Tallinna Ülikool

Informaatika Instituut

Näidisrakendusel põhinev Microsoft Silverlight 3 õppematerjal

Microsoft Silverlight 3 Tutorial for Creation of a Data Driven Application

Bakalaureusetöö

Autor: Ilja Šmorgun

Juhendaja: Andrus Rinde

Autor: ..... "....." ..... 2009. a.

Juhendaja: ..... "....." ..... 2009. a.

Instituudi direktor: ..... "....." ..... 2009. a.

Tallinn 2009

Autorideklaratsioon

Olen koostanud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Kuupäev:

Autor: Allkiri:

Sisukord

Sissejuhatus

Tänapäeval mängib internet väga tähtsat rolli. On raske kujutada ette päeva ilma e-kirjade lugemise, uudiste sirvimise ja videote vaatamiseta. Viimastel aastatel on ilmunud uus suund veebitarkvaras – *RIA* rakendused. Enamlevinud tehnoloogiad *RIA* rakenduste loomiseks on olnud *Adobe Flash* ja *AJAX*. Praeguseks on neile tekkinud konkurent *Microsoft Silverlight*, mis pole veel väga tuntud ja selle kasutajate arv on väiksem.

Käesoleva töö autori tutvumine antud *.NET Raamistiku* osaga algas *Windows Presentation Foundation'ist* aastal 2006. Tehnoloogia avaldas muljet võimalustega lihtsalt ja kiiresti rakendusi luua, mis näevad hämmastavalt ilusad välja, on suutelised kasutama arvuti graafikakaardi ressursse näitamaks kasutajale animatsioone, graafikuid, pilte ning esitada andmeid ja infot arusaadaval ja põneval kujul.

2007. aasta sügisel avalikustati *Microsoft Silverlight*, *WPF-i* kärbitud versioon, mõeldud brauserite pluginaks, mis oleks suuteline töötama igal platvormil. *Silverlight* oli *WPF-st* samm edasi, kõik tööriistad ja töövoog jäid samaks. *Silverlight-i on* stabiilselt arendatud mitu aastat, ning hetkel on tehnoloogia jõudnud kolmanda versiooni beetani.

Silverlight 3 pakub selliseid võimalusi nagu 3D graafika kasutamine veebirakendustes ning rikkaliku sisuga programmide loomist, mis oskavad suhelda andmebaasidega ja selliste *Cloud teenustega* (*Cloud Services*) nagu *Amazon Elastic Cloud* ja *Windows Azure*. Väga huvitavaks *Silverlight 3* rakenduste aspektiks on võimalus käivitada programme väljastpoolt brauserit ja isegi *offline* režiimis.

Kahjuks teeb huvilistele tehnoloogiaga tutvumise raskeks asjaolu, et struktureerituid eestikeelseid materjale on suhteliselt vähe. Suurem osa olemasolevaid materjale on killustatud.

Kuna töö autor on ise tükk aega tegelenud *Silverlight-i* rakenduste arendamisega, siis tekkis huvi põhjaliku õppematerjali kirjutamiseks. Koostöös *Veebistuudiumi* meeskonnaga mõeldi välja näidisrakenduse kontseptsioon, mille loomise käigus oleks võimalik tutvuda peamiste *Silverlight-i* võimalustega.

Käesoleva töö põhieesmärk ongi koostada õppematerjali prototüüp. Õppematerjali järgi oleks võimalik tutvuda *Silverlight-i* rakenduste põhiliste omadustega ning sammhaaval *Silverlight 3* rakenduse luua. Prototüüp on aluseks *Veebistuudiumi* õppematerjalile.

Püstitatud eesmärgi saavutamiseks:

- annab autor ülevaate uuest *Microsofti* veebitehnoloogiast nimega *Silverlight*, mis on osa *Microsoft .NET Raamistikust*;
- vastavalt Veebistuudiumi koolituse aluseks mõeldud näidisrakenduse ideele koostas autor näidisrakenduse spetsifikatsiooni;
- lõi autor spetsifikatsioonile vastava reaalselt töötava rakenduse;
- koostas autor näidisrakenduse loomiseks vajalike harjutuste komplekti, mille täiendamisel saab mahukama koolitusmaterjali *Veebistuudiumi* jaoks.

1. Silverlight

Silverlight-i esimene versioon ilmus 2007. aasta sügisel. Selle omadused olid piiratud – *Silverlight 1.0* oli digitaalse meedia esitamise platvorm. *Silverlight* võimaldas kliendile vaadata kõrge kvaliteediga videoid, ilma et oleks vaja oodata, kuna brauser laeb piisava tüki sisust. Üks huvitavamatest näidetest oli *Foxi* filmistuudio poolt loodud proovirakendus *Fox Trailers*, kus oli võimalik vaadata peatselt saabuvate filmide klippe.

1.1. Silverlight-i esimese versiooni puudused

Rakenduste loomine oli raske, kuna koodi oli võimalik kirjutada ainult kasutades *JavaScript-i*. Kasutajaliidese loomiseks olid saadaval primitiivsed kujundid, ja selleks, et luua midagi tarka, arendajal tuli näha päris palju vaeva *JavaScript-i* manipuleerimisega.

1.2. Silverlight 2

Teine versioon oli välja kuulutatud samal ajal, kui ilmus *Silverlight-i 1* lõppversioon. Arendamisefaas kestis peaaegu aasta, ja 2'st finaalversioon ilmus 2008. aasta oktoobri keskel.

Silverlight-i teine versioon tõi kaasa palju uuendusi. Nüüd olid saadaval igasugused kasutajaliidese komponendid, millest igaühe jaoks sai luua oma stiliseeritud kujunduse. Rakenduste koodi kirjutatakse *C#*, mis on mitmeid kordi edasijõudnum keel, kui *JavaScript*. See on ka arusaadav, kuna *JavaScript* ei ole selliste ülesannete jaoks mõeldud.

1.3. Silverlight 3

2009. aasta kevadel ilmus *Silverlight 3* beeta versioon. *Silverlight 3* toob kaasa selliseid omadusi nagu:

- *H.264* kõrgkvaliteediga videote tugi;
- *3D* kujunduste kasutamise võimalused;
- Uued animatsioonid ja efektid, varjude lisamine ja *Pixel Shading*;
- Võimalus luua *RIA* rakendusi (*Rich Internet Applications*);
- Uued kasutajaliidese komponendid, näiteks *DataForm*;

- Võimalus käivitada programme väljastpoolt brauserit;
- Võimalus käivitada programme ilma interneti ühenduseta;
- Võimalus kasutada arvuti graafikakaadri ressursse keerukate kujunduste näitamiseks.

2. Töö struktuur

Käesoleva töö põhiliseks osaks on *Silverlight 3* rakenduse loomise õppematerjal. Materjal koosneb järgmistest osadest:

- rakenduse struktuuri kirjeldus – tehnoloogiad, komponendid, projekti puu;
- andmebaasi loomine andmete hoidmiseks kasutades *Microsoft SQL Server 2008 Express* versiooni;
- rakenduse kasutajaliidese loomine *Microsoft Expression Blend 3* ja *Microsoft Visual Studio 2008* abil;
- registreerimisvormi loomine kasutades *Silverlight 3* uue komponendi – *DataForm-i*;
- kasutajate nimekirja loomine kasutades *DataGrid* komponenti;
- *DataForm* ja *DataGrid-i* sidumine andmebaasiga kasutades *Microsoft .NET RIA Services* paketti;
- Pildigalerii loomine kasutades *Microsoft Deep Zoom Composer* tarkvara ja pärast selle integreerimine olemasolevaga rakendusega;

Materjali struktuur on omapärane. Alguses luuakse andmebaasi ning seotakse olemasoleva projektiga, ja ainult siis luuakse programmi kasutajaliides. Põhjuseks on see, et kuna käesoleva materjali kirjutamise ajal oli *.NET RIA Services* pakett beeta versioonis, siis projekti loomist tuli alustada just infrastruktuuri paigaldamisest. Muul juhul oli suur tõenäosus vigade tekitamiseks, ning rakendus ei pruugiks üldse töötada.

Esmapilgul võib tekkida tunne, et algajatele mõeldud õppematerjali jaoks on tegemist suhteliselt keerulise projektiga, on see siiski mõeldud selleks, et alustada lihtsatest asjadest, nagu *Silverlight-i* baaskomponentide loomine (nupud, *Grid*, *Path*, nuppude funktsioonid) ja siis sujuvalt edasi liikuda keeruliste asjade poole. Õppejuhendi eesmärk on *Silverlight 3* professionaalse taseme rakendus ning selliste oskuste omandamine nagu:

- Erinevate kasutajaliidese komponentide sidumine funktsioonidega;
- Kasutajaliidese komponentide jagamine *ChildWindow-ite* kaupa ja nende kontrollide käivitamine;

- Animatsioonidega manipuleerimine;
- Komponentide sidumine andmetega (*DataBinding*);

Materjali läbitöötamisel peaksid õppijatel tekkima piisavad baasteadmised, et hiljem hõlpsalt iseseisvalt edasi õppida.

2.1. Rakenduses kasutatud tehnoloogiad

Rakendus on ehitatud *Microsoft Silverlight 3* baasil.

Rakenduse koodi loomiseks kasutatakse *Microsoft Visual Studio 2008 Professional Edition* paketti.

Rakenduse kasutajaliidese loomiseks arvestades hea *UX-iga (user experience*) kasutatakse *Microsoft Expression Blend 3 March 09 Preview* versiooni.

Andmete salvestamiseks kasutatakse *Microsoft SQL 2008 Express Edition* andmebaasi.

Andmebaasi ja *Silverlight* rakenduse omavaheliseks suhtlemiseks kasutatakse *.NET RIA Services March 09 Preview* tehnoloogiat.

Pildigalerii jaoks kasutatakse *Silverlight Deep Zoom* komponenti.

2.2. Põhjus üleminekuks Silverlight 3-le

Alguses oli kavas rakenduse loomiseks kasutada *Microsoft Silverlight 2 lõppversiooni*. Kuid rakenduse arendamise käigus tuli välja *Silverlight 3 beeta* versioon. Kolmas versioon pakkus mõningaid uusi võimalusi, mida töö autoril tekis soov katsetada. Nende seas olid näiteks järgmised:

- Võimalus importida *Adobe Illustrator* faile otse *Blendi* ilma *Expression Design-i* kasutamata.
- Võimalus kasutada uusi animatsioone.
- *Expression Blend* toetab *XAML* koodi kirjutamist *IntelliSense-i* abil.
- *Expression Blend* oskab otseselt redigeerida *C#* koodfaile ilma *Visual Studio-t* kasutamata.

2.3. .NET RIA Services

.NET RIA Services on eraldi pakett, mis on saadaval siit:

http://www.microsoft.com/downloads/details.aspx?FamilyID=76bb3a07-3846-4564-b0c3-27972bcaabce&displaylang=en

Antud tehnoloogia võimaldab omavahel suhteliselt lihtsalt siduda olemasolev *Silverlight-i* rakendus *SQL Server-i* andmebaasiga. Vajadus sellise tehnoloogia vastu seisneb selles, et *Silverlight-i* rakendused ei oska suhelda otseselt andmebaasidega või teiste infoteenustega, seega nende kahe komponentide vahel peab olema eraldi kiht, mis korraldaks info liikumist. Varem oli samasuguse suhtluskihi loomine tunduvalt keerulisem, kuna arendaja oli sunnitud defineerima vajalikud tükid käsitsi.

.NET RIA Services muudab tavalise *Silverlight-i* lahenduse struktuuri, lisades *ASP.NET veebiteenuse* osa, mille kaudu toimub andmevahetus *DataProvider-iga*, ehk siis näiteks andmebaasiga.

2.4. Rakenduse kasutajaliides

Rakenduse komponentide joonistamiseks kasutatakse vektorgraafikat. Selleks sobib väga hästi mõni professionaalne vektorgraafika programm, näiteks *Adobe Illustrator* või *Microsoft Expression Design*. Pärast on mugav loodud kujundeid eksportida ja teisaldada *XAML* koodile. Vektorgraafika kujundid paistavad lõpprakenduses paremini välja, kui tavalised pildid. Jooned on teravamad, värvid erksamad. Vektorgraafika kasutamine tagab ka mugavust hilisema töötlemise või kujundite muutmise korral.

Pärast rakenduse käivitamist ekraanile ilmuvad neli nupu valikutega. Nuppude siltide esialgsed variandid on:

- Üritusest (ürituse kirjeldus)
- Registreerimine
- Juba registreeritud
- Pildigalerii

Soovides saada ettekujutuse kasutajaliidesest *vt. Lisa 1*.

2.4.1. Rakenduse töövoog

Kasutaja klõpsab soovitud valikul. Animatsiooniga avatakse valitud osa rakendusest.

2.4.1.1 Esimene stsenaarium

Mulli avamisel, muutub see ristküliku taoliseks kujundiks, mille sees on vastav sisu. *(vt. Lisa 2).*

2.4.1.2 Teine stsenaarium

Pärast nupule vajutamist kõik valikud muudavad oma asukohta *(vt. Lisa 3)* ning ekraani keskele ilmub ristkülik, mille sees on vastav sisu.

Mõned ristkülikud võivad olla teineteisega seotud, näiteks pärast registreerimisvormi täitmist on võimalik vajutada noolele *(vt. Lisa 3)*, mille tulemusena näidetakse registreerunud kasutajate nimekirja.

2.4.1.3 Lõplik stsenaarium

Valikud asuvad kogu aeg ekraani keskel. Valikute sildid on järgmised:

- Üritusest – ürituste kirjeldus;
- Registreerimine – registreerimisvorm;
- Juba registreeritud – registreeritud isikute nimekiri;
- Galerii – interaktiivne pildigaleeri ürituse fotodega.

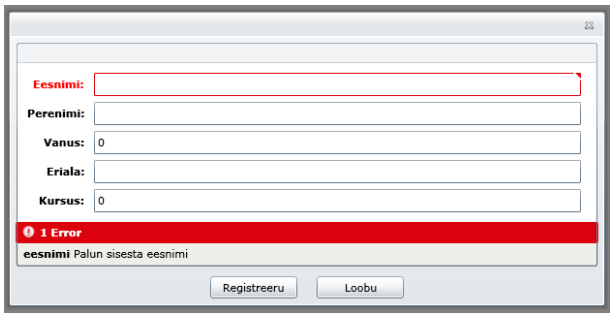
Pärast igal valikul klõpsamist kuvatakse *pop-up* aken, mille sees on vastav sisu. Valikud jäävad paika. *Pop-up* akna avamisel käivitatakse animatsioon, samamoodi ka sulgemisel.

Registreerimisvormil on olemas *validaatorid*, seega kui kasutaja sisestab vale väärtuse, või jätab lahtri täitamata, kuvatakse veateade.

2.5. Rakenduse kasutajaliidese komponendid

2.5.1. Registreerimisvorm

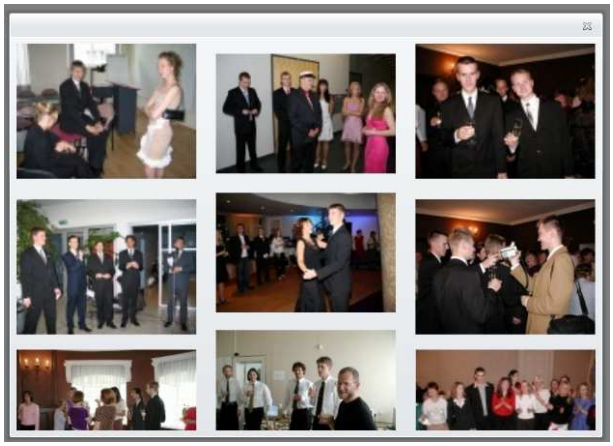
Registreerimisvormi jaoks kasutatakse *Silverlight 3* uue *DataForm* komponenti, mille abil on võimalik lihtsalt deklareerida vajalikud väljad ja nupud. *DataForm* pakub suured eelised võrreldes samasuguse vormi käsitsi loomisega. Üks nendest on võimalus määrata tingimusi, millele sisestatud info peab vastama. Tingimustele mitte vastamise puhul näitab *DataForm* kasutajale veateateid, näiteks tehakse punane kast selle lahtri ümber, kus info puudub või on valel kujul (vt. joonis 1).



Joonis 1. Silverlight 3 DataForm koos andmete valideerimisega

2.5.2. Pildigalerii

Pildigalerii luuakse kasutades *Deep Zoom-i*. See võimaldab luua kollaaži, ning kasutada kõrgkvaliteetseid ja suure resolutsiooniga pilte. Piltide suumimine ja nende vahel liikumine toimub sujuvalt ning ei sõltu interneti ühenduse kiirusest (vt. joonis 2).

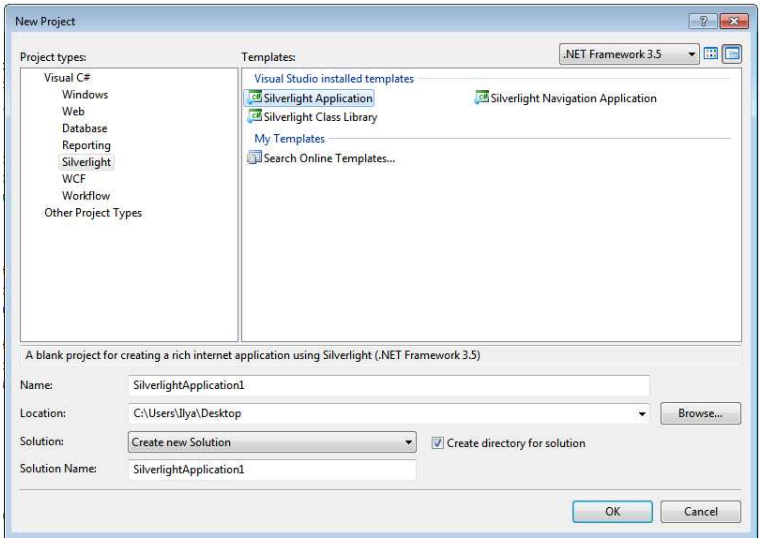


Joonis 2. Deep Zoom galerii

3. Õppematerjal

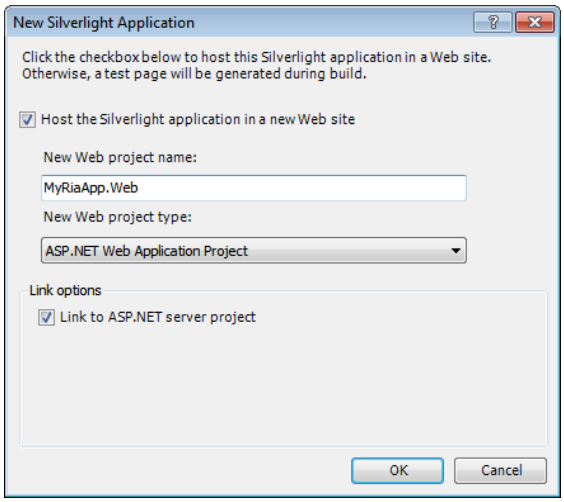
3.1. Uue rakenduse loomine

Uue projekti loomiseks tuleb käivitada *Microsoft Visual Studio C# Developer 2008* ja valida menüüst: *File -> New -> Project -> Silverlight Application* (vt. joonis 3).



Joonis 3. Uue Silverlight-i Projekti Loomine

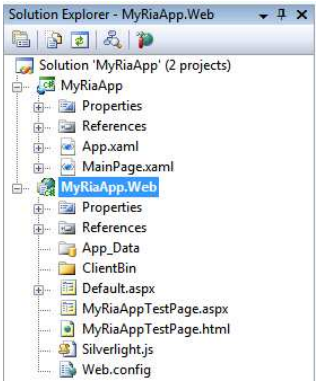
Määrake oma projektile nimi (meie näites: *MyRiaApp*). Klõpsake nupul *OK*, avaneb uus dialoogaken (vt. joonis 4).



Joonis 4. Silverlight-i rakenduse sidumine ASP.NET veebisaidiga

Sellist tüüpi dialoogaken ilmub ainult juhul, kui *.NET RIA Services* pakett on arvutile paigaldatud. *.NET RIA Services* võimaldab siduda *Silverlight-i* rakenduse (*projekti kliendi pool*) andmebaasiga või andmeserveriga (*andmepool*) *ASP.NET* liidese kaudu (*vahekiht*), kuna *Silverlight-i* rakendused ja andmebaasid ei oska teineteisega otseselt suhelda.

Pärast *OK* nupu klõpsamist loob *Visual Studio* uue projekti omapärase struktuuriga (vt. joonis 5).



Joonis 5. .NET RIA projekti struktuur

.NET RIA projekt koosneb kahest osast: *Silverlight-i* kliendi pool ja serveri *ASP.NET* pool. *.NET RIA* projekti kliendi poolel ja tavalisel *Silverlight-i* rakendusel on olemas ühesugused komponendid:

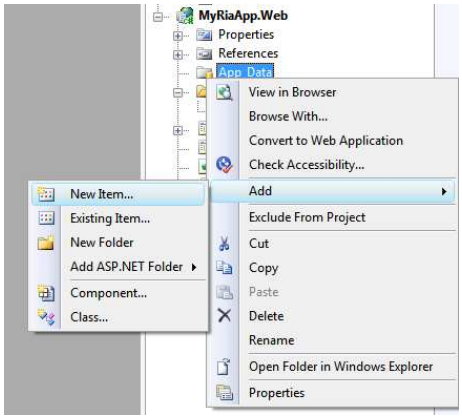
- *MainPage.xaml* – see on rakenduse põhileht, mida vaikimisi kuvatakse kohe pärast rakenduse käivitamist.
- *App.xaml* – nimetatakse ka *ressursside sõnastikuks* (*Resource Dictionary*). Sinna salvestatakse kõik objektide stiilid, selleks, et nad oleksid terve projekti skoobi sees saadaval.

3.2. Projekti sidumine andmete allikaga

Selleks, et registreerunud kasutajaid salvestada ja vaadata ürituse osalejate nimekirja, tuleb luua andmebaasi ja siduda see projektiga. Selleks läbitakse järgmised sammud.

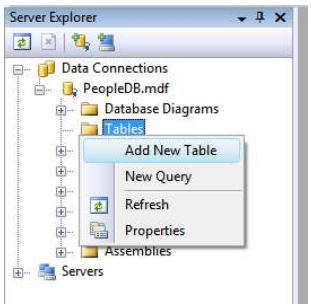
3.2.1. Uue andmebaasi loomine

Uue andmebaasi loomiseks tuleb avada projekt *Visual Studio-s* ja teha paremklõps *App_Data* kaustal, mis asub veebiosa juure all (*MyRiaApp.Web*). Kontekst-menüüst valitakse *Add -> New Item* (vt. joonis 6) ja ilmuvast dialoogaknast valitakse *SQL Server Database*. Nimeks võiks panna *PeopleDB.mdf*.



Joonis 6. Uue objekti lisamine App_Data kataloogi

Nüüd kui andmebaas on loodud peaks ta olema nähtav ka *Server Explorer-i* paneelist *Data Connections* kategooria alt. Vajutame plussi peale, selleks et vaadata, mis on andmebaasi faili sees. Ilmub andmebaasi struktuur. Teeme paremklõps *Tables* kataloogi peal ja valime *Add New Table* (vt. joonis 7).



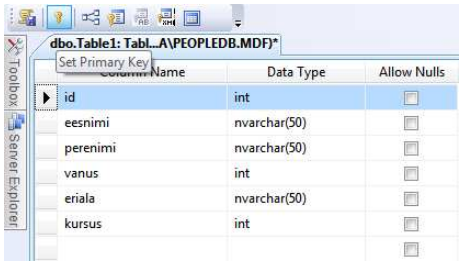
Joonis 7. Tabeli lisamine andmebaasi

Ilmub uus vaade, kust on võimalik määrata baasi andmeväljasid. Lisame järgmised väärtused (vt. tabel 1):

Column name	Data type	Allow Nulls
id	Int (arv, numbriline väärtus)	No
eesnimi	Nvarchar(50) (võtab vastu sõnu, mille maksimum pikkus on 50 märki)	No
perenimi	Nvarchar(50)	No
vanus	Int	No
eriala	Nvarchar(50)	No
kursus	Int	No

Tabel 1. Andmebaasi tabeli väärtuste määramine.

Nüüd kui tabeli veerud on määratud, valime *id* veeru ja vajutame võtme ikooniga nupule, mis muudab vastava väärtuse *Primary Key-ks* (vt. joonis 8). *Primary key* on vajalik selleks, et oleks võimalik identifitseerida iga rida andmebaasi tabelis.



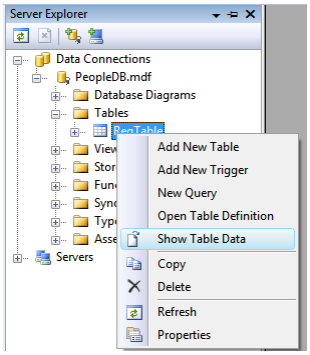
Joonis 8. Primary Key määramine

Pärast *Primary Key* määramist paneme paika ka õiged seadistused *Column Properties* sektsioonist. Avame rühma *Identity Specification* ja valime *(Is Identity) Yes*. Selle liigutuse tulemusena luuakse *id* veeru väärtused automaatselt ning suurendatakse iga uue kirje kohta ühe võrra. Nüüd kui kõik tabeli seadistused on paigas, vajutame salvestamisnupule ja paneme tabeli nimeks *RegTable*.

3.2.2. Kirje lisamine andmebaasi tabelisse

Kui uus tabel on loodud, lisame sinna ühe kirje, selleks, et see oleks pärast *DataGrid-ist* nähtav. See aitab katsetada, kas kõik komponendid töötavad ootuspäraselt, või on kusagil viga sees.

Uue kirje lisamiseks tabelisse, teeme *Server Explorer-is* paremklõps *RegTable* peale ja valime *Show Table Data* (vt. joonis 9).



Joonis 9. Andmebaasi sees olevate kirjete vaatamine

Ekraanile ilmub eelnevalt loodud tabel koos vastavate veergudega. Lisame sinna mingi fiktiivse kirje, näiteks:

Juku, Juurikas, 21, Füüsika, 2

ID jaoks ei ole vaja väärtust sisestada, kuna see määratakse automaatselt vastavalt eelnevalt määratud seadistustele.

3.2.3. Uue Entity Data Model-i loomine

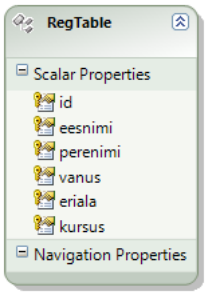
Entity Data Model on model andmebaasist. See mudel kirjeldab andmebaasi *Domain Service* teenuse jaoks (millised on andmeväljad ja milliseid väärtuseid aktsepteeritakse). *Domain Service-i* loomist käsitletakse järgmises peatükis.

Uue *Entity Data Model-i* tekitamiseks teeme paremklõps *MyRiaApp.Web* juure peal ja valime *Add -> New Item -> ADO.NET Entity Data Model* ning paneme nimeks *MyDataModel.edmx*.

Ekraanile ilmub viisardi dialoogaken, kus küsitakse kas me soovime luua *Data Model-i* andmebaasist või tekitada tühja mudeli ja siis täita see käsitsi ära. Valime *Generate from database* ja vajutame *Next*.

Järgmine vaade pakub võimalust määrata ühendust andmebaasiga. Siin andmed peaksid olema juba automaatselt sisestatud. Vajutame *Next* nupu.

Kolmas samm annab võimaluse valida andmebaasi objekte mudeli genereerimiseks. Märgime valiku lahtrisse *Tables*, mille all on meie andmebaasi tabel *RegTable(dbo)*. Vajutame nupule *Finish*. Tulemusena näidetakse ekraanile loodud andmemudel (vt. joonis 10).

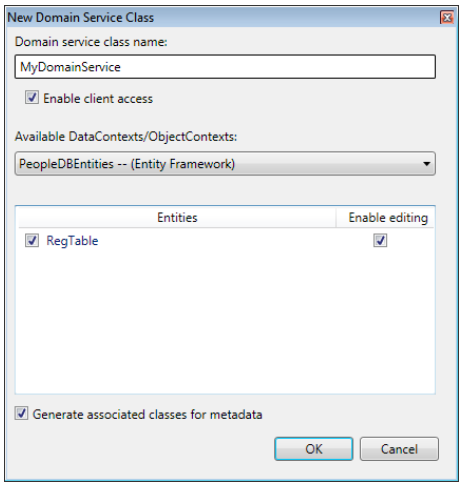


Joonis 10. Uus Entity Data Model

3.2.4. Uue Domain Service-i loomine

Andmebaas ja selle mudel on loodud. Viimase sammuna tuleb tekitada uus *Domain Service*, mis vastutab selle eest, et rakenduse kasutajaliidese osa suudaks andmebaasiga suhelda. Enne tuleb projekti uuesti kompileerida, muidu meie andmemudel ei oleks *Domain Service-le* nähtav. Valime *Build -> Build Solution* või vajutame klahvile *F6*.

Teeme paremklõpsu *MyRiaApp.Web* peal ja valime *Add -> New Item -> Domain Service Class* ning paneme nimeks *MyDomainService.cs*. Ekraanile ilmub dialoogaken, kus tuleb määrata andmekonteksti, ehk siis selle andmebaasi tabeli, mida me soovime kasutada andmeallikana. Märgime valikud *Entities: RegTable*, *Enable editing* ja *Generate associated classes for metadata* (vt. joonis 11).



Joonis 11. Uue Domain Service klassi defineerimine

Selle tulemusena peaks uus *Domain Service* võimaldama andmeid lisada, kustutada ja uuendada kasutajaliidese kaudu. Ka peaksid olema genereeritud vastavad *metaandmete* klassid, mis võimaldavad määrata valideerimisreegleid andmebaasi kirjete jaoks.

3.2.5. Valideerimisreeglite määramine

Valideerimisreeglite määramiseks on eraldi fail. Meie rakenduses on selleks *MyDomainService.metadata.cs*. Antud failis on automaatselt paika pandud andmetüübid vastavate andmebaasi veergude jaoks. Edasi on valideerimisreeglite määramine suhteliselt lihtne, tuleb vaid lisada mõned koodiread (vt. koodinäide 1). Kõik kood meie rakenduses on kirjutatud *Visual C#* programmeerimiskeeles. *Silverlight-i* rakenduste puhul on veel võimalik kasutada *Visual Basic-ut*.

```
[Bindable(false)]
public int id;

[Required(ErrorMessage = "Palun sisesta eesnimi")]
public string eesnimi;

[Required(ErrorMessage = "Palun sisesta perekonnanimi")]
public string perenimi;

[Required(ErrorMessage = "Palun sisesta vanus")]
[Range(18, 99, ErrorMessage = "Vanus peab olema 18 ja 99 vahel")]
public int vanus;

[Required(ErrorMessage = "Palun sisesta oma eriala")]
public string eriala;

[Required(ErrorMessage = "Palun sisesta vaartus")]
[Range(1, 5, ErrorMessage = "Kursus peab olema 1 ja 5 vahel")]
public int kursus;
```

Koodinäide 12. Valideerimisreeglite määramine

Nagu koodinäitest näha, pole sidumine *id* veeruga enam lubatud, et ta ei paistaks registreerimisvormis ega ka inimeste nimekirjas, kuna selle väärtus on niikuinii automaatselt genereeritud. Kõik muud väljad on kohustuslikud ja kui väärtus on puudu, genereeritakse kasutajale automaatselt veateade koos meie poolt määratud teavitusega. Näiteks eesnime puhul on selleks „Palun sisesta eesnimi“. Vanuse ja kursuse jaoks on määratud piirangud, näiteks vanus peab olema 18 ja 99 vahel ja kursus 1 ja 5 vahel. Valideerimisreeglite määramise võimalus on väga lai, näiteks saab defineerida regulaaravaldisi jne.

3.2.6. Kokkuvõtte andmete sidumisest

Väga tähtis on jälgida, et nii andmemudelite kui valideerimisreeglite määramine toimub serveri osas, ning sellel on olemas põhjus. Pärast seda, kui rakendus on valmis ja töötab kuskil reaalses serveris, tahetakse et kõik andmetega seotud toimingud oleksid realiseeritud just serveri poolt ja mitte kasutajaliideses. See teeb rakenduse kasutamise palju mugavamaks ja kiiremaks.

3.3. Rakenduse kasutajaliidese kujundamine

Silverlight-i rakenduste ehitamise töövoog on jagatud kaheks osaks: üks on disaineri pool ja teine – arendaja pool. Igale poolele on mõeldud omad tööriistad, mis on teineteisega kitsalt seotud, selleks, et arendamine oleks võimalikult mugav ja paindlik.

Disainerite jaoks on olemas *Microsoft Expression Blend 3*, mille abil saab luua rakenduse kujunduse, lisada objektidele efekte ja animatsioone ilma koodi kirjutamata.

Arendajate jaoks on mõeldud *Microsoft Visual Studio*, kus on olemas kõik vajalikud võimalused keerulise programmiloogika loomiseks.

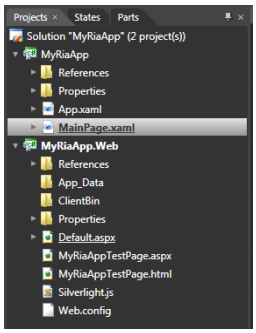
Selline lähtumine väljendub ka *Silverlight* rakenduste koosseisus olevate failide struktuuris. Näitena võib tuua *MainPage.xaml* faili, mis tegelikult koosneb kahest objektist: *XAML* koodist ja *code-behind .cs* laiendiga failist, kui programmeerimiseks kasutatakse *C#* keelt. *XAML* on deklaratiivne *XML*-põhine keel, mille abil on väga lihtne kirjeldada selliseid kasutajaliidese objekte, nagu *nuppe*, *comboboxe*, *grid-de* jne.

Silverlight 2 põlvkonna rakendustel olid *Blend-i* ja *Visual Studio* rollid suhtelised rangelt eraldatud. *Blend 3 beeta versiooniga* on *Microsoft* võtnud uue suuna. Nüüd *Blend*-ist on võimalik otseselt redigeerida *code-behind* faile, kui selleks tekib vajadus. Eelmine versioon võimaldas ainult *XAML* koodi redigeerimist.

Blend-i ja *Visual Studio* kitsas integreerimine võimaldab arendada rakendust mõlemast programmist. Tehes muudatusi ühe programmi sees, näitab teine dialoogakent, mis teavitab tehtud muudatustest.

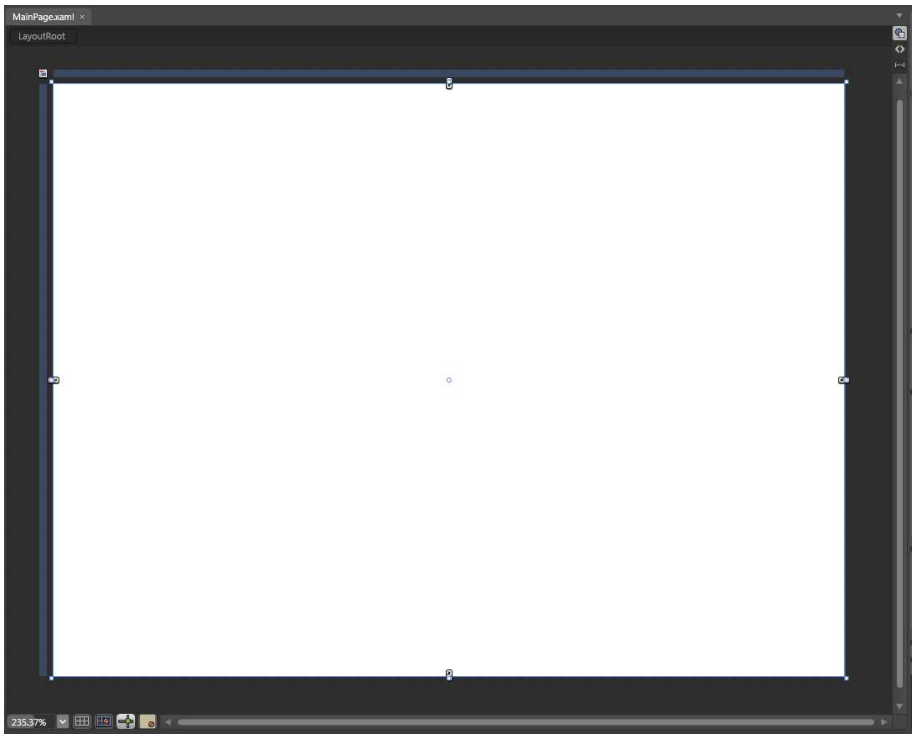
3.3.1. Pealehe kujundamine

Teeme meie projekti *Expression Blend-is* lahti ja näeme täpselt samasugust struktuuri, nagu *Visual Studio-s* (vt. joonis 13).



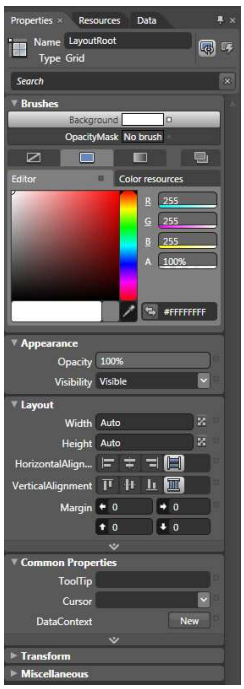
Joonis 13. Projekti struktuur Expression Blend-is

Ekraani keskele kuvatakse stseen, kuhu saab lisada kasutajaliidese komponente (vt. joonis 14).



Joonis 14. Expression Blend-i stseen

Paremal pool näidatakse objekti omaduste riba (*Properties Pane*), kus on võimalik muuta värvi, suurust, paigutust, fonti jne. (vt. joonis 15).



Joonis 15. Objekti omaduste riba

Vasakul pool on olemas riba, mis on kohe tuttav inimestele, kes on varem kasutanud graafika redigeerimisprogramme, nagu *Adobe Photoshop* või *Adobe Illustrator* (vt. joonis 16).

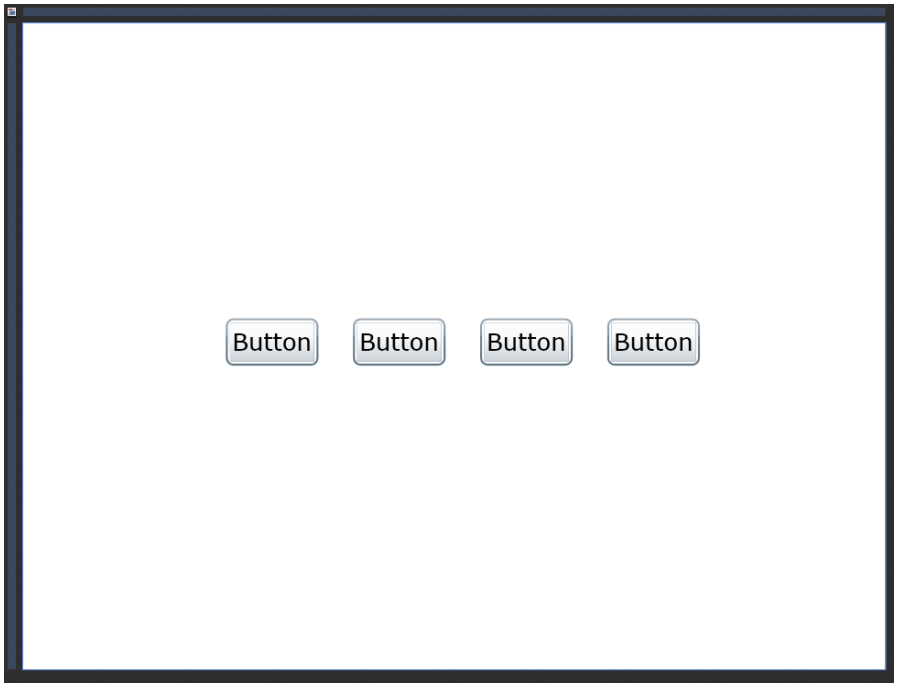
Joonis 16. Blend-i tööriistade riba

Objektide lisamiseks stseenile on olemas kaks võimalust:

- Valida sobiv objekt ja lohistada see stseenile;

- Vajutada topelt-klõpsuga objekti ikoonile, mis loob vasakpoolse ülemise servale vaikimisi suurusega vastava komponendi stseeni.

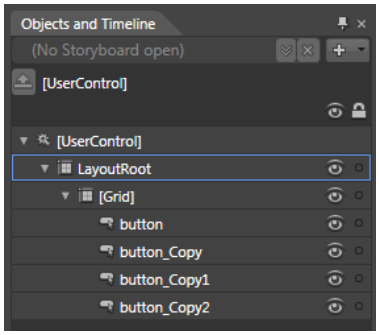
Lisame stseenile neli nuppu ja paigutame need keskele (vt. joonis 17).



Joonis 17. Nuppude lisamine stseenile

Selleks, et nuppudega oleks hiljem lihtsam töötada grupeerime need *grid*-i. Valime kõik nupud, teeme paremklõpsu ning valime *Group Into -> Grid*.

Object and Timeline paneelist on näha objektide hierarhiat (vt. joonis 18).



Joonis 18. Objektide hierarhia

Nagu illustratsioonilt näha, kõige kõrgem komponent on *UserControl*. *Selle* sees on *Grid* nimega *LayoutRoot*, millesse paigutatakse kõik teised objektid. Hetkel on selle sees grupeeritud nupud.

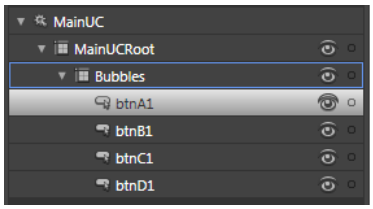
Nüüd, kui stseenil on olemas hulk objekte, muudame nende nimesid ja teisi omadusi. Meie näites nimetame *UserControl*-i *MainUC*-ks, ja määrame talle automaatse suuruse. Nimi võib olla täiesti suvaline. Siiski on tähtis seda meeles pidada, et saaks koodis õige objekti poole pöörduda. *Horizontal* ja *Vertical Alignement* peaksid olema *Stretch* ja kõik *Margin*-id 0.

Nimetame *LayoutRoot* komponendi *MainUCRoot*-uks. Suurus peaks olema samamoodi automaatne, mõlemad *Alignement*-i väärtused *Stretch* peal ja *Margin*-id 0.

Pärast kõikide väärtuste muutmist on näha, et nuppude suurus ja paigutus on muutunud. Põhjuseks on see, et suurused pole staatilised, seega *Blend* venitab nad vastavalt seadistustele välja.

Paneme nuppude *grid*-ile nimeks *Bubbles*, suuruseks valime 460x100 ja *Alignement-i* väärtuseks *Center*.

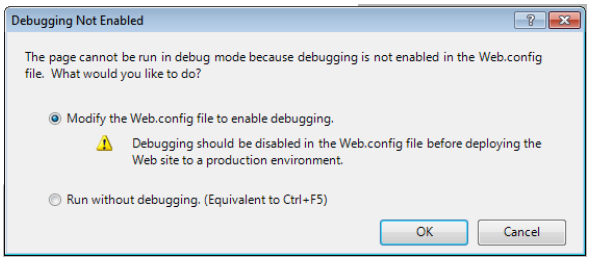
Valime kõik nupud korraga (hoides neil klõpsamise ajal all *Control* või *Shift* klahvi) ja määrame nende suuruseks 100x100 pikslit. Muudame ükshaaval ka nende nimed. Tulemus peaks olema selline (vt. joonis 19):



Joonis 19. Komponentide uuendatud omadused

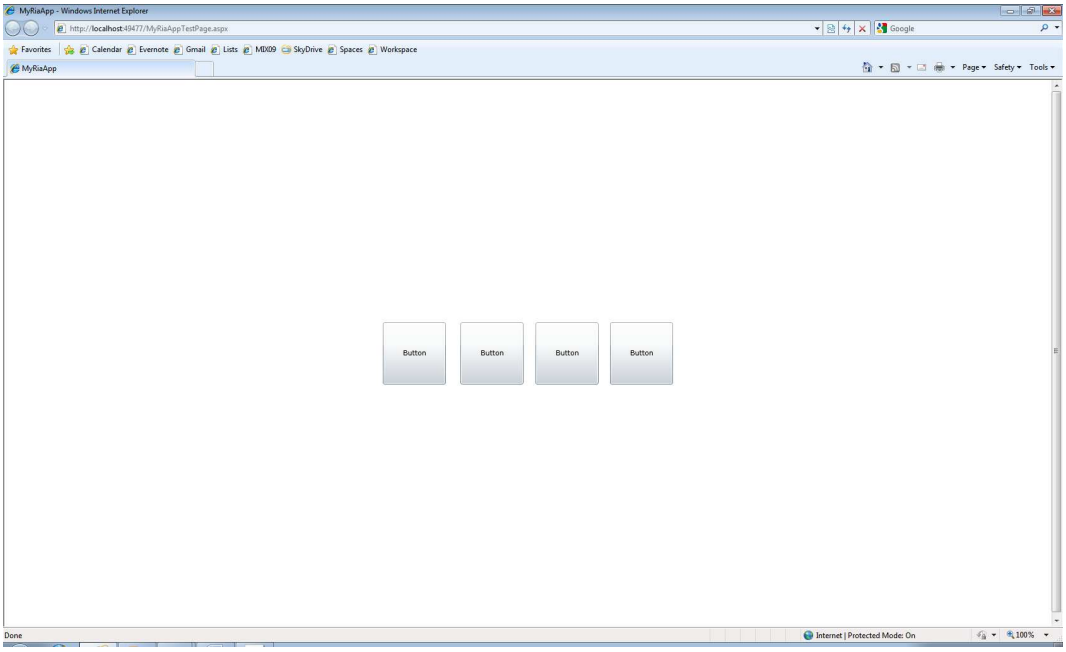
Pöördume tagasi *Visual Studio* poole. Ekraanile ilmub aken, mis annab teada, et projektis on tehtud muudatusi ja küsib, kas tahame faile uuendada. Klõpsame nupul *Yes to All*.

Nüüd on aeg vaadata, kuidas meie senine looming välja näeb! Vajutame klahvi *F5*, see annab *Visual Studio*-le käsu rakendus kompileerida ja näidata tulemust brauseris. Ekraanile ilmub uus aken, kus *Visual Studio* pakub *debugimist* (vt. joonis 20). Debugimine on vajalik vigade parandamiseks. Kui kood oli valesti kirjutatud, siis pärast käivitamist *Visual Studio* katkestab programmi töö ja annab teate tekkinud probleemi kohta. Vajutame *OK* nupule.



Joonis 20. Visual Studio võimaldab debugimist

Avatud brauseri aknas on võimalik katsetada programmi töötamist (vt. joonis 21). Kompileerinud rakendusel on olemas ka üks omapära, seda käivitatakse kasutaja arvutis (*localhost*) virtuaalses *ASP.NET Development Serveris*. *.NET RIA* rakendusi ei ole võimalik käivitada otse arvutist, eriti kui on olemas seos *SQL* andmebaasiga. Seega virtuaalne server võimaldab arendajal läbi proovida programmi tööd, ilma et ta oleks sunnitud pärast iga muudatust lähtekoodi kuskile reaalsesse serverisse laadima.



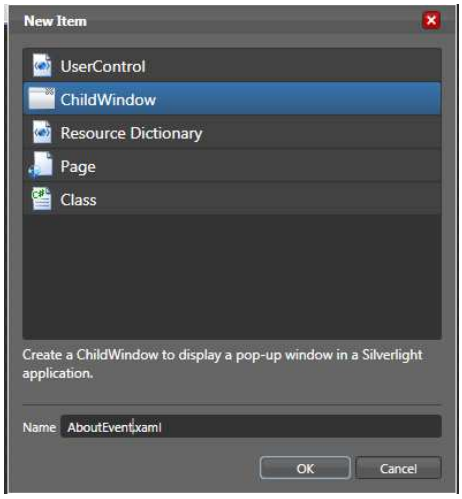
Joonis 21. Kompileeritud rakendus valmis katsetamiseks brauseri aknas

3.3.2. Tekstiinfot sisaldava lehe loomine

Nüüd, kui pealeht ja vastavad nupud on valmis, tekitame lehe, kus hakkab asuma ürituse kirjeldus. *Silverlight 3* versioonis on saadaval uus kasutajaliidese komponent – *ChildWindow*. Sellel on vaikimisi olemas sulgemisnupp ja kui *ChildWindow-i* avatakse, siis *Silverlight* näitab automaatselt animatsiooni ja teeb tausta tumedamaks, mis võimaldab kasutajal paremini keskenduda.

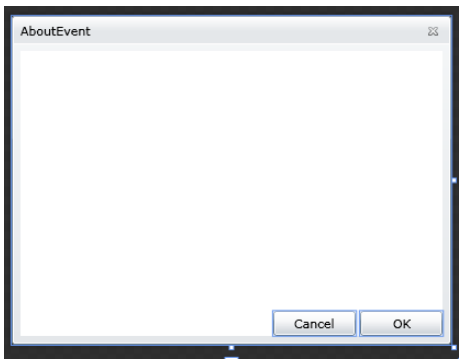
Alustame *ChildWindow* loomist *Expression Blend-is*, hiljem loome rakenduse teiste osade jaoks aknad otse *Visual Studio-s.*

Uue *ChildWindow-i* loomiseks teeme paremklõps *MyRiaApp* peal *Blend-i* projektipuus ja valime *Add New Item…* Ekraanile ilmub dialoogaken, kust valime *ChildWindow* ja paneme nimeks *AboutEvent.xaml* (vt. joonis 22).



Joonis 22. Uue ChildWindow-i loomine

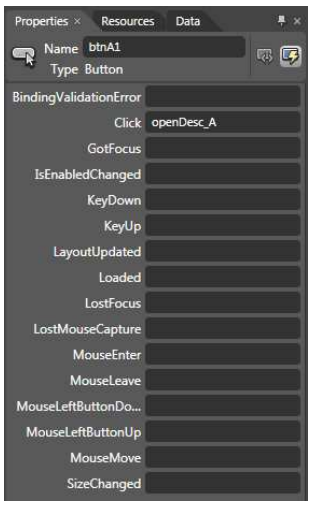
Näeme, et *ChildWindow-i* struktuur on praktiliselt samasugune, nagu *UserControl-i* puhul, väljaarvutud mõned erinevused: *ChildWindow-il* on olemas raam, vaikimisi on genereeritud sulgemisnupp, ning *Blend* lisab automaatselt ka *Cancel* ja *OK* nupud (vt. joonis 23).



Joonis 23. Automaatselt genereeritud ChildWindow

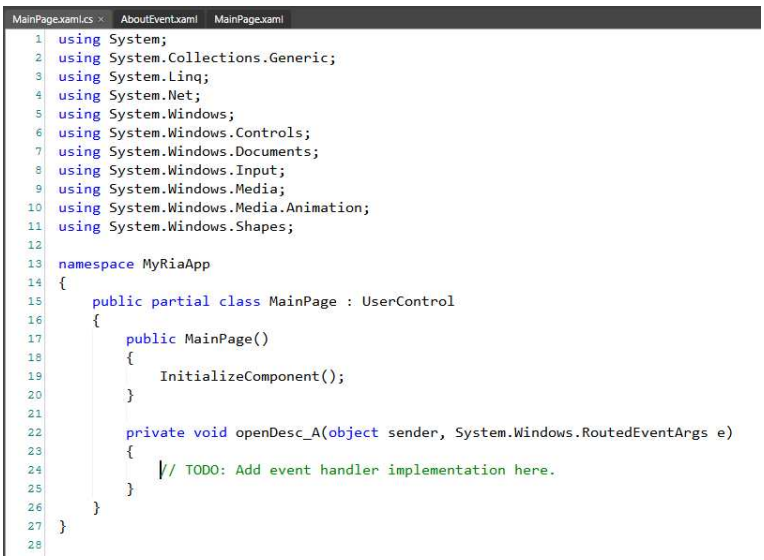
Teeme mõned muudatused, et aken vastaks meie vajadustele. Alguseks kustutame *OK* ja *Cancel* nupud. Lisame uue *TextBlock-i* ja määrame tema laiuseks 600px. Joonduseks (*Alignement*) peaks määrama *Strech.* Kõrgus peaks olema automaatselt määratav (*Auto*), seega akna kõrgus muutub vastavalt sellele, kui palju teksti me soovime kuvada. Objekti omaduste paneelist on võimalik määrata teksti suurust ja fonti, valime näiteks *Trebuchet Ms* ja *12pt* ning lisame katseks lõigu teksti.

Akna kujundus on valmis ja viimaseks sammuks oleks siduda antud akna kuvamine meie pealehe esimese nupu abil. Selleks teeme lahti *MainPage.xaml Blend-is* ja valime esimese nupu *btnA1*. Nuppude funktsioonide määramiseks on *Blendi omaduste paneelis* olemas eraldi vaade, nimega *Events.* Valime *Click* lahtri ja sisestame nime *openDesc_A* (vt. joonis 24).



Joonis 24. Nupu Click funktsiooni määramine

Vajutades *Enter* klahvi avatakse automaatselt *MainPage.xaml.cs* fail, *Blend-i* uus versioon toetab ka *code-behind* koodi kirjutamist (vt. joonis 26).



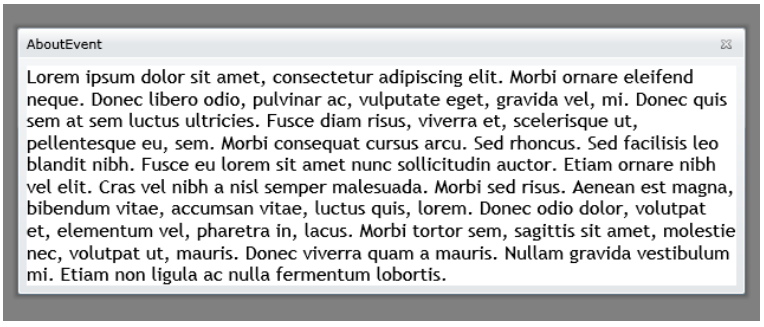
Joonis 25. Code-behind failide redigeerimine Blend-is

Siiski on hetkel selle jaoks mugavam kasutada *Visual Studio-t*, sest *Blend 3* beeta versioonis ei ole veel koodi kirjutamise tugi piisavalt küpsis. Sellega avame *Visual Studio-s* meie projekti ja teeme lahti *MainPage.xaml.cs* faili. Näeme, et vastav funktsioon on loodud, kuid see on hetkel veel tühi. Lisame sinna vajalikud read (vt. koodinäide 2).

```
private void openDesc_A(object sender, RoutedEventArgs e)
{
        var aboutEventWindow = new AboutEvent();
        aboutEventWindow.Show();
}
```

Koodinäide 2. Kood AboutEvent.xaml akna avamiseks

Kompileerimine rakenduse ja vajutades esimesele nupule näeme, et tõepoolest ekraanile ilmub uus aken (vt. Joonis 26).



Joonis 26. AboutEvent aken

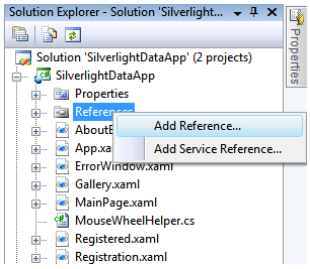
3.3.3. Loetelu loomine andmebaasi andmetest

Järgmiseks sammuks oleks luua osalejate nimekirja. Selleks kasutame *DataGrid* komponenti, mis hakkab näitama andmebaasis olevaid kirjeid.

Avame *Visual Studio*. Tekitame uue *ChildWindow-i* ja nimetame selle näiteks: *Registered.xaml*. Nüüd kui aken on loodud, lisame sinna vajalikud komponendid. Selleks, et *DataGrid* suudaks näidata infot, tuleb deklareerida uus *Domain Data Source*, ehk andmete allikas. Seda on võimalik teha kas deklaratiivselt *XAML* koodis või *C#* vahenditega. Meie kasutame deklaratiivset meetodid, mis lihtsustab objektide kasutamise arusaamist.

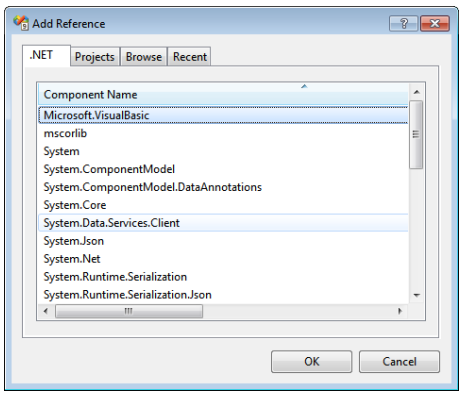
3.3.3.1 Objektide teekide lisamine projektile

Silverlight-i programmid on ehitatud samal põhimõttel nagu kõik teised *Microsoft .NET Framework-i* rakendused. Idee seisneb selles, et kasutatakse olemasolevaid komponente, mis asuvad teekides. Võrreldes teiste tehnoloogiatega on teegid justkui klassid. Mingisuguse komponendi kasutamiseks tuleb projektile lisada teek, mis seda sisaldab, ehk teegile viidata (*reference*). Lisame ka meie projektile vajalikud viited, et saaksime hiljem nende sees kirjeldatud komponente kasutada. Viite lisamiseks tuleb teha paremklõps *References* kataloogi peal projekti puus ja valida *Add Reference* (vt. joonis 27).



Joonis 27. Uue viite (*Reference*) lisamine

Ekraanile ilmub aken koos kõikide võimalikke *reference-de* nimekirjaga (vt. joonis 28).



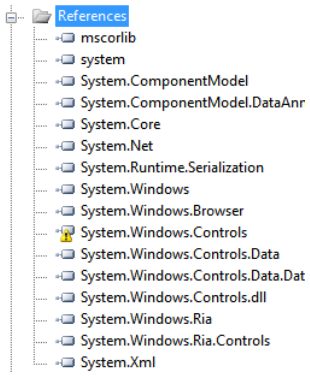
Joonis 28. Reference-de nimekiri

Lisame projektile järgmised teegid:

- *System.Windows.Controls*
- *System.Windows.Controls.Data*
- *System.Windows.Controls.Data.DataForm*
- *System.Windows.Controls.Ria*
- *System.Windows.Controls.Ria.Controls*

Need teegid sisaldavad kõike objekte, mida kavatseme edaspidi oma rakenduses kasutada: *DataForm*, *DataGrid* jne.

Näeme, et vastavad teegid ilmuvad *Reference-de* nimekirjas (vt. joonis 29).



Joonis 29. Uued teegid Reference-de nimekirjas

Nüüd tuleb lisada ainult veel üks teek – *ActivityControl*. See teek on vajalik selleks, et me saaksime näidata kasutajale progressi indikaatorit ajal kui *DataGrid* suhtleb andmebaasiga.

Siis saab kasutaja aru, et programm ikka teeb midagi. Selleks avame veelkord *Add Reference* dialoogakna, kuid nüüd vajutame ülaservas *Browse* nuppu ja leiame koolituse materjalide kataloogist *ActivityControl.dll* faili. Vastav teek peaks ilmuma *References* kataloogi. *ActivityControl-i* teegi lisamine toimub käsitsi failist, kuna see ei ole veel vaikimisi *Silverlight 3* beetaga saadaval. Käesolev objekt peaks olema lisatud *Silverlight 3* lõppversioonis.

3.3.3.2 XML viidete teekidele lisamine

Vajalikud teegid on lisatud ja saame kasutada nende sees olevaid komponente. Avame *Registered.xaml* faili ja lisame päisesse vajalikud kirjed (vt. koodinäide 3).

```
<controls:ChildWindow x:Class="MyRiaApp.Registered"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls"
  xmlns:data="clr-
namespace:System.Windows.Data;assembly=System.Windows.Ria.Controls"
  xmlns:datagrid="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
  xmlns:dds="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"
  xmlns:activity="clr-namespace:System.Windows.Controls;assembly=ActivityControl"
  xmlns:App="clr-namespace:MyRiaApp.Web" >
```

Koodinäide 3. XML namespace-de lisamine Registered.xaml faili päisse.

Koodilõigust on näha, et lisasime *Registered.xaml* faili päisesse mõned *XML Namespace-id*. Need töötavad nagu viited vastavatele teekidele, selleks, et *Visual Studio* teaks, kust kohast võtta õiged komponendid.

3.3.3.3 Andmebaasi kirjete näitamiseks vajalike komponentide määramine

Vaatame *Registered.xaml* struktuuri. *Visual Studio* lisas aknale vaikimisi kaks nuppu. Kustutame nad, kuna me ei hakka neid kasutama. *Grid-il* on vaikimisi määratud lahtrite kirjeldused, ka neid pole tarvis.

Paneme *Grid-i* suuruseks 400x600 pikslit. Selleks lihtsalt lisame vastavad *Height* ja *Width* väärtused. *Visual Studio* aitab, pakkudes võimalikud variandid. Tulemus peaks olema selline (vt. koodinäide 4).

```
<controls:ChildWindow x:Class="MyRiaApp.Registered"
```

```
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls"
  xmlns:data="clr-
namespace:System.Windows.Data;assembly=System.Windows.Ria.Controls"
  xmlns:datagrid="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
  xmlns:dds="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"
  xmlns:activity="clr-namespace:System.Windows.Controls;assembly=ActivityControl"
  xmlns:App="clr-namespace:MyRiaApp.Web" >
    <Grid x:Name="LayoutRoot" Height="400" Width="600" >
    </Grid>
</controls:ChildWindow>
```

Koodinäide 4. Registered.xaml ilma ebavajalike komponentideta.

Nüüd kui kood on puhastatud, lisame puuduvad osad. Esimese sammuna deklareerime uue *Domain Data Source*, mis haldab interaktsioone rakenduse kasutajaliidese ja andmeallika vahel. Selleks tuleb lisada järgmine kood (vt. koodinäide 5).

```
<dds:DomainDataSource x:Name="dds"
  LoadMethodName="LoadRegTable"
  AutoLoad="True"
  LoadSize="20">
    <dds:DomainDataSource.DomainContext>
      <App:MyDomainContext></App:MyDomainContext>
    </dds:DomainDataSource.DomainContext>
</dds:DomainDataSource>
```

Koodinäide 5. Domain Data Source-i lisamine XAML koodi abil.

Koodist on näha, et *Domain Data Source-i* nimeks määrame *dds*. *LoadMethodName-ks* on määratud *LoadRegTable*. See on vajalik selleks, et meie *Domain Data Source* teaks millise meetodi abil ta andmebaasist kirjed kätte saab. *AutoLoad-i* väärtus on pandud *True-ks*. Sellega andmeid hakatakse laadima automaatselt akna avamise korral. *LoadSize-i* väärtus on 20. Selle asemel võib olla ka mingi muu suvaline väärtus. Antud juhul andmeid hakatakse lugema 20 kaupa. Tulemusena töötab rakendus paremini ja ei hangu kui andmeid on liiga palju.

Domain Data Source-i sees on deklareeritud *Domain Context*, milleks meie näite puhul on *MyDomainContext*. *DomainContext* töötab nagu kliendipoolne sild *DomainService-i* jaoks ning sisaldab tarkust, mis haldab päringuid kasutajaliidese ja andmebaasi vahel.

Nüüd oskab rakendus andmeid küsida. Viimaseks sammuks on tekitada komponent, mis suudaks neid andmeid kuvada. Selleks sobib ilusasti *DataGrid*, mille lisamiseks kasutame järgmist koodi (vt. koodinäide 6).

```
<activity:Activity IsActive="{Binding IsBusy, ElementName=dds}">
  <datagrid:DataGrid x:Name="RegisteredDataGrid"
    HorizontalAlignment="Stretch"
    ItemsSource="{Binding Data, ElementName=dds}"
    AutoGenerateColumns="False"
    IsReadOnly="True"
    Height="400" Width="600">
      <datagrid:DataGrid.Columns>
        <datagrid:DataGridTextColumn Header="Eesnimi" Binding="{Binding eesnimi}"
/>
        <datagrid:DataGridTextColumn Header="Perenimi" Binding="{Binding
perenimi}" />
        <datagrid:DataGridTextColumn Header="Vanus" Binding="{Binding vanus}" />
        <datagrid:DataGridTextColumn Header="Eriala" Binding="{Binding eriala}" />
        <datagrid:DataGridTextColumn Header="Kursus" Binding="{Binding kursus}" />
      </datagrid:DataGrid.Columns>
    </datagrid:DataGrid>
  </activity:Activity>
```

Koodinäide 6. DataGrid-i deklareerimine.

Koodist on näha, et meil on loodud *DataGrid* nimega *RegisteredDataGrid*. Elemendi *ItemSource-ks* on määratud *dds*, ehk enne loodud *Domain Data Source*. *Binding Data* tähendab, et element on seotud *Domain Data Source-i* andmetega. *AutoGenerateColumns* väärtuseks on *false*, kuna selle näite puhul me tahame deklareerida vajalikud tabeli veerud käsitsi. *DataGrid* võimaldab andmeid otseselt redigeerida. Käesoleva näite puhul me seda ei taha, seega *IsReadOnly* väärtuseks on pandud *true*. *DataGrid-i* suurus 400x600 pikslit.

Kuna me ei soovinud *DataGrid-i* veergusid automaatselt genereerida, siis edaspidi on need kirjeldatud käsitsi. Igal veerul on määratud *Header*, näiteks *Eesnimi* ja seos andmebaasi veeruga, näiteks *Binding="{Binding eesnimi}"*.

DataGrid-i ümber on *ActivityControl*, mis näitab animatsiooni siis, kui *DataGrid* loeb andmebaasist andmeid. *ActivityControl* on samamoodi seotud *Domain Data Source-iga*.

3.3.3.4 Akna avamise funktsiooni sidumine pealehe nupuga

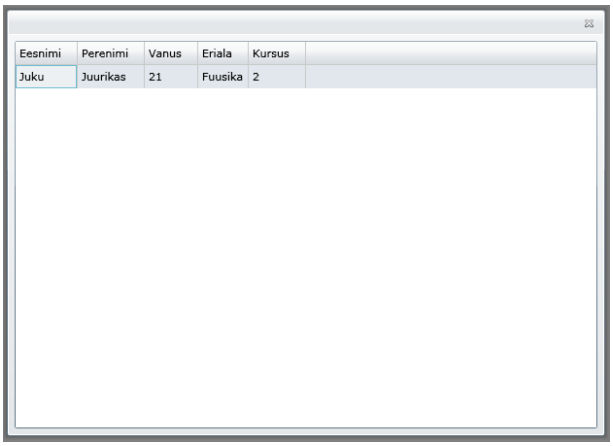
Baasi andemete loetelu näitamise komponent on valmis. Enne katsetamist seome akna vastava nupuga. Selleks teeme lahti *Expression Blend* ja määrame soovitud nupu (meie näites

kolmanda) *Click* sündmuseks *openRegistered_C*. Lisame ka vajaliku funktsiooni (vt. koodinäide 7).

```
public void openRegistered_C(object sender, System.Windows.RoutedEventArgs e)
{
  var registeredEventWindow = new Registered();
  registeredEventWindow.Show();
}
```

Koodinäide 7. Funktsioon osalejate nimekirja avamiseks.

Vajutame *F5* klahvi ja testime tulemust. Pärast kolmanda valiku vajutamist peaks ilmuma aken koos osalejate nimekirjaga (vt. joonis 30). Andmete lugemise ajal näidatakse progressiindikaatorit.



Joonis 30. Osalejate nimekiri

3.3.4. Registreerimisvormi loomine

Registreerimislehe aluseks on uus *Silverlight-i* komponent – *DataForm. DataForm* võimaldab lihtsalt määrata andmete lahtreid, mis on vajalikud andmebaasi kirje loomiseks ehk meie näites uue inimese registreerimiseks.

Alustame uue *ChildWindow-i* loomisest ja nimetame selle *Registration.xaml*. Jätame nupud ja *Grid-i* read alles, kuna neid läheb hiljem vaja. *Grid-i* suurus peaks olema *Auto* ja laius 600 pikslit, mis vastab rakenduse kujundusele, kuna on samasugune kui teistel loodud osadel.

Lisame faili päisesse uue *namespace-i* (vt. koodinäide 8), et *Visual Studio* teaks meie soovist kasutada *DataForm-i*.

```
xmlns:dataControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataForm"
```

Koodinäide 8. XML namespace DataForm-i kasutamiseks.

3.3.4.1 DataForm-i tekitamine

DataForm-i jaoks lisame järgmise koodi (vt. koodinäide 9).

```
<dataControls:DataForm x:Name="MyDataForm"
  AutoGenerateFields="False"
  CommandButtonsVisibility="None"
  CanUserDeleteItems="False"
  AutoEdit="True" >
    <dataControls:DataForm.Fields>
        <dataControls:DataFormTextField FieldLabelContent="Eesnimi: "
Binding="{Binding Mode=TwoWay, Path=eesnimi}" />
        <dataControls:DataFormTextField FieldLabelContent="Perenimi: "
Binding="{Binding Mode=TwoWay, Path=perenimi}" />
         <dataControls:DataFormTextField FieldLabelContent="Vanus: "
Binding="{Binding Mode=TwoWay, Path=vanus}" />
         <dataControls:DataFormTextField FieldLabelContent="Eriala: "
Binding="{Binding Mode=TwoWay, Path=eriala}" />
         <dataControls:DataFormTextField FieldLabelContent="Kursus: "
Binding="{Binding Mode=TwoWay, Path=kursus}" />
    </dataControls:DataForm.Fields>
</dataControls:DataForm>
```

Koodinäide 9. DataForm-i deklareerimine

Koodist on näha, et meie *DataForm-i* nimeks on pandud *MyDataForm.* Sarnaselt eelnevalt loodud *DataGrid-iga* määrame selle lahtreid käsitsi. Seega *AutoGenerateFields* on *false. DataForm-i* puhul on võimalik kohe genereerida salvestamis- ja loobumisnuppe, kuid me

hakkame kasutama nuppe, mis olid *ChildWindow-iga* kaasas. Määrame *CommandButtonsVisibility* väärtuseks *false.*

DataForm võimaldab andmete kustutamist. Kuna meie näite puhul seda funktsioonlasust ei kasutata, määrame *CanUserDeleteItems false-iks.* Kui registreerimisaken avatakse, peab *DataForm* olema andmete sisestamiseks valmis, selleks määrame *AutoEdit* väärtuseks *true*, vastasel korral peab kasutaja andmete sisestamiseks veel eraldi nupul klõpsama.

Edasi järgneb *DataForm-i* lahtrite kirjeldamine. Iga lahtri juures on olemas silt, ning lahter on seotud andmebaasiga kahesuunaliselt. Selle abil *DataForm* saab aru, et sisestatud andmeid tuleb andmebaasi salvestada.

3.3.4.2 Registreerimisakna elementide paigutamine

Paneme tähele, et meil on *Grid*, mille sees on *DataForm* ja kaks nuppu. Me tahame kõiki akna elemente õigesti positsioneerida ja selleks kasutame *Grid.RowDefinitions* omadust (vt. koodinäide 10).

```
<Grid.RowDefinitions>
  <RowDefinition Height="Auto"/>
  <RowDefinition Height="10"/>
  <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

Koodinäide 10. Grid-i ridade deklareerimine.

Vaikimisi paneb *Visual Studio* kõik elemendid ühte ritta. Me tahame, et *DataForm-i* ja nuppude jaoks oleksid eraldi read, ning nende vahel väike vahemik. Selle tulemusena näeksid aknad paremad välja. Nagu koodist on näha, meil on defineeritud kolm rida, kust esimese ja kolmanda kõrgused on automaatsed, ning teise rea kõrgus on 10 pikslit. Viimane samm oleks paigaldada nupud ja määrata nende sildid. Selleks lisame nuppudele järgmised parameetrid (vt. koodinäide 11).

```
<Button x:Name="CancelButton" Content="Loobu" Click="CancelButton_Click"
HorizontalAlignment="Right" Width="90" Height="23" VerticalAlignment="Bottom"
Margin="0,0,200,0" Grid.Row="2" />
<Button x:Name="OKButton" Content="Registreeru" Click="OKButton_Click"
HorizontalAlignment="Left" Width="90" Height="23" VerticalAlignment="Bottom"
Margin="200,0,0,0" Grid.Row="2" />
```

Koodinäide 11. Nuppudele lisaparameetrite määramine.

Nagu koodist näha on igale nupule lisatud silt. Näiteks loobumisnupu puhul on *Content=“Loobu“*. Nupud on joondatud kas vasak- või parempoole suhtes. Kõrgus on 90 pikslit, laius 23 pikslit. *VerticalAlignement* on *Bottom. Margin-id* on *Loobu* nupul 0, 0, 200, 0; ning *Registreeru* nupul 200, 0, 0, 0. Neid positsioneerimisomadusi saab määrata käsitsi. Soovides teha seda interaktiivselt, saab kasutada *Expression Blend-i.*

Et nupud oleksid paigaldatud õigesse ritta, määrame *Grid.Row* parameetri ja paneme selle väärtuseks 2, kuna *Grid-i* ridade arvutamine algab 0 ja mitte 1.

3.3.4.3 Nuppude hiire vajutamisfunktsioonide määramine

Nüüd kui kõik elemendid on paigas, lisame nuppude *Click* funktsioonidele vastavad meetodid. Avades *Registration.xaml.cs* faili näeme, et *Registreerimisnupule* on juba määratud funktsioon

```
this.DialogResult = true;
```

ning *Loobumisnupule*

```
this.DialogResult = false;
```

Loobumisnupu puhul tähendab see seda, et pärast nupu vajutamist pannakse aken kinni muudatusi salvestamata, mis on just see, mida me tahame.

3.3.4.4 Uue kirje deklareerimine

Nüüd tuleb lisada vaid natuke koodi, et isikute registreerimine töötaks. Selleks lisame enne klassi konstruktorit *public Registration()* uue isiku deklaratsioon (vt. koodinäide 12).

```
public RegTable newPerson;
```

Koodinäide 12. Uue isiku deklareerimine.

Koodist on näha, et uus isik on tüübist *RegTable.* Sellega *Visual Studio* teab, et uuel isikul on olemas sellised omadused nagu eesnimi, perekonnanimi, vanus, eriala ja kursus, kuna need on määratud andmebaasi tabelis.

Lisame konstruktorile järgmised koodiread (vt. koodinäide 13):

```
newPerson = new RegTable();
MyDataForm.CurrentItem = newPerson;
```

Koodinäide 13. Uue isiku klassi instanseerimine ja DataForm-i jooksva elemendi määramine.

Koodist on näha, et me käivitame uue isiku tüübist *RegTable* ja ütleme *DataForm-ile*, et hetkel valitud element ongi uus isik.

Lisame *Registreerimisnupu* funktsioonile järgmine rida koodi (vt. koodinäide 14).

```
this.MyDataForm.CommitItemEdit();
```

Koodinäide 14. DataForm-i andmete lokaalne salvestamine.

Eespool toodud meetod võtab *DataForm-i* sisestatud andmeid ja salvestab neid lokaalselt. Viimase sammuna edastame andmed andmebaasile.

Viitame faili päises projekti veebiserveri osale, selleks et *Visual Studio* teaks, kust kohast on võetud *RegTable* element (vt. koodinäide 15).

```
using MyRiaApp.Web;
```

Koodinäide 15. Projekti veebiserveri osa lisamine.

Vaatame veelkord üle, mis sai kuhu kohta lisatud (vt. koodinäide 16).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using MyRiaApp.Web;

namespace MyRiaApp
{
    public partial class Registration : ChildWindow
    {
        public RegTable newPerson;

        public Registration()
        {
```

```
            InitializeComponent();

            newPerson = new RegTable();
            MyDataForm.CurrentItem = newPerson;
        }

        private void OKButton_Click(object sender, RoutedEventArgs e)
        {
            this.MyDataForm.CommitItemEdit();
            this.DialogResult = true;
        }

        private void CancelButton_Click(object sender, RoutedEventArgs e)
        {
            this.DialogResult = false;
        }
    }
}
```

Koodinäide 16. Registration.xaml.cs koos õigetesse kohtadesse lisatud koodi juppidega.

3.3.4.5 Registreerimisakna sidumine pealehe nupuga

Avame *MainPage.xaml.cs* faili ja eelkõige lisame teisele nupule *Click* meetodi täpselt samamoodi, nagu eelneva kahe nupu puhul. Ainuke erinevus seisneb selles, et meetodile tuleb lisada natuke rohkem koodi, kuna meil oleks vaja salvestada uue isiku andmeid pärast seda, kui registreerimisaken on suletud. (vt. koodinäide 17).

```
private void openReg_B(object sender, RoutedEventArgs e)
{
  var registrationEventWindow = new Registration();

  registrationEventWindow.Closed += registrationEventWindow_Closed;
  registrationEventWindow.Show();
}
```

Koodinäide 17. Ürituse registreerimisnupu Click meetod.

Koodist on näha, et esialgu käivitatakse *Registration* akna. Teine rida koodi määrab funktsiooni, mida kutsutakse välja siis, kui kasutaja paneb *Registreerimisakna* kinni. Sellist funktsiooni nimetatakse *EventHandler-iks.* Viimasena näidatakse registreerimisakent.

Edasi tuleb lisada *registrationEventWindow_Closed* meetod (vt. koodinäide 18). Seda võib panna kohe pärast *openReg_B* funktsiooni.

```
void registrationEventWindow_Closed(object sender, EventArgs e)
{
  var win = sender as Registration;
```

```
  var context = RegisteredUC.dds.DomainContext as MyDomainContext;

  if (win.DialogResult == true)
  {
    context.RegTables.Add(win.newPerson);
    RegisteredUC.dds.SubmitChanges();
  }
}
```

Koodinäide 18. registrationEventWindow_Closed meetod.

Koodist on näha, et alguses määratakse sündmuse saatjaks registreerimisaken. Siis tuleb deklareerida *Domain Context-i*. Edasi järgneb tingimus: kui kasutaja vajutas *Registreeri* nuppu, siis tuleb võtta uue isiku andmeid ja lisada neid kontekstile. Lõpuks kutsume *Domain Data Source-i*, mis oli juba deklareeritud *Registered.xaml* failis, ning käivitame *SubmitChanges()* meetodi. See meetod võtab uue isiku andmed ja salvestab need andmebaasi.

Kuna me ei tahtnud deklareerida *Domain Data Source-i* teist korda, siis tuleb seletada *Visual Studio-le*, kus kohas olemasolev kirjeldus asub. Selleks lisame enne *MainPage* konstruktorit järgmise koodirea (vt. koodinäide 19).

```
Registered RegisteredUC = new Registered();
```

Koodinäide 19. RegisteredUC deklareerimine.

Lisame ka muutaja *context* kirjelduse, et *Visual Studio* teaks, mis asi see ikka on (vt. koodinäide 20).

```
public MyDomainContext context = new MyDomainContext();
```

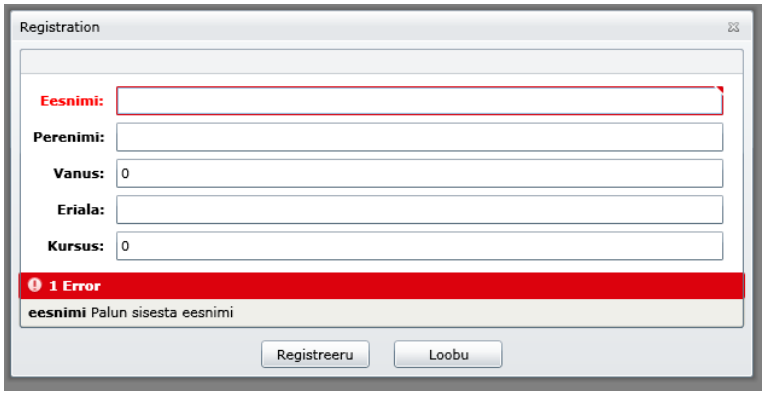
Koodinäide 20. Muutuja context deklareerimine.

Lisame faili päisesse projekti veebiosa deklaratsiooni, et *Visual Studio* teaks, kus kohas asub *MyDomainContext* (vt. koodinäide 21).

```
using MyRiaApp.Web;
```

Koodinäide 21. Projekti veebiosa deklareerimine.

Vajutame *F5* klahvi ning testime tulemuse. Kuna valideerimisreeglid olid juba määratud, siis meie uus *DataForm* oskab ilusti näidata ka veateateid (vt. joonis 31).



Joonis 31. Uus DataForm koos andmete valideerimisega.

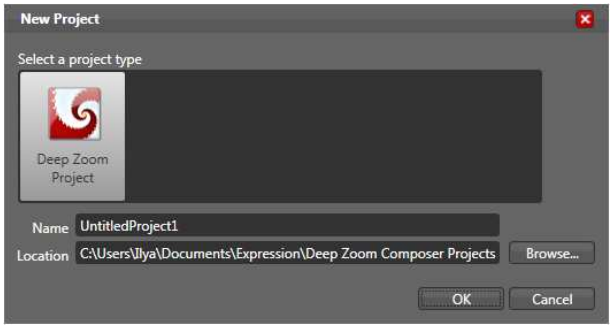
3.3.5. Deep Zoom galerii loomine

Järgmine samm oleks luua interaktiivne pildigalerii ja integreerida see olemasoleva rakendusega. Galerii loomiseks kasutame *Microsoft Deep Zoom Composer* tarkvara.

Deep Zoom-i omapära seisneb selles, et galerii eksportimise ajal jagatakse pildid väikesteks tükkideks, ning pannakse nende asukohad kirja *dzc_output.xml* faili. Sedasi on *Deep Zoom* galeriis võimalik kasutada väga kõrge resolutsiooniga pilte. Nende suumimine toimub sujuvalt isegi aeglase interneti ühendusega.

3.3.6. Deep Zoom Composer

Teeme *Deep Zoom Composer-i* lahti ning valime *New Project.* Ekraanile ilmub aken, kus tuleb määrata projekti nimi ja asukoht (vt. joonis 32).



Joonis 32. Uue Deep Zoom projekti loomine

Nimeks paneme *MyGalleryProject* ning *Location-iks Desktop-i.* Klõpsame *OK* nuppu.

Uus projekt on loodud ning programm näitab tühja stseeni, kuhu tuleb lisada pilte. Klõpsame nupul *Add image* ning impordime pildid kataloogist. Parempoolse ribale peaks ilmuma nimekiri imporditud piltidest (vt. joonis 33).

Joonis 33. Nimekiri imporditud piltidest.

Paneme tähele, et *Composer-i* akna üleval pool on kolm nupu: *Import*, *Compose* ja *Export* (vt. joonis 34). Need nupud peegeldavad *Deep Zoom Composer-i* töövoogu. Esimene samm on piltide importimine. Teisel sammul loob kasutaja kollaaži, määrates piltide suuruse ja asukoha. Viimane samm on eksportida loodud galerii.

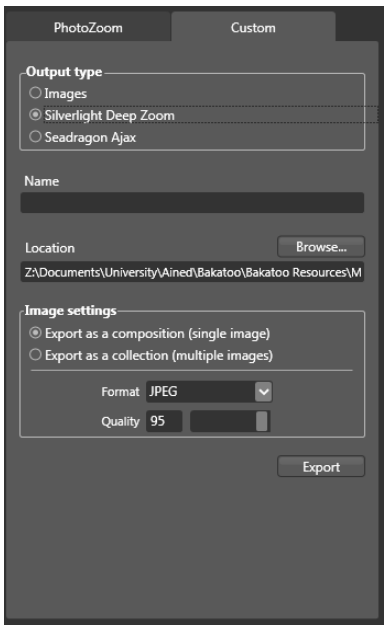
Joonis 34. Deep Zoom Composer-i töövoogu peegeldavad nupud.

Pildid on imporditud ja saab liikuda teise sammu poole. Vajutame *Compose* nupule. *Compose* vaates on täpselt samasugused elemendid, keskel tühi stseen ja paremal pool piltide nimekiri. Lohistame pildid stseenile soovitud järjekorras ning määrame nende suurused ja asukohad. Tulemus võiks olla näiteks selline (vt. joonis 35).

Joonis 35. Pildid organiseeritud suvaliselt Deep Zoom Composer-i stseenil.

Kui pildid on paigas, liigume viimase sammu poole. Klõpsame nupul *Export*.

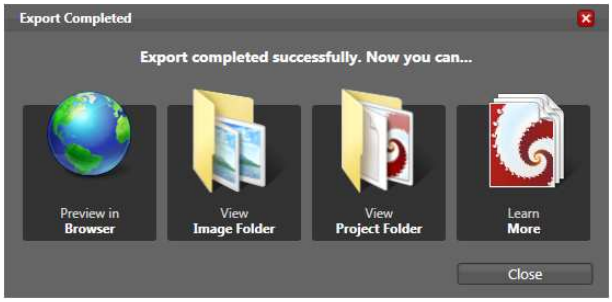
Paneme tähele, et *Deep Zoom Composer* pakub vaikimisi võimalust sisse logida *Microsoft PhotoZoom* teenusele *Windows Live ID-ga* ning publitseerida loodud galerii internetis. Hetkel me seda ei soovi, seega vajutame *Custom* nupule. Siin pakutakse mitu valikut galerii eksportimiseks (vt. joonis 36).



Joonis 36. Deep Zoom galerii eksportimisvõimalused.

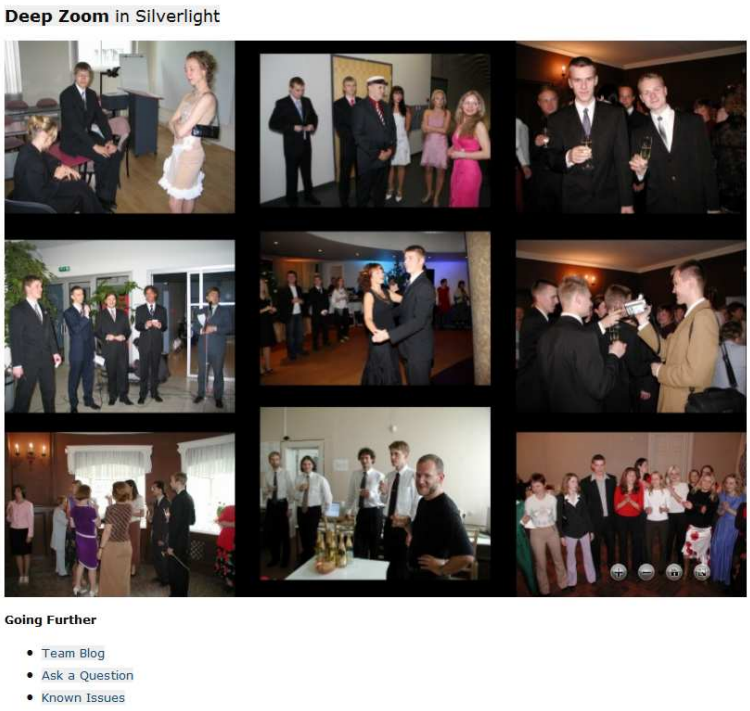
Eksportimiseks on olemas kolm võimalust. Esimene oleks lihtsalt pildid salvestada. Teine võimalus on luua *Silverlight-i* rakendus, ning kolmas – luua *Seadragon Ajax* veebirakendus. Valime teise võimaluse, kuna me soovime integreerida loodud galeeri olemasoleva rakendusega ja meil oleks vaja genereerida koodi, mis oskaks hallata hiire nuppude vajutamisi, ning kerimisrulliga suumimist.

Määrame rakendusele nime. See võiks olla näiteks *DeepZoomProject*. Asukohaks määrame *Desktop-i*. Kui eksportimine on valmis, pakub *Deep Zoom Composer* võimalust avada loodud galerii brauseris, avada piltide kataloogi või avada loodud rakenduse kataloogi (vt. joonis 37). Valime *Preview in Browser*.



Joonis 37. Võimalikud toimingud pärast galerii eksportimist.

Brauser avatakse ja saame katsetada oma interaktiivset pildigaleriid (vt. joonis 38).



Joonis 38. Deep Zoom galerii brauseri aknas.

Galerii on loodud. Järgmise sammuna integreerime selle olemasoleva rakendusega.

3.3.6.1 Galerii akna loomine MyRiaApp projektis

Selleks, et pildigaleriid oleks võimalik meie rakenduses kuskile paigutada, tekitame *MyRiaApp* projektis uue *ChildWindow-i*. Nimeks paneme *Gallery.xaml*. Kustutame kõik mittevajalikud komponendid ära ning jätame ainult *Grid-i*, mille suurused väärtusteks oleksid *Auto*.

Grid-i sisse paigutame *Border-i*, mille suurus on 600x400 pikslit. Raam on vajalik selleks, et piirata hiire nuppude vajutamispiirkonda. Ilma selleta hakkab rakendus viskama veateateid. *Border-i* sisse lisame *MultiScaleImage* komponendi, mille nimeks paneme *msi*, ning suuruseks määrame 600x400 pikslit (vt. koodinäide 22). *MultiScaleImage-t* kasutatakse *Deep Zoom* galerii näitamiseks.

```
<MultiScaleImage x:Name="msi" Source="/GeneratedImages/dzc_output.xml" Width="600"
Height="400"/>
```

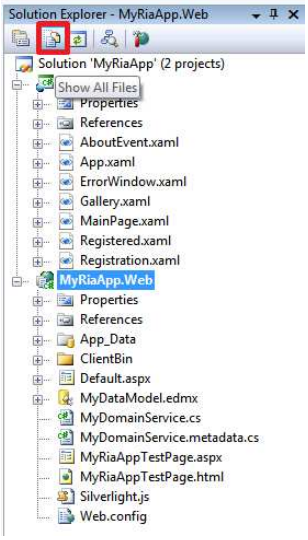
Koodinäide 22. MultiScaleImage-i lisamine.

Koodist on näha, et *MultiScaleImage-i* allikaks on *dzc_output.xml*, mis asub kataloogis *GeneratedImages.* See *XML* fail hoiab kõikide piltide asukohti meie galeriis, ning on *Deep Zoom Composer-i* poolt automaatselt genereeritud.

3.3.6.2 GeneratedImages kataloogi lisamine projektile

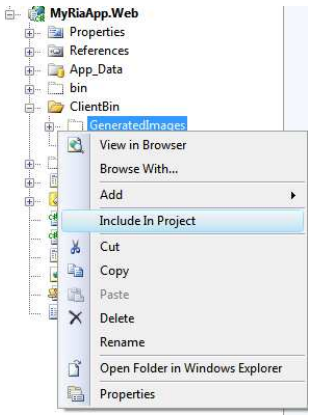
Nüüd tuleb projektile lisada *GeneratedImages* kataloog koos *dzc_output.xml* failiga. Selleks teeme lahti *Desktop\deepzoomproject\DeepZoomProjectWeb\ClientBin* kausta, kus asub meie eksporditud *Deep Zoom-i* pildigalerii. Leiame sealt kausta *GeneratedImages* ning kopeerime selle *Desktop\MyRiaApp\MyRiaApp.Web\ClientBin* kataloogi, mille sees peaks praegu olema ainult *MyRiaApp.xap* fail.

Galerii pildid on lisatud, kuid *Visual Studio* sellest veel ei tea, kuna lisamine toimus väljaspool keskkonda. Selleks, et lisada pildigalerii kataloogi projekti puuse, avame *Visual Studio*, valime *MyRiaApp.Web* juure (*root*) ning vajutame *Show All Files* nuppu (vt. joonis 39).



Joonis 39. Show All Files nupp Visual Studio-s.

Tulemusena peaks *ClientBin* kataloogi ilmuma kaust *GeneratedImages*. Paneme tähele, et kataloogil on punktiiriline kujund, mis tähendab, et ta on küll füüsiliselt olemas, kuid projektile pole veel lisatud. Teeme paremklõps *GeneratedImages* kaustal ning valime *Include In Project* (vt. joonis 40).



Joonis 40. GeneratedImages kataloogi lisamine projektile.

Nüüd *Visual Studio* teab, kus kohas asuvad pildid ja nende parameetreid määrav *XML* fail.

3.3.6.3 Hiire sündmustele reageerimine

Galerii on küll rakendusega integreeritud, kuid meie rakendus ei oska hiire sündmusi hallata. Selleks lisame ühe *Deep Zoom Composer-i* poolt genereeritud klassi ning natuke koodi.

Teeme paremklõpsu *MyRiaApp* juure peal ning valime *Add -> Existing Item.* Lisame *MouseWheelHelper.cs* faili kataloogist *Desktop\deepzoomproject\DeepZoomProject.* See klass sisaldab koodi, mis oskab hallata hiire kerimisrulli sündmusi.

Teeme *MouseWheelHelper.cs* faili lahti, ning vahetame *namespace-i DeepZoomProject MyRiaApp* vastu (vt. koodinäide 23).

```
namespace MyRiaApp
```

Koodinäide 23. MouseWheelHelper.cs namespace-i muutmine.

Teeme lahti *Gallery.xaml.cs* faili, ning kustutame *OK* ja *Cancel* nuppude sündmused, kuna neid kasutama ei hakka. Lisame meetodid, mis olid automaatselt genereeritud *Deep Zoom Composer-i* poolt. Tulemus peaks olema selline (vt. koodinäide 24).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace MyRiaApp
{
  public partial class Gallery : ChildWindow
  {
  //All of the code below is generated automatically by DeepZoom Composer. The
code handles mouse events, such as double-click, zoom in
  //with the mouse wheel etc.
    double zoom = 1;
    bool duringDrag = false;
    bool mouseDown = false;
    Point lastMouseDownPos = new Point();
    Point lastMousePos = new Point();
    Point lastMouseViewPort = new Point();


    public double ZoomFactor
    {
      get { return zoom; }
      set { zoom = value; }
    }

    public Gallery()
    {
      // Required to initialize variables
      InitializeComponent();

      //
      // Firing an event when the MultiScaleImage is Loaded
      //
      this.msi.Loaded += new RoutedEventHandler(msi_Loaded);

      //
      // Firing an event when all of the images have been Loaded
      //
      this.msi.ImageOpenSucceeded += new
RoutedEventHandler(msi_ImageOpenSucceeded);

      //
      // Handling all of the mouse and keyboard functionality
      //
      this.MouseMove += delegate(object sender, MouseEventArgs e)
      {
        lastMousePos = e.GetPosition(msi);

        if (duringDrag)
        {
          Point newPoint = lastMouseViewPort;
          newPoint.X += (lastMouseDownPos.X - lastMousePos.X) / msi.ActualWidth *
msi.ViewportWidth;
          newPoint.Y += (lastMouseDownPos.Y - lastMousePos.Y) / msi.ActualWidth *
msi.ViewportWidth;
          msi.ViewportOrigin = newPoint;
        }
      };

      this.MouseLeftButtonDown += delegate(object sender, MouseButtonEventArgs e)
      {
```

```
        lastMouseDownPos = e.GetPosition(msi);
        lastMouseViewPort = msi.ViewportOrigin;

        mouseDown = true;

        msi.CaptureMouse();
      };

      this.MouseLeftButtonUp += delegate(object sender, MouseButtonEventArgs e)
      {
        if (!duringDrag)
        {
          bool shiftDown = (Keyboard.Modifiers & ModifierKeys.Shift) ==
ModifierKeys.Shift;
          double newzoom = zoom;

          if (shiftDown)
          {
            newzoom /= 2;
          }
          else
          {
            newzoom *= 2;
          }

          Zoom(newzoom, msi.ElementToLogicalPoint(this.lastMousePos));
        }
        duringDrag = false;
        mouseDown = false;

        msi.ReleaseMouseCapture();
      };

      this.MouseMove += delegate(object sender, MouseEventArgs e)
      {
        lastMousePos = e.GetPosition(msi);
        if (mouseDown && !duringDrag)
        {
          duringDrag = true;
          double w = msi.ViewportWidth;
          Point o = new Point(msi.ViewportOrigin.X, msi.ViewportOrigin.Y);
          msi.UseSprings = false;
          msi.ViewportOrigin = new Point(o.X, o.Y);
          msi.ViewportWidth = w;
          zoom = 1 / w;
          msi.UseSprings = true;
        }

        if (duringDrag)
        {
          Point newPoint = lastMouseViewPort;
          newPoint.X += (lastMouseDownPos.X - lastMousePos.X) / msi.ActualWidth *
msi.ViewportWidth;
          newPoint.Y += (lastMouseDownPos.Y - lastMousePos.Y) / msi.ActualWidth *
msi.ViewportWidth;
          msi.ViewportOrigin = newPoint;
        }
      };

      new MouseWheelHelper(this).Moved += delegate(object sender,
MouseWheelEventArgs e)
```

```
        {
          e.Handled = true;

          double newzoom = zoom;

          if (e.Delta < 0)
          newzoom /= 1.3;
          else
          newzoom *= 1.3;

          Zoom(newzoom, msi.ElementToLogicalPoint(this.lastMousePos));
          msi.CaptureMouse();
        };
      }

      void msi_ImageOpenSucceeded(object sender, RoutedEventArgs e)
      {
        //If collection, this gets you a list of all of the MultiScaleSubImages
        //
        //foreach (MultiScaleSubImage subImage in msi.SubImages)
        //{
        //    // Do something
        //}

        msi.ViewportWidth = 1;
      }

      void msi_Loaded(object sender, RoutedEventArgs e)
      {
        // Hook up any events you want when the image has successfully been opened
      }

      private void Zoom(double newzoom, Point p)
      {
        if (newzoom < 0.5)
        {
          newzoom = 0.5;
        }

        msi.ZoomAboutLogicalPoint(newzoom / zoom, p.X, p.Y);
        zoom = newzoom;
      }

      private void ZoomInClick(object sender, System.Windows.RoutedEventArgs e)
      {
        Zoom(zoom * 1.3, msi.ElementToLogicalPoint(new Point(.5 * msi.ActualWidth,
.5 * msi.ActualHeight)));
      }

      private void ZoomOutClick(object sender, System.Windows.RoutedEventArgs e)
      {
        Zoom(zoom / 1.3, msi.ElementToLogicalPoint(new Point(.5 * msi.ActualWidth,
.5 * msi.ActualHeight)));
      }

      private void GoHomeClick(object sender, System.Windows.RoutedEventArgs e)
      {
        this.msi.ViewportWidth = 1;
        this.msi.ViewportOrigin = new Point(0, 0);
        ZoomFactor = 1;
      }
```

```
      private void GoFullScreenClick(object sender, System.Windows.RoutedEventArgs
e)
      {
        if (!Application.Current.Host.Content.IsFullScreen)
        {
          Application.Current.Host.Content.IsFullScreen = true;
        }
        else
        {
          Application.Current.Host.Content.IsFullScreen = false;
        }
      }

      // Handling the VSM states
      private void LeaveMovie(object sender, System.Windows.Input.MouseEventArgs e)
      {
        VisualStateManager.GoToState(this, "FadeOut", true);
      }

      private void EnterMovie(object sender, System.Windows.Input.MouseEventArgs e)
      {
        VisualStateManager.GoToState(this, "FadeIn", true);
      }


      // unused functions that show the inner math of Deep Zoom
      public Rect getImageRect()
      {
        return new Rect(-msi.ViewportOrigin.X / msi.ViewportWidth, -
msi.ViewportOrigin.Y / msi.ViewportWidth, 1 / msi.ViewportWidth, 1 /
msi.ViewportWidth * msi.AspectRatio);
      }

      public Rect ZoomAboutPoint(Rect img, double zAmount, Point pt)
      {
        return new Rect(pt.X + (img.X - pt.X) / zAmount, pt.Y + (img.Y - pt.Y) /
zAmount, img.Width / zAmount, img.Height / zAmount);
      }

      public void LayoutDZI(Rect rect)
      {
        double ar = msi.AspectRatio;
        msi.ViewportWidth = 1 / rect.Width;
        msi.ViewportOrigin = new Point(-rect.Left / rect.Width, -rect.Top /
rect.Width);
      }
    }
}
```

Koodinäide 24. Kood hiire sündmuste haldamiseks.

3.3.6.4 Akna avamissündmuse lisamine Galerii nupule

Galerii ja hiiresündmuste haldamisfunktsioonid on lisatud. Nüüd peaksime lisama pealehel asuvale neljandale nupule koodi, et ta saaks galerii avada. Avame *MainPage.xaml* faili, ning lisame *btnD1* nupule *Click* sündmuse (vt. koodinäide 25).

```
<Button  x:Name="btnD1"  HorizontalAlignment="Right"  Content="Button"  Width="100"
Height="100" Click="openGallery_D"/>
```

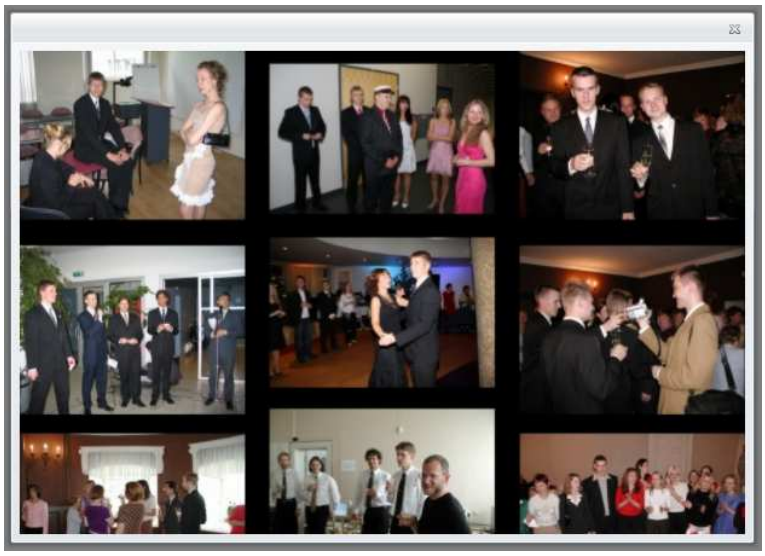
Koodinäide 25. BtnD1 Click sündmuse lisamine.

Teeme lahti *MainPage.xaml.cs* faili, ning lisame ka vastava meetodi (vt. koodinäide 26).

```
public void openGallery_D(object sender, System.Windows.RoutedEventArgs e)
{
  var galleryEventWindow = new Gallery();
  galleryEventWindow.Show();
}
```

Koodinäide 26. Galeriiakna avamismeetod.

Paneme rakenduse *F5* klahviga käima ning katsetame tulemust. Pealehe neljanda nupu vajutamisega peaks ekraanile ilmuma aken interaktiivse pildigaleriiga (vt. joonis 41).



Joonis 41. Interaktiivne ürituse pildigalerii.

Kokkuvõtte

Käesoleva töö eesmärgiks oli luua õppematerjali prototüüp, mille põhjal saaks õppija tuttavaks *Microsoft Silverlight-i* tehnoloogiaga ning sellel põhinevate rakenduste loomise töövooga.

Töö kirjutamise tulemusena:

- Loodi *Veebistuudiumi* koolituse eesmärkidele vastav näidisrakendus;
- Koostati näidisrakendusel põhinev õppematerjali prototüüp, millest lähtudes luuakse *Veebistuudiumi* koolitusmaterjalid.

Õppematerjal püüab detailselt illustreerida *Microsoft Silverlight-i* tehnoloogial põhineva *.NET RIA* (*Rich Internet Application*) rakenduse loomist. Materjali on püütud teha lihtsaks ja arusaadavaks. Õppijale tuleb kasuks eelnev programmeerimis- ja *HTML* koodi kirjutamiskogemus.

Materjal illustreerib selliseid *.NET RIA* rakenduse loomise etappe nagu:

- Uue projekti tekitamine;
- Andmebaasi loomine ja andmete sisestamine;
- Lihtsa rakenduse kasutajaliidese loomine;
- Registreerimisvormi ja registreerunud isikute nimekirja loomine;
- Interaktiivse pildigaleeri loomine ja selle integreerimine olemasoleva rakendusega.

Töö autor kavatseb ka edaspidi jälgida *Microsoft Silverlight-i* arengut ja on kindel, et sellised tehnoloogiad muudavad inimeste ettekujutust nii veebist kui ka selle kasutamisest.

Annotation

Microsoft Silverlight 3 Tutorial for Creation of a Data Driven Application as an Example.

BSc Thesis by Ilya Shmorgun

The purpose of the given thesis was creating a thorough tutorial, which would serve as an introduction to the *Microsoft Silverlight* technology as well as *Silverlight* application creation workflow.

The result of writing the given thesis is:

- Creation of an example application, which corresponds to the goals of the *WebStudio* educational seminar;
- Creation of a tutorial for developing applications, similar to the example. The tutorial will serve as a basis for creation of a more thorough *WebStudio* educational material.

The created tutorial tries to closely illustrate the process of creating a *Microsoft .NET RIA* application using *Silverlight* technology. The goal was to create a material, which would be simple and easy to understand even for a person, who is not deeply familiar with *Microsoft .NET Framework* or *Silverlight* based application development. Still experience with programming or *HTML* coding and some familiarity with *XML* will greatly simplify the process of understanding different steps of the tutorial.

The material illustrates such steps of creating a *.NET RIA* application as:

- Creation of a new project;
- Creation of a database and entering data;
- Creation of a simple user interface for the application;
- Creation of a registration form and list of registered individuals;
- Creation of an interactive photo gallery and it's integration with an existing *Silverlight* application.

The author of this material is determined to keep a close look on the development of *Silverlight* and is sure, that technologies of this kind will change people's perception of the web itself and ways of interaction with it.

Kasutatud kirjandus

The Official Microsoft Silverlight Site Forums: http://silverlight.net/forums/ (kasutamise kuupäev: 20. Aprill 2009. a.)

The Official Microsoft Silverlight Site: http://silverlight.net (kasutamise kuupäev: 20. Aprill 2009. a.)

MicroApplications, Inc. Web Log: http://www.microapplications.com/blog/ (kasutamise kuupäev: 20. Aprill 2009. a.)

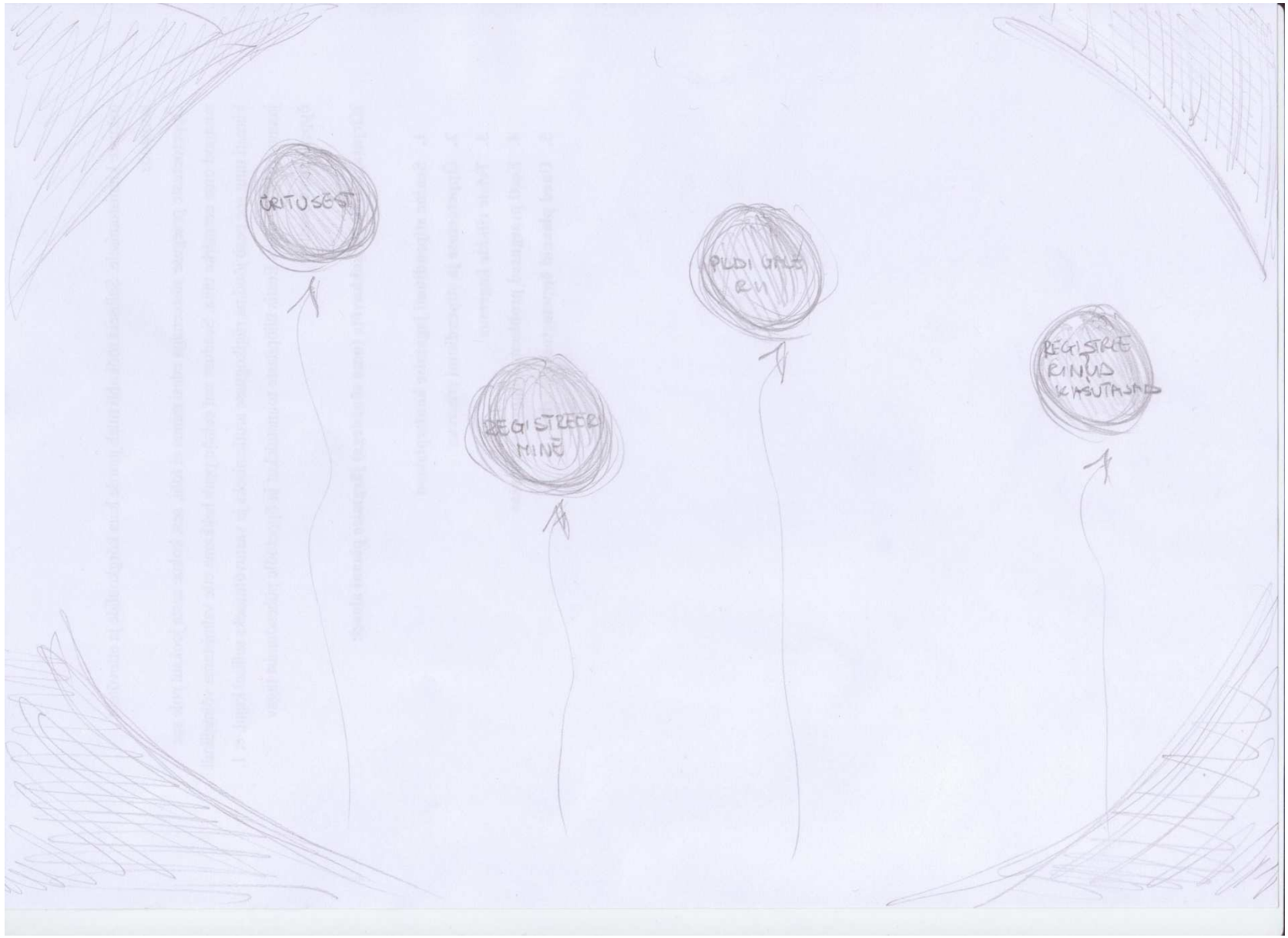
Abrams, B. Design Guidelines, Managed code and the .NET Framework: http://blogs.msdn.com/brada/ (kasutamise kuupäev: 20. Aprill 2009. a.)

Microsoft .NET RIA Services Overview. Data-Driven RIA with Silverlight: http://www.microsoft.com/downloads/details.aspx?FamilyID=76bb3a07-3846-4564-b0c3-27972bcaabce&displaylang=en (kasutamise kuupäev: 20. Aprill 2009. a.)

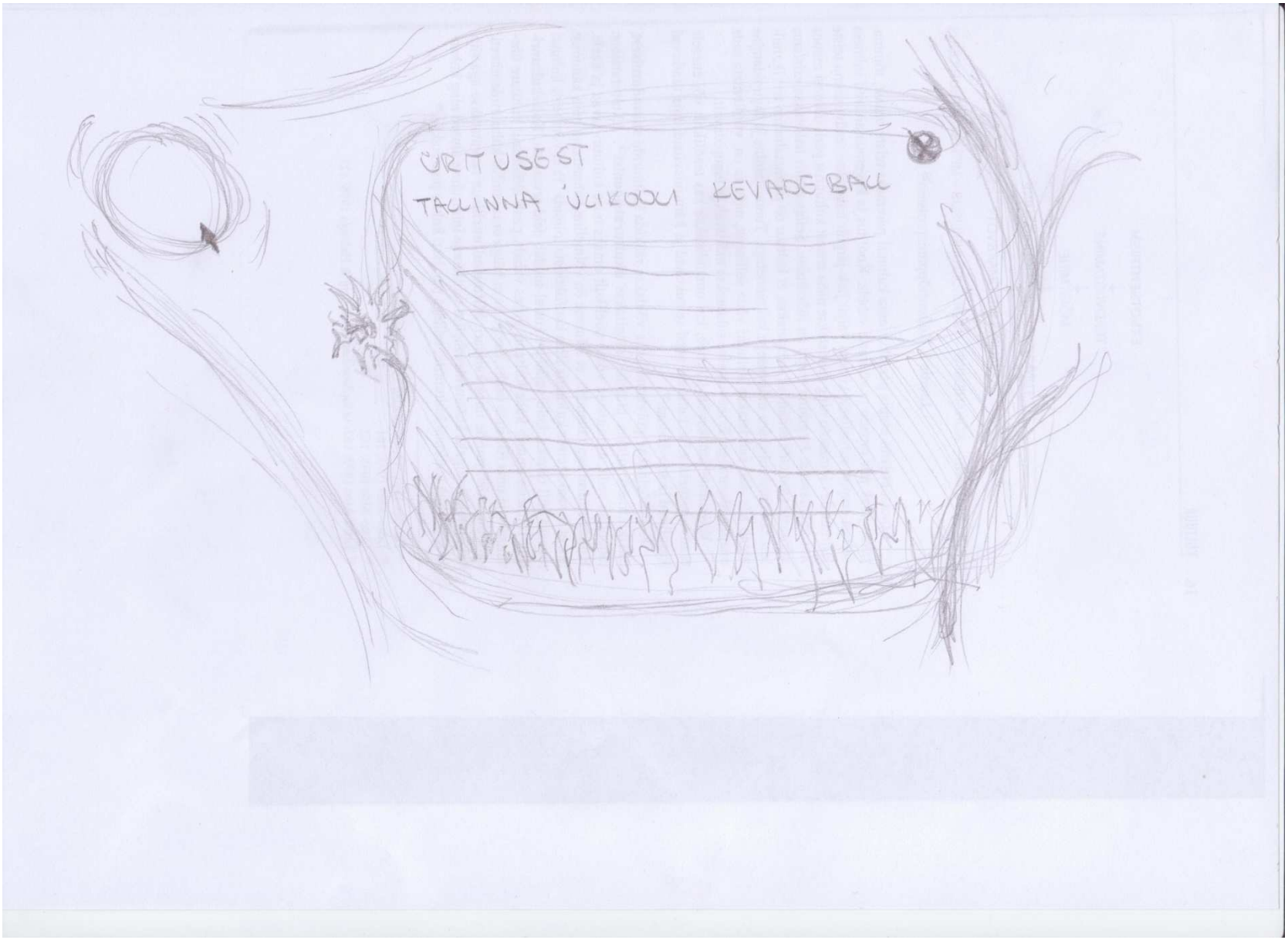
Moroney, L. (2008). *Introducing Microsoft Silverlight 2.*

LISAD

Lisa 1. Rakenduse töövoogu esimene variant.



Lisa 2. Rakenduse töövoogu teine variant.



Lisa 3. Rakenduse töövoogu kolmas variant.