|  |  |
| --- | --- |
| WebPartManager | Haldab lehel WebPart kontrolle. Igal lehel on vaja ühte ja ainult ühte WebPartManager’i |
| CatalogZone | Sisaldab CatalogPart kontrolle. Seda kasutatakse kasutajale WebPartide kataloogi loomiseks |
| EditorZone | Sisaldab EditorPart kontrolle. Selles tsoonis saab kasutaja WebParte muuta |
| WebPartZone | Üldine WebPartide paigutus lehel. Selliseid tsoone võib lehel olla rohkem kui üks. |
| ConnectionsZone | Sisaldab WebPartConnection kontrolle ning pakub kasutajaliidest ühenduste haldamiseks. |
| WebPart | Kasutajaliidese element, mille paigutuse üle saab kasutaja otsustada |
| CatalogPart | Sisaldab loetelu WebPartidest, mida kasutaja saab lehele lisada |
| WebPartConnection | Tekitab ühenduse kahe WebParti vahele. Ühest neist WebPartidest saab klient ja teisest andmete pakkuja |
| EditorPart | Lubab kasutajal muuta WebParti põhiomadusi |

### Veebiteenused

Veebiteenus on protseduur, mis asub kauges serveris. Sisuliselt on tegemist tavalise klassiga, mille mõned meetodid on märgistatud kui veebist kasutatavad meetodid.

Veebiteenused on uus põlvkond protseduuri kaugkutseid, mida varem on püütud realiseerida COM, COM+ ja RPC abil. Veebiteenused võimaldavad lihtsalt korraldada rakenduste ja organisatsioonide vahelist andmevahetust, kuna andmevahetus käib XML kujul SOAP kirjadega ning ei ole vahet, kes ja kuidas need kirjad kokku paneb, peaasi, et nad on korrektses vormingus.

#### Veebiteenuste tegemine

Veebiteenuste faililaiendiks on .asmx ning selle faili sisu on ülimalt lihtne. Lisame näiteks faili Service.asmx. Selle faili sees tuleks öelda, et tegemist on veebiteenusega, mis keeles selle teenuse koodi plaanime kirjutada, kuhu (mis faili ja klassi) kood on salvestatud:

<%@ WebService Language="C#" CodeBehind="~/Service.cs" Class="Service" %>

Koodi keerukus sõltub loomulikult teenuse funktsionaalsusest. Siinsel juhul teeme ühe hästi lihtsa teenuse, millel on vaid üks meetod „Tere”, mis välja kutsudes ütleb „Teretulemast veebiteenuste maailma!”

using System;

using System.Web;

using System.Web.Services;

using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]

public class Service : System.Web.Services.WebService

{

[WebMethod]

public string Tere() {

return "Teretulemast veebiteenuste maailma!";

}

}

Kuna tegemist on XML veebiteenusega, tuleb meil ära määrata nimeruum. Niisama katsetamiseks võib kasutada tempuri.org’i, kuid kui on omal mõni domeen kasutada, mida ise ka kontrollite, siis võite seda nime kasutada.

Veebiteenuse klass tuleb pärida klassist System.Web.Services.WebService, mis tagab kogu vajaliku funktsionaalsuse.

Kõik meetodid, mida tohib veebist välja kutsuda, tuleb märgistada [WebMethod] atribuudiga.

Veebiteenuse seadistamine käib web.config faili abil nagu ASP.NET rakendusel, kogu ülejäänud programmeerimine on täpselt nagu tavalisel klassil.

#### WCF teenuste tegemine

|  |  |
| --- | --- |
| WCF – Windows Communication Foundation on uus meetod programmidevahelise infovahetuse korraldamiseks. Selliseid teenuseid saab teha alates raamistikust 3.0. Uue WCF teenuse loomiseks lisage oma veebi uus objekt, mis on „WCF Service“ tüüpi: |  |

Selle tulemusena tekitatakse teile kaks faili. <teenusenimi>.svc, mille sisu on väga minimalistlik:

<%@ ServiceHost Language="C#" Debug="true" Service="Tere" CodeBehind="~/App\_Code/Tere.cs" %>

Ning selle teenuse nägu (paigutatakse App\_Code kausta) <teenusenimi>.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

// NOTE: If you change the interface name "ITere" here, you must also update the reference to "ITere" in Web.config.

