

# **TCS Windows Phone Terminal**

Tehtävienhallintasovellus Windows Phonelle

**Kalle Tanskanen**

Opinnäytetyö

---



Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Kalle Tanskanen	
Työn nimi TCS Windows Phone Terminal	
Päiväys 24.11.2012	Sivumäärä/Liitteet 38/0
Ohjaaja(t) lehtori Jussi Koistinen, tuotantopäällikkö Ville Pietikäinen	
Toimeksiantaja/Yhteistyökumppani(t) Ecomond Oy	
Tiivistelmä <p>Opinnäytetyön tavoitteena oli toteuttaa Ecomond Oy:lle tehtävienhallinnan päätelaitesovellus Windows Phone -alustalle ja siihen liittyvä palvelinrajapinta. Työn tarkoituksena oli myös kokeilla CSLA.NET-sovelluskehiksen käyttöä Windows Phone -sovelluksen toteuttamisessa. Järjestelmä koostuu Windows Phone -sovelluksesta, CSLA.NET:n avulla toteutetusta bisneslogiikkaluokkakirjastosta sekä palvelimella sijaitsevasta WCF-palvelusta ja tietokantakerroksesta.</p> <p>Työ toteutettiin käyttämällä Microsoftin Visual Studio 2010 -ohjelmistokehitysympäristöä ja C#-ohjelmointikieltä. Päätelaitesovellus toteutettiin käyttämällä MVVM-suunnittelumallia Windows Phone 7.1 -käyttöjärjestelmälle, palvelinrajapinta käyttää .NET 4.0 -ohjelmistokomponenttikirjastoa. Sekä sovellus että palvelimen tietokantakerros käyttävät CSLA.NET-sovelluskehystä.</p> <p>Työn tuloksena saatiin vaatimusmäärittelyn mukainen järjestelmä ja todettiin CSLA.NET:n soveltuvan hyvin Windows Phone -sovelluksen ja sen palvelinrajapinnan toteuttamiseen.</p>	
Avainsanat Windows Phone, CSLA.NET, MVVM	

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Kalle Tanskanen			
Title of Thesis TCS Windows Phone Terminal			
Date	24 November 2012	Pages/Appendices	38/0
Supervisor(s) Mr. Jussi Koistinen, Lecturer, Mr. Ville Pietikäinen, Production Manager			
Client Organisation /Partners Ecomond Oy			
<p>Abstract</p> <p>The primary goal of the thesis was to implement a task management terminal application for the Windows Phone platform and its server interface. The secondary goal was to test the suitability of the CSLA.NET framework for Windows Phone. The thesis was commissioned by Ecomond Oy.</p> <p>The system developed in this thesis consists of a Windows Phone application, a business logic class library made with CSLA.NET, WCF service and data access layer. It was programmed using Microsoft Visual Studio 2010 and the C# programming language. The terminal application was implemented using MVVM design pattern to Windows Phone 7.1 operating system. The server interface uses the .NET 4.0 framework. Both the terminal application and the data access layer use the CSLA.NET framework.</p> <p>The thesis resulted in a Windows Phone application and server interface that met the requirement specifications. Furthermore, it was confirmed that CSLA.NET is well suited for implementing this type of Windows Phone application.</p>			
Keywords Windows Phone, CSLA.NET, MVVM			

## ALKUSANAT

Tämä opinnäytetyö tehtiin Ecomond Oy:lle. Aloitin työn tekemisen keväällä 2012 ja sain sen valmiiksi talvella 2012. Kiitän työn ohjaajia, lehtori Jussi Koistista ja lehtori Sami Lahtea Savonia-ammattikorkeakoulusta. Lisäksi iso kiitos kuuluu Ecomond Oy:n tuotantopäällikkö Ville Pietikäiselle ja muulle yrityksen henkilökunnalle kaikesta työn aikana saamastani avusta ja tuesta.

Kuopiossa 24.11.2012

Kalle Tanskanen

## SISÄLTÖ

1	JOHDANTO .....	8
2	ECOMOND OY .....	9
3	KÄYTETYT TEKNIIKAT .....	10
3.1	Windows Phone 7 .....	10
3.2	Model-View-ViewModel (MVVM) .....	10
3.3	Basic Xaml Framework (BXF) .....	10
3.4	Windows Communication Foundation (WCF) .....	10
3.5	Language Integrated Query (LINQ) .....	11
4	CSLA:N RAKENNE.....	12
4.1	Bisnesluokat .....	13
4.1.1	BusinessBase .....	15
4.1.2	BusinessListBase .....	16
4.2	N-tasoinen kumoamistoiminto .....	16
4.3	Verkkorajapinta .....	17
4.4	Tietokantakerros.....	17
4.4.1	Uuden oliion luominen (Create) .....	18
4.4.2	Tietojen haku (Fetch).....	19
4.4.3	Tietojen lisääminen ja päivittäminen (Update).....	19
4.4.4	Olioiden poistaminen (Delete).....	19
4.4.5	Komennon suorittaminen (Execute).....	20
4.5	DataPortal .....	20
4.6	Verkkopalvelu ja käytettävä välityspalvelin .....	21
5	JÄRJESTELMÄN RAKENNE .....	22
5.1	Sovelluksen rakenne.....	23
6	KÄYTTÖLIITTYMÄ .....	26
6.1	Sisään kirjautuminen ja muut aloitusnäytöt .....	26
6.2	Tehtävälisan valinta .....	27
6.3	Tehtävälisan näyttö.....	27
6.4	Tehtävä .....	28
7	YHTEENVETO.....	30
	LÄHTEET .....	32

## TERMIT JA LYHENTEET

BISNESLUOKKA	CSLA:n avulla luotu bisneslogiikkaluokka
BISNESOLIO	CSLA:n avulla luodun bisneslogiikkaluokan ilmentymä
XAML	Extensible Application Markup Language
.NET	Microsoftin kehittämä sovelluskehys Windows-sovellusten tekemiseen
REFLECTION	.NET-sovelluskehysten tekniikka, jolla voidaan kutsua luokkia tai metodeja ajon aikana

## 1 JOHDANTO

Tässä opinnäytetyössä suunnitellaan ja toteutetaan Windows Phone 7 -laitteelle tehtävienhallintasovellus, joka on yhteydessä Ecomond Oy:n muihin järjestelmiin verkon yli. Sovelluksella kokeillaan myös CSLA.NET-sovelluskehiksen sopivuutta Windows Phone 7 -ohjelman toteuttamiseen. CSLA.NET-sovelluskehiksen käyttäminen opinnäytetyössä on Ecomond Oy:n vaatimus. CSLA.NET:n käytöllä pyritään nopeuttamaan ohjelmistojen kehittämistä ja helpottamaan niiden ylläpitoa. Opinnäytetyö keskittyy kuvaamaan, mikä on CSLA.NET ja miten sen avulla rakennetaan asiakassovelluksen bisneslogiikka ja palvelimen tietokantakerros.

Opinnäytetyön aihe sai alkunsa Ecomond Oy:n tarpeesta saada päätelaitesovellus Symbian- ja Windows Mobile 6.5 -sovellusten rinnalle. Windows Phone 7 -alustaan päädyttiin, koska sen oletettiin muita alustoja helpommin integroituvan Microsoftin muihin ohjelmistoihin.

Opinnäytetyössä tehtyä Windows Phone -sovellusta käytetään tehtävienhallintaan. Ajatuksena on, että käyttäjälle asetetaan palvelimella tehtävälista. Käyttäjä kirjautuu sisään Windows Phone -sovellukseen, lataa itselleen tehtävälistan palvelimelta ja suorittaa listalla olevia tehtäviä. Hän tekee tehtäviin muutoksia ja merkitsee ne valmiiksi. Valmis tehtävälista tallennetaan sen jälkeen takaisin palvelimelle. Tehtävälistan lisäksi sovelluksella voidaan näyttää käyttäjälle erilaisia viestejä ja ohjeita. Sovellus tallentaa tehtävälisat puhelimeen, joten verkkoyhteyttä ei tarvita tehtävälistan lataamisen jälkeen.

Opinnäytetyöraportti koostuu seuraavista osista: Ecomond Oy:n esittely, käytettyjen tekniikoiden esittely, CSLA.NET-sovelluskehiksen rakenne, varsinaisen sovelluksen ja palvelinrajapinnan kuvaus ja yhteenveto.



## 2 ECOMOND OY

Ecomond Oy on vuonna 2002 perustettu kuopiolainen ohjelmistoyritys. Yrityksen pääosaamisalue on toiminnanohjausjärjestelmät logistiikan alalle.

Ecomondin tärkeimmät tuotteet ovat TCS-Transport Control System ja TCS-Opti. TCS on logistiikkajärjestelmä, johon sisältyy ohjelmistot ajoneuvoihin, toimistoon ja palvelimille. TCS-Opti on työkalu, jolla voidaan optimoida kuljetusreittejä ja -aikatauluja. Lisäksi sillä voidaan simuloida erilaisia reittivaihtoehtoja. (Ecomond.)

Vuonna 2011 Ecomond Oy:ssä työskenteli 15 henkilöä ja sen liikevaihto oli 817 000 €.

### 3 KÄYTETYT TEKNIIKAT

Tässä luvussa esitellään lyhyesti opinnäytetyössä käytettyjä tekniikoita ja ohjelmistokirjastoja. CSLA.NET esitellään omana kokonaisuutenaan luvussa 4.

#### 3.1 Windows Phone 7

Windows Phone 7 on Microsoftin kehittämä puhelinkäyttöjärjestelmä, joka korvaa aiemmin käytössä olleen Windows Mobile -käyttöjärjestelmän. Toisin kuin edeltäjänsä se on suunnattu enemmän kuluttajamarkkinoille. Windows Phone 7:lle voidaan tehdä sovelluksia Visual Basic- ja C# -kielillä. (Lecrenski, Watson, Fonseca-Ensor 2011.)

#### 3.2 Model-View-ViewModel (MVVM)

Model-View-ViewModel on Microsoftin kehittämä suunnittelumalli. Sen tarkoituksena on erottaa näyttö (View) ja malli (Model) toisistaan omiksi loogisiksi kokonaisuuksikseen. Näiden välissä on ViewModel, joka lähinnä muuttaa mallin tiedot näytettävään muotoon ja ohjaa käyttäjän tekemät toimet mallille. (Lecrenski ym. 2011.)

#### 3.3 Basic Xaml Framework (BXF)

Basic Xaml Framework on CSLA.NET:n kehittäjän Rocford Lhotkan kehittämä MVVM-kehys, jonka tarkoitus on helpottaa MVVM-suunnittelumallin käyttöä. Se on varsin minimalistinen toteutus, joka toteuttaa vain ehdottoman tarpeelliset toiminnallisuudet. BXF:n tukemia tekniikoita Windows Phone 7:n lisäksi ovat WPF ja Silverlight. (CodePlex.)

#### 3.4 Windows Communication Foundation (WCF)

Windows Communication Foundation kuuluu Microsoftin .NET-kirjastoon. Sen avulla voidaan tehdä palvelukeskeisen arkkitehtuurin (SOA) verkkopalvelu, johon voi ottaa myös asynkronisia yhteyksiä. (Microsoft 2012.)

### 3.5 Language Integrated Query (LINQ)

Language Integrated Query on osa Microsoftin .NET-kirjastoa. Sen avulla voidaan suorittaa tietokantatoimintoja ilman SQL-kieltä. Näin kaikki tietokantatoiminnot voidaan lisätä koodina luokkiin. (Microsoft 2007.)

#### 4 CSLA:N RAKENNE

CSLA.NET (Component-based Scalable Logical Architecture) on amerikkalaisen Rocford Lhotkan kehittämä avoimen lähdekoodin sovelluskehys pääasiassa Microsoft Windows -käyttöjärjestelmälle. Sen kehitys on aloitettu vuonna 1999 ja nykyinen versionumero on 4.3. CSLA:n tukemia tekniikoita ovat nykyisin Visual Studio 2010, Microsoft .NET 4, Silverlight 4 ja 5 sekä Windows Phone 7.5. Seuraava CSLA:n versio 4.5 tukee myös Windows 8:aa ja Windows Phone 8:aa. Tästä eteenpäin CSLA.NET-sovelluskehuksesta käytetään nimitystä CSLA. Tämä luku pohjautuu lähteeseen Expert C# 2008 Business Objects. (Lhotka 2009.)

CSLA on monikerrosarkkitehtuuri, joka helpottaa erityisesti verkon yli tietoa siirtävien sovelluksien toteuttamista. CSLA:lla tehdyt sovellukset ovat hyvin rakenteellisia ja kaikki arkkitehtuurin kerrokset ovat jaettavissa helposti eri projekteiksi. CSLA:n avulla toteutettua luokkakirjastoa voidaan käyttää helposti muilla tuetuilla alustoilla ja muissa sovelluksissa.

CSLA:n toiminta perustuu *bisnesolioihin*, jotka näyttävät siirtyvän verkon yli sellaisenaan. Tämä on mahdollista siitä syystä, että sama bisnesluokkakirjasto on saatavilla sekä asiakas- että palvelinpäässä. Siirrettäessä bisnesoliota verkon yli CSLA serialisoi olion palvelimen päässä ja lukee sen asiakaspäässä samantyyppiseen olioon. Näin sama olio näyttää siirtyvän verkon yli.

Bisnesolioilla on määreitä, joiden perusteella olio tietää, onko sen sisältämä tieto muuttunut. Osaa näistä määreistä hallitsee CSLA, muut täytyy asettaa käsin. Nämä määreet kertovat, onko olio uusi (IsNew) ja onko olion sisältämä tieto muuttunut viimeisimmän tallennuksen jälkeen (IsDirty). Näiden tietojen perusteella voidaan päätellä tietokantakerroksessa, miten olioita täytyy käsitellä. Lisäksi CSLA:ssa on tekniikka, jolla voidaan määrittellä jokaiseen bisnesolioon liittyviä sääntöjä ja tietojen oikeellisuustarkistuksia. CSLA:han on myös sisään rakennettu n-tasoinen kumoamistoiminto.

CSLA:n avulla tehdyssä sovelluksessa on aina vähintään käyttöliittymä, bisnesluokkakirjasto ja tietokantakerros. Yleensä itse käyttöliittymä ja tietokantakerros sijaitsevat eri laitteissa, ja näiden välissä olevaa verkkoyhteyttä hallinnoi DataPortal-niminen osa, jonka avulla tiedot välitetään verkon yli. CSLA:n perusajatuksiin kuuluu myös,

että bisnesluokat ovat saatavilla kaikilla sovelluksen logiikkatasoilla. Alla olevassa kuvassa (kuva 1) on esitetty tämä rakenne.



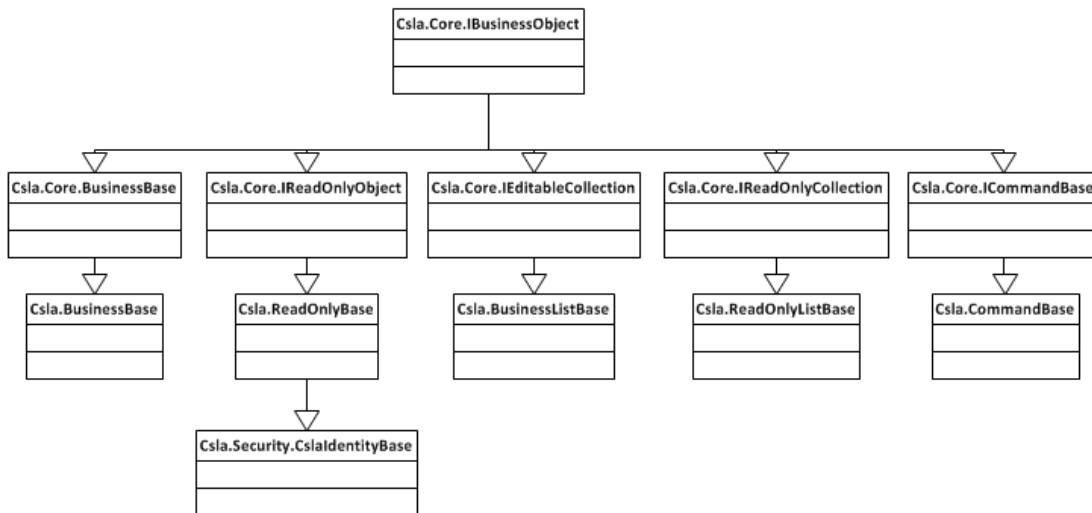
KUVA 1. CSLA:n avulla toteutetun sovelluksen rakenne

Seuraavissa luvuissa kuvataan tarkemmin CSLA:n verkkorajapinnan, bisnesluokkien, DataPortal:n ja tietokantakerroksen toimintaa.

#### 4.1 Bisnesluokat

Bisnesluokkakirjasto koostuu yksittäisistä bisnesluokista, jotka tietosisällöltään vastaavat käyttöliittymän näyttöjä. Näin ollen bisnesluokkakirjastoon ei edes yritetä tehdä yleisluontoisia luokkia, vaan luokkia jotka toimivat pääasiassa yhden näytön kanssa. Luokat ovat yleensä myös sisäkkäisiä siten, että niillä on lapsiluokkia ja lapsiluokilla edelleen uusia lapsiluokkia. Tällä tavoin voi muodostua pitkiäkin luokkaketjuja. CSLA:ssa luokkaketjun kaikki tietokantatoiminnot tapahtuvat juuriluokan kautta.

Bisnesluokat perivät aina jonkin CSLA:n kantaluokista. Kaikkien luokkien perustoiminnot on määriteltävä IBusinessObject-rajapinnassa alla olevan kuvion (kuvio 2) mukaisesti. CSLA sisältää muitakin kantaluokkia, joista tässä on kuitenkin esitelty vain yleisimmät käytössä olevat.



KUVIO 2. CSLA:n yleisimmät kantaluokat

CSLA:ssa luokat jaetaan tallennettavaa ja vain luettavaa tietoa sisältäviin luokkiin. Näille molemmille on olemassa erilliset kantaluokat. Lisäksi CSLA:ssa on erilliset kantaluokat kokoelmatyypisille ja yksittäisille luokille. ReadOnlyBase:a ja ReadOnlyListBase:a käytetään bisnesluokille, joissa on vain luettavaa tietoa. Näillä luokilla ei ole ollenkaan tiedon tallennustoimintoja. Vastaavasti BusinessBasea ja BusinessListBasea käytetään luokille, joiden tietoihin voidaan tehdä muutoksia ja joita pitää voida tallentaa.

CommandBase:a käytetään luomaan komentotyyppinen bisnesolio, jolla voidaan suorittaa jokin komento palvelimella. Tämän tyyppisiä olioita voidaan käyttää vain yhdessä bisnesolion Execute-metodin kanssa.

CslIdentityBase:n avulla luodaan Identity-luokka. Identity-luokkaa käytetään käyttäjän tunnistamiseen CSLA:n avulla tehdyssä järjestelmässä. Identity-olio kulkee jokaisessa tietokantakerrospyynnössä mukana, joten sen avulla voidaan tarkistaa käyttäjän oikeudet jokaisella kerralla.

Lisäksi CSLA:ssa on CriteriaBase-niminen kantaluokka, jota käyttämällä luodaan Criteria-tyyppisiä olioita. Criteria-olioita käytetään, kun halutaan antaa metodille enemmän kuin yksi parametri. Criteria-luokat ovat yleensä bisnesluokan sisäluokkia, koska niiden sisältö on riippuvainen bisnesluokasta.

Seuraavissa luvuissa esitellään BusinessBasen ja BusinessListBasen periviä bisnesluokkia esimerkkien avulla. Muuntyyppiset bisnesluokat ovat samankaltaisia,

joten niitä ei erikseen esitellä. Nämä esimerkit eivät ole itse opinnäytetyöstä vaan tehty esittelemään bisnesluokkien rakennetta.

#### 4.1.1 BusinessBase

BusinessBase-kantaluokkaa käytetään luomaan yksittäinen bisnesluokka, joka voi olla juuriluokka tai toisen luokan lapsi. Bisnesluokille määritellään kaikki kyseisellä näytöllä tarvittavat jäsenmuuttujat. Lisäksi sille voidaan määritellä monia erilaisia sääntöjä ja tietojen oikeellisuustarkistuksia (business and validation rules). Esimerkiksi onko syötetty tieto oikean pituinen tai salliiiko käyttäjän rooli tietojen muutoksen.

```
namespace BusinessLibrary
{
    [Serializable]
    public class Order : BusinessBase<Order>
    {
        public static PropertyInfo<int> IdProperty = RegisterProperty<int>(p => p.Id);
        public int Id
        {
            get { return GetProperty(IdProperty); }
            private set { SetProperty(IdProperty, value); }
        }

        public static PropertyInfo<string> LastNameProperty = RegisterProperty<string>(c => c.LastName);
        [Required]
        public string LastName
        {
            get { return GetProperty(LastNameProperty); }
            set { SetProperty(LastNameProperty, value); }
        }

        public static PropertyInfo<string> FirstNameProperty = RegisterProperty<string>(p => p.FirstName);
        public string FirstName
        {
            get { return GetProperty(FirstNameProperty); }
            set { SetProperty(FirstNameProperty, value); }
        }
    }
}
```

#### KUVA 3. Esimerkki bisnesluokan lähdekoodista

Yllä olevassa kuvassa (kuva 3) on esimerkki lähdekoodista, jossa on BusinessBase-luokasta peritty Order-niminen bisnesluokka, jolla on Id-, LastName- ja FirstName-nimiset ominaisuudet. Esimerkkinä yksinkertaisesta oikeellisuustarkistuksesta, LastName on määrätty pakolliseksi tiedoksi Required-määreellä.

Jäsenmuuttujien arvot asetetaan SetProperty- tai LoadProperty-metodeilla. SetProperty-metodia kutsumalla CSLA suorittaa muuttujille määritellyt säännöt ja oikeellisuustarkistukset. Lisäksi käyttämällä SetProperty-metodia CSLA asettaa olion tilan muuttuneeksi eli asettaa IsDirty-muuttujan tilaksi tosi. LoadProperty-metodilla void-

aan asettaa arvoja ilman oikeellisuustarkistuksien suorittamista. GetProperty-metodia käytetään tietojen lukemiseen muuttujista.

#### 4.1.2 BusinessListBase

```
namespace BusinessLibrary
{
    [Serializable]
    [Cslla.Server.ObjectFactory("DataAccess.OrderFactory, DataAccess", "FetchList")]
    public class OrderList : BusinessListBase<OrderList, Order>
    {
        public static void GetList(EventHandler<DataPortalResult<OrderList>> callback)
        {
            DataPortal.BeginFetch<OrderList>(callback);
        }
    }
}
```

KUVA 4. Esimerkki kokoelmatyyppisen bisnesluokan lähdekoodista

BusinessListBase-kantaluokkaa käytetään kokoelmatyyppisen bisnesluokan luomisessa. Sillä ei ole ollenkaan omia jäsenmuuttujia vaan se sisältää kantaluokan tyyppiä määriteltyjä bisnesolioita. Yllä olevassa kuvassa (kuva 4) on esimerkki BusinessListBase:n perivästä luokasta, jossa on OrderList-niminen lista, joka sisältää Order-nimisiä olioita. Lisäksi luokkaan on määritelty asynkroninen GetList-niminen metodi, joka suorittaa tietokantakerroksen. Valmistumisestaan se ilmoittaa callback-nimiselle tapahtumankäsittelijälle.

Tietokantakerroksen metodien nimet annetaan bisnesluokan ObjectFactory-määreeseen tekstinä yllä olevan kuvan mukaisesti (kuva 4). ObjectFactory-määreeseen annetaan luokan, nimiavaruuden ja metodin nimet. Näitä määreitä käyttämällä palvelimen tietokantakerros osaa kutsua oikean ObjectFactory-luokan oikeaa metodia. GetList-metodi kutsuu BeginFetch-metodia. BeginFetch kutsuu ObjectFactory-määreessä asetettua FetchList-metodia, joka sijaitsee DataAccess-nimiavaruudessa ja OrderFactory-luokassa tietokantakerroksella.

#### 4.2 N-tasoinen kumoamistoiminto

CSLA-sovelluskehikseen on sisään rakennettu n-tasoinen kumoamistoiminto. (n-level undo) Kumoamistoiminto on käytössä vain BusinessBase- tai BusinessListBase-luokan perivillä luokilla. Kumoamistoimintoon kuuluu kolme metodia: BeginEdit, ApplyEdit ja CancelEdit. BeginEdit-metodia kutsutaan, kun halutaan aloittaa kumoamis-



toiminnon käyttäminen. CSLA tallentaa BeginEdit-metodin kutsumisen jälkeen kaikki muuttujiin tehtävät muutokset erilliseen listaan ja pitää alkuperäisen tiedon ennallaan. Jos sen jälkeen kutsutaan ApplyEdit-metodia, CSLA tallentaa listan viimeisen arvon muuttujan arvoksi. CancelEdit-metodia kutsumalla CSLA pitää muuttujan arvon ennallaan ja tyhjentää muuttuneiden tietojen listan. Bisnesoliota ei voi tallentaa ennen kuin joko ApplyEdit- tai CancelEdit-metodia on kutsuttu.

### 4.3 Verkkorajapinta

CSLA:n DataPortal:n verkkorajapinta on hyvin yksinkertainen. Siinä on viisi metodia: Create (uuden olion luominen), Fetch (olion haku), Update (olion päivitys), Delete (olion poisto) ja Execute (komennon suoritus). Bisnesluokkien kantaluokissa on määriteltä edellä mainitut metodit ja niitä voi tarpeen mukaan ylikirjoittaa. Lisäksi jokaisessa bisnesluokassa, joka suorittaa tietokantatoimintoja, on määritetty mitä metodia DataPortal kutsuu, kun kutsutaan tietokantakerrosta.

Verkkorajapinnan metodimäärytykset voidaan antaa kahdella eri tavalla. Aiempi tapa oli kirjoittaa metodit suoraan bisnesluokkaan, jolloin DataPortal luo tietokantapalvelimella kyseisen bisnesolion ja kutsuu sen metodia. Huonona puolena tässä tavassa on se, että tietokerroksen logiikka sotkeentuu bisneslogiikan kanssa ja kerrosarkkitehtuurin ylläpitäminen on haastavampaa.

Uudempi tapa käyttää ObjectFactory:a, jolloin bisnesluokkaan määritetään ObjectFactory-attribuutti ja sille annetaan tekstinä haluttu tiedoston, luokan ja metodin nimi. Tällöin DataPortal kutsuu palvelimen päässä *reflectionia* hyväksi käyttäen oikeaa metodia. Tämän tekniikan hyötynä on pitää bisnesluokat ja tietokantakerros loogisesti erillään toisistaan. Huonona puolena on reflectionin käytöstä koitua ongelma, jossa virheet tulevat esiin vasta ajon aikana, sillä kääntäjä ei osaa tarkistaa reflection-kutsujen oikeellisuutta. Tästä huolimatta opinnäytetyössä päädyttiin käyttämään ObjectFactory-tekniikkaa.

### 4.4 Tietokantakerros

Tietokantakerros koostuu ObjectFactory-luokan perivistä luokista. Näihin luokkiin määritellään rajapinnan Create-, Fetch-, Update-, Delete- ja Execute-metodit tarpeen mukaan. Lähdekoodiesimerkissä (kuva 5) on nämä metodit lukuun ottamatta Execute-metodia. Mikäli bisnesoliossa on vain luettavaa tietoa, ei ole tarpeellista määri-

tellä kuin Fetch-metodi. Edellä mainittujen metodien sisään määritellään varsinaiset tiedon haku- ja tallennustoiminnot haluttua tekniikkaa käyttäen. CSLA ei ota mitään kantaa siihen, mitä tiedon tallennustekniikkaa sovelluksessa käytetään. Seuraavissa luvuissa käydään läpi edellä mainitut toiminnot asiakassovelluksen näkökulmasta.

```
namespace DataAccess
{
    public class OrderFactory : ObjectFactory
    {
        //Uuden oliion luominen
        public Order Create()
        {
            var obj = (Order)MethodCaller.CreateInstance(typeof(Order));
            //luetaan tiedot jotain tallennustekniikkaa käyttäen
            return obj;
        }

        //Tietojen haku
        public OrderList FetchList()
        {
            var obj = (OrderList)MethodCaller.CreateInstance(typeof(OrderList));
            //luetaan tiedot jotain tallennustekniikkaa käyttäen
            return obj;
        }

        //Tietojen lisääminen ja päivittäminen
        public OrderList Update(OrderList obj)
        {
            foreach (Order o in obj)
            {
                if (o.IsDirty)
                {
                    if (o.IsNew)
                    {
                        //lisätään uusi tieto
                    }
                    else
                    {
                        //päivitetään olemassa olevaa tietoa
                    }
                }
            }

            return obj;
        }

        //Olioiden poistaminen
        public void Delete(Order obj)
        {
            //poistetaan jotain tallennustekniikkaa käyttäen
        }
    }
}
```

KUVA 5. Esimerkki ObjectFactory-luokasta

#### 4.4.1 Uuden oliion luominen (Create)

Bisnesolio voidaan luoda kahdella tavalla riippuen siitä, tarvitaanko uudelle oliolle arvoja tietokantakerrokselta. Jos uuteen olioon tarvitaan tietoja tietokantakerrokselta, kutsutaan bisnesolion Create-metodia. Create-metodi palauttaa tietokantakerrokselta halutun tyyppisen uuden oliion. Jos uuteen olioon ei tarvita tietoja tietokantakerrokselta, se voidaan lisätä myös paikallisesti ilman tietokantakerroksua.

#### 4.4.2 Tietojen haku (Fetch)

Bisnesolioon haetaan tietoja Fetch-toiminnolla. Fetch-metodilla on korkeintaan yksi parametri. Jos parametreja haluttaisiin antaa enemmän, ne kapseloidaan Criteria-tyyppiseen olioon, joka annetaan Fetch-metodille. Fetch palauttaa tietokantakerrokselta halutun tyyppisen bisnesolion, jossa on parametrien mukainen tieto.

#### 4.4.3 Tietojen lisääminen ja päivittäminen (Update)

CSLA:ssa tietoja lisätään ja päivitetään asiakassovelluksen kannalta samalla tavalla. Päätös siitä, lisätäänkö uusi tieto vai päivitetäänkö vanhaa tietoa, tehdään tietokantakerroksella. Tietojen lisääminen tai päivittäminen aloitetaan kutsumalla bisnesolion Save-metodia, joka kutsuu edelleen Update-metodia. Update-metodi ottaa kopion kyseisestä bisnesoliosta ja lähettää sen tietokantakerrokselle. Tietokantakerroksella päätetään olion IsDirty-määreen avulla, onko tieto muuttunut viimeisimmän tallennuksen jälkeen. IsNew-määreen perusteella päätetään, tallennetaanko tieto uutena vai päivitetäänkö vanhaa. Jos IsNew on tosi, tieto tallennetaan uutena tietona, ja jos epätosi, päivitetään olemassa olevaa. Tietojen tallennuksen jälkeen olio palautetaan takaisin asiakaspäähän, joten tallennuksessa voidaan tehdä muutoksia olioon ja muutokset siirtyvät asiakassovellukseen.

#### 4.4.4 Olioiden poistaminen (Delete)

Olioita voidaan poistaa kahdella tavalla sen mukaan, halutaanko ne poistaa välittömästi vai vasta seuraavan tietojen tallentamisen yhteydessä. Jos olio halutaan poistaa heti, kutsutaan bisnesolion Delete-metodia. Delete-metodi kutsuu välittömästi halutun olion Delete-metodia tietokantakerroksessa.

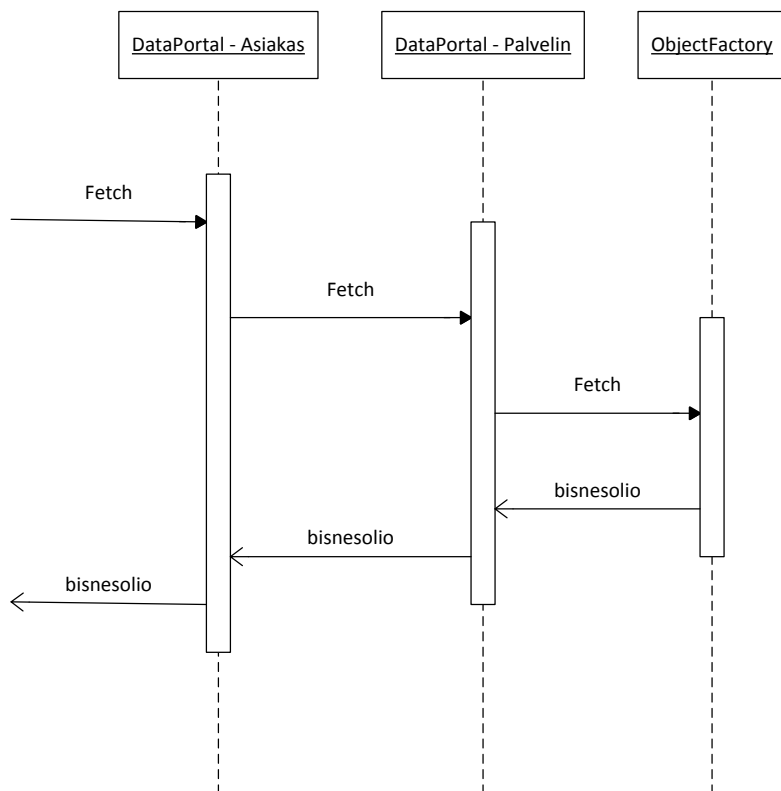
Olio voidaan poistaa myös käyttämällä kokoelmatyyppisen bisnesolion Remove-metodia, jolloin poistettava olio siirtyy varsinaisesta kokoelmasta Deleted-listaan. Tällä tavalla olio näyttää poistuvan kokoelmasta, mutta sen poisto käsitellään tietokantakerroksella vasta seuraavan tiedon päivittämisen yhteydessä.

#### 4.4.5 Komennon suorittaminen (Execute)

Execute-kutsulla voidaan suorittaa tietokantakerroksella toimintoja, jotka eivät liity edellä mainittuihin tehtäviin. Execute-komennolla suoritettavat toiminnot rakennetaan aina Command-tyyppisen bisnesolion sisään.

#### 4.5 DataPortal

DataPortal on CSLA:n osa, joka piilottaa verkkoyhteyden eri laitteilla sijaitsevien ohjelmiston osien väliltä. DataPortal on staattinen luokka ja sen metodeja kutsutaan aina, kun halutaan kutsua jotain tietokantakerroksen metodia. Kutsulle annetaan parametriksi bisnesolion tyyppi.



KUVIO 6. CSLA DataPortal kutsu

Yllä olevassa kuviossa (kuvio 6) on esitetty tiedon haku CSLA:n avulla palvelimelta. Kun jokin luokka haluaa tietoa palvelimelta, se kutsuu DataPortalin geneeristä Fetch-metodia. Fetch-metodille annetaan parametrina hakukriteeri. DataPortal kutsuu edelleen verkkoasetuksissa määriteltyä palvelinta ja siellä olevaa DataPortalia. Palvelinpään DataPortal kutsuu bisnesluokassa määritettyä ObjectFactoryn perivän luokan

metodia. Metodi luo uuden bisnesolion parametrina tulleen tyypin mukaan ja lukee siihen halutut tiedot. Olio palautetaan takaisin DataPortalille ja sieltä verkkoyhteyden yli takaisin alkuperäiselle kutsujalle.

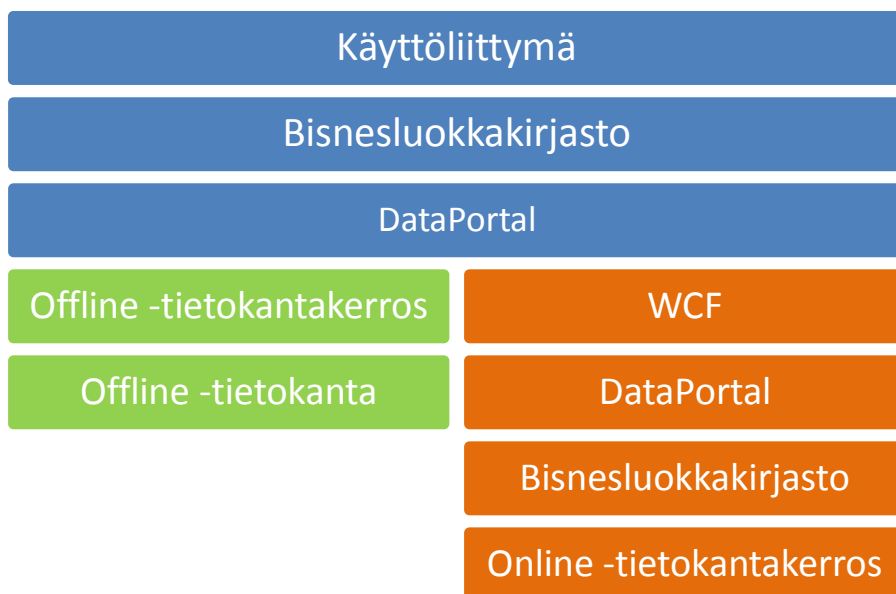
#### 4.6 Verkkopalvelu ja käytettävä välityspalvelin

CSLA tukee yleisimpiä verkkopalveluja kuten webservicea ja WCF:ää. Näille verkkopalveluille on CSLA:ssa valmiina välityspalvelinluokat, jotka määrittävät käytettävän palvelun asiakassovelluksessa. Jos halutaan käyttää jotain harvinaisempaa verkkopalvelua, voidaan välityspalvelin määritellä myös itse. Oletuksena CSLA käyttää WCFProxya. Jos halutaan käyttää useampaa tietokantakerrosta ja useampaa verkkopalvelua, täytyy ylikirjoittaa ProxyFactory-luokka, joka määrää, mitä välityspalvelinta käytetään.

## 5 JÄRJESTELMÄN RAKENNE

Järjestelmä koostuu Windows Phone 7 -sovelluksesta, puhelimessa olevasta tietokantakerroksesta ja tietokannasta sekä palvelimella olevista WCF-palvelusta ja tietokantakerroksesta. Varsinainen palvelimen tietokanta ei kuulu tämän työn piiriin.

Puhelimen tietokantaa käytetään tiedon välivarastona ja sinne tallennetaan kaikki tiedot jotka ladataan palvelimelta. Sovellus tallentaa myös kaikki tietojen muutokset sekä paikalliseen että palvelimen tietokantaan. Näin sovellusta voidaan käyttää vaikka verkkoyhteys palvelimeen menetetään. Sovellus on pyritty tekemään siten, että käyttäjä ei välttämättä edes huomaa verkkoyhteyden katkeamista.

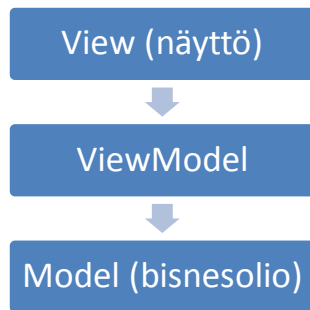


KUVA 7. Järjestelmän rakenne

Yläpuolella olevassa kuvassa (kuva 7) on kuvattu järjestelmä loogisina kerroksina. Sinisellä värillä olevat kerrokset kuvaavat Windows Phone -sovellusta. Vihreällä on merkitty puhelimen paikallinen tiedon tallennuskerros ja oranssilla palvelimella oleva tiedon tallennuskerros. Sekä puhelimessa että palvelimella oleva Bisnesluokkakirjasto on itse asiassa sama kirjasto, joka on vain käännetty eri alustalle. Seuraavissa kappaleissa käydään tarkemmin läpi järjestelmän osia. Windows Phone -sovelluksen käyttöliittymä on esitelty luvussa 6.

## 5.1 Sovelluksen rakenne

Windows Phone -sovellus on toteutettu käyttämällä MVVM-suunnittelumallia, jossa yksittäinen näyttö koostuu näytöstä (View), mallista (Model) ja niiden väliin asettuvasta viewmodelista, joka muuntaa tietoa edellä mainittujen välillä. CSLA:n avulla toteutetussa sovelluksessa mallina toimii bisnesolio. MVVM-suunnittelumallissa käytetään DataBinding-tekniikkaa, jolla käyttöliittymän komponentit sidotaan joko suoraan bisnesolioon tai viewmodeliin, joka on vastaavasti yhteydessä bisnesolioon alla olevan kuvan mukaisesti (kuva 8). Viewmodelia käytetään pääasiassa muuntamaan bisnesolion sisältämä tieto näytettävään muotoon.



KUVA 8. Model-View-ViewModel

Windows Phone -sovelluksissa näytöt määritellään XAML-kielellä. XAML-tiedostoihin liittyy myös .cs-päätteinen taustakooditiedosto (code-behind), mutta MVVM-mallissa tämä pyritään jättämään tyhjäksi, sillä kaiken bisneslogiikan on tarkoitus sijaita Model:ssa eli bisnesoliossa.

Kuvassa 9 on esimerkki XAML-tiedostosta, jossa on määritelty ListBox-komponentti. ListBox:n rivien lähteenä (ItemsSource) toimii kokoelmatyyppinen bisnesolio ja valitun rivin lähteenä viewmodelissa sijaitseva SelectedItem-muuttuja. SelectedItem on määritelty kaksisuuntaiseksi, jolloin myös käyttäjän valinta välittyy viewmodelille.

Kuvan (kuva 9) lähdekoodissa näkyvät myös usein käytetty ValueConverter, jolla voidaan helposti muuntaa muuttujan arvo näyttökomponentille sopivaan muotoon. ValueConverter:lla asetetaan tekstin väri IsRead-muuttujan arvon mukaan. Seuraavassa kuvassa (kuva 10) on esitelty edellä mainitun ValueConverter:n lähdekoodi.

```

<Grid
  x:Name="ContentPanel"
  Grid.Row="1"
  Margin="20,0,12,0">
  <ListBox
    ItemsSource="{Binding Path=Model}"
    SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay}">
    <ListBox.ItemTemplate>
      <DataTemplate>
        <StackPanel>
          <TextBlock
            x:Name="item"
            Text="{Binding Path=Subject}"
            Style="{StaticResource PhoneTextLargeStyle}"
            Foreground="{Binding Path=IsRead, Converter={StaticResource BoolColorConverter}}"
            TextWrapping="Wrap" />
        </StackPanel>
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>
</Grid>

```

### KUVA 9. Esimerkki XAML-lähdekoodista

```

public class BoolColorConverter : IValueConverter
{
  public bool Invert { get; set; }

  public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
  {
    var v = (bool)value;

    if (v)
      return (Brush)App.Current.Resources["PhoneDisabledBrush"];
    else
      return (Brush)App.Current.Resources["PhoneForegroundBrush"];
  }

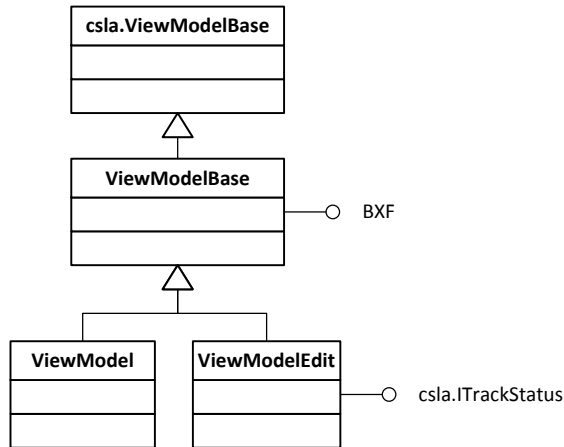
  public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
  {
    return value;
  }
}

```

### KUVA 10. BoolColorConverter

CSLA:ssa on ViewModelBase-kantaluokka, jossa on valmiina toimintoja liittyen esimerkiksi tietojen hakuun ja tallennukseen. Sovelluksessa käytetään myös BXF-nimistä MVVM-kehystä, joka helpottaa siirtymistä näyttöjen välillä ja dialogien näyttämistä. CSLA:n ViewModelBase-luokka huolehtii myös tarvittavan bisnesolion luomisesta.





KUVA 11. ViewModel-hierarkia

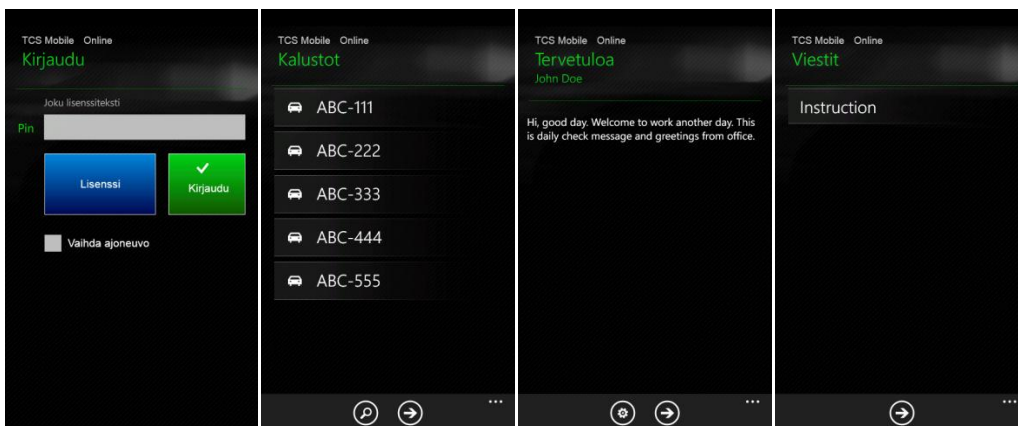
Sovellukseen tehtiin oma `ViewModelBase`-kantaluokka, joka perii edellä mainitun CSLA:n `ViewModelBase` ja toteuttaa `BXF`-rajapinnan (kuva 11). Tällä tavalla näyttöjen yhteiset toiminnot on saatu kerättyä yhteen luokkaan. Tästä `ViewModelBase`sta on peritty edelleen `ViewModelEdit`-luokka tallennettaville ja `ViewModel`-luokka vain luettaville bisnesolioille. Näiden ero on se, että `ViewModelEdit`-luokka toteuttaa CSLA:n `ITrackStatus`-rajapinnan, jolloin myös `viewmodel` on tietoinen bisnesolion tilan muutoksista. Lisäksi `ViewModelEdit`-luokkaan on määritelty `Save`-, `OnSaving`- ja `OnSaved`-metodit helpottamaan tallentamista.

## 6 KÄYTTÖLIITTYMÄ

Tässä luvussa esitellään Windows Phone -sovelluksen käyttöliittymän näytöt ja toiminnot. Käyttöliittymä on pyritty tekemään siten, että siinä on mahdollisimman vähän valintoja ja että se olisi helppokäyttöinen. Käyttöliittymä on lokalisoitu suomeksi, ruotsiksi ja englanniksi. Sovelluksen kieli vaihtuu puhelimen kieliasetuksen mukaan.

### 6.1 Sisään kirjautuminen ja muut aloitusnäytöt

Sovelluksen käynnistyessä käyttäjälle näytetään sisään kirjautuminen -näyttö (kuva 12), jossa käyttäjä kirjautuu sisään henkilökohtaisella PIN-koodilla. Lisäksi voidaan valita Vaihda ajoneuvoa -valintalaatikolla halutaanko vaihtaa käytössä oleva ajