Tallinna Ülikool

Informaatika Instituut

# Catel raamistik ja MVVM muster WPF rakendustes

**Bakalaureusetöö**

Autor: Lauri Mattus

Juhendaja: Jaagup Kippar

Autor: ..... „ ..... „2014

Juhendaja: ..... „ ..... „2014

Instituudi direktor: ..... „ ..... „2014

Tallinn 2014

# Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

..... .....

(kuupäev) (autor)

# Sisukord

# Sissejuhatus

Windows on siiani enim kasutatav operatsioonisüsteem ja ei saa jätta tähelepanuta, et kuigi paljud rakendused kolivad tänapäeval ära veebi või nutitelefonidesse, on Windowsi töölauarakendustel siiski säilinud suur osakaal.

Windowsi rakendusi saab luua erinevates programmeerimiskeeltes ning erinevates keskkondades, kuid kõige mugavam on Microsofti poolt loodud operatsioonisüsteemi tarvis kirjutada ka programme kasutades Microsofti tehnoloogiad, milleks on Visual Studio arenduskeskkond, .NET raamistik, C# programmeerimiskeel ning WPF rakendusetüüp.

WPF rakendustes on soovitatav kasutada MVVM mustrit, et hoida lahus äriloogika, kasutajaliides ning andmed. Antud muster on liikumas ka teistesse arenduskeskkondadesse, seega on see teema aktuaalne, kuid puudub eesti keelne materjal sellega tutvumiseks.

WPF ja MVVM mustri kasutamiseks on loodud erinevaid raamistikke, mis teevad arendaja eest palju samme ära, et arendaja saaks keskenduda rohkem loogikale mitte koodi kirjutamisele. Üheks nendest raamistikest on Catel, kuid puudub hea õppematerjal sellega tutvumiseks. Selle raamistiku tutvustamiseks ongi pandud kokku järnev töö, mille lisana valmib õppematerjal.

Esimeses peatükis tutvustatakse Catel raamistikku ning MVVM mustrit. Antakse ülevaade olemasolevatest materjalidest, erinevate platvormide toest ning võrreldakse seda teiste alternatiivsete raamistikega.

Teises peatükis kirjeldab autor ära kes on õppematerjali sihtrühm. Kirjeldatakse ära ka õppematerjali struktuur ning õpiväljundid.

Kolmas peatükk keskendub õppematerjali testimisele ning testimise tulemustele.
Valmiv õppematerjal Catel raamistiku kohta on esitatud kääsoleva bakalaureusetöö lisana.

# Olulisemad mõisted

MVC – Model-View-Controller muster

WPF – Windows Presentation Foundation. Vahendid Windowsi rakenduse tegemiseks kasutades
.NET raamistikku.

Silverlight – Veebiraamistik interaktiivsete komponentide ja animatsioonide tegemiseks.

Windows Phone – Operatsioonisüsteem mobiiltelefonidele.

WinRT – Operatsioonisüsteem tahvelarvutitele.

ASP.NET MVC – Veebiraamistik veebilehekülgede tegemiseks.

.NET raamistik – Tarkvara raamistik, mis on loodud peamiselt Windowsi rakenduste jooksutamiseks.

Line of Business application – Äritarkvara, üldiselt tarkvara, mis on töökindel, mis peaks toetama
palju kasutajaid ja palju samaaegseid päringuid (ingl.k request).

API – (ingl.k Application Program Interface) kireldab rutiinid, protokollid ja vahendid tarkvara arendamiseks

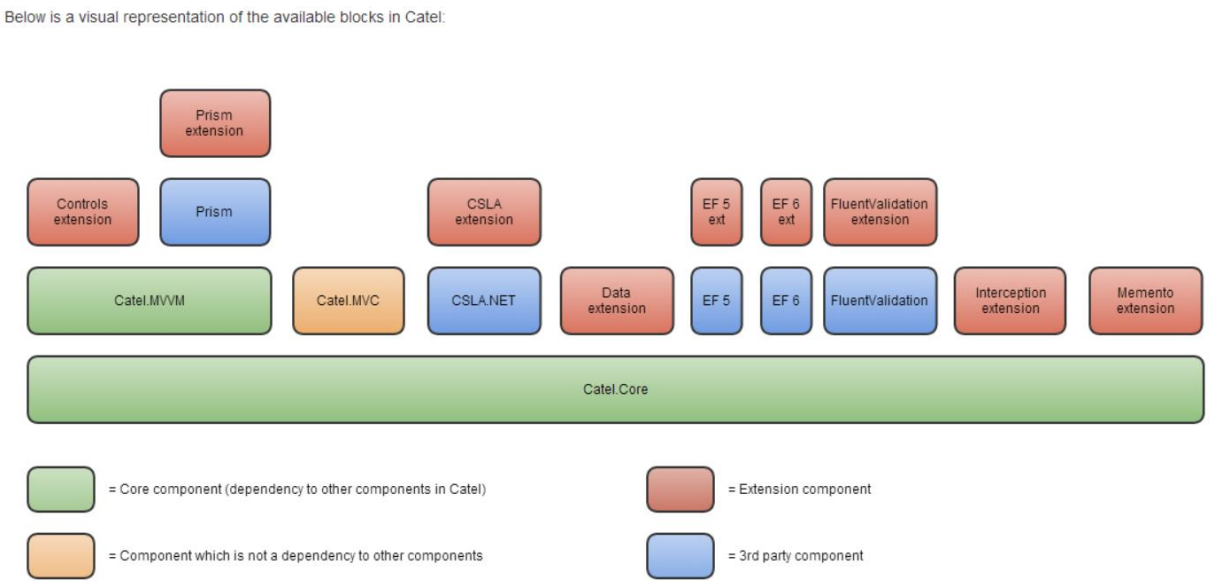
Code Snipet – koodiplokk, mida on võimalik automaatselt genereerida ja siis vajalikus kohas kasutada.

# 1. Catel raamistik

Catel on tarkvare arenduse platvorm, mille fookuses on MVVM (ingl.k Model-View-Viewmodel) muster (WPF, Silverlight, Windows Phone ja WinRT) ning MVC (ingl.k Model-View-Controller) muster (ASP.NET MVC). Cateli eesmärk on pakkuda moodulite komplekti Line Of Business rakenduste kirjutamiseks kõigis .NET tehnoloogiates, alates kliendist kuni serverini (Geert van Horrik, 2014).

Antud töös on autor koostanud õppematerjali WPF rakenduse loomiseks kasutades MVVM mustrit.

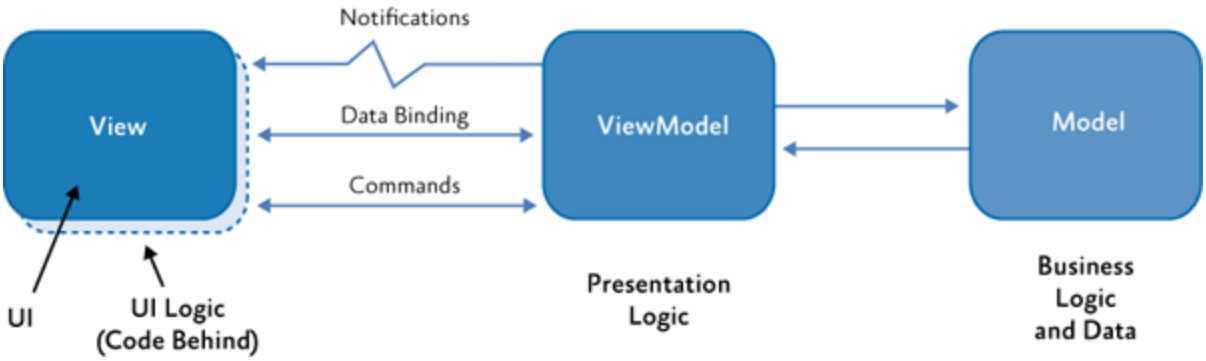
Lisaks Cateli tuumale, on olemas palju täiendusi (ingl.k extensions), mida kasutavad saavad valida. Nende seas on näiteks Dynamic Objects,  Objects, Entity Framework, Fluent Validation, Memento mustrid ning Prism  (Geert van Horrik, 2014).




Joonis 1. Visuaalne representatsioon saadaval olevatest Cateli plokkidest (Geert van Horrik, 2014)

## 1.1. MVVM muster

Mudel-Vaade-VaateMudel muster (ingl.k Model-View-ViewModel pattern) aitab eraldada rakenduse äriloogika ning presenteerimise loogika. Rakenduse loogika ning kasutajaliidese eraldamine toob kaasa lihtsama testmise, haldamise ning arendamise. See suurendab koodi taaskasutamise võimalusi ja lubab arendajal ja kasutajaliidese disaineril teha paremini koostööd. Kasutades MVVM mustrit, jagatakse kasutajalidies, presentatsioon ning äriloogika kolme eraldi klassi: vaade, mis sisaldab ainult kasutajaliidest ja kasutajaliidese loogikat; vaate mudel, mis sisaldab presentatsiooni loogikat ja olekut (ingl.k state); mudel, mis sisaldab rakenduse äriloogikat ja andmeid (Microsoft, 2014).




Joonis 2. MVVM komponentide suhtlus (Microsoft, 2014).

View jagatakse kaheks. Kasutajaliides UI (ingl.k User Interface) sisaldab ainult disaini, mis on xml kujul failis laiendiga .xaml. Teine osa on UI loogika, kuhu on võimalik kirja panna programmikoodi.

ViewModel on ühendatud View'ga läbi sidumise (Ingl.k. Binding). Bindingu võimalusteks on: Data Binding (andmete sidumine), mis seob ViewModeli mingi parameetri view mingi väljaga; Command Binding (käskude sidumine), mis seob mingi UI käsu ViewModeli käsuga (näiteks nupu käsk klikk, seotakse ViewModel'i meetodiga, mis käivitatakse selle nupu klikkimisel).

Model on tavaliselt andmemudel, mis on seotud näiteks andmebaasis samanimelise tabeliga.

ViewModelis on kirjeldatud parameetrid mudelitena ja need mudelid on seotud siis UI mingite elementidega.

## 1.2. Ülevaade saadaval olevatest materjalidest

**Catel examples**

Autor: Geert von Horrik

Viimane muudatus: 2014

Aadress: https://github.com/Catel/Catel.Examples (12.12.2014)

Cateli meeskonna poolt on loodud GitHubi konto, kus on saadaval näited. Seal on näited Silverlighti, Windows 8 ja Windows Phone 8 kohta.

Näited on komponentide põhised ning on abiks autori poolt koostatud materjali kõrvale tutvumiseks. Rohkem asjalikke näiteid autor Cateli kohta ei leidnud.

**Catel - Part 5 of n: Building a WPF example application with Catel in 1 hour**

Autor: Geert von Horrik

Viimane muudatus: 2011

Aadress. *http://www.codeproject.com/script/Articles/View.aspx?aid=151491* (12.12.2014)

On veel Cateli ühe looja poolt tehtud õpetus, aga kahjuks, selle näite järgi tehtud programm ei käivitu, sest tegu on 2011 aastal koostatud koodiga sotsiaalmeediaga suhtlemiseks, kuid sotsiaalmeediaga suhtlemiseks loodud API'd muutuvad väga kiiresti ja tihti. Seega, see õpetus ei sobi Cateliga tutvuse tegemiseks, kuna puudub tagasiside töötava rakenduse näol. Niiet autor otsustaski luua oma õppematerjali.

**Catel**

Autor: Geert von Horrik

Viimane muudatus: 2014

Aadress: https://github.com/catel/catel (12.12.2014)

Siit saab leida cateli lähtekoodi, et uurida, kuidas süsteem siseselt toimib.

## 1.3. Cateli komponentide tugi erinevatele platvormidele

| | NET40 | NET45 | SL4 | SL5 | WP7.8 | WP8 | WIN80 | WIN81 | PCL |
|---|---|---|---|---|---|---|---|---|---|
| Catel.Core | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Catel.MVVM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Catel.Mvc | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | | | | | | | | | |
| Catel.Extensions.Controls | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Catel.Extensions.CSLA | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Catel.Extensions.Data | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Catel.Extensions.EntityFramework5 | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Catel.Extensions.FluentValidation | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Catel.Extensions.Interception | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Catel.Extensions.Memento | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ |
| Catel.Extensions.Prism | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |

Tabel 1. Platvormide toe selgitus (Geert von Horrik, 2013)

Tabeli selgitus:

NET40, NET45 – .NET raamistiku versioonid 4.0 ja 4.5

SL4, SL5 – Silverlight versioonid 4.0 ja 4.5

WP7.8, WP8, WP81 – Windows Phone versioonid 7.8, 8.0 ja 8.1

WIN80, WIN81 – Windows 8 ja Windows 8.1

PCL – Portable Client Library – jagatav kood erinevate platvormide vahel

Kuna Catelit saab kasutada nii serveri kui kliendipoolsetel Windowsi platvormidel, sobib see nii ASP.NET MVC veebirakendustesse kui ka WPF, Silverlight, Windows Phone ning WinRT platvormidele. Multiplatvormi toe eelis on see, et tulemuseks on väga suur koodi taaskasutuse protsent. Sellest tuleneb veel eelis, mis lubab luua komponente, mis saab eraldada nii kliendist kui serverist (Geert von Horrik, 2013).

## 1.4. Võrdlus teiste MVVM raamistikega

| Framework | Description | Website |
|---|---|---|
| Catel | Catel on rohkem kui lihtsalt MVVM töövahend. See sisaldab endas kasutaja juhtelemente (ingl.k. user control) ja palju enterprise library klasse. MVVM töövahend erineb teistest raamistikest lihtsuse poolest ja lahendab ära "nested user control" probleemi dünaamiliste vaatemudelitega kasutaja juhtelementidele. | http://catel.codeplex.com |
| Caliburn.Micro | Väike, aga võimas implementatsioon Caliburn raamistikule disainitud WPF'ile, Silverlightile ja WP7'le. See raamistik implementeerib paljusi kasutajaliidese mustreid, lahendades reaalelu probleeme. Mustrid, mis on toetatud on: MVC, MVP, MVVM ja Application Controller. | http://caliburnmicro.codeplex.com |
| Cinch (V2) | Cinch on MVVM raamistik, mis sisaldab paljusid abiklasse, mis annab arendaja kasutusse | http://cinch.codeplex.com |

| Framework | Description | Website |
|---|---|---|
| | võimaluse luua väga kiiresti skaleeruvaid testitavaid MVVM rakendusi. | |
| MVVM light | MVVM Light Toolkit on komponentide kogum, mis aitab kasutajatel algust teha MVVM mustriga Silverlight'is ja WPF'is. See on väga kerge raamistik, mis sisaldab ainult väga elementaarseid vajalikke komponente. | http://mvvmlight.codeplex.com |
| Simple MVVM toolkit | Simple MVVM Toolkit teeb lihtsaks Silverlight, WPF and WP7 rakenduste arendamise MVVM mustriga. Eesmärk on pakkuda lihtsat raamistikku ja töövahendite kogumikku, et üles seada rakendusi MVVM baasil väga kiiresti. Keskendatud on lihtsusel, kuid see sisaldab endas kõike vajalikku MVVM rakenduste loomiseks. | http://simplemvvmtoolkit.codeplex.com/ |
| WAF | The WPF Application Framework (WAF) on kergekaaluline raamistik, mis aitab luau | http://waf.codeplex.com/ |

| Framework | Description | Website |
| --- | --- | --- |
| | stuktureeritud WPF rakendusi. See toetab mitmekihilist arhitektuuri ja MVVM ning PresentationModel mustrit. | |

Tabel 2. Võrdlus teiste MVVM raamistikega (Geert von Horrik, 2013)

# 2. Õppematerjali koostamine

Õppematerjali eesmärk on tutvustada lugejale Catel raamistiku võimalusi ning MVVM mustri kasutust. Õppematerjali läbides valmib Windowsi rakendus, milles on võimalik hallata firmasid ning firmaga seotud isikuid.

Autor soovib tutvustada õppematerjalis järgmiseid teemasid:

- Cateli projekti loomine Visual Studios
- Mudeli (ingl.k model) loomine
- Vaate (ingl.k view) loomine
- Vaatemudeli (ingl.k view model) loomine
- Mudelite kasutamine vaatemudelis
- Vaatemudelis mudelite sidumine (ingl.k. Binding) vaatega
- Teenuse (ingl.k service) loomine
- IoC konteineri kasutamine
- Andmebaasi SQLite kasutamine ja lisamine programmi
- Vaate juhtelemendi (ingl.k. control) käsu (ingl.k. Command) sidumine vaatemudeli käsuga
- CRUD (ingl.k. Create-Read-Update-Delete) funktsionaalsuse lisamine

## 2.1. Sihtrühm

Antud õppematerjali sihtrühmaks on programmeerimise huvilised ja IT tudengid, kellel on mõningane programmeerimise kogemus. Materjal sobib ka kogenud WPF arendajale, kes tahab arendust läbi Catel raamistiku lihtsustada. Sihtrühma kuuluval lugejal peaksid olema järgnevad baasteadmised:

- Tunneb mingit objekt-orienteeritud programmeerimiskeelt, soovitavalt C# või Java.
- Teab põhilisi andmestruktuure ja algoritme.
- Soovituslikult võiks olla kogemust Visual Studioga või mõne teise arenduskeskkonnaga.

Sihtrühmaks sobivad näiteks Tallinna Ülikooli üliõpilased, kes on läbinud järgnevad ained:

- Programmeerimise alused
- Aloritmid ja andmestuktuurid
- Programmeerimise põhikursus
- .NET raamistik (soovituslik valikaine)

Õppematerjal on koostatud nii, et piisab esimesest kolmest kursusest ja on loetav ka Java programmeerimiskeele tundjale (Programmeerimise põhikursus on Java baasil). Kõik toimingud, mis on tehtud Visual Studio arenduskeskonnaga on autori poolt illustreeritud piltide ja kirjeldustega, seega peaks olema materjal läbitav ka ilma selle keskkonna eelneva tundmiseta.

## 2.2. Õppematerjali struktuur

Õppematerjal on jagatud 12 peatükiks nii, et iga peatükk üritab keskenduda mingi kindla komponendi loomisele või kindlale protsessile, mis tuleb programmi kirjutamisel läbida.

Esimene peatükk kirjeldab ära toimingud, mida lugeja peab täitma enne õppematerjali läbimist.

Teine peatükk seletab, kuidas luua uus Catel raamistikul põhinev projekt Visual Studios. Samuti sisaldab õpetust kuidas lisada projekti NuGet pakett projektile.

Kolmandas peatükis keskendub autor mudeli loomisele. Eraldi on alapeatükk klassi loomiseks, mille põhjal mudel luuakse. Õpetatakse kasutama code snippet'eid ning lisama mudelile property'sid.

Neljas peatükk õpetab looma vaatemudelit. Seletatakse lahti vaatemudeli ülesehitus ning komponendid. Lisatakse ka käsk, mis kävitatakse nupu vajutamisel. Tutvutakse uue tüübiga ObservableCollection.

Viies peatükk räägib vaate ülesehitusest ning õpetab looma uut vaadet. Räägitakse vaatemudeli ning vaate elementide sidumisest. Samuti õpetatakse vaadet kujundama.

Kuues peatükk kirjeldab, kuidas luua kasutajaliides nii, et selle osad oleksid dünaamiliselt muudetavad. Räägitakse lähemalt, mis komponendid tuleb selleks lisada ja mis muudatused koodis sisse viia.

Kaheksas peatükk tegeleb teenuse ning selle liidse loomisega ja õpetab, kuidas need IoC konteinerisse lisada ning hiljem vajalikus kohas sealt välja küsida.

Üheksas peatükk räägib lugejale SQLite andmebaasist ja sellest, kuidas kirjutada valmis juba varem lisatud teenus ja selle liides. Teenusesse saab kirja kõik vajalik andmebaasi tabelite salvestamiseks, andmete küsimiseks, muutmiseks, lisamiseks ja kustutamiseks.

Kümnes peatükk käsitleb firma vaatemudeli mudelite ja käskude sidumist andmebaasiteenuse meetoditega.

Üheteistkümnes sisaldab endas Person vaate ja vaatemudeli loomist ning sidumist andmebaasiga.

Kaheteistkümnes peatükk liidab endas kokku kahe valminud vaate vahel navigeerimise võimaluse loomise. Programmi vasakpoolne regioon saab sisuks vaadete vahetamise funktsionaalsuse.

## 2.3. Õppematerjali analüüs

Õppematerjal on koostatud kasutades palju selgitavaid pilte arenduskeskkonna erinevatest akendes ning vaadetest. Koodiread on paigutatud õppematerjalis vahele, kus konkreetse koodi kirjutamisest juttu tuleb. Materjal on jälgitav iga protsessi kaupa ja iga lisatava koodiploki kaupa.

Koodinäidetes on kasutatud samu värve nagu Visual Studio's vastavas koodiaknas, et kasutaja saaks kontrollida, kas tal on koodis esinenud vigu. Kui koodinäites värv erineb kasutaja poolt kirjutatud koodist, siis ilmselt on kuskil viga ja seda viga on nii lihtne leida. Koodinäited on paigutatud kasti sisse, et eristada muust tekstist.

Enamus materjali peatükke on jaotatnud nii, et üks peatükk on üks komponent. Nii saab vajadusel lihtsasti pöörduda mingi komponendi ülevaatamiseks selle peatüki poole. Samuti on lihtne viidata analoogse protsessi läbiviimisele, mis on juba kord tehtud.

Autor on üritanud jälgida ka seda, et kõik peatükid oleksid loogilises järjekorras, nii et programmi loomise seisukohalt on komponentide loomise järjekord sama.

Muutujate nimed on inglise keelsed ja tuginedes testijate tagasisidele on ka palju komponenentide nimedes kasutatud inglise keelt, sest nii on materjal palju jälgitavam.

## 2.4. Õpiväljundid

Antud peatükis kirjeldab autor eeldatavaid õpitulemusi õppematerjali läbijale. Autor toob välja ka materjali täiendamise võimalusi ning edasiõppimise võimalusi.

Õppematerjali läbitöötaja tunneb ja oskab luua ning kasutada järgmiseid Catel raamistiku elemente:

- Uus Catel projekt
- Mudel
- Vaade
- Vaatemudel
- Teenus

Õppematerjali läbija peaks tundma järgmisi protsesse:

- Nuget paketi lisamine projekti
- Vaate jagamine regioonideks ning nende sisu dünaamiline vahetamine
- Teenuse lisamine ja küsimine IoC konteinerist
- SQLite andmebaasi loomine ning sidumine programmiga
- Vaatemudeli ning vaate elementide (parameetrid ja käsud) sidumine

Õppematerjali võiks tulevikus täienda veel erinevate teenustega näiteks mõne veebiteenusega suhtlemise teenusega ja vigade logimise ning andmete valideerimisega. Samuti võiks õppematerjali lisada sündmuste edasiandmise võimaluse erinevate vaatemudelite vahel.

Iseseisvaks õppimiseks soovitab autor lugeda Cateli komponentide põhiseid näiteid (Geert van Horrik, 2014)

# 3. Õppematerjali testimine

Õppematerjali testimiseks kasutas autor kolme Tallinna ülikooli informaatika tudengi abi. Kaks tudengit õpivad bakalaureuse astmes III kursusel ning üks II kursusel. Testimise eesmärk oli välja selgitada kas koostatud õppematerjal on sobilik autori poolt pakutud sihtgrupile. Samuti oli eesmärk leida võimalikke vigu, ebatäpsuseid ning kontrollida õpetuse jälgitavust ning arusaadavust.

Testimine viidi läbi kahes etapis. Esimeses neist lasti tudengitel õppematerjali abil proovida jõuda autori poolt soovitud tulemuseni. Iga arusaamatuse või kommentaari puhul palus autor pöörduda koheselt tema poole. Nii selgitati pideva kommunikatsiooni tulemusel välja õppematerjali puudused. Lisaks andis autor tudengitele küsimused, millele soovis vastust. Küsimused olid mõeldud peamiselt õppematerjali sisu hindamiseks tagasiside saamiseks, selle kohta, mida võiks parandada. Samuti soovis autor saada tagasisidet selle kohta, kas antud õppematerjal andis antud teema kohta vajaliku ülevaate. Tagasiside põhjal viis autor sisse materjali parandused.

Testimise teises etapis andis autor tudengitele parandatud versiooni, mis valmis esimese testimise tagajärjel ning palus tagasisidet, kas parandused vastavad ootustele ning õppematerjali viimane variant on kasutajatele arusaadav.

## 3.1.Testimise esimene etapi tulemused

Vaatleme õppematerjali testimise kulgu peatükkide kaupa. Toome eraldi välja teemad, mille kohta testija andis tagasisidet. Märgime siia ka parandused, mis autor peale seda nendesse peatükkidesse sisse viis.

### 3.1.1. Nõuded enne õppematerjali läbimist

Puudu oli Catel versioon, mis tuleks alla laadida ja paigaldada. Autor tegi õppematerjali alguses versioonile 3.9, kuid jättis selle kohta info õppematerjali kohta lisamata. Vahepeal (17.11.2014) ilmus uus Cateli versioon 4.0 ning testijad proovisid automaatselt õppematerjali selle versiooni peal. Nende versioonide vahel on mõned erinevused ja autor otsustas õppematerjali viia vastavusse uusima versiooniga. Esimene erinevus tuli selles peatükis, kus autor märkis ära eraldi aadressid, kust tuleks alla laadida snippet'id ning template'd, need lingid enam ei töötanud. Versioonis 4.0 on snippet'id ja template'd installerisse juba lisatud. Autor viis sisse muudatused vastavalt sellele.

### 3.1.2. Vaatemudeli lisamine

Siin peatükis oli mainimata jäänud 3 nimeruumi, mida CompanyViewModel peaks kasutama.

```
using System.Collections.ObjectModel;
using Models;
using Catel.Data;
```

### 3.1.3. Vaate lisamine

Leiti üks trükiviga MainWindow.xaml kirjapildis ning selle sama faili sisus mõned puuduvad ja üleliigsed jutumärgid. Autor viis sisse parandused.

### 3.1.4. Prism laienduspaketi lisamine

Esines kirjaviga paketi nimetuses, mis sai parandatud.

### 3.1.5. Bootstrapperi lisamine

Testijale jäi segaseks, kuhu täpselt tuleks Bootstrapper.cs lisada. Autor lisas täiendavad selgitused. Enviroment.RegisterDefaultViewModelServices() on uues versioonis viidud üle

CatelEnviroment.RegisterDefaultViewModelServices() ning Catel.MVVM.Services on paigutaud versioonis 4.0 nüüd Catel.Services. OnStartup meetodis ei ole võimalik enam kutsuda välja Catel.Logging.LogManager.RegisterDebugListener(), see tuleks eemaldada. MainWindowViewModelis peaks olema pakett System.Threading.Tasks. Erinevus on Initialize meetodis võrreldes eelmise versiooniga on tagastusväärtus Task tuuli (vana void asemel). Samuti on erinevuseks see, et `IUIVisualizerService` asemel on `UICompositionService` Autor viis peale tagasisidet muudatused sisse.

### 3.1.6. SQLite andmebaas

Puudusid nimeruumid System.Data ja System.Data.SQLite, mis said lisatud. Andmebaaside loomisel SQL süntaksis viga, NOT NULL peab järgnema märksõnale AUTOINCREMENT. Parandus sisse viidud.

### 3.1.7. Company vaatemudeli täitmine andmetega andmebaasist

Puudu oli nimeruum CompanyManager.Services.Interfaces, mis sai lisatud.

## 3.2. Küsimused testijatele ja nende vastused

**Kas olete varem kokku puutunud C# programmeerimiskeelega?**

Kaks testijat kolmest oli C# keelega kokku puutunud, üks mitte.

**Milliste programmeerimiskeeltega oled kokku puutunud?**

Kõik olid puutunud kokku keeltega Python, Java, Php. Üks vastanutest märkis ära veel C keele ja Objective-C.

**Kas piltidest oli õppematerjali läbimisel kasu?**

Kõik vastanud olid ühel meelel, et piltidest oli kas. Põhjenduseks toodi see, et siis saad olla kindel, et kui ees on sama pilt, mis õppematerjali koostajal, siis oled õigel teel. Samuti toodi põhjenduseks, et see aitab programmis paremini navigeerida.

**Kas õppematerjal oli arusaadav või oleks pidanud mõnda osa rohkem lahti seletama? Kui jah, siis millist osa?**

Üks testijatest oli MVVM mustriga ja C# keelega tuttav ning leidis, et see aitas tal paremini õppematerjali läbida. Toodi välja veel, et pilte ja täpseid seletusi, mis klassi mida lisada, oleks võinud rohkem olla.

**Kas õppematerjalis kirjeldatud teemad on loogilises järjekorras või tuleks järjestust muuta?**

Kõik olid ühel nõul, et teemad on loogilises järjekorras ning järjestust ei tuleks muuta.

**Kas programmikood on loogiliselt ülesehitatud ja paigutatud või tuleks midagi muuta?**

Programmikoodi ülesehituse ja paigutusega olid kõik vastanud rahul ja ei muudaks midagi.

**Kas värvilised koodiread aitavad koodi kontrollimise juures? Kui jah, siis kuidas?**

Vastuseks märgiti, et aitavad aru saada, kas tegu on muutuja, meetodi või tüübiga, samuti aitab värvide kontroll aru saada, kas on toimitud õigesti. Kui värvidega midagi ei klapi, siis ilmselt on koodis viga.

**Kas antud õppematerjal on piisav ülevaate andmiseks Catel raamistuku kohta?**

Üks vastaja märkis, et algteadmisteks on see õppematerjal piisav ning edasi võib juba õppida töö käigus. Teised vastajad märkisid samuti, et said hea ülevaate teema kohta.

**Mida on omapoolse kommentaarina lisada või soovitada?**

Soovitatakse materjali ette võtnud isikul varuda aega, sest Cateli ja MVVM mustri mõistmine võtab aega. Samuti on soovitused autorile vaadata üle grammatiga ja kirjapilt. Soovituseks oli ka konkreetselt osade terminite kasutamise soovitus inglisekeelselt ehk mitte tõlkida.

## 3.3. Testimise teise etapi tulemused

Pärast parandusi, mis autor tegi peale kasutajate kommentaare ja vastuseid küsimustele, lasi autor uuesti õppematerjali üle vaadata ja ootas kommentaare. Kasutajad olid muudatustega rahul ning leidsid, et materjal on nüüd palju paremini jälgitav ning raskesti mõistetavaid peaaegu et pole.

## 3.4. Testimise kokkuvõte

Testimisest oli väga palju kasu, sest tulid välja vead, mida autor ise ei osanud märgata. Samuti sai autor teada, millised punktid vajavad paremat lahtiseletamist ja rohkem illustreerivaid pilte. Üldiselt jäi autor rahule sellega, et kui jätta välja segased kohad ning testijate täiendavad küsimused, said testijad väga hästi aru, mida autor on tahtnud õppematerjaliga saavutada.

Vastused küsimustele aitasid kaasa paremale ülesehitusele, paremale sõnastusele ning täpsustuste lisamisele. Samuti andsid vastuse, et sihtgrupp oli õigesti valitud ning materjali läbitöötamine andis ülevaate Catel raamistikust ja selle kasutamisest.

# Kokkuvõte

Käesoleva bakalaureusetöö tulemuseks on õppematerjal Catel raamistikus WPF rakenduste loomiseks. Õppematerjal sobib mõningase programmeerimiskogemusega IT huvilisele, soovitavalt Java või C# keele tundjale. Õppematerjali sobivust on testitud kolme informaatika tudengi peal ja tulemus vastab autori ootustele. Testijate tagasisidet on autor kasutanud õppematerjali parandamiseks ja viimistlemiseks.

Õppematerjalis on antud ülevaade Catel 4.0.0 raamistikust WPF rakenduse loomiseks ning MVVM mustrist. Testimise tagasiside põhjal võib autor öelda, et lugeja on saanud õppematerjalist vajalikud teadmised Catel raamistiku ja MVVM mustriga tuvumiseks.

# Summary

The aim of current Bachelor's thesis is study material for using Catel framework for WPF application development. The study material is suitable for people interested in IT and who have some programming experience, preferably in Java or C#. The suitability of material is tested by 3 informatics students and the result meets authors expectations. The feedback from testers has been used by author to fix the material and make it better.

There is given summary about Catel 4.0.0 framweork for building WPF applications using MVVM pattern. According to testers feedback, the autor can say, that reader has received needed knowledge about Catel framework and MVVM pattern.

The study material is added as appendix to the Bachelor's thesis.

# Kasutatud kirjandus


Microsoft (2014). *Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF.* http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx (12.12.2014).

Geert van Horrik (2014). *Catel documentation.* https://catelproject.atlassian.net/wiki/display/CTL/Catel+documentation+Home (12.12.2014).

Geert van Horrik (2014). *Catel examples.* https://github.com/Catel/Catel.Examples (12.12.2014).

Geert van Horrik (2011). *Catel - Part 5 of n: Building a WPF example application with Catel in 1 hour. http://www.codeproject.com/script/Articles/View.aspx?aid=151491 (12.12.2014) .*

Geert van Horrik (2013). *Platform support explanation*. https://catelproject.atlassian.net/wiki/display/CTL39/Platform+support+explanation (12.12.2014).

Geert van Horrik (2012). *MVVM framework comparison sheet*. https://catelproject.atlassian.net/wiki/display/CTL/MVVM+framework+comparison+sheet (12.12.2014).

Geert van Horrik (2014). *Catel.Extensions.Prism* https://catelproject.atlassian.net/wiki/display/CTL/Catel.Extensions.Prism (12.12.2014).

Geert van Horrik (2014). *Catel.Extensions.WPf.Prism* https://github.com/Catel/Catel.Examples/tree/master/src/NET/Catel.Examples.WPF.Prism (12.12.2014)

Martin Fowler (2005). *InversionOfControl*. http://martinfowler.com/bliki/InversionOfControl.html (12.12.2014).

Lisad

# Lisa 1. Õppematerjal

## 1. Nõuded enne õppematerjali läbimist

Enne õppematerjali läbimist on vajalik paigaldada arvutisse järgmised programmid ning teha järgmised toimingud.

- Visual Studio 2013.

  Allalaetav

  http://www.visualstudio.com/downloads/download-visual-studio-vs

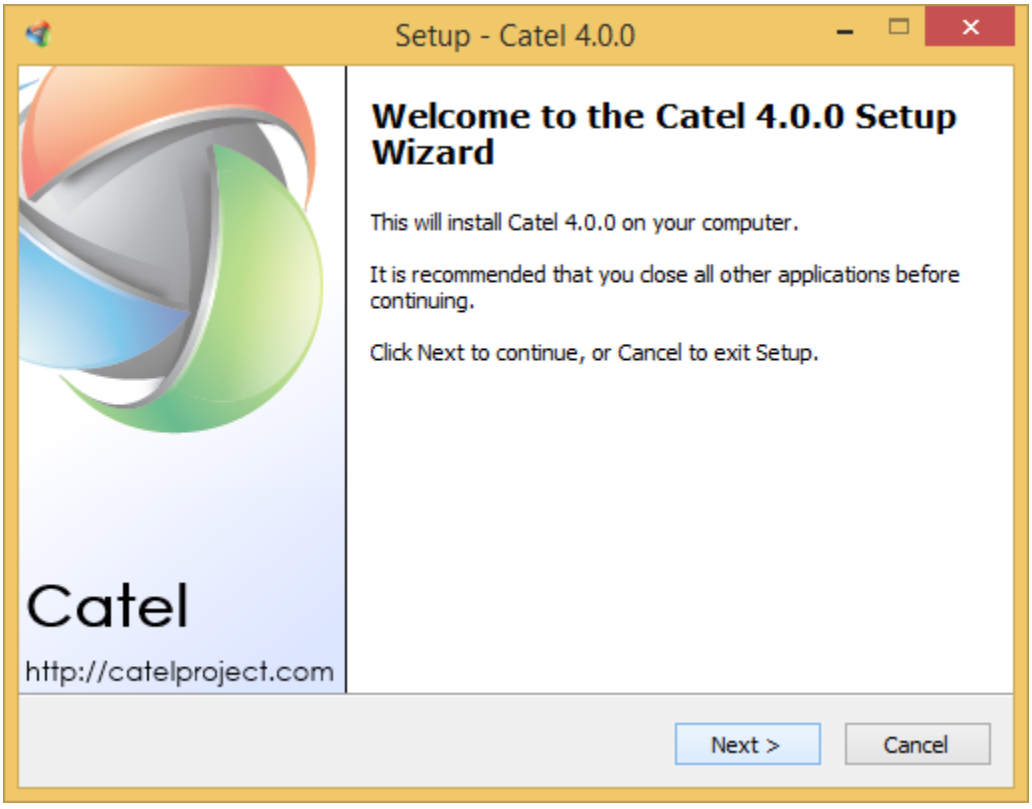
  90 päeva saab ilma litsentsita proovida. Sobivad versioonid Ultimate, Premium ja Professional.

  Installimisel tuleks tähele panna, et saaks installeeritud C# keeleversioon.

- Catel 4.0.0.

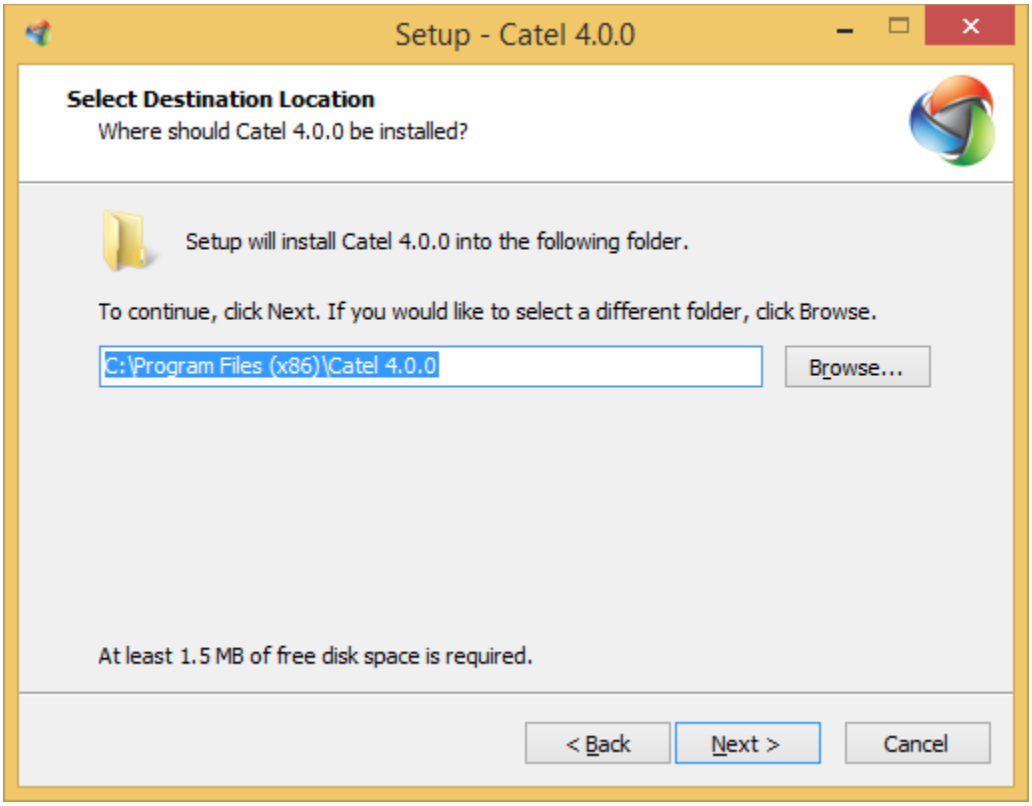
  https://github.com/Catel/Catel/releases/download/4.0.0/Catel.4.0.0.setup.exe

  Peale allalaadimist ja käivitamist tuleks klikkida Next (vt joonis 1).



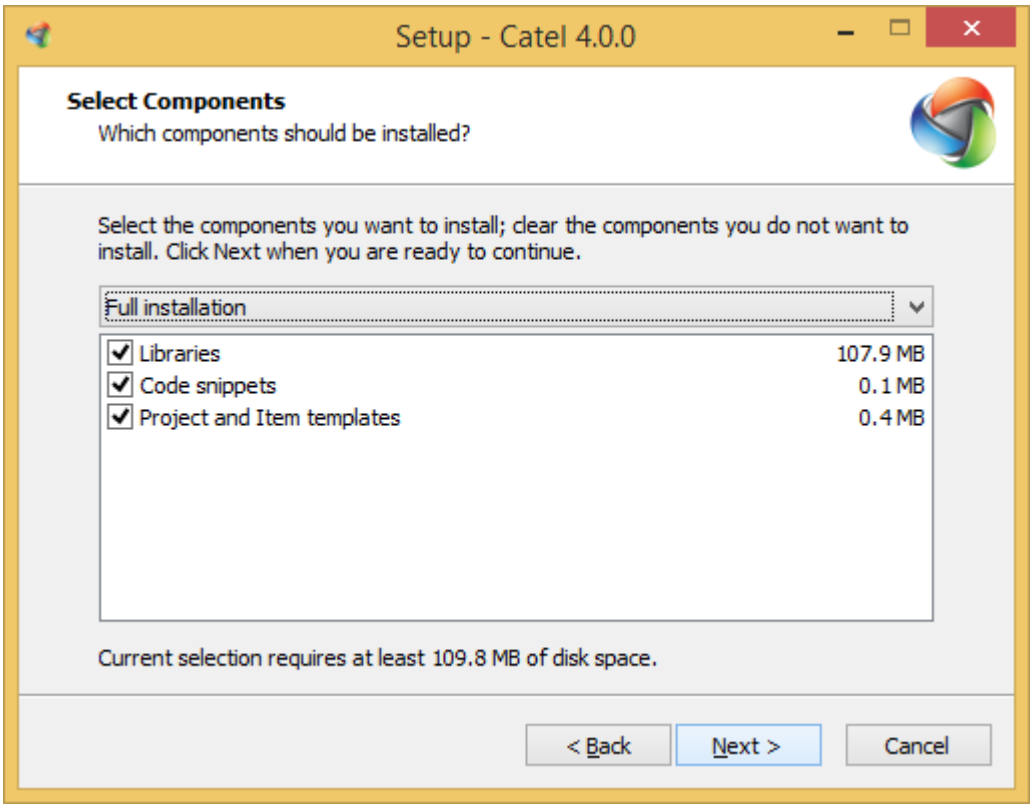

Joonis 1. Catel installimine esimene vaade

Uues avanevas aknas tuleks märkida ära kaust, kuhu soovitakse Cateli failid installida. See teekond tuleks nüüd meelde jätta (soovitavalt jätta samaks, mis vaikeväärtus) ja klikkida Next (vt joonis 2).




Joonis 2. Catel installimise teine vaade

Järgnevas avanevas aknas tuleb kindlasti valikusse jätta Libraries, Code snippets ja Project and Item Templates, seejärel klikkida Next (vt joonis 3).




Joonis 3. Catel installimise kolmas vaade

Järgnevates akendes klikkida Next ja seejärel Install.

- Catel templates

  Peale allalaadimist tuleks liikuda kausta, kuhu sai Catel installitud ning võtta sealt kausta \templates\C# sisu (vaikeväärtust kasutades C:\Program Files (x86)\Catel 4.0.0\templates\C# sisu) ja kopeerida kausta Documents\Visual Studio 2013\Templates

- Catel snippets

  Documents\Visual Studio 2013\Code Snippets\ kausta võiks teha uue kausta Catel.

  Peale allalaadimist tuleks liikuda kausta, kuhu sai Catel installitud ning võtta sealt kausta \snippets\C# sisu (vaikeväärtust kasutades C:\Program Files (x86)\Catel 4.0.0\snippets\C# sisu) ja kopeerida kausta Documents\Visual Studio 2013\Code Snippets\Catel

  Lisatud Catel kaust koos uute snippetitega tuleb Visual Studiosse lisada. Selleks

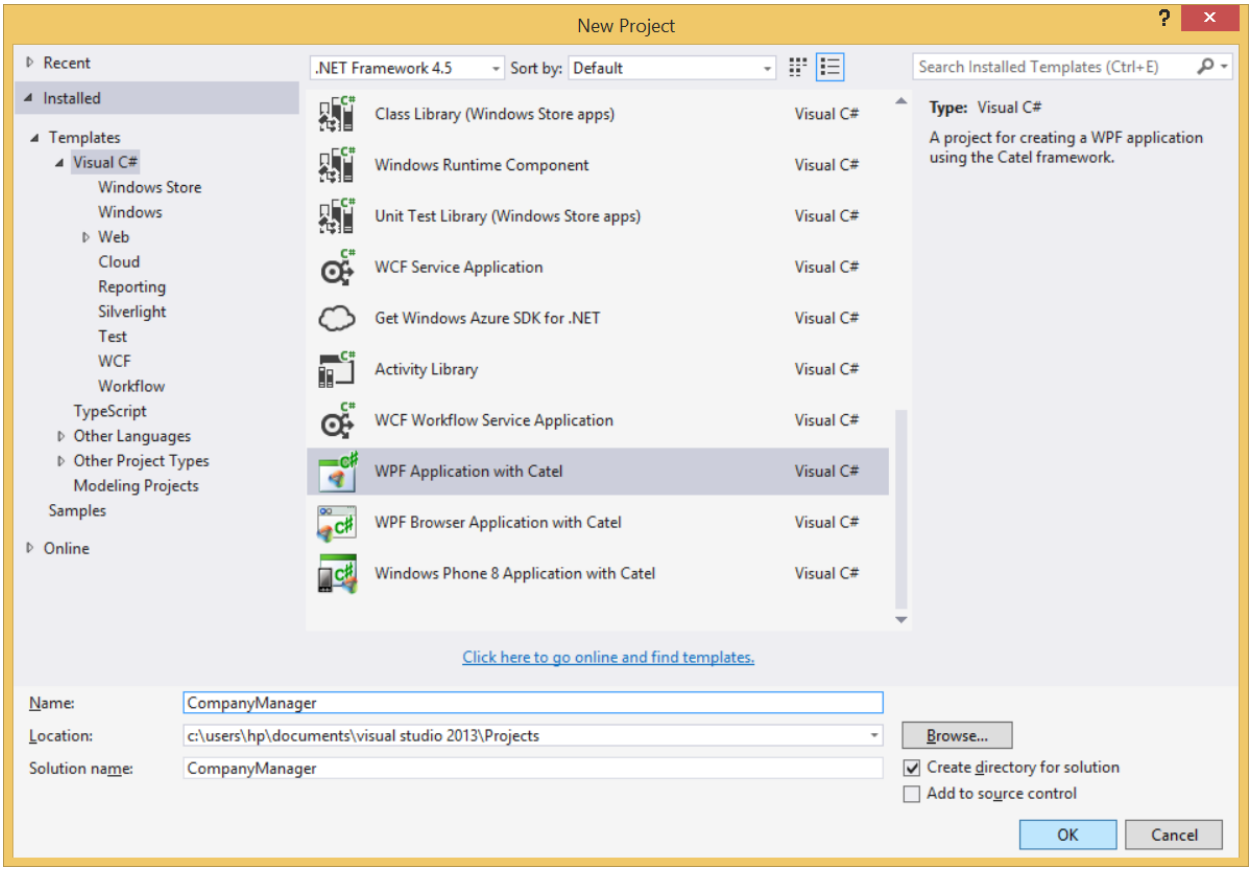
  Tuleb avada Tools->Code Snippets Manager. Avanevas aknas valida Language Visual C#

Ning klikkida Add ja valida kaust Documents\Visual Studio 2013\Code Snippets\Catel

# 2. Programmi koostamine

Avame Visual Studio ning teeme uue projekti (File->New->Project…)

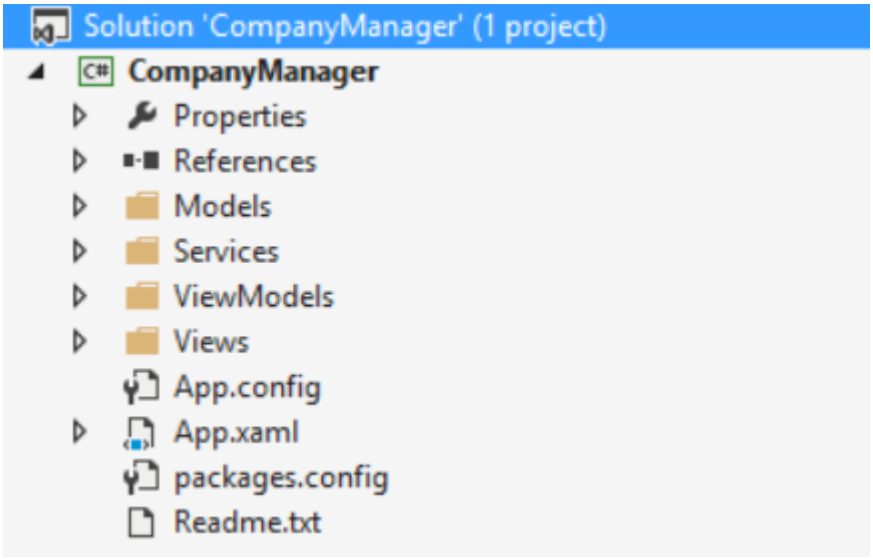
Avanevas aknas valime projekti tüübiks WPF Application with Catel ja paneme nimeks CompanyManager (vt joonis 4).




Joonis 4. Projekti loomine

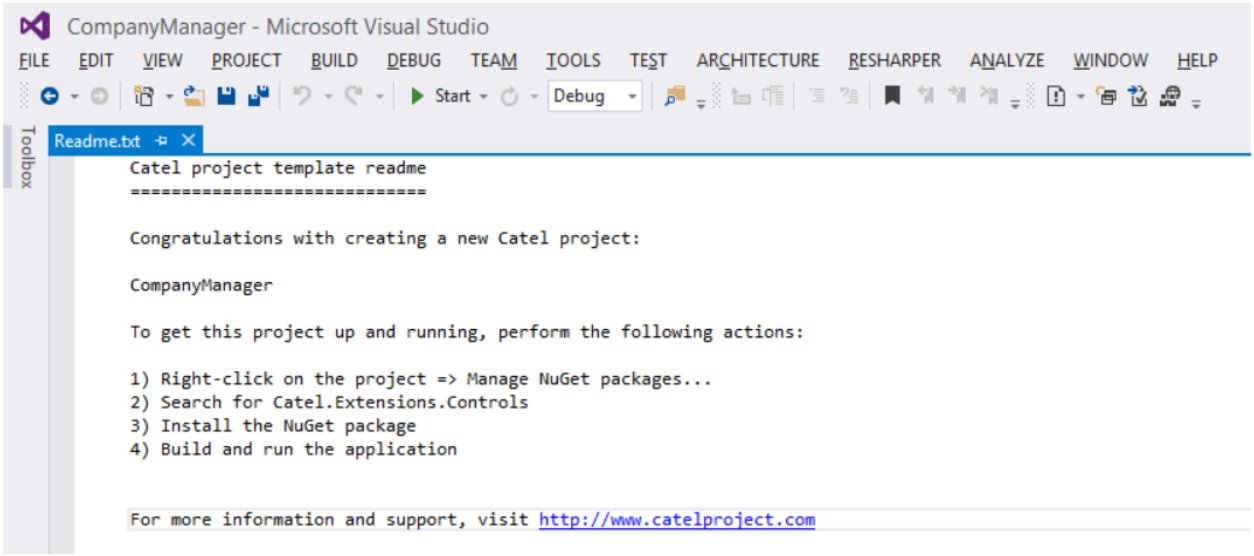
Tuleb jälgida, et vasakul oleks valitud keeleks Visual C#.

Klikkides OK genereeritakse meile projekt, mis sisaldab järgnevaid komponente (vt joonis 5):




Joonis 5. Projekti CompanyManager sisu

Ja põhiknas avatakse siit Readme.txt dokument, mis automaatselt projekti genereeritakse (vt joonis 6).

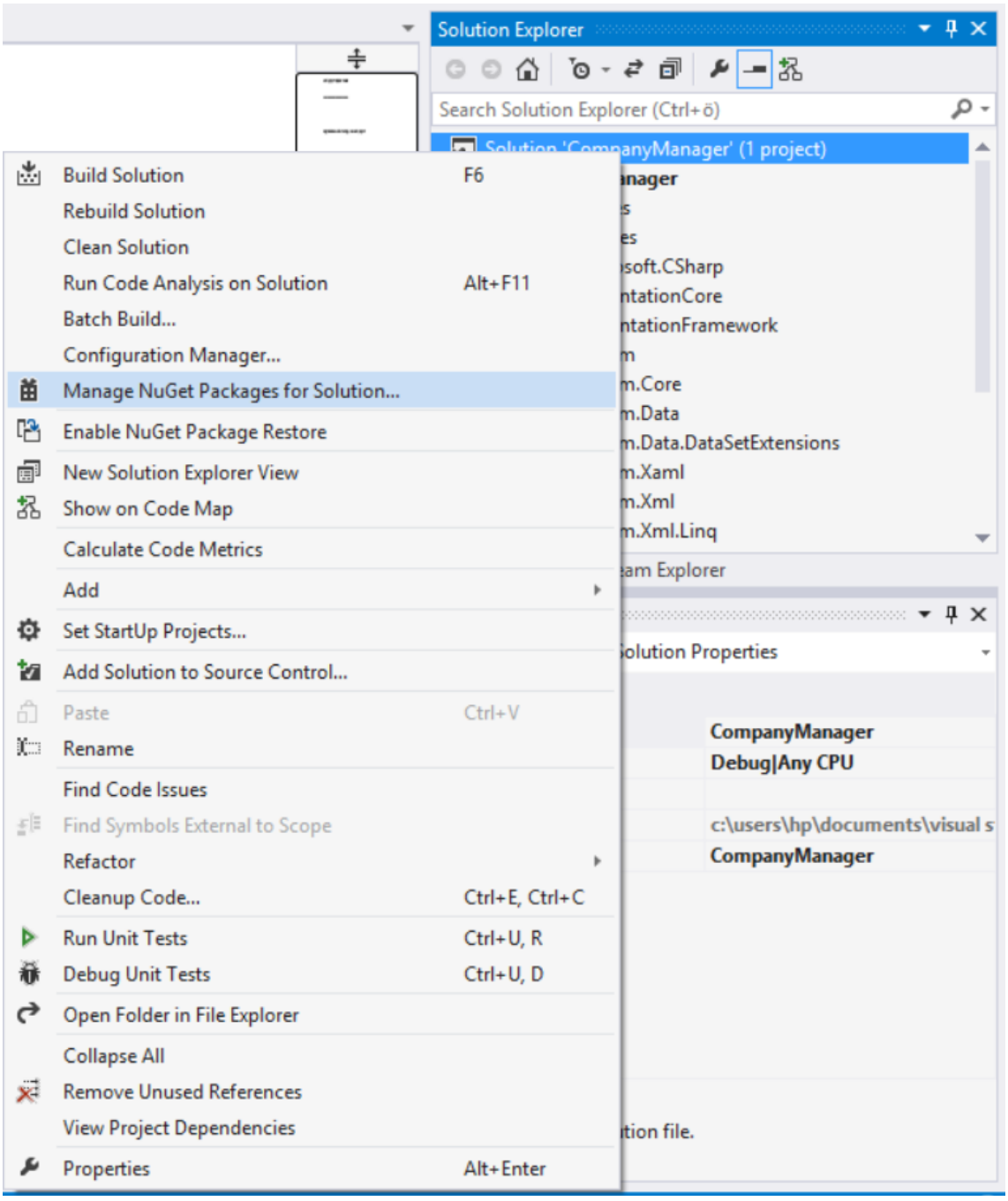



Joonis 6. Readme.txt sisu

Siin on selgitus, mis tuleks järgnevaks teha, et projekti genereerimine lõpule viia. Järgime õpetust ning lisame NuGet paketi nimega Catel.Extensions.Controls

## 3. NuGet paketi lisamine projekti

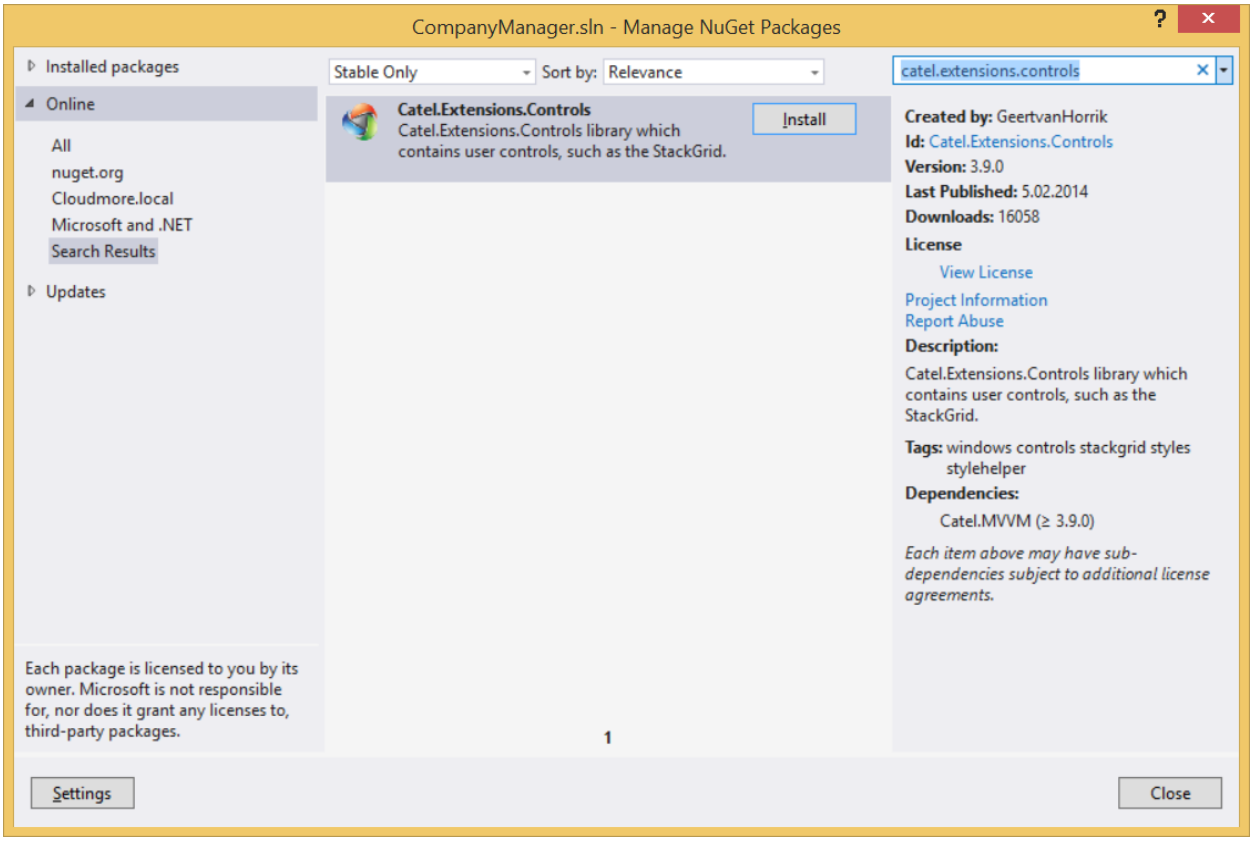
NuGet paketi lisamiseks klikime Solution'i peal hiire parema klahviga ja valime Manage NuGet Packages for Solution (vt joonis 7).




Joonis 7. Nuget paketi lisamine

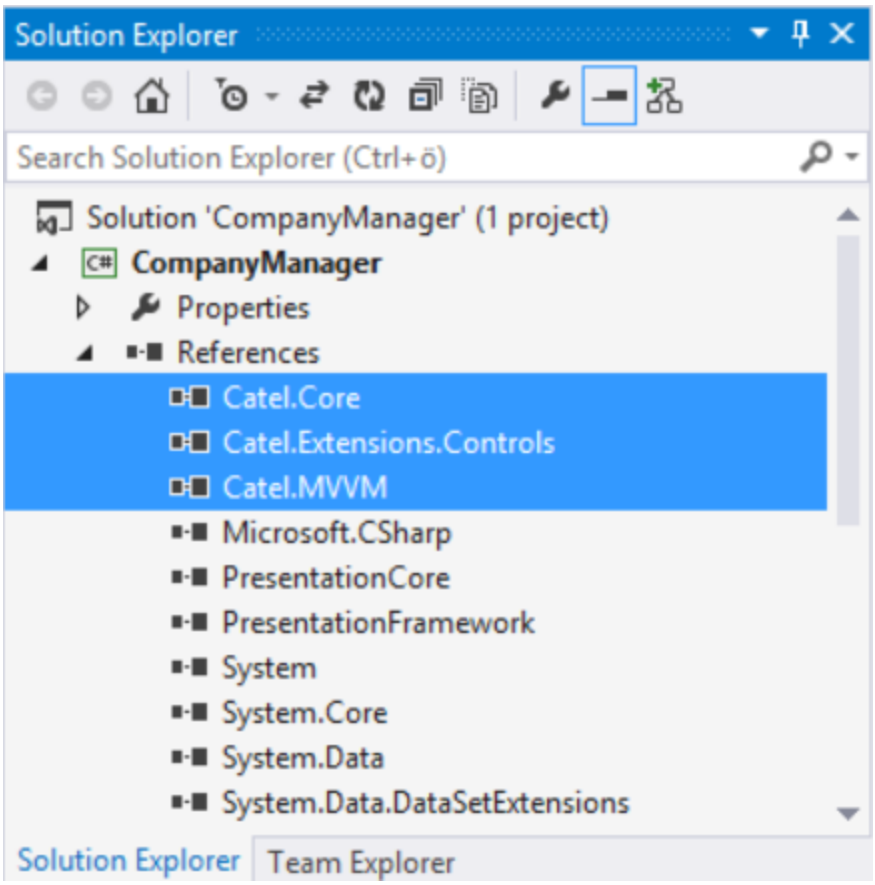
Avanenud aknast paremal üleval nurgas olevast otsingulahtrist otsime Catel.Extensions.Controls,

ja klikime Install (vt joonis 8).




Joonis 8. Catel.Extensions.Controls Nuget paketi lisamine

Peale edukat installeerimist tekib Solution Explorer'is all paiknevas References alla kolm uut dll'i (vt joonis 9).



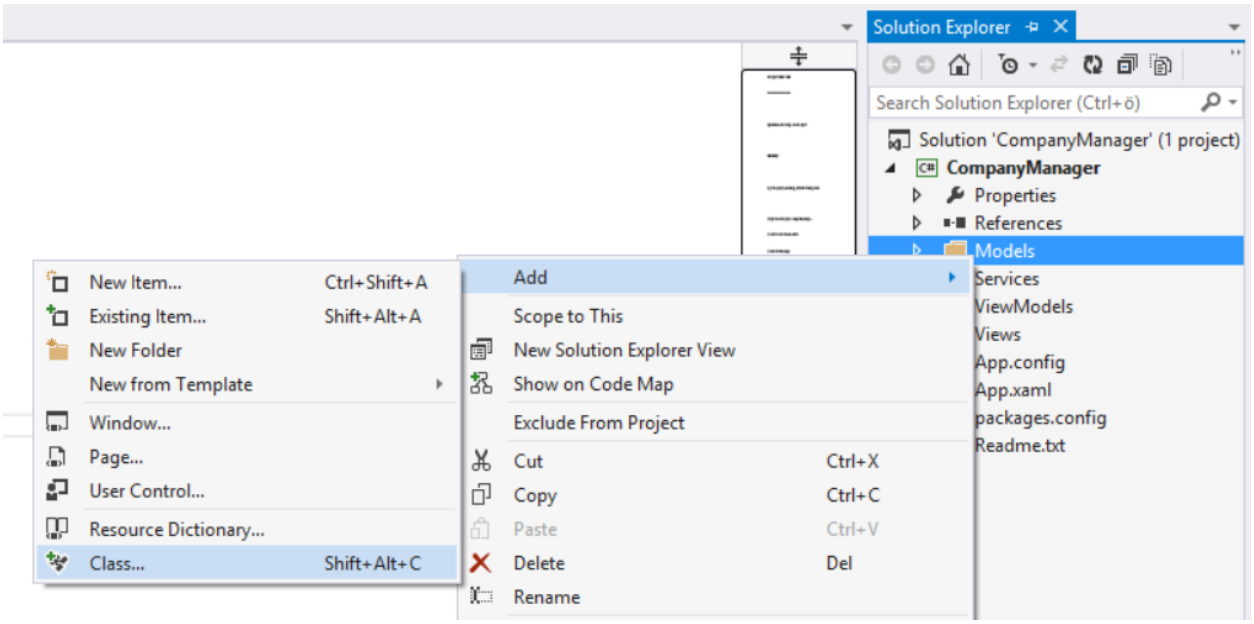

Joonis 9. References all kolm uut dll.

Nüüd oleme valmis alustama programmi komponentide loomisega.

# 4. Mudeli loomine

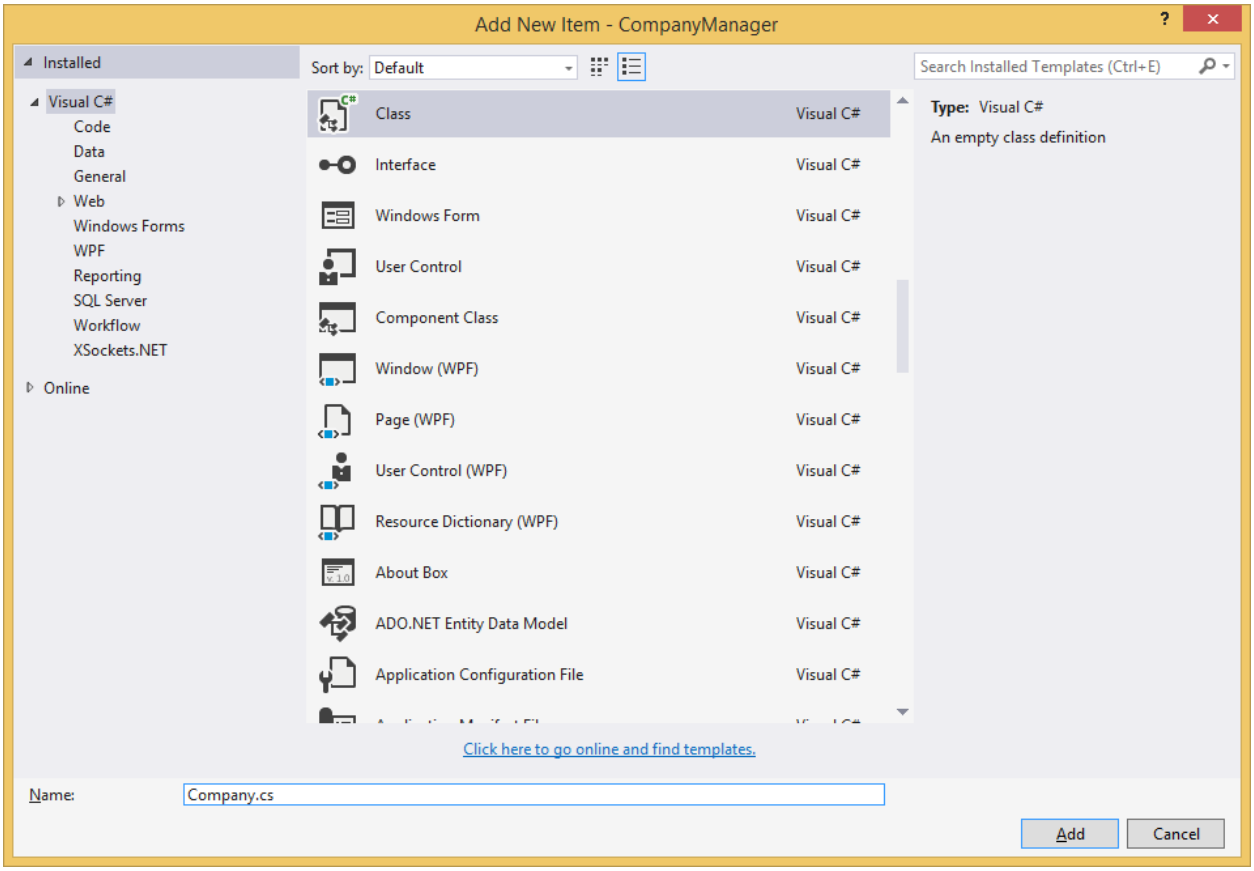
## 4.1. Klassi loomine

Esimeseks loome ühe klassi, selleks toimime järgnevalt (vt joonis 10).




Joonis 10. Uue klassi lisamine projekti.

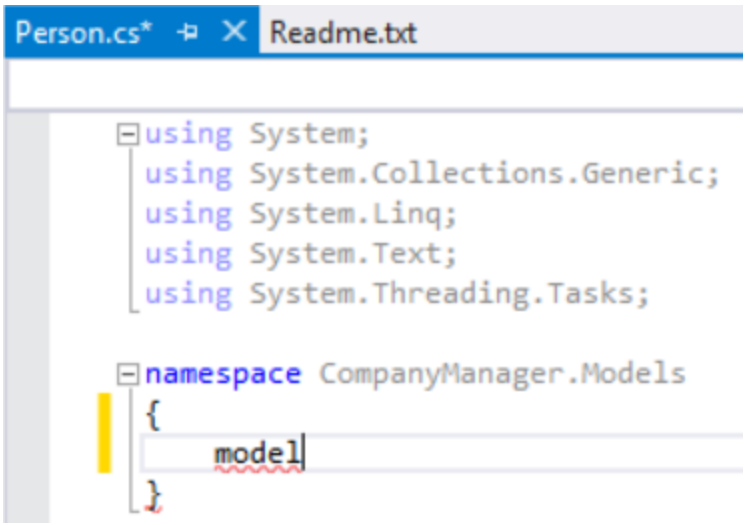
Avanevast aknast valime järgmise(vt joonis 11).




Joonis 11. Klassi loomise valikud.

## 4.2. Mudeli loomine klassi põhjal

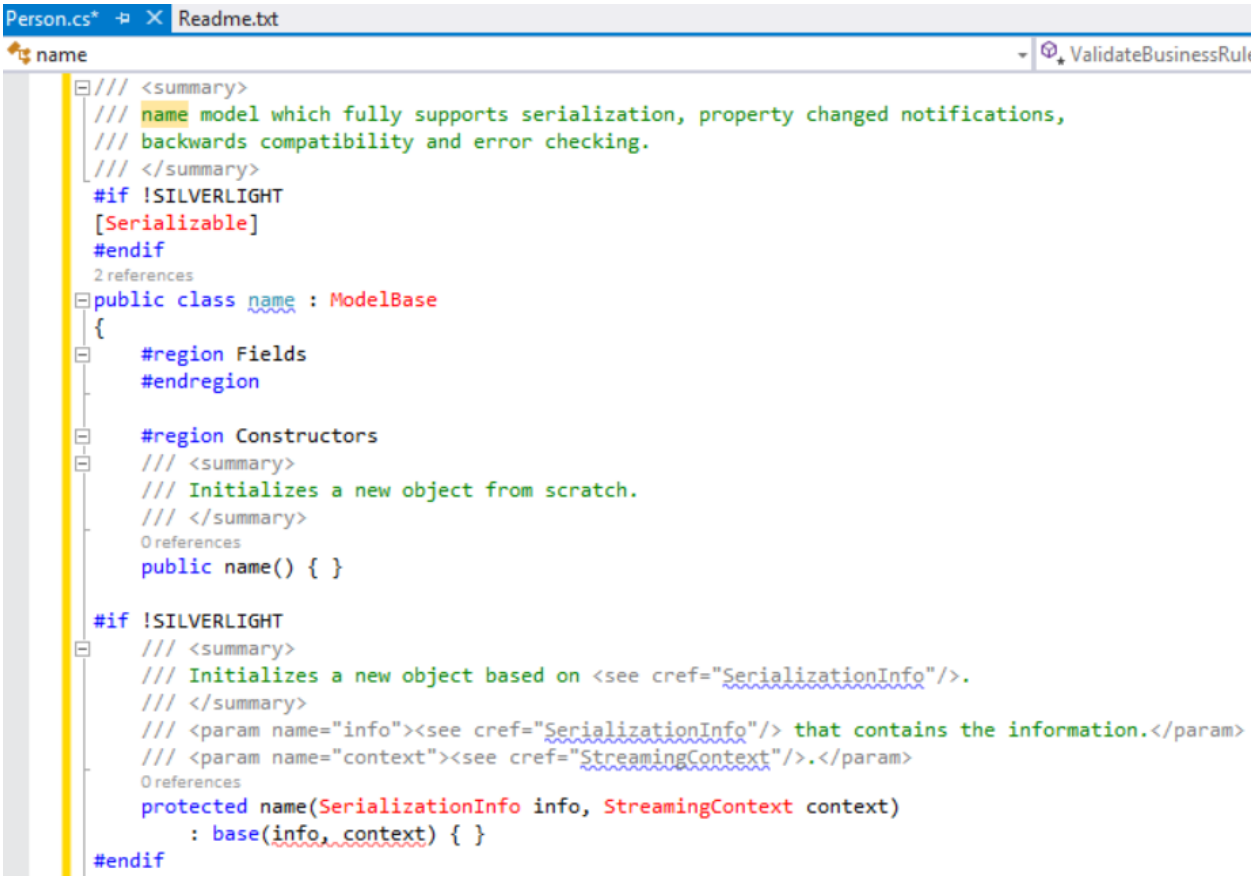
Nüüd laseme Catelil teha sellest klassist mudel. Selleks on meil eelnevalt installeeritud code snippet'idl millest saame kasutada nüüd snippetit nimega model. Toimive järgnevalt. Kõigepealt kustutame ära kogu Person klassi sisu ja kirjutame sinna sisse lihtsalt model (vt joonis 12).




Joonis 12. Mudeli loomine klassist.

Vajutame tab klahvi. Ning Catel genereerib meile vaikeväärtustega ModelBase klassist pärineva klassi ehk siis mudeli, mille parameetreid me saame koheselt seadistada.

Järgneva pildi peal on ülemine osa sellest klassist (klassi definitsioon ning kaks konstruktorit). Paneme tähele, et vaikimisi on klassi nimi ning konstruktorite nimed märgitud nimega name ja ülemises blokis on see selekteeritud ning me saame selle vaikeväärtuse asemele kirjutada meile sobiva mudeli nime. Antud juhul kirjutan Person ja vajutan Enter klahvi. Vahel ei pruugi see õnnestuda, seljuhul tuleks käsitsi see name asendada klassi nimega nendes 3 kohas (vt joonis 13).




Joonis 13. Automaatselt genereeritud mudel, kus name tuleb asendada soovitava mudeli nimega.

Järgnevalt tuleks lisada kaks nimeruumi, kus asuvad vajalikud puuduvad tüübid. Selleks lisame faili päisesse kaks rida(vt koodinäide 1):

```
using System.Runtime.Serialization;
using Catel.Data;
```

Koodinäide 1. Nimeruumide lisamine

Nüüd on meil valmis tühi mudel Person, kuhu saame hakata lisama vajalikke property'sid. Selleks on Catel juba meile teinud eraldi regiooni, kuhu sisse võiks kõik property'd minna. Samuti on kommentaarina pandud ka õpetus, kuidas seda teha. Kirjutame sinna modelprop ja vajutame nuppu Tab (vt koodinäide 2). Teine võimalus kasutada snippeteid on hiire parema

klahviga valitud kohas valida Insert Snippet ja otsida üles, kus vajalik snippet asub. Antud juhul Catel-> Define a property for the ModelBase class.

```
#region Properties
    // TODO: Define your custom properties here using the modelprop code snippet
#endregion
```

Koodinäide 2. Modelprop snippet

Tekitatakse järgnev kood, kus asendame type tüübiga ning name property nimega (vt koodinäide 3).

```
/// <summary>
/// Gets or sets the property value.
/// </summary>
public type name
{
        get { return GetValue<type>(nameProperty); }
    set { SetValue(nameProperty, value); }
}
/// <summary>
/// Register the name property so it is known in the class.
/// </summary>
public static readonly PropertyData nameProperty =
    RegisterProperty("name", typeof(type), null);
```

Koodinäide 3. Property nime asendamine.

Olgu meil Person mudelil parameeter nimega FirstName tüübiga string. Nüüd muutes type ja klikkides enter, muudab ta type ära kõigis 3 kohas ning sama tehes name'ga, muutub name 5 kohas. Tulemus (vt koodinäide 4):

```
/// <summary>
/// Gets or sets the property value.
/// </summary>
public string FirstName
{
        get { return GetValue<string>(FirstNameProperty); }
        set { SetValue(FirstNameProperty, value); }
}

/// <summary>
/// Register the Name property so it is known in the class.
/// </summary>
public static readonly PropertyData FirstNameProperty =
        RegisterProperty("FirstName",  typeof(string), null);
```

Koodinäide 4. Property peale asendust.

Analoogiliselt lisame veel parameetrid LastName, mis on samuti tüübilt string ja siis veel Id, tüübiga int ning CompanyId, mis määrab ära firma id, millega antud Person on seotud.

Lisame projekti veel ühe mudeli nimega Company ettevõtte informatsiooni hoidmiseks. Company mudelile lisame string tüüpi propertied Name ning int tüüpi property Id. Nüüd on meil 2 mudelit, mis näevad välja järgnevalt (vt koodinäide 5 ja koodinäide 6):

Company.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;
using Catel.Data;

namespace CompanyManager.Models
{
    /// <summary>
    /// Company model which fully supports serialization, property changed notifications,
    /// backwards compatibility and error checking.
    /// </summary>
#if !SILVERLIGHT
    [Serializable]
#endif
    public class Company : ModelBase
    {
        #region Fields
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new object from scratch.
        /// </summary>
        public Company() { }

#if !SILVERLIGHT
        /// <summary>
        /// Initializes a new object based on <see cref="SerializationInfo"/>.
        /// </summary>
```

```
        /// <param name="info"><see cref="SerializationInfo"/> that contains the information.</param>
        /// <param name="context"><see cref="StreamingContext"/>.</param>
        protected Company(SerializationInfo info, StreamingContext context)
            : base(info, context) { }
#endif
        #endregion

        #region Properties
        // TODO: Define your custom properties here using the modelprop code snippet
        /// <summary>
        /// Id.
        /// </summary>
        public int Id
        {
            get { return GetValue<int>(IdProperty); }
            set { SetValue(IdProperty, value); }
        }

        /// <summary>
        /// Register the Id property so it is known in the class.
        /// </summary>
        public static readonly PropertyData IdProperty = RegisterProperty("Id", typeof(int), null);

        /// <summary>
        /// Gets or sets the property value.
        /// </summary>
        public string Name
        {
            get { return GetValue<string>(NameProperty); }
            set { SetValue(NameProperty, value); }
        }

        /// <summary>
        /// Register the Name property so it is known in the class.
        /// </summary>
        public static readonly PropertyData NameProperty = RegisterProperty("Name", typeof(string), null);

        #endregion

        #region Methods
        /// <summary>
        /// Validates the field values of this object. Override this method to enable
```

```
        /// validation of field values.
        /// </summary>
        /// <param name="validationResults">The validation results, add additional results to this list.</param>
        protected override void ValidateFields(List<IFieldValidationResult> validationResults)
        {
        }

        /// <summary>
        /// Validates the field values of this object. Override this method to enable
        /// validation of field values.
        /// </summary>
        /// <param name="validationResults">The validation results, add additional results to this list.</param>
        protected override void ValidateBusinessRules(List<IBusinessRuleValidationResult> validationResults)
        {
        }
        #endregion
    }
}
```

Koodinäide 5. Company mudel.

Person.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.Serialization;
using Catel.Data;
using Catel.MVVM;

namespace CompanyManager.Models
{
    /// <summary>
    /// Person model which fully supports serialization, property changed notifications,
    /// backwards compatibility and error checking.
    /// </summary>
#if !SILVERLIGHT
    [Serializable]
#endif
```

```
  public class Person : ModelBase
  {
    #region Fields
    #endregion

    #region Constructors
    /// <summary>
    /// Initializes a new object from scratch.
    /// </summary>
    public Person() { }

#if !SILVERLIGHT
    /// <summary>
    /// Initializes a new object based on <see cref="SerializationInfo"/>.
    /// </summary>
    /// <param name="info"><see cref="SerializationInfo"/> that contains the
information.</param>
    /// <param name="context"><see cref="StreamingContext"/>.</param>
    protected Person(SerializationInfo info, StreamingContext context)
      : base(info, context) { }
#endif
    #endregion

    #region Properties
    // TODO: Define your custom properties here using the modelprop code snippet
    /// <summary>
    /// Gets or sets the property value.
    /// </summary>
    public string FirstName
    {
      get { return GetValue<string>(FirstNameProperty); }
      set { SetValue(FirstNameProperty, value); }
    }

    /// <summary>
    /// Register the FirstName property so it is known in the class.
    /// </summary>
    public static readonly PropertyData FirstNameProperty = RegisterProperty("FirstName",
typeof(string), null);

    /// <summary>
    /// Gets or sets the property value.
    /// </summary>
    public string LastName
```

```
    {
        get { return GetValue<string>(LastNameProperty); }
        set { SetValue(LastNameProperty, value); }
    }

    /// <summary>
    /// Register the LastName property so it is known in the class.
    /// </summary>
    public static readonly PropertyData LastNameProperty = RegisterProperty("LastName",
typeof(string), null);

    /// <summary>
    /// Gets or sets the property value.
    /// </summary>
    public int Id
    {
        get { return GetValue<int>(IdProperty); }
        set { SetValue(IdProperty, value); }
    }

    /// <summary>
    /// Register the Id property so it is known in the class.
    /// </summary>
    public static readonly PropertyData IdProperty = RegisterProperty("Id", typeof(int), null);


    #endregion

    #region Methods
    /// <summary>
    /// Validates the field values of this object. Override this method to enable
    /// validation of field values.
    /// </summary>
    /// <param name="validationResults">The validation results, add additional results to this
list.</param>
    protected override void ValidateFields(List<IFieldValidationResult> validationResults)
    {
    }

    /// <summary>
    /// Validates the field values of this object. Override this method to enable
    /// validation of field values.
    /// </summary>
```

```
        /// <param name="validationResults">The validation results, add additional results to this list.</param>
        protected override void ValidateBusinessRules(List<IBusinessRuleValidationResult> validationResults)
        {
        }
        #endregion
    }
}
```

Koodinäide 6. Person mudel.

# 5. Vaatemudeli lisamine

Vaatemudelis (ingl.k ViewModel) hakkavad asuma objektid andmetaga, mis on vajalik sellega seotud vaate kuvamiseks. Näiteks kirjeldatakse ära mudelid või kollektsioonid mudelitest, mida vaade saab siis hiljem kuvada.

Vaatemudeli lisamine käib järgnevalt. Solution Exploreris ViewModels kaustale parema klahviga klikkides ning valides Add->New Item->ViewModel (Catel). Paneme nimeks CompanyViewModel.

See hakkab sisaldama siis endas äriloogikat firmade vaate jaoks.

Esimene element siin võiks olla kollektsioon firmade hoidmiseks, selleks sobib hästi tüüp ObservableCollection<Company>. ObservableCollection on dünaamiline kollektsioon, mis saadab notification'eid kui elemente lisatakse, kustutakse või listi uuendatakse (http://msdn.microsoft.com/en-us/library/ms668604%28v=vs.110%29.aspx).

Lisame paketid, mida on vaja mudelite kasutamiseks ja ObservableCollectioni kasutamiseks (vt koodinäide 7).

```
using System.Collections.ObjectModel;
using Models;
using Catel.Data;
```

Koodinäide 7. Vajalike pakettide lisamine

Nüüd lisatud CompanyViewModel sisaldab endas õpetust mudeli lisamiseks, meie jaoks hakkabki olema esimene mudel tüübist ObservableCollection<Company>. Selleks jälgime õpetust (vt koodinäide 8).

```
// TODO: Register models with the vmpropmodel codesnippet
```

Koodinäide 8. Õpetus vaatemudelile mudeli lisamiseks

Kusagil klassi sees kirjutame vmpropmodel ja vajutame tab ning asendame tüübi ja nime samamoodi nagu mudeli propery genereerimisel (vt mudeli property genereerimist joonis 2, 3 ja 4).

Samamoodi lisame veel ühe mudeli nimekirjas aktiivse ehk valitud firma hoidmiseks mudeli tüübist Company nimega SelectedCompany (vt koodinäide 9).

```
/// <summary>
/// Gets or sets the property value.
/// </summary>
[Model]
public ObservableCollection<Company> Companies
{
        get { return GetValue<ObservableCollection<Company>>(CompaniesProperty); }
       private set { SetValue(CompaniesProperty, value); }
}

/// <summary>
/// Register the Companies property so it is known in the class.
/// </summary>
public static readonly PropertyData CompaniesProperty =
    RegisterProperty("Companies", typeof(ObservableCollection<Company>));


/// <summary>
/// Gets or sets the property value.
/// </summary>
[Model]
public Company SelectedCompany
{
        get { return GetValue<Company>(SelectedCompanyProperty); }
    private set { SetValue(SelectedCompanyProperty, value); }
}
/// <summary>
/// Register the SelectedCompany property so it is known in the class.
/// </summary>
public static readonly PropertyData SelectedCompanyProperty =
    RegisterProperty("SelectedCompany", typeof(Company));
```

Koodinäide 9. Companies ja SelectedCompany property'de lisamine

Lisame veel ühe property, kus hoiame lisatava firma nime, olgu nimeks NewCompanyName (vt koodinäide 10).

```
/// <summary>
/// Gets or sets the property value.
/// </summary>
public string NewCompanyName
{
        get { return GetValue<string>(NewCompanyNameProperty); }
    private set { SetValue(NewCompanyNameProperty, value); }
}

/// <summary>
/// Register the NewCompanyName property so it is known in the class.
/// </summary>
public static readonly PropertyData NewCompanyNameProperty =
    RegisterProperty("NewCompanyName", typeof(string));
```

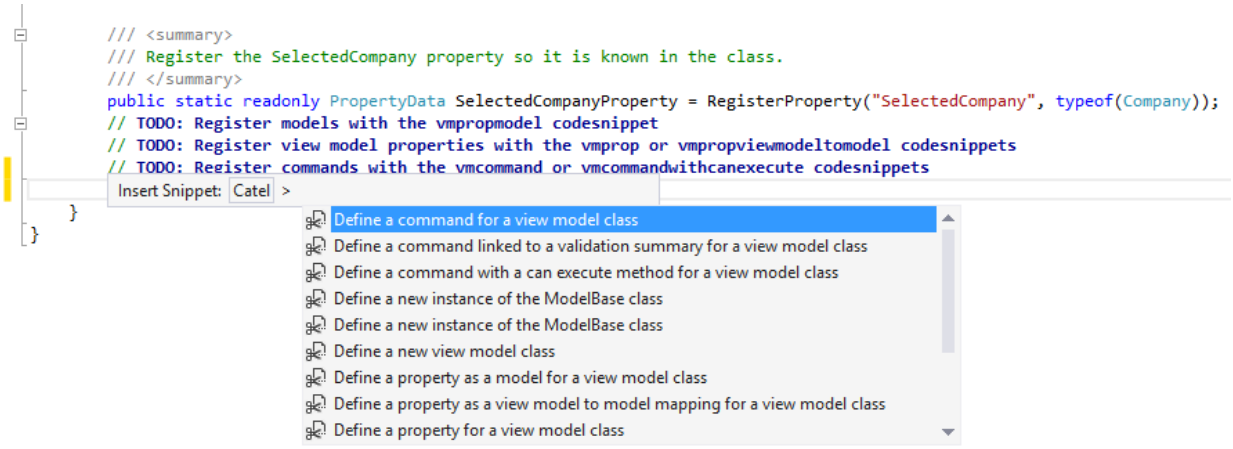
Koodinäide 10. NewCompanyName lisamine.

Lisame veel ka ühe Command'i, mis käivitatakse hiljem uue firma lisamisel. Vaadates viewmodeli automaatselt genereeritud koodi, leiame snippet'i Commandi loomiseks (vt koodinäide 11).

```
// TODO: Register commands with the vmcommand or vmcommandwithcanexecute codesnippets
```

Koodinäied 11. Command'i lisamise snippeti näide.

Kirjutame klassi sisse vmcommand ning klikime Tab nuppu. Vahel ei pruugi see õnnestuda, siis saame toimida nii nagu järgneval pildil. Klikkides parema klahviga soovitud kohal, valides Insert snippet. Ja sealt edasi Define a command for view model class (vt joonis 12).




Joonis 12. Command' snippeti lisamine.

Asendame name igal pool Command'i nime InsertCompany'ga (vt koodinäide 12).

```
/// <summary>
/// Gets the InsertCompany command.
/// </summary>
public Command InsertCompany { get; private set; }


/// <summary>
/// Method to invoke when the InsertCompany command is executed.
/// </summary>
private void OnInsertCompanyExecute()
{
   // TODO: Handle command logic here
}
```

Koodinäide 12. Command nimi peale asendust.

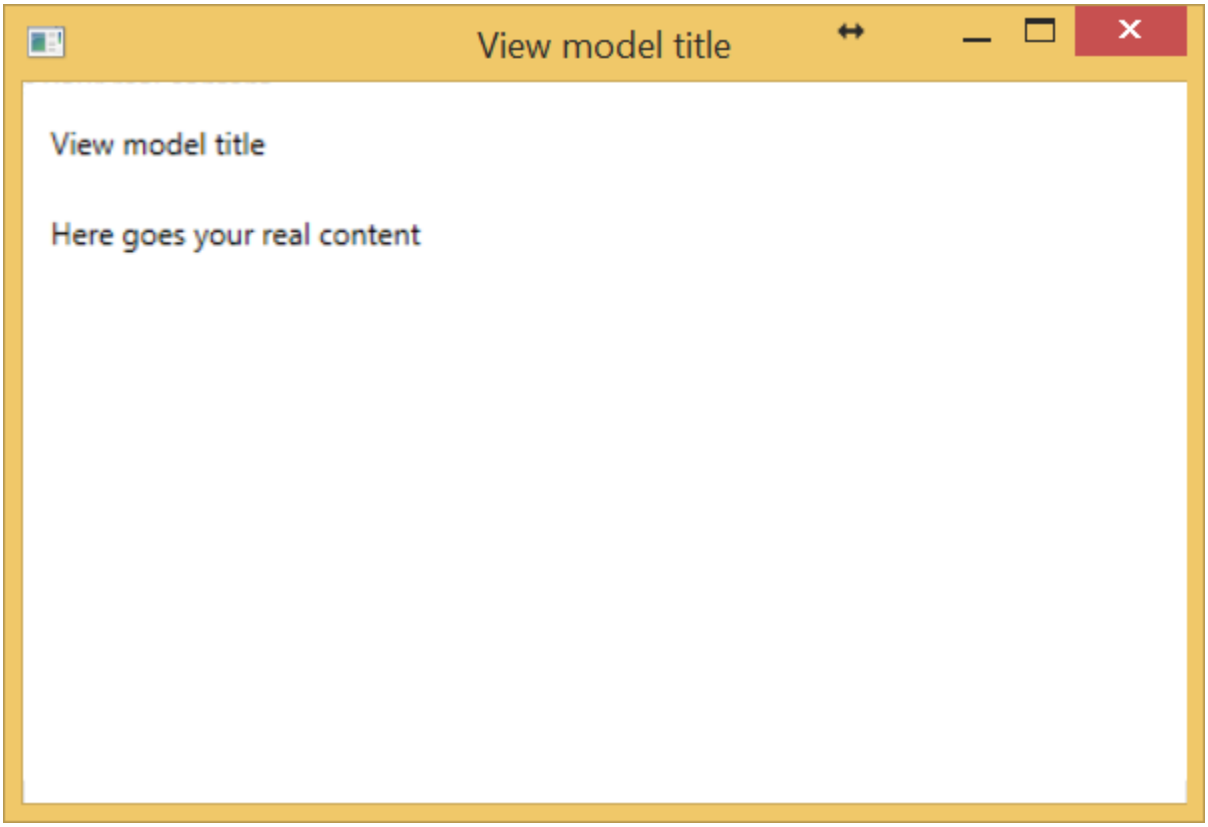
Seejärel liigutame järgmise rea CompanyViewModel konstruktorisse (vt koodinäide 13):

```
InsertCompany = new Command(OnInsertCompanyExecute);
```

Koodinäide 13. Commandi liigutamine konstruktorisse.

# 6. Vaate lisamine

Vaade (ingl.k View) on koht, kus hakkab paiknema programmi mingi osa kujundus. Uue projekti loomisel, luuakse vaikimisi MainWindow komponent, mis avatakse programmi käivitamisel. Kui praeguse koodiga käivitada (debugida), siis tekib tühi aken, kus on kirjas 2 rida (vt joonis 13).




Joonis 13. Aken preaguse koodiga.

Vaatame selle komponendi koodi. Selleks avame faili Solution Explorerist kaustast Views nimega MainWindow.xaml (vt koodinäide 14).

```
<catel:DataWindow x:Class="CompanyManager.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                        xmlns:catel="http://catel.codeplex.com"
        ShowInTaskbar="True"      ResizeMode="CanResize"      SizeToContent="Manual"
WindowStartupLocation="Manual" WindowState="Maximized">

  <!-- Resources -->
```

```
  <catel:DataWindow.Resources>
  </catel:DataWindow.Resources>

  <!-- Content -->
   <catel:StackGrid x:Name="LayoutRoot">
                <catel:StackGrid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="Auto" />
                </catel:StackGrid.RowDefinitions>

                <Label Content="{Binding Title}" />
     <Label Content="Here goes your real content" />
  </catel:StackGrid>
</catel:DataWindow>
```

Koodinäied 14. MainWindow.xaml sisu.

Näeme, et komponent ise on tüübilt catel:Datawindow ning selle sees on catel:StackGrid, mis endast kujutab põhimõtteliselt võimalust kujundust sättida paika tabeli kujul, jagades akna veergudeks ja ridadeks , võimaldades sinna komponente ilusasti paigutada. Antud juhul on tehtud 2 rida ehk jagatud aken horisontaalselt kaheks osaks. Height="Auto" tähendab, et nende kahe piirkonna suurused sõltuvad konkreetselt sisu kõrgusest. Antud juhul on esimene ülemine rida pealkirja kõrgune ja teine on automaatselt kogu ülejäänud akna piirkond.

Uue vaate lisamine käib järgnevalt. Solution Exploreris Views kaustale parema klahviga klikkides ning valides Add->New Item->User Control (WPF with Catel). Paneme nimeks CompanyView.xaml.

## 6.1. Kujundame Company vaate

Kõigepealt näitaks selles vaates nimekirja firmadest. Selleks sobib hästi control ListView

Kustutame kõigepealt sellest vaatest automaatselt genereeritud 2 Label'it. Kustutame järgnevad read failist CompanyView.xaml (vt koodinäide 15):

```
<Label Content="{Binding Title}" />
<Label Content="Here goes your real content" />
```

Koodinäide 15. Kustutame labelid.

Ning asemele paneme (vt koodinäide 16).

```
<ListView ItemsSource="{Binding Companies}" SelectedItem="{Binding SelectedCompany}"
DisplayMemberPath="Name"></ListView>
```

Koodinäide 16. Asendame labelid.

Selle reaga ütleme, et tekita listview milles elemendid seo (Binding) vastava viewmodeli (CompanyViewModel) property'ga Companies ning valitud element seo sama viewmodeli property'ga SelectedCompany ning DisplayMemberPath näitab millist välja objektist näidata nimekirjas.

Kuna meil praegu firmasid ei ole, siis programmi käivitamisel näidatakse tühja ListView'd. Lisame võimaluse uut firmat sisestada. Selleks lisame labeli, tekstivälja ja ühe nupu. Need võiks paikneda firmade nimekirjast näiteks paremal pool, selleks kasutan catel:StackGridi (vt stackgridi kasutamist). Panen paika ka mingid esialgsed väljade suurused, mida hiljem võib vajadusel muuta. Nupul (Button kontrol) lisame atribuudi Command ning seome ära Command tüüpi property'ga viewmodel'is. Tekstiväljal (TextBox) seome atribuudi Text viewmodeli property'ga NewCompanyName.

CompanyViev võiks välja näha nüüd järgnev (vt koodinäide 17):

```
<catel:UserControl x:Class="CompanyManager.Views.CompanyView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                                 xmlns:catel="http://catel.codeplex.com">

  <!-- Resources -->
  <UserControl.Resources>
  </UserControl.Resources>

  <!-- Content -->
  <catel:StackGrid>
                  <catel:StackGrid.RowDefinitions>
       <RowDefinition Height="Auto" />
           <RowDefinition Height="Auto" />
    </catel:StackGrid.RowDefinitions>
```

```
      <catel:StackGrid.ColumnDefinitions>
        <ColumnDefinition Width="150"></ColumnDefinition>
        <ColumnDefinition Width="Auto"></ColumnDefinition>
      </catel:StackGrid.ColumnDefinitions>
      <Label Grid.ColumnSpan="2" Content="Companies List"></Label>
      <ListView MinHeight ="400"  ItemsSource="{Binding Companies}" SelectedItem="{Binding
SelectedCompany}" DisplayMemberPath="Name"></ListView>
      <catel:StackGrid>
        <catel:StackGrid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />

        </catel:StackGrid.RowDefinitions>
        <catel:StackGrid.ColumnDefinitions>
          <ColumnDefinition Width="50"></ColumnDefinition>
          <ColumnDefinition Width="Auto"></ColumnDefinition>
        </catel:StackGrid.ColumnDefinitions>
        <Label Content="Name"></Label>
        <TextBox Text="{Binding NewCompanyName}" Width="200"></TextBox>
        <Button      Grid.ColumnSpan="2"      Width="200"      HorizontalAlignment="Right"
Content="Insert" Command="{Binding InsertCompany}"></Button>
      </catel:StackGrid>

    </catel:StackGrid>
</catel:UserControl>
```

Koodinäide 17. Disainitud CompanyView.xaml.

Grid.ColumnSpan="2" tähendab, et kontrol ulatub üle kahe veeru ning HorizontalAlignment="Right"

tähendab, et kontrol on joondatud paremale (see annab selle, et nupp paikneb tekstivälja all)

# 7. Vaate jagamine regioonideks ning nende sisu dünaamiline vahetamine

Kui me tahame jagada programmiakna erinevateks regioonideks nii, et valikutest ühes regioonis muutuks teise regiooni sisu. Näiteks kui vasakus regioonis on navigeerimise nupud kahe vaate vahel ning tahame, et neil nuppudel klikkides muutuks parema regiooni sisu, siis tuleks programmi lisada järgmised kompnendid: Prism laienduspakett ja Bootstrapper.

## 7.1. Prism laienduspaketi lisamine

Prism on juhis komponentidest üles ehitatud programmi koostamiseks. Üks selle komponente on kasutajaliidese kompositsioon. Põhimõtteliselt on see kasutajaliidse jagamine regioonideks, mille sisu tuleb eraldi komponentidest ja nende regioonide sisu on dünaamiliselt vahetatav (Geert van Horrik, 2014).

Kui on vaja, et üks regioon oleks dünamiliselt muudetav, siis installime vastava nuget paketi. Nimega Catel.Extensions.Prism (vt Nuget paketi lisamine projekti)

## 7.2. Kujundame peaakna

Koostame peaakna disaini nii, et see koosneb kahest osast. Vasak ja parem. Vasakule piirkonnale jätame programmi navigeerimise nupud ning vastavalt valikule hakkab parema piirkonna sisu muutuma.

Kustutame Label'id ja teeme ümber stackgridi nii, et see koosneks kahe rea asemel kahest veerust.

Jätame esialgu vasaku regiooni tühjaks ja panema sinna lihtsalt teksti Left-Region. Selleks sobib hästi selline kontrol nagu Label . catel:StackGrid täidab tabeli vasakult paremale ridade

kaupa. Kuna meil on kaks veergu, siis automaatselt esimene kontrol pannakse vasakusse regiooni ja teine paremasse.

Lisaks tuleb MainWindow.xaml päisesse lisada catel:DataWindow tagi sisse (vt koodinäide 18).

```
xmlns:regions="http://www.codeplex.com/CompositeWPF"
```

Koodinäide 18. Regions lisamine xamli.

Siis saame määrata eelpool tehtud stackgridi paremaks komponendiks dünaamilise regiooni nii (vt koodinäide 19):

```
<ContentControl regions:RegionManager.RegionContext="Right-Region"></ContentControl>
```

Koodinäied 19. Dünaamiline regioon.

Nüüd on MainWindow.xaml sisu järgnev (vt koodinäide 20):

```
<catel:DataWindow x:Class="CompanyManager.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                        xmlns:catel="http://catel.codeplex.com"
                        xmlns:regions="http://www.codeplex.com/CompositeWPF"
                        ShowInTaskbar="True"                ResizeMode="CanResize"
SizeToContent="Manual" WindowStartupLocation="Manual" WindowState="Maximized">

  <!-- Resources -->
  <catel:DataWindow.Resources>
  </catel:DataWindow.Resources>

  <!-- Content -->
   <catel:StackGrid x:Name="LayoutRoot">
            <catel:StackGrid.RowDefinitions>
                <RowDefinition Height="Auto" />
            </catel:StackGrid.RowDefinitions>
            <catel:StackGrid.ColumnDefinitions>
      <ColumnDefinition Width="100"></ColumnDefinition>
      <ColumnDefinition Width="Auto"></ColumnDefinition>
    </catel:StackGrid.ColumnDefinitions>
            <Label Content="Left region" />
    <ContentControl regions:RegionManager.RegionName="RightRegion">
      </ContentControl>
```

```
    </catel:StackGrid>
</catel:DataWindow>
```

Koodinäide 20. MainWindow sisu dünaamiliste regioonidega.

Siin ColumnDefinition read määravad ära, mitu veergu tabelis on ning RowDefinition määravad ridade arvu. Antud juhul siis on tabel ühe reaga ja kahe veeruga.

## 7.3. Bootstrapperi lisamine

Selleks, et Prismi kasutajaliidese kompositsiooni kasutada on vaja programmi viia sisse mõned muudatused. Tuleb programmi lisada bootstrapper ning panna programm seda kasutama. Bootstrapper on põhimõtteliselt üks vahemoodul, mis hoolitseb programmi käivitamise eest. Töötav näide bootsrapperi kasutamisest on vaadatav siit (Geert van Horrik, 2014).
Kasutame seda näidet ja lisame sealt Bootstrapper.cs otse CompanyManager projekti alla (vt Klassi lisamine)
Kustutame mitte vajalikud read (vt koodinäide 21):

```
Container.RegisterType<IDepartmentRepository, DepartmentRepository>();
Container.RegisterType<IEmployeeRepository, EmployeeRepository>();
```

Koodinäide 21. Ebavajalike ridade kustutamine Bootstrapperist.

Ning muudame

```
public class Bootstrapper : BootstrapperBase<ShellView, ConfigurationModuleCatalog>
```

Asendades shellview mainviewga

```
public class Bootstrapper : BootstrapperBase<MainWindow, ConfigurationModuleCatalog>
```

ning muudame ära nimeruumi:

```
namespace Catel.Examples.WPF.Prism
```

asemele

```
namespace CompanyManager
```

Bootstrapper.cs sisu oleks nüüd järgnev (vt koodinäide 22):

```
using Catel;
using CompanyManager.Views;
using Microsoft.Practices.Prism.Modularity;
using Microsoft.Practices.Prism.Regions;

namespace CompanyManager
{
    /// <summary>
    /// The bootstrapper.
    /// </summary>
    public class Bootstrapper : BootstrapperBase<MainWindow, ConfigurationModuleCatalog>
    {
        #region Method Overrides
        /// <summary>
        /// Configures the <see cref="IUnityContainer"/>. May be overwritten in a derived
class to add specific
        /// type mappings required by the application.
        /// </summary>
        protected override void ConfigureContainer()
        {
            base.ConfigureContainer();

            //Container.RegisterType<IDepartmentRepository, DepartmentRepository>();
            //Container.RegisterType<IEmployeeRepository, EmployeeRepository>();
        }

        /// <summary>
        /// Configures the <see
cref="T:Microsoft.Practices.Prism.Modularity.IModuleCatalog"/> used by Prism.
        /// </summary>
        protected override void ConfigureModuleCatalog()
        {
            ModuleCatalog.Initialize();
        }

        /// <summary>
        /// Configures the default region adapter mappings to use in the application, in
order
        /// to adapt UI controls defined in XAML to use a region and register it
automatically.
        /// </summary>
        /// <returns>The RegionAdapterMappings instance containing all the
mappings.</returns>
        protected override RegionAdapterMappings ConfigureRegionAdapterMappings()
        {
            // Call base method
            var mappings = base.ConfigureRegionAdapterMappings();
            return ObjectHelper.IsNull(mappings) ? null : mappings;
        }
        #endregion
    }
}
```

Koodinäide 22. Bootstrapper.cs peale vajalikke muudatusi.

Nüüd kui bootstrapper on valmis, siis tuleb öelda programmile, et see seda kasutaks

Võtame App.xaml.cs ning näite järgi lisame siia konstruktori (vt koodinäide 23):

```
public App()
{
    var dispatcherService = ServiceLocator.Default.ResolveType<IDispatcherService>();

    dispatcherService.BeginInvoke(() =>
    {
        CatelEnvironment.RegisterDefaultViewModelServices();

        var bootstrapper = new Bootstrapper();
        bootstrapper.Run();
    });
}
```

Koodinäide 23. App.xaml.cs konstruktori muudatused.

Samas failis meetodis OnStartup() kustutame bloki (vt koodinäide 24):

```
#if DEBUG
            Catel.Logging.LogManager.RegisterDebugListener();
#endif
```

Koodinäide 24. Bloki kustutamine OnStartup() meetodist.

Ning lisame kolm paketti, kus asub vajalik info kasutatud tüüpide ja meetodite kohta (vt koodinäide 25).

```
using Catel;
using Catel.IoC;
using Catel.Services;
```

Koodinäide 25. Vajalike nimeruumide lisamine.

Nüüd kui meil on Bootstrapper lisatud, siis tuleks kustutada App.xaml'is ära StartUpUri (vt koodinäide 26).

```
<Application x:Class="CompanyManager.App"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        StartupUri="/Views/MainWindow.xaml">
```

Koodinäide 26. StartUpUri kustutamine.

Järele jääb (vt koodinäide 27).

```
<Application x:Class="CompanyManager.App"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Application.Resources>

    <!-- Generic Catel theme -->
    <ResourceDictionary
Source="/Catel.Extensions.Controls;component/themes/generic.xaml" />

  </Application.Resources>
</Application>
```

Koodinäide 27. App.xaml peale StartUpUri kustutamist.

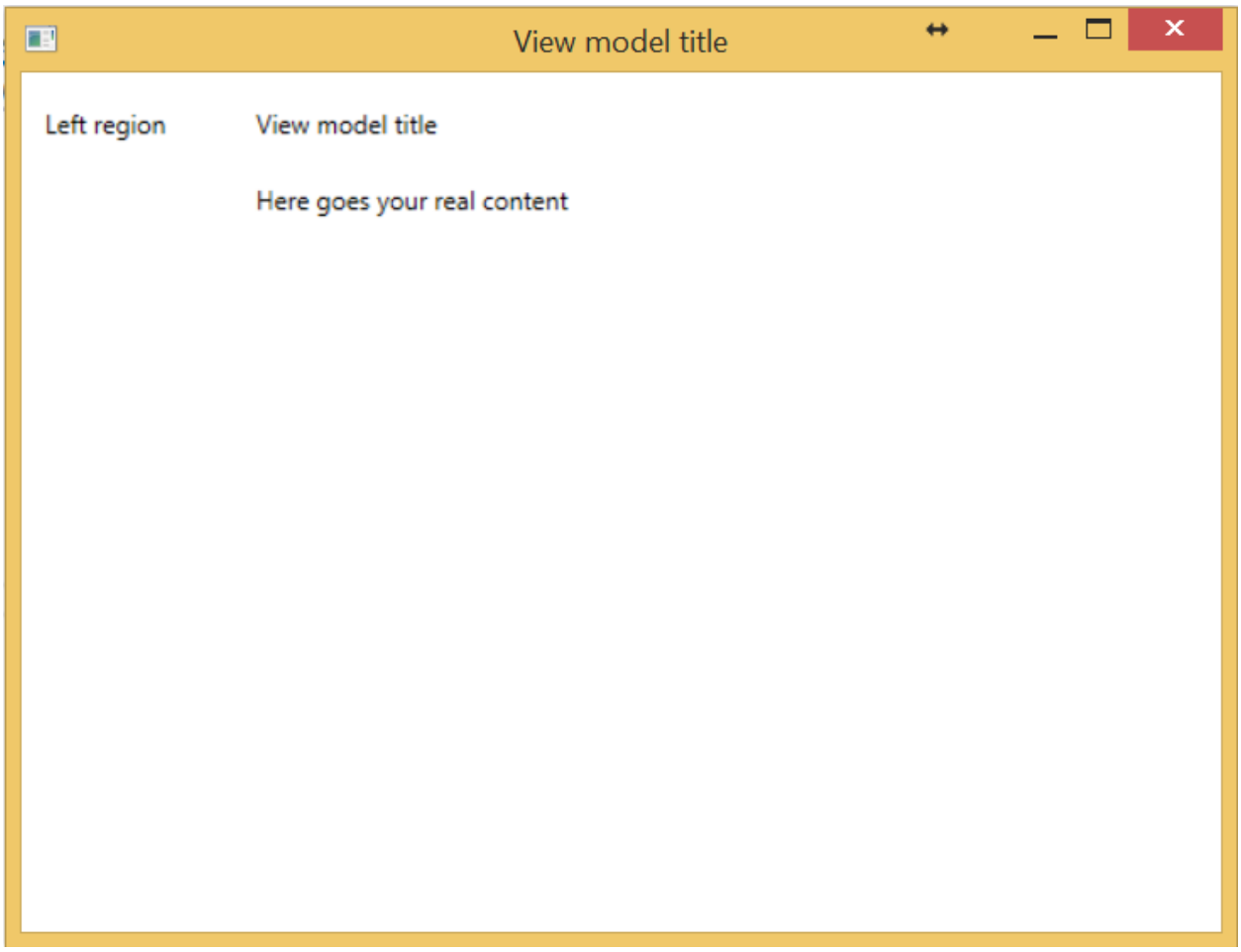
MainViewModelis lisan `using System.Threading.Tasks; ja` kirjutan üle Initialize (vt koodinäide 28).

```
protected override Task Initialize()
{
    var visualizerService =
        Catel.IoC.ServiceLocator.Default.ResolveType<UICompositionService>();
    visualizerService.Activate(new CompanyViewModel(), this, "RightRegion");
    return base.Initialize();
}
```

Koodinäide 28. Nimeruumi lisamine ja initialize meetodi muudtus.

Nüüd kui käivitada programm, siis näeb see välja nii, vasakul pool on left regioon

Paremal siis right regioon, kus hetkel on companyview sisu (vt joonis 14).



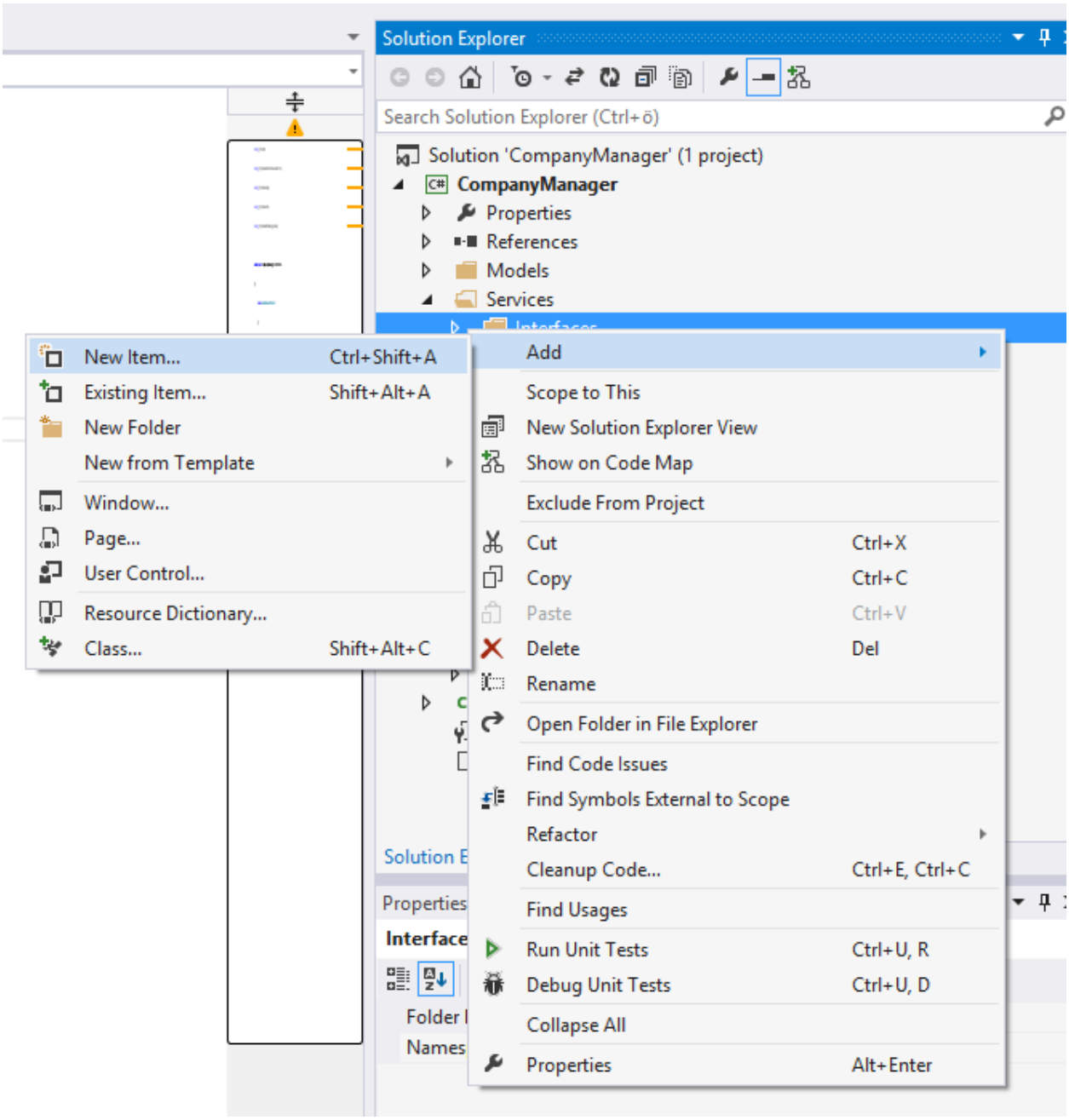

Joonis 14. Programmiaken kahe regiooniga.

# 8. Teenused ja nende lisamine IoC konteinerisse

## 8.1. Teenused

Catelis paigutatakse kogu äriloogika eraldi teenustesse (ingl.k Service). Teemegi valmis esimese teenuse, mis hakkab tegelema andmebaasiga suhtlemisega. Selleks lisame Solution Exploreris Service kausta klassi DatabaseService.cs (vt Klassi lisamine).

Nüüd lisame veel liidese (ingl.k Interface) selle komponendi poole pöördumiseks. Liideste jaoks on eraldi kaust Services kausta sees, nimega Interfaces. Klikime parema klahviga peale ning valime Add New Item (vt joonis 15).




Joonis 15. Uue elemendi lisamine.

Avaneb aken, kust valime Interface ning paneme nimeks IDatabaseService.cs . .NET'i stiilis on kõigi interface nimede ees suur I täht.

Nüüd paneme DatabaseService klassi implementeerima seda IDatabaseService liidest. Selleks lisame kõigepealt uue nimeruumi, kus interface'd paiknevad: using CompanyManager.Services.Interfaces;

DatabaseService.cs sisu oleks nüüd järgmine (vt koodinäide 29):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CompanyManager.Services.Interfaces;

namespace CompanyManager.Services
{
   class DatabaseService : IDatabaseService
   {
   }
}
```

Koodinäide 29. DatabaseService.cs sisu.

Nüüd on meil olemas esimene teenus ja selle liides ning Catelile saab nüüd öelda, et pane need IoC konteinerisse.

## 8.2. IoC konteiner

IoC konteiner (ingl.k Inversion of Control container) on konteiner, kus on võimalik siduda liidesed seda implementeeriva klassiga. Antud õppematerjalis ei hakka IoC teemal pikemalt peatuma. Kui lugejat teema lähemalt huvitab, siis selle kohta on internetis leida palju materjali, näiteks Inversion Of Control (Martin Fowler, 2005). Antud õppematerjalis on oluline teada seda, et see konteiner lihtsustab komponentide hoidmist liidestena ühes konteineris, kust on neid vajadusel lihtne välja küsida või asendada teise komponendiga, mis seda liidest implementeerib.

Selleks, et sellesse konteinerisse teenust lisada, tuleks kirjutada rida App.xaml.cs faili meetodisse OnStartup()(vt koodinäide 30).

```
ServiceLocator.Default.RegisterType<IDatabaseService, DatabaseService>();
```

Koodinäide 30. Teenuse registreerimine.

See annab meile võimaluse terve programmi piires küsida hiljem sellest konteinerist välja liidesele IDatabaseService vastava teenuse DatabaseService.
Konteinerist küsimiseks tuleb toimida nii (vt koodinäide 31):

```
var databaseService = Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
```

Koodinäide 31. Teenuse küsimine IoC konteinerist.

Paneme tähele, et küsimine käib alati liidese järgi, mis tähendab, et konteineris endas on info selle kohta olemas, mis teenus täpselt liidsega on sinna salvestatud.

# 9. SQLite andmebaas

Programmis tuleks andmeid kuskil hoida ning selleks sobib väga hästi SQLite andmebaas. Selleks lisame projekti uue NuGet Package (vt nuget package lisamist). Package nimi on System.Data.SQLite (x86/x64). SQLite andmebaasi enda kohta võib lähemalt lugeda (http://www.sqlite.org/about.html). Lühidalt võib öelda, et SQLite on selline andmebaas, milles toimub tegelikult koguaeg faili kirjutamine ning failist lugemine. See annab võimaluse seda andmebaasi väga lihtsalt ühest programmist teise liigutada ja programmiga kaasa panna installides.

Alustasime service ja interfacega loomisega andmebaasiga suhtlemiseks.

Lisame interface'le IDatabaseService nüüd vajalikud meetodid andmebaasiga toimetamiseks. Esimene meetod võiks olla CreateTables(), mis genereerib meile kõik vajalikud tabelid, kui neid veel ei ole (esmakordsel käivitamisel).

Järgnevalt võiks lisada vajalikud meetodid nii firmade kui kasutajate lisamiseks, muutmiseks küsimisesks ning kustutamiseks.

Selleks, et saaks kasutada mudeleid Person ja Company, lisame nimeruumi: using CompanyManager.Models;

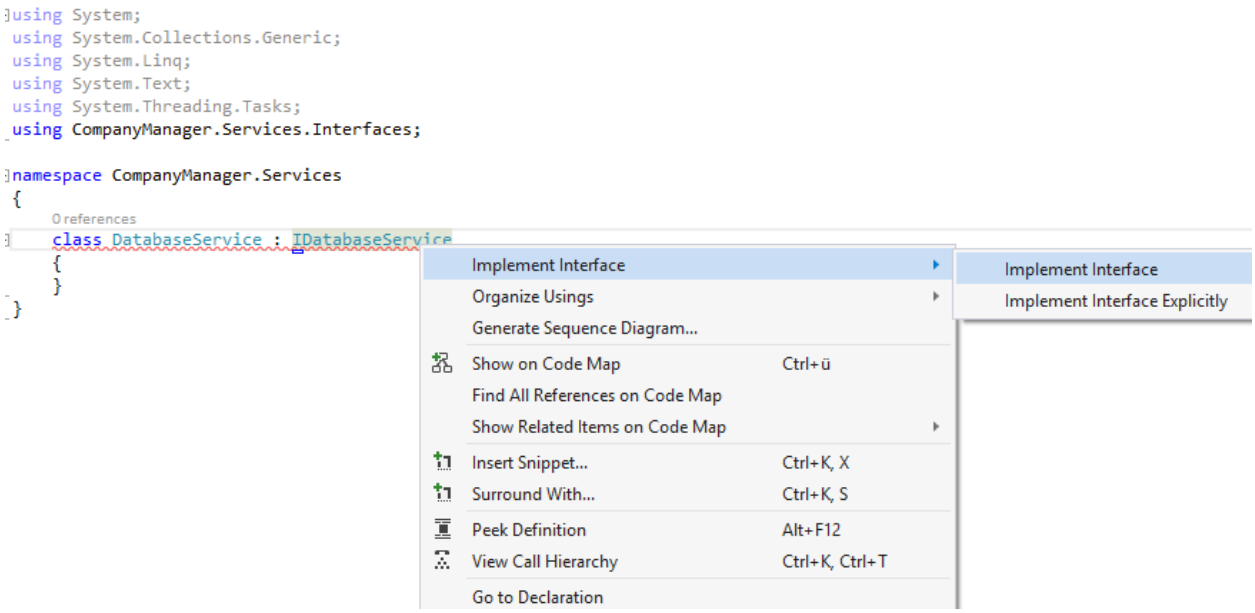
Iterface võiks välja näha siis järgmine (vt koodinäide 32):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CompanyManager.Models;

namespace CompanyManager.Services.Interfaces
{
   interface IDatabaseService
   {
      // Genereerib vajalikud tabelid
      void CreateTables();
```

```
        // Tagastab kõik firmad
        List<Company> GetCompanies();
        // Lisab firma
        void InsertCompany(Company company);
        // Muudab firmat
        void UpdateCompany(Company company);
        // Kustutab firma
        void DeleteCompany(int companyId);


        // Tagastab kõik firma isikud
        List<Person> GetCompanyPersons(int companyId);
        // Lisab firma
        void InsertPerson(Person person);
        // Muudab isikut
        void UpdatePerson(Person person);
        // Kustutab isiku
        void DeletePerson(int personId);
    }
}
```

Koodinäied 32. IDatabaseService liidese sisu.

Nüüd tuleks lisada kõik need meetodid ka DatabaseService'le, seda saab lihtsalt teha läbi Visual Studio, klikkides parema klahviga DatabaseService failis seal, kus on kirjas implementeerimise osa (vt joonis 16) .




Joonis 16. Liidese implementeerimine.

Nüüd peaks olema DatabaseService.cs sisu järgmine (vt koodinäide 33):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CompanyManager.Services.Interfaces;

namespace CompanyManager.Services
{
   class DatabaseService : IDatabaseService
   {
      public void CreateTables()
      {
         throw new NotImplementedException();
      }

      public List<Models.Company> GetCcompanies()
      {
         throw new NotImplementedException();
      }

      public void InsertCompany(Models.Company company)
      {
```

```
            throw new NotImplementedException();
        }

        public void UpdateCompany(Models.Company company)
        {
            throw new NotImplementedException();
        }

        public void DeleteCompany(int companyId)
        {
            throw new NotImplementedException();
        }

        public List<Models.Person> GetCompanyPersons(int companyId)
        {
            throw new NotImplementedException();
        }

        public void InsertPerson(Person person)
        {
            throw new NotImplementedException();
        }

        public void UpdatePerson(Models.Person person)
        {
            throw new NotImplementedException();
        }

        public void DeletePerson(int personId)
        {
            throw new NotImplementedException();
        }
    }
}
```

Koodinäide 33. DatabaseService.cs peale liidese implementeerimist.

Iga meetodi sisu on hetkel throw new NotImplementedException, mis tähendab, et kui neid meetodeid välja kutsuda, siis antakse exception, et selline meetod on implementeerimata. Järgmiseks tuleks valmis teha kõik need meetodid, kuid veel enne seda tuleks luua ühendus andmebaasiga.

Lisame paketid, mida on vaja andmebaasiteenusel (vt koodinäide 34):

```
using System.Data;
using System.Data.SQLite;
```

Koodinäied 34. Vajalike nimeruumide lisamine DatabaseService.cs'i.

Lisame privaatse meetodid Open(), mis avab ühenduse andmebaasiga ning privaatse muutuja connection ühenduse hoidmiseks (vt koodinäide 35).

```
private SQLiteConnection connection;

private void Open()
{
        connection = new SQLiteConnection("Data Source=CompanyManager.sqlite;Version=3");
    connection.Open();
}
```

Koodinäide 35. Ühenduse loomine andmebaasiga

Open meetodis antakse connection muutujale väärtus nii, et pannakse konstruktorisse kaasa connection string, mille kohta rohkem infot võib lugeda (http://www.connectionstrings.com/connection-strings-explained/). Oluline on siinkohal CompanyManager.sqlite, mis on SQLite andmebaasi faili nimi. Kui selline fail puudub, siis connection.Open() välja kutsdes, see fail luuakse. Version=3 tähendab, SQLite versiooni.
Pärast ühenduse kasutamist tuleb see ilusasti sulgeda ning teeme selleks ka privaatse meetodi Close()(vt koodinäide 35).

```
private void Close()
{
        if(connection != null && connection.State != ConnectionState.Closed)
        connection.Close();
}
```

Koodinäied 35. Andmebaasi ühenduse sulgemine.

Siin meetodis kõigepealt kontrollime, kas muutujal on väärtus (kas on varem üldse Open välja kutsutud) ning kas ühendus juba ei ole suletud staatuses. Ning vajadusel sulgeme.

Edasi lähme esimese automaatselt genereeritud meetodi CreateTables() juurde. Kustutame kõigepealt praeguse sisu ja loome ühenduse andmebaasiga (vt koodinäide 36).

```
public void CreateTables()
{
        Open();
}
```

Koodinäide 36. Tabelite loomise meetod.

Seejärel loome string tüüpi muutuja sql, kuhu paneme kirja Company tabeli loomise SQL käsu (vt koodinäide 37).

```
var sql = "CREATE TABLE IF NOT EXISTS Company (" +
      "Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +
      "Name NVARCHAR(128))";
```

Koodinäide 37. Tabeli loomise SQL käsk.

Antud materjalis ei hakka lahti seletama SQL käske andmebaasiga toimetamiseks. Võib lisaks vaadata SQLite kodulehel toodud SQL süntaksi dokumentatsiooni (http://www.sqlite.org/lang.html)

Selleks et SQL käsk käivitada, tuleb luua kõigepealt SQLiteCommand ning see siis omakorda käivitada. SQLiteCommand omab erinevaid konstruktoreid, meile sobib hetkel selline, mis esimese parameetrina võtab SQL päringu stringi ning teise parameetrina SQLiteConnection'i. Kui tegemist on SQL käsuga, mis mingeid andmeid tagastama ei pea, saab selle käsu käivitada SQLiteCommand'i meetodiga ExecuteNonQuery()(vt koodinäide 38).

```
var command = new SQLiteCommand(sql, connection);
command.ExecuteNonQuery();
```

Koodinäide 38. SQLiteCommand loomine.

Järgmiseks tuleks luua teine tabel Person, analoogiliselt. Kasutada saab juba deklareeritud muutujaid sql ja command.

Kui need 2 tabelit on nüüd valmis, siis ei tasu unustada, et ühendus tuleb ka sulgeda (vt koodinäide 39).

```
public void CreateTables()
{
        Open();
    var sql = "CREATE TABLE IF NOT EXISTS Company (" +
          "Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +
          "Name NVARCHAR(128))";

    var command = new SQLiteCommand(sql, connection);
    command.ExecuteNonQuery();

    sql = "CREATE TABLE IF NOT EXISTS Person (" +
                "Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +
                "FirstName NVARCHAR(128), " +
                "LastName NVARCHAR(128), " +
                "CompanyId INTEGER)";

    command = new SQLiteCommand(sql, connection);
    command.ExecuteNonQuery();

    Close();
}
```

Koodinäide 39. Tabelite loomine.

CreateTables() meetodi välja kutsumise võib tegelikult panna kohe DatabaseService konstruktorisse. Lisame klassile konstruktori (vt koodinäide 40).

```
public DatabaseService()
{
        CreateTables();
}
```

Koodinäide 40. Tabelite loomine konstruktorisse.

Järgmiseks võtame meetodi InsertCompany(). Jällegi kõigepealt loome ühenduse, muutujad sql ning command vastavalt siis SQL käsu ning SQLiteCommand'i hoidmiseks ning käivitame käsuga ExecuteNonQuery()(vt koodinäide 41).

```
public void InsertCompany(Models.Company company)
{
        Open();
    var sql = "INSERT INTO Company (Name) " +
          "VALUES (@Name)";

    var command = new SQLiteCommand(sql, connection);
    command.Parameters.AddWithValue("@Name", company.Name);

     command.ExecuteNonQuery();

     Close();
}
```

Koodinäide 41. Company sisestamine.

Siin @Name märgiga on tähistatud parameeter, mida tahame hiljem asendada. AddWithValue asendab hiljem päringus @Name company.Name väärtusega.

Analoogiliselt täidame meetodid UpdateCompany ja DeleteCompany (vt koodinäide 42).

```
public void UpdateCompany(Models.Company company)
{
        Open();
    var sql = "UPDATE Company SET Name = @Name " +
          "WHERE Id = @Id";

    var command = new SQLiteCommand(sql, connection);
    command.Parameters.AddWithValue("@Name", company.Name);
        command.Parameters.AddWithValue("@Id", company.Id);

    command.ExecuteNonQuery();

    Close();
}

public void DeleteCompany(int companyId)
{
        Open();
    var sql = "DELETE FROM Company WHERE Id = @Id";

    var command = new SQLiteCommand(sql, connection);
        command.Parameters.AddWithValue("@Id", companyId);
```

```
    command.ExecuteNonQuery();

    Close();
}
```

Koodinäide 42. Company Update ja Delete.

Järgnevalt teeme firmade nimekirja küsimise päringu ehk GetCompanies()

Algus on sama, kutsume välja meetodi Open() ja loome stringi sql ning commandi (vt koodinäide 43).

```
Open();
var sql = "SELECT Id, Name FROM Company";
var command = new SQLiteCommand(sql, connection);
```

Koodinäide 43. Company select.

Teeme kollektsiooni, kus hoida kõiki firmasid, mis meetod hakkab tagastama (vt koodinäide 44).

```
var companies = new List<Company>();
```

Koodinäide 44. List Company'de hoidmiseks.

Edasi tuleb käivitada command'i meetod ExecuteReader(), mis tagastab SQLiteDataReader tüüpi objekti.

Selle objekti salvestame muutujasse nimega reader (vt koodinäide 45).

```
var reader = command.ExecuteReader();
```

Koodinäide 45. SQLiteDataReader loomine.

Seda reader'it saame nüüd kasutada kõikide ridade vajalike andmete küsimiseks. Selleks küsime reader'lt meetodiga Read() while tsükli sees, kas leidub järgmine rida. Read() nimelt liigutab readeri järgmise rea juurde, kui see eksisteerib tagastab true, kui mitte, siis false.

While tsüklis teeme Company tüüpi objekti, kus hoiame iga rea kohta andmeid, mida hiljem kollektsiooni lisada (vt koodinäide 46).

```
var company = new Company();
```

Koodinäied 46. Uue company objekti loomine.

Nüüd saab reader'ilt küsida antud rea kohta väärtused. Iga väärtus tuleb veel teisendada õigeks tüübiks (vt koodinäide 47).

```
company.Id = Convert.ToInt32(reader["Id"]);
company.Name = Convert.ToString(reader["Name"]);
```

Koodinäide 47. Company väljadele väärtuste andmine reederist.

Saadud objekti lisame nüüd kollektsiooni companies (vt koodinäide 48).

```
companies.Add(company);
```

Koodinäide 48. Company lisamine companies listi.

Peale while tsüklit tagastame meetodis selle sama kollektsiooni, kus nüüd on kõik filmid koos andmetega.

Meetodi sisu on nüüd järgnev (vt koodinäide 49).

```
public List<Models.Company> GetCompanies()
{
        Open();
    var sql = "SELECT Id, Name FROM Company";
    var command = new SQLiteCommand(sql, connection);

    var reader = command.ExecuteReader();

    var companies = new List<Company>();

    while (reader.Read())
    {
        var company = new Company();
         company.Id = Convert.ToInt32(reader["Id"]);
         company.Name = Convert.ToString(reader["Name"]);
         companies.Add(company);
    }
    return companies;
}
```

Koodinäide 49. GetCompanies() meetodi sisu.

Nüüd on kõik vajalik firmadega toimetamiseks olemas. Jätame lugeja ülesandeks lisada sisu ülejäänud meetoditele, mis on vajalikud Person tabeliga toimetamiseks.

# 10. Andmebaasi teenuse sidumine firma vaatemudeliga

## 10.1. Company vaatemudeli mudelite täitmine andmetega andmebaasist

Meil on vajalikud meetodid DatabaseService'l nüüd olemas andmetega toimetamiseks. Kõigepealt lisame vajalikud paketid (vt koodinäide 50)

```
using Catel.IoC;
using CompanyManager.Services.Interfaces;
```

Koodinäide 50. Vajalike nimeruumide lisamine.

Et seda service't viewmodelis nüüd kasutada, tuleb see IOC konteinerist välja küsida. Teeme seda konstruktoris ja täidame Companies kollektsiooni andmebaasist võetavate andmetega (vt koodinäide 51).

```
var databaseService = Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
Companies = new ObservableCollection<Company>(databaseService.GetCompanies());
```

Koodinäide 51. Companies täitmine andmetega andmebaasist.

## 10.2. Company vaatemudeli käskude sisu täitmine

Meil on CompanyView's nupp Insert, mis on seotud Command'iga InsertCompany. Eelnevalt sai genereeritud küll Command ise, aga sellel puudus sisu.

Nupule vajutades peaks toimuma kaks asja. Esiteks peaks Company lisatama firmade nimekirja ja teiseks salvestatama andmebaasi. Paneme veel NewCompanyName väärtuse null'iks, et peale lisamist oleks tekstiväli jälle tühi (vt koodinäide 52).

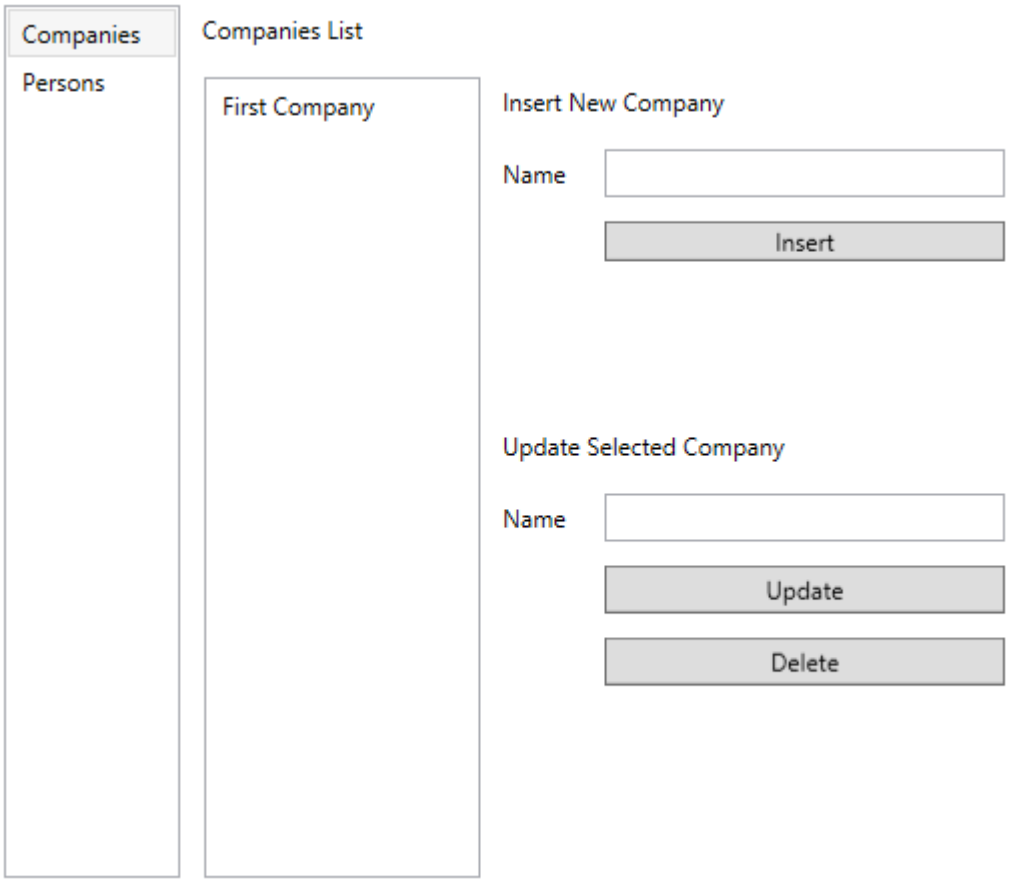
```
private void OnInsertCompanyExecute()
{
        var newCompany = new Company();
    newCompany.Name = NewCompanyName;
```

```
    Companies.Add(newCompany);
    var databaseService =
            Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
    databaseService.InsertCompany(newCompany);
        NewCompanyName = null;
}
```

Koodinäide 52. Company lisamise meetod.

## 10.3. Company andmete muutmise võimaluse lisamine

Lihtsuse mõttes ei hakka me eraldi vormi firma muutmiseks tegema vaid lisame selle funktsionaalsuse samasse aknasse kus lisamine ja kuvamine. Peale muudatusi võiks CompanyView välja näha järgnev (vt joonis 17).




Joonis 17. Company vaade peale muudatusi.

Lisasime Label'i Insert New Company, et eraldada lisamise plokki ning Label'i Update Selected Company, kus plokis muutmine ning kustutamine. Muutmiseks ja kustutamiseks eraldi nupud ning muutmiseks veel eraldi tekstiväli, kus kuvatakse nimekirjast valitud firma nime. Disainikood võiks näha välja järgnev (vt koodinäide 53).

```
<catel:UserControl x:Class="CompanyManager.Views.CompanyView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                                  xmlns:catel="http://catel.codeplex.com">

  <!-- Resources -->
  <UserControl.Resources>
  </UserControl.Resources>

  <!-- Content -->
  <catel:StackGrid>
        <catel:StackGrid.RowDefinitions>
       <RowDefinition Height="Auto" />
       <RowDefinition Height="Auto" />
        </catel:StackGrid.RowDefinitions>
    <catel:StackGrid.ColumnDefinitions>
      <ColumnDefinition Width="150"></ColumnDefinition>
      <ColumnDefinition Width="Auto"></ColumnDefinition>
    </catel:StackGrid.ColumnDefinitions>
    <Label Grid.ColumnSpan="2" Content="Companies List"></Label>
    <ListView MinHeight ="400"  ItemsSource="{Binding Companies}" SelectedItem="{Binding
SelectedCompany}" DisplayMemberPath="Name"></ListView>
    <catel:StackGrid>
      <catel:StackGrid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="100" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </catel:StackGrid.RowDefinitions>
      <catel:StackGrid.ColumnDefinitions>
        <ColumnDefinition Width="50"></ColumnDefinition>
        <ColumnDefinition Width="Auto"></ColumnDefinition>

      </catel:StackGrid.ColumnDefinitions>
      <Label Grid.ColumnSpan="2" Content="Insert New Company"></Label>
```

```
        <Label Content="Name"></Label>
        <TextBox Text="{Binding NewCompanyName}" Width="200"></TextBox>
        <Button Grid.ColumnSpan="2" Width="200" Height="20" VerticalAlignment="Top"
HorizontalAlignment="Right" Content="Insert" Command="{Binding
InsertCompany}"></Button>

        <Label Grid.ColumnSpan="2" Content="Update Selected Company"></Label>
        <Label Content="Name"></Label>
        <TextBox Text="{Binding SelectedCompany.Name}" Width="200"></TextBox>
        <Button Grid.ColumnSpan="2" Width="200" HorizontalAlignment="Right"
Content="Update" Command="{Binding UpdateCompany}"></Button>
        <Button Grid.ColumnSpan="2" Width="200" HorizontalAlignment="Right"
Content="Delete" Command="{Binding DeleteCompany}"></Button>
    </catel:StackGrid>

  </catel:StackGrid>
</catel:UserControl>
```

Koodinäide 53. CompanyView.xaml peale muudatusi.

Paneme tähele, et uus lisatud TextBox on otse seotud SelectedCompany.Name väärtusega, seega saab see tekstiväli automaatselt hetkel valitud firma nime. Näeme veel, et lisatud kahel nupul on Command seotud väärtustega UpdateCompany ja DeleteCompany. Lisame need meetodid viewmodelisse ning täidame ka sisu (vt Command'i lisamine viewmodelisse). Meetodite sisud oleks siis järgnevad (vt koodinäide 54).

```
/// <summary>
/// Method to invoke when the UpdateCompany command is executed.
/// </summary>
private void OnUpdateCompanyExecute()
{
  if (SelectedCompany != null)
  {
    var databaseService =
            Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
    databaseService.UpdateCompany(SelectedCompany);
  }

}
```

```
/// <summary>
/// Method to invoke when the DeleteCompany command is executed.
/// </summary>
private void OnDeleteCompanyExecute()
{
   if (SelectedCompany != null)
   {
      var databaseService =
                  Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
      databaseService.DeleteCompany(SelectedCompany.Id);
      Companies.Remove(SelectedCompany);
   }
}
```

Koodinäide 54. Company Update ja delete meetodid.

Mõlemal juhul kutsume välja vastava andmebaasi funktsiooni. Firma muutmisel ei ole vaja eraldi muudatust kuhugi mudelisse lisada, sest tekstiväli, mida muudame on juba seotud otse SelectedCompany'ga. Mõlemal juhul kontrollime kõigepealt, kas listist on ülde firma valitud.

Nüüd on meil kõik vajalik firmade lisamiseks, muutmiseks ja kustutamiseks olemas. Järgnevalt teeme kasutajate vaate PersonView.

# 11. Person vaate ja vaatemudeli lisamine

## 11.1. Person vaate lisamine

Lisame projektile uue vaate (vt Vaate lisamine). Nimeks paneme PersonView.xaml. Siin vaates võiks saada valida firma, näidata selle firma kasutajaid, lisada firmale kasutajaid ning muuta neid ja kustutada. Olgu eelduseks, et iga kasutaja saab olla seotud ainult 1 firmaga. Lisame ka viewmodeli PersonViewModel.cs (vt viewmodeli lisamine). Jätkame PersonView.xaml kujundamisega. Kõigepealt firma valik, võiks olla ComboBox (vt koodinäide 55).

```
<ComboBox ItemsSource="{Binding Companies}" SelectedItem="{Binding SelectedCompany}"
DisplayMemberPath="Name" Width ="250" HorizontalAlignment="Left"></ComboBox>
```

Koodinäide 55. Combobox'I lisamine.

Lugeja võib disaini ise paigutada, üks võimalus faili PersonView.xaml sisuks oleks selline (vt koodinäide 56).

```
<catel:UserControl x:Class="CompanyManager.Views.PersonView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                                  xmlns:catel="http://catel.codeplex.com">

  <!-- Resources -->
  <UserControl.Resources>
  </UserControl.Resources>

  <!-- Content -->
  <catel:StackGrid>
    <catel:StackGrid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
    </catel:StackGrid.RowDefinitions>
    <catel:StackGrid.ColumnDefinitions>
      <ColumnDefinition Width="150"></ColumnDefinition>
      <ColumnDefinition Width="Auto"></ColumnDefinition>
```

```
    </catel:StackGrid.ColumnDefinitions>
    <Label Content="Select Company"></Label>
    <ComboBox ItemsSource="{Binding Companies}" SelectedItem="{Binding SelectedCompany}" DisplayMemberPath="Name" Width ="250" HorizontalAlignment="Left"></ComboBox>
    <Label Grid.ColumnSpan="2" Content="Persons List"></Label>
    <ListView MinHeight ="400" ItemsSource="{Binding Persons}" SelectedItem="{Binding SelectedPerson}" DisplayMemberPath="FirstName"></ListView>
    <catel:StackGrid>
      <catel:StackGrid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />

        <RowDefinition Height="100" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />

        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </catel:StackGrid.RowDefinitions>
      <catel:StackGrid.ColumnDefinitions>
        <ColumnDefinition Width="150"></ColumnDefinition>
        <ColumnDefinition Width="Auto"></ColumnDefinition>

      </catel:StackGrid.ColumnDefinitions>
      <Label Grid.ColumnSpan="2" Content="Insert New Person"></Label>

      <Label Content="First Name"></Label>
      <TextBox Text="{Binding NewPersonFirstName}" Width="200"></TextBox>
      <Label Content="Last Name"></Label>
      <TextBox Text="{Binding NewPersonLastName}" Width="200"></TextBox>

      <Button Grid.ColumnSpan="2" Width="200" Height="20" VerticalAlignment="Top" HorizontalAlignment="Right" Content="Insert" Command="{Binding InsertPerson}"></Button>

      <Label Grid.ColumnSpan="2" Content="Update Selected Person"></Label>
      <Label Content="First Name"></Label>
      <TextBox Text="{Binding SelectedPerson.FirstName}" Width="200"></TextBox>
      <Label Content="Last Name"></Label>
      <TextBox Text="{Binding SelectedPerson.LastName}" Width="200"></TextBox>
```

```
        <Button        Grid.ColumnSpan="2"        Width="200"        HorizontalAlignment="Right"
Content="Update" Command="{Binding UpdatePerson}"></Button>
        <Button        Grid.ColumnSpan="2"        Width="200"        HorizontalAlignment="Right"
Content="Delete" Command="{Binding DeletePerson}"></Button>
     </catel:StackGrid>

   </catel:StackGrid>
</catel:UserControl>
```

Koodinäide 56. PersonView.xaml sisu.

Nagu näha on disain sarnane CompanyView disainiga. Lisatud on vaid ComboBox firma valikuks ning

lisatud väljad kasutaja andmete jaoks. Bindinguna on ära märigud järgmised mudelid: Companies, SelectedCompany, Persons, SelectedPerson, NewPersonFirstName, NewPersonLastName ja käsud (Command): InsertPerson, UpdatePerson, DeletePerson.

Need tuleks nüüd lisada PersonViewModel.cs'i. Nende lisamine jääb lugeja ülesandeks (vt command lisamine ja property lisamine). Samuti saab lugeja täita command'ide sisu Company view näite järgi. Põhiline erinevus on, et Comboboxi valiku muutudes peaks uuendama kasutajate nimekirja (andmebaasist paring CompanyId järgi) ning kasutaja lisamisel tuleks CompanyId (comboboxi valikust) salvestada kasutaja kirje külge andmebaasis.

Samuti tuleks konstruktorisse lisada Companies täitmine andmebaasist, SelectedCompany valimine (vaikimisi esimene firma), kindlasti enne seadistamist tuleks kontrollida, kas üldse leiti mingi firma,selle firma põhjal firma isikute (Persons) täitmine andmebaasist (samuti kontrollida, kas üldse leiti ning SelectedPerson valimine (vaikimisi esimene isik). Konstruktoris peaksid olema ka 3 commandi initsialiseerimine (vt CompanyViewModel.cs näitel). Konstruktori sisu võiks olla järgmine (vt koodinäide 57):

```
public PersonViewModel()
{
   var databaseService =
         Catel.IoC.ServiceLocator.Default.ResolveType<IDatabaseService>();
   Companies = new ObservableCollection<Company>(databaseService.GetCompanies());

   if(Companies != null && Companies.Count > 0)
      SelectedCompany = Companies[0];
```

```
   if (SelectedCompany != null)
   {
           Persons = new

ObservableCollection<Person>(databaseService.GetCompanyPersons(SelectedCompany.Id));


      if (Persons != null && Persons.Count > 0)
         SelectedPerson = Persons[0];
   }

   InsertPerson = new Command(OnInsertPersonExecute);
   UpdatePerson = new Command(OnUpdatePersonExecute);
   DeletePerson = new Command(OnDeletePersonExecute);

}
```

Koodinäide 57. PersonView konstruktor.

# 12. Navigeerimismenüü lisamine

Teeme valmis vasakpoolse menüü, kust saab edaspidi valida firmade ja isikute vaadete vahel. MainWindow.xaml'is lisasime alguses rea, mis lihtsalt näitas meile, et tegu on vasakpoolse regiooniga (vt koodinäide 58)

```
<Label Content="Left region" />
```

Koodinäide 58. Label Left region.

Selle võime nüüd asendada näiteks samuti ListView'ga kus on kaks elementi ja neist üks saab olla selekteeritud. Selles listviews võivad olla staatilised elemendid, näiteks string tüüpi. Lisame MainWindowViewModel'isse property vaadete valiku hoidmiseks. Kuna see nimekiri hakkab olema staatiline, siis ei ole vaja kasutada ObservableCollection'it, piisab List'ist (vt koodinäide 59).

```
[Model]
public List<string> Views
{
   get { return GetValue<List<string>>(ViewsProperty); }
   private set { SetValue(ViewsProperty, value); }
}

/// <summary>
/// Register the Views property so it is known in the class.
/// </summary>
public static readonly PropertyData ViewsProperty = RegisterProperty("Views",
typeof(List<string>));
```

Koodinäide 59. Views list lisamine.

Samuti lisame property valitud vaate hoidmiseks (vt koodinäide 60).

```
[Model]
public string SelectedView
{
   get { return GetValue<string>(SelectedViewProperty); }
   private set { SetValue(SelectedViewProperty, value); }
}
```

```
/// <summary>
/// Register the SelectedView property so it is known in the class.
/// </summary>
public static readonly PropertyData SelectedViewProperty =
        RegisterProperty("SelectedView", typeof(string));
```

Koodinäide 60. SelectedView property lisamine.

Konstruktorisse lisame vaadete väärtused ja paneme vaikevalikuks esimese (Companies) (vt koodinäide 61).

```
public MainWindowViewModel()
   : base()
{
   Views = new List<string>() { "Companies", "Persons"};
   SelectedView = Views[0];
}
```

Koodinäide 61. Views täitmine 2 vaatega ja esimese valimine.

Selleks, et menüüs view muutmisel nüüd ka view ise muutuks, tuleb SelectedView set meetodisse lisada funktsionaalsus vajaliku view kuvamiseks (vt koodinäide 62).

```
[Model]

public string SelectedView
{
   get { return GetValue<string>(SelectedViewProperty); }
   private set
   {
      SetValue(SelectedViewProperty, value);

      var visualizerService =
                  Catel.IoC.ServiceLocator.Default.ResolveType<IUIVisualizerService>();

      if(SelectedView == "Companies")
         visualizerService.Activate(new CompanyViewModel(), this, "RightRegion");
      else
      {
         visualizerService.Activate(new PersonViewModel(), this, "RightRegion");
      }
   }
}
```

Koodinäide 62. Vaate valiku kood.

Tegelikult võib nüüd Initialize meetodist ära kustutada vaikimisi company view kuvamise, sest see SelectedView Set kutsutakse konstruktoris nagunii välja ja esimese view'ga. Niiet kustutame need 2 rida (vt koodinäide 63).

```
var visualizerService =
        Catel.IoC.ServiceLocator.Default.ResolveType<IUIVisualizerService>();
visualizerService.Activate(new CompanyViewModel(), this, "RightRegion");
```

Koodinäide 63. Kustutame üleliigsed read.

MainWindow.xaml võiks välja näha nüüd järgnev (vt koodinäide 64).

```
<catel:DataWindow x:Class="CompanyManager.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                        xmlns:catel="http://catel.codeplex.com"
                        xmlns:regions="http://www.codeplex.com/CompositeWPF"
                        ShowInTaskbar="True"                ResizeMode="CanResize"
SizeToContent="Manual" WindowStartupLocation="Manual" WindowState="Maximized">

  <!-- Resources -->
  <catel:DataWindow.Resources>
  </catel:DataWindow.Resources>

  <!-- Content -->
   <catel:StackGrid x:Name="LayoutRoot">
            <catel:StackGrid.RowDefinitions>
                <RowDefinition Height="Auto" />
            </catel:StackGrid.RowDefinitions>
            <catel:StackGrid.ColumnDefinitions>
      <ColumnDefinition Width="100"></ColumnDefinition>
      <ColumnDefinition Width="Auto"></ColumnDefinition>
    </catel:StackGrid.ColumnDefinitions>
            <ListView      ItemsSource="{Binding      Views}"      SelectedItem="{Binding
SelectedView}"></ListView>
    <ContentControl                                                    Grid.Column="1"
regions:RegionManager.RegionName="RightRegion"></ContentControl>
  </catel:StackGrid>
</catel:DataWindow>
```

Koodinäide 64. MainWindows peale muudatusi.

Programm on nüüd valmis. Funtsionaalsusest on olemas vaate vahetamine, Company lisamine, muutmine, kustutamine. Samuti on olemas Person'i lisamine ja sidumine Company'ga, tema muutmine ja kustutamine. Muudatused salvestatakse andmebaasi.
Õppematerjalile on lisatud CD, kuhu on salvestatud programmi lähtekood, millega on võimalik oma valminud programmi võrrelda.

Kasutatud kirjandus

Geert van Horrik (2014). *Catel.Extensions.Prism* https://catelproject.atlassian.net/wiki/display/CTL/Catel.Extensions.Prism (12.12.2014).

Geert van Horrik (2014). *Catel.Extensions.WPf.Prism* https://github.com/Catel/Catel.Examples/tree/master/src/NET/Catel.Examples.WPF.Prism (12.12.2014)

Martin Fowler (2005). *InversionOfControl*. http://martinfowler.com/bliki/InversionOfControl.html (12.12.2014).