[ServiceContract]

public interface ITere

{

[OperationContract]

void DoWork();

}

Nägu on selles, et tutvustada oma teenust selle teenuse kasutajatele. Lisaks sellele peate selle näo järgi tegema ka ühe klassi, mis hakkab reaalset teenust pakkuma! Nägu on sisuliselt leping teenusepakkuja ja teenuse tarbija vahel.

Loome näiteks veebiteenuse, mis oskab teenusekasutajat tervitada. Seda nii anonüümselt kui ka nimeliselt. Selleks teen esmalt väikesed muudatused selles teenuse näos. Ütleme, et meie teenusel saab olema kaks tervitamise meetodit. Üks neist vajab parameetrit ja teine mitte ning mõlemad tagastavad tervituse teksti kujul:

[ServiceContract()]

public interface IGeometryService

{

[OperationContract]

string Tervita();

[OperationContract]

string Tervita(string nimi);

}

Järgmiseks loome klassi, mis annab sellele näole funktsionaalsuse:

#### Veebiteenuse kasutamine

Veebiteenust saab kasutada igas .NET rakenduses ning on variant seda kasutada ka varem loodud programmeerimisvahenditega nagu nt VBA (Visual Basic for Applications).

.NET rakenduses kasutamiseks tuleb teha viide veebiteenusele. Kõige mugavam on seda teha VisualStudioga valides menüüst Website\WebReference, näitate ära teenuse aadressi nt <http://localhost:52876/VeebiTeenus/Service.asmx> ning klõpsate nupule AddReference.

Samas võib seda protsessi teha ka ilma VisualStudio abita.

Selleks tuleb esmalt lisada oma veebikausta alamkaust App\_WebReferences, mille alla teete teenuse nimega alamkaust. Alamkausta nimi peaks olema selline, mida teil on hiljem mugav koodis kasutada nt TereTeenus ning sinna alla tulevad kõik ülejäänud failid, mida teenuse kasutamiseks vaja.

Esmalt on teil vaja teenuse päris nimega wsdl faili. Meie näite puhul oleks selleks Service.wsdl. Tegemist on XML failiga, mis räägib raamistikule, kuidas selle teenusega tuleb käituda. Selle faili sisu saate kopeerida teenuse juurest pannes asmx faili järgi võtme ?wsdl nt

<http://localhost:52876/VeebiTeenus/Service.asmx?wsdl>

Meie lihtsa teenuse puhul genereerub sealt järgmine XML:

<?xml version="1.0" encoding="utf-8"?>

<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://tempuri.org/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:types>

<s:schema elementFormDefault="qualified"

targetNamespace="http://tempuri.org/">

<s:element name="Tere">

<s:complexType />

</s:element>

<s:element name="TereResponse">

<s:complexType>

<s:sequence>

<s:element minOccurs="0" maxOccurs="1" name="TereResult"

type="s:string" />

</s:sequence>

</s:complexType>

</s:element>

</s:schema>

</wsdl:types>

<wsdl:message name="TereSoapIn">

<wsdl:part name="parameters" element="tns:Tere" />

</wsdl:message>

<wsdl:message name="TereSoapOut">

<wsdl:part name="parameters" element="tns:TereResponse" />

</wsdl:message>

<wsdl:portType name="ServiceSoap">

<wsdl:operation name="Tere">

<wsdl:input message="tns:TereSoapIn" />

<wsdl:output message="tns:TereSoapOut" />

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="Tere">

<soap:operation soapAction="http://tempuri.org/Tere"

style="document" />

<wsdl:input>

<soap:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">

<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="Tere">

<soap12:operation soapAction="http://tempuri.org/Tere"

style="document" />

<wsdl:input>

<soap12:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap12:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="Service">

<wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">

<soap:address

location="http://localhost:52876/VeebiTeenus/Service.asmx" />

</wsdl:port>

<wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">

<soap12:address

location="http://localhost:52876/VeebiTeenus/Service.asmx" />

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

Teiseks on vaja lisada veebiteenuse nimega .disco fail. See fail räägib raamistikule, kuidas seda teenust edaspidi leida. Nagu .wsdl fail on ka .disco XML fail ning selle faili sisu saate teenuse juurest, lisades .asmx järgi võtme ?disco. Meie näite puhul oleks selleks

<http://localhost:52876/VeebiTeenus/Service.asmx?disco> ning sealt avaneks järgmine XML:

<?xml version="1.0" encoding="utf-8"?>

<discovery xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns="http://schemas.xmlsoap.org/disco/">

<contractRef ref="http://localhost:52876/VeebiTeenus/Service.asmx?wsdl"

docRef="http://localhost:52876/VeebiTeenus/Service.asmx"

xmlns="http://schemas.xmlsoap.org/disco/scl/" />

<soap address="http://localhost:52876/VeebiTeenus/Service.asmx"

xmlns:q1="http://tempuri.org/" binding="q1:ServiceSoap"

xmlns="http://schemas.xmlsoap.org/disco/soap/" />

<soap address="http://localhost:52876/VeebiTeenus/Service.asmx"

xmlns:q2="http://tempuri.org/" binding="q2:ServiceSoap12"

xmlns="http://schemas.xmlsoap.org/disco/soap/" />

</discovery>

Kolmandaks tuleb luua teenuse nimega .discomap fail, mis räägib, miks te need kaks eelmist faili tegite. Sinna tuleb lisada kaks elementi: esimene ütleb, kuhu faili panite kasutamise info e. lepingu so wsdl fail; ning teine ütleb, kus on info teenuse leidmiseks so. disco fail. Nagu kõik eelmised, on ka .discomap XML fail ning meie näite puhul võiks tema sisu olla järgmine:

<?xml version="1.0" encoding="utf-8"?>

<DiscoveryClientResultsFile

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Results>

<DiscoveryClientResult

referenceType="System.Web.Services.Discovery.ContractReference"

url="http://localhost:52876/VeebiTeenus/Service.asmx?wsdl"

filename="Service.wsdl" />

<DiscoveryClientResult

referenceType="System.Web.Services.Discovery.DiscoveryDocumentReference"

url="http://localhost:52876/VeebiTeenus/Service.asmx?disco"

filename="Service.disco" />

</Results>

</DiscoveryClientResultsFile>

Sellega on viide loodud. Järgmiseks tuleb seda teenust kasutada. Selleks loome kohas, kus soovime teenust kasutada, selle teenuse instantsi ning kutsume välja sobiva meetodi:

TereTeenus.Service srv = new TereTeenus.Service();

Label1.Text = srv.Tere();

Nii lihtsalt see lähebki. Loodetavasti tekib rohkelt mõtteid, kus selliseid teenuseid oleks võimalik kasutada. Mõelge, kui tore oleks järgmine stsenaarium: loote veebipoodi ning teil on vaja teada, kui palju klient peab tellitud toodete pealt makse maksma. Selle välja selgitamiseks kasutate maksuameti poolt pakutavat maksuarvutusteenust. Seejärel on vaja teada, kui palju maksab tellitud toodete kliendile viimine: selleks kasutate mõne kullerfirma hinnapäringuteenust jne. Nii on võimalik väga lihtsa vaevaga pakkuda oma rakenduses väga keerulist funktsionaalsust.

PS! Ka Google otsing on saadaval veebiteenusena ;)

### IIS

IIS teenus on olemas kõigil Windowsi serveritel, kõrgema klassi kodu ja ärikasutuse tööjaama Windowsidel nagu (Windows 2000 Pro, Windows XP Pro, Vista Business, Windows 7 Home Premium ja Professional jne).

## Kokkuvõte

ASP.NET abil saab mitmesuguseid veebilehestikke kokku panna. Siin on kasutada küllalt palju seadistatavaid valmiskomponente ning nende kaudu saab standardolukordi lahendada mugavamalt, kui kõike ise algusest kirjutades. Samas on komponendid piisavalt paindlikud, et anda tulemuseks just selline HTML nagu rakenduse jaoks tarvilik on, nii et pole vaja muretseda tehnoloogia poolt ette seatavate piirangute pärast.

## Lisad

Järgnevalt mõned pikemad koodijupid, mida võib suurema ja dünaamilisema rakenduse loomisel vaja minna.

#### Ressursside hoidmine SQL Serveris – Resource Provider

Järgnev kood kirjutab üle mootori, mis tegeleb ressursside lugemisega. See võimaldab ressursse hoida .resx failide asemel SQL serveri andmebaasis.

Selleks, et allolev näide tööle läheks, tuleb:

* Lisada konfiguratsiooni ühendustekst, mis ühendab ressursse hoidva andmebaasiga. Ühendusteksti nimi omistage konstandile connStrName.

Lisada konfiguratsiooni element  
<globalization culture="auto" uiCulture="auto" resourceProviderFactoryType="OmaProjekt.SqlResourceProviderFactory" />

Loomulikult on võimalik allolev klass kirjutada märksa keerukamaks, nii et seda oskaks kasutada ka VisualStudio ressursside võlur ning samuti on võimalik lisada funktsionaalsus puuduvate ressursside automaatseks lisamiseks.

using System;

using System.Collections;

using System.Collections.Specialized;

using System.Data;

using System.Data.SqlClient;

using System.Data.SqlTypes;

using System.Globalization;

using System.Resources;

using System.Text;

using System.Web;

using System.Web.Compilation;

namespace OmaProjekt

{

public sealed class SqlResourceProviderFactory : ResourceProviderFactory

{

const string connStrName = "yhendusTxt";

public override IResourceProvider

CreateGlobalResourceProvider(string classKey)

{

return new SqlResourceProvider(null, classKey);

}

public override IResourceProvider

CreateLocalResourceProvider(string virtualPath)

{

virtualPath = System.IO.Path.GetFileName(virtualPath);

return new SqlResourceProvider(virtualPath, null);

}

private sealed class SqlResourceProvider : IResourceProvider

{

private string \_virtualPath;

private string \_className;

private IDictionary \_resourceCache;

private static object CultureNeutralKey = new object();

public SqlResourceProvider(string virtualPath, string className)

{

\_virtualPath = virtualPath;

\_className = className;

}

private IDictionary GetResourceCache(string cultureName)

{

object cultureKey;

if (cultureName != null)

{

cultureKey = cultureName;

}

else

{

cultureKey = CultureNeutralKey;

}

if (\_resourceCache == null)

{

\_resourceCache = new ListDictionary();

}

IDictionary resourceDict =

\_resourceCache[cultureKey] as IDictionary;

if (resourceDict == null)

{

resourceDict = SqlResourceHelper.GetResources(\_virtualPath,

\_className, cultureName, false, null);

\_resourceCache[cultureKey] = resourceDict;

}

return resourceDict;

}

object IResourceProvider.GetObject(string resourceKey,

CultureInfo culture)

{

string cultureName = null;

if (culture != null)

{

cultureName = culture.Name;

}

else

{

cultureName = CultureInfo.CurrentUICulture.Name;

}

object value = GetResourceCache(cultureName)[resourceKey];

if (value == null)

{

// vajaliku kultuuri ressurss on puudu, kasutame ilma

// kultuuriinfota ressurssi

HttpContext.Current.Trace.Warn("Puuduv ressurss",

"resourceKey=" + resourceKey +

";virtualPath=" + \_virtualPath +

";className=" + \_className +

";cultureName=" + cultureName);

value = GetResourceCache(null)[resourceKey];

}

if (value == null)

{

// Vajalikku ressurssi pole üldse olemas

HttpContext.Current.Trace.Warn("Puuduv ressurss",

"resourceKey=" + resourceKey +

";virtualPath=" + \_virtualPath +

";className=" + \_className);

}

return value;

}

IResourceReader IResourceProvider.ResourceReader

{

get

{

return new SqlResourceReader(GetResourceCache(null));

}

}

}

private sealed class SqlResourceReader : IResourceReader

{

private IDictionary \_resources;

public SqlResourceReader(IDictionary resources)

{

\_resources = resources;

}

IDictionaryEnumerator IResourceReader.GetEnumerator()

{

return \_resources.GetEnumerator();

}

void IResourceReader.Close()

{

}

IEnumerator IEnumerable.GetEnumerator()

{

return \_resources.GetEnumerator();

}

void IDisposable.Dispose()

{

}

}

internal static class SqlResourceHelper

{

public static IDictionary GetResources(string virtualPath,

string className, string cultureName, bool designMode,

IServiceProvider serviceProvider)

{

SqlConnection con = new SqlConnection(

System.Configuration.ConfigurationManager.

ConnectionStrings[connStrName].ToString());

SqlCommand com = new SqlCommand();

if (!String.IsNullOrEmpty(virtualPath))

{

// Lokaalsete ressursside küsimine

if (string.IsNullOrEmpty(cultureName))

{

// ilma kultuuriinfota ressursid

com.CommandType = CommandType.Text;

com.CommandText = "select resource\_name, resource\_value" +

" from ASPNET\_GLOBALIZATION\_RESOURCES" +

" where resource\_object = @virtual\_path" +

" and culture\_name is null";

com.Parameters.AddWithValue("@virtual\_path", virtualPath);

}

else

{

com.CommandType = CommandType.Text;

com.CommandText = "select resource\_name, resource\_value" +

" from ASPNET\_GLOBALIZATION\_RESOURCES " +

"where resource\_object = @virtual\_path " +

"and culture\_name = @culture\_name ";

com.Parameters.AddWithValue("@virtual\_path", virtualPath);

com.Parameters.AddWithValue("@culture\_name", cultureName);

}

}

else if (!String.IsNullOrEmpty(className))

{

// Globaalsete ressursside küsimine

if (string.IsNullOrEmpty(cultureName))

{

// ilma kultuuriinfota ressursid

com.CommandType = CommandType.Text;

com.CommandText = "select resource\_name, resource\_value" +

" from ASPNET\_GLOBALIZATION\_RESOURCES " +

"where resource\_object = @class\_name" +

" and culture\_name is null";

com.Parameters.AddWithValue("@class\_name", className);

}

else

{

com.CommandType = CommandType.Text;

com.CommandText = "select resource\_name, resource\_value " +

"from ASPNET\_GLOBALIZATION\_RESOURCES where " +

"resource\_object = @class\_name and" +

" culture\_name = @culture\_name ";

com.Parameters.AddWithValue("@class\_name", className);

com.Parameters.AddWithValue("@culture\_name", cultureName);

}

}

else

{

// Probleem lähteandmetega

throw new Exception("SqlResourceHelper.GetResources()" +

" – puuduvad parameetrid virtualPath või className");

}

ListDictionary resources = new ListDictionary();

try

{

com.Connection = con;

con.Open();

SqlDataReader sdr =

com.ExecuteReader(CommandBehavior.CloseConnection);

while (sdr.Read())

{

string rn = sdr.GetString(sdr.GetOrdinal("resource\_name"));

string rv = sdr.GetString(sdr.GetOrdinal("resource\_value"));

resources.Add(rn, rv);

}

}

catch (Exception e)

{

throw new Exception(e.Message, e);

}

finally

{

if (con.State == ConnectionState.Open)

{

con.Close();

}

}

return resources;

}

}

}

}

#### Kasutajatunnuste hoidmine andmebaasis - MembershipProvider

using System;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

using System.Web;

using System.Web.Security;

public class KasutajaProvider : MembershipProvider

{

#region abstraktsed meetodid ja propertid, mis tuleb kindlasti realiseerida

public override string ApplicationName

{

get

{

throw new Exception("The method or operation is not implemented.");

}

set

{

throw new Exception("The method or operation is not implemented.");

}

}

public override bool ChangePassword(string username, string oldPassword, string newPassword)

{

throw new Exception("The method or operation is not implemented.");

}

public override bool ChangePasswordQuestionAndAnswer(string username, string password, string newPasswordQuestion, string newPasswordAnswer)

{

throw new Exception("The method or operation is not implemented.");

}

public override MembershipUser CreateUser(string username, string password, string email, string passwordQuestion, string passwordAnswer, bool isApproved, object providerUserKey, out MembershipCreateStatus status)

{

throw new Exception("The method or operation is not implemented.");

}

public override bool DeleteUser(string username, bool deleteAllRelatedData)

{

throw new Exception("The method or operation is not implemented.");

}

public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize, out int totalRecords)

{

throw new Exception("The method or operation is not implemented.");

}

public override int GetNumberOfUsersOnline()

{

throw new Exception("The method or operation is not implemented.");

}

public override string GetPassword(string username, string answer)

{

throw new Exception("The method or operation is not implemented.");

}

public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)

{

throw new Exception("The method or operation is not implemented.");

}

public override MembershipUser GetUser(string username, bool userIsOnline)

{

throw new Exception("The method or operation is not implemented.");

}

public override string GetUserNameByEmail(string email)

{

throw new Exception("The method or operation is not implemented.");

}

public override bool EnablePasswordReset

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override bool EnablePasswordRetrieval

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override MembershipUserCollection FindUsersByEmail(string emailToMatch, int pageIndex, int pageSize, out int totalRecords)

{

throw new Exception("The method or operation is not implemented.");

}

public override MembershipUserCollection FindUsersByName(string usernameToMatch, int pageIndex, int pageSize, out int totalRecords)

{

throw new Exception("The method or operation is not implemented.");

}

public override int MaxInvalidPasswordAttempts

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override int MinRequiredNonAlphanumericCharacters

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override int MinRequiredPasswordLength

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override int PasswordAttemptWindow

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override MembershipPasswordFormat PasswordFormat

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override string PasswordStrengthRegularExpression

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override bool RequiresQuestionAndAnswer

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override bool RequiresUniqueEmail

{

get { throw new Exception("The method or operation is not implemented."); }

}

public override string ResetPassword(string username, string answer)

{

throw new Exception("The method or operation is not implemented.");

}

public override bool UnlockUser(string userName)

{

throw new Exception("The method or operation is not implemented.");

}

public override void UpdateUser(MembershipUser user)

{

throw new Exception("The method or operation is not implemented.");

}

public override bool ValidateUser(string username, string password)

{

SqlCommand cmd = new SqlCommand("TuvastaKasutaja\_proc",

new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString));

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add("@username", SqlDbType.NVarChar, 50).Value = username;

cmd.Parameters.Add("@password", SqlDbType.NVarChar, 50).Value = password;

cmd.Parameters.Add("@KasutajaID", SqlDbType.Int).Direction = ParameterDirection.Output;

cmd.Connection.Open();

cmd.ExecuteNonQuery();

cmd.Connection.Close();

bool KasutajaOK = false;

if (cmd.Parameters["@KasutajaID"].Value != System.DBNull.Value)

{

HttpContext.Current.Session["ActiveInimeneID"] = cmd.Parameters["@KasutajaID"].Value.ToString();

KasutajaOK = true;

}

return KasutajaOK;

}

#endregion

}

#### Kasutajagruppide hoidmine andmebaasis – RoleProvider

using System;

using System.Collections;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

using System.Web;

using System.Web.Security;

public class SqlRoleProvider : RoleProvider

{

public override void AddUsersToRoles(string[] usernames, string[] roleNames)

{

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "AddUserToRole\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.Add("@UserName", SqlDbType.NVarChar, 50);

cmd.Parameters.Add("@RoleName", SqlDbType.VarChar, 50);

cmd.Connection.Open();

foreach (string user in usernames)

{

cmd.Parameters["@UserName"].Value = user;

foreach (string role in roleNames)

{

cmd.Parameters["@RoleName"].Value = role;

cmd.ExecuteNonQuery();

}

}

cmd.Connection.Close();

}

public override string ApplicationName

{

get

{

return "Tallinna Täiskasvanute Gümnaasiumi testimiskeskus";

}

set

{

throw new Exception("The method or operation is not implemented.");

}

}

public override bool IsUserInRole(string username, string roleName)

{

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "IsUserInRole\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@UserName", username);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Parameters.Add("@OK", SqlDbType.Bit).Direction = ParameterDirection.Output;

cmd.Connection.Open();

cmd.ExecuteNonQuery();

cmd.Connection.Close();

return Convert.ToBoolean(cmd.Parameters["@OK"].Value);

}

public override string[] GetRolesForUser(string username)

{

string[] rollid = new string[1] { "ADMINISTRATORS" };

return rollid;

}

public override void CreateRole(string roleName)

{

if (HttpContext.Current.User.IsInRole("ADMINISTRATORS"))

{

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "CreateRole\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Connection.Open();

cmd.ExecuteNonQuery();

cmd.Connection.Close();

}

}

public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)

{

bool tulemus = false;

if (HttpContext.Current.User.IsInRole("ADMINISTRATORS"))

{

SqlCommand cmd = new SqlCommand();

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Connection.Open();

if (throwOnPopulatedRole)

{

cmd.CommandText = "RolePopulated\_proc";

cmd.Parameters.AddWithValue("@Tulemus", throwOnPopulatedRole).Direction = ParameterDirection.Output;

cmd.ExecuteNonQuery();

if (Convert.ToBoolean(cmd.Parameters["@Tulemus"].Value))

{

throw new Exception("Ei saa kustutada rolli, kuhu kuuluvad kasutajad!");

}

else {

cmd.Parameters.Remove(cmd.Parameters["@Tulemus"]);

cmd.CommandText = "DeleteRole\_proc";

cmd.ExecuteNonQuery();

tulemus = true;

}

}

else

{

cmd.Parameters.Remove(cmd.Parameters["@Tulemus"]);

cmd.CommandText = "DeleteRole\_proc";

cmd.ExecuteNonQuery();

tulemus = true;

}

cmd.Connection.Close();

}

return tulemus;

}

public override bool RoleExists(string roleName)

{

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "RoleExists\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Parameters.Add("@Tulemus", SqlDbType.Bit).Direction = ParameterDirection.Output;

cmd.Connection.Open();

cmd.ExecuteNonQuery();

cmd.Connection.Close();

return Convert.ToBoolean(cmd.Parameters["@Tulemus"].Value);

}

public override void RemoveUsersFromRoles(string[] usernames, string[] roleNames)

{

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "RemoveUserFromRole\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.Add("@UserName", SqlDbType.NVarChar, 50);

cmd.Parameters.Add("@RoleName", SqlDbType.VarChar, 50);

cmd.Connection.Open();

foreach (string user in usernames)

{

cmd.Parameters["@UserName"].Value = user;

foreach (string role in roleNames)

{

cmd.Parameters["@RoleName"].Value = role;

cmd.ExecuteNonQuery();

}

}

cmd.Connection.Close();

}

public override string[] GetUsersInRole(string roleName)

{

ArrayList kasutajad = new ArrayList();

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "GetUsersInRole";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Connection.Open();

SqlDataReader lugeja = cmd.ExecuteReader();

while (lugeja.Read()) {

kasutajad.Add(lugeja["KNimi"].ToString());

}

lugeja.Close();

cmd.Connection.Close();

return (string[])kasutajad.ToArray(typeof(string));

}

public override string[] GetAllRoles()

{

ArrayList rollid = new ArrayList();

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "GetAllRoles";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Connection.Open();

SqlDataReader lugeja = cmd.ExecuteReader();

while (lugeja.Read())

{

rollid.Add(lugeja["Nimi"].ToString());

}

lugeja.Close();

cmd.Connection.Close();

return (string[])rollid.ToArray(typeof(string));

}

public override string[] FindUsersInRole(string roleName, string usernameToMatch)

{

ArrayList kasutajad = new ArrayList();

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "FindUsersInRole\_proc";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Connection = new SqlConnection(ConfigurationManager.ConnectionStrings["yhendusTekst"].ConnectionString);

cmd.Parameters.AddWithValue("@RoleName", roleName);

cmd.Parameters.AddWithValue("@usernameToMatch", usernameToMatch);

cmd.Connection.Open();

SqlDataReader lugeja = cmd.ExecuteReader();

while (lugeja.Read())

{

kasutajad.Add(lugeja["KNimi"].ToString());

}

lugeja.Close();

cmd.Connection.Close();

return (string[])kasutajad.ToArray(typeof(string));

}

}

1. IIS – Internet Information Service so Windowsi teenus, mis võimaldab veebi serveerimist nii sisevõrku kui Internetti. Rohkem infot leheküljel 481 [↑](#footnote-ref-1)
2. FTP – File Transferee Protocol [↑](#footnote-ref-2)
3. HTTP – Hyper Text Transfeere Protocol – standard veebilehtede edastamiseks üle Interneti [↑](#footnote-ref-3)
4. HTTPS – HTTP turvaline versioon, kus andmevahetus serveri ja kliendi (brauseri) vahel on krüpteeritud. [↑](#footnote-ref-4)
5. SharePoint – Microsoft Office SharePoint Server ja Microsoft Windows SharePoint Services on keskkonnad portaalide ehitamiseks. Neid kasutatakse enamasti intranetis töökorralduse ja failihalduse lahendamiseks. Rohkem infot veebist [www.microsoft.com/sharepoint](http://www.microsoft.com/sharepoint) [↑](#footnote-ref-5)
6. Kui soovite veateateid väljastada vaid konkreetsele IP aadressile siis tuleb selle koha peal kasutada pisut kavalamat protseduuri. Täpsemad juhised leiate mitmetest foorumitest või MSDNist (<http://msdn2.microsoft.com/en-us/library/aa479319.aspx>) [↑](#footnote-ref-6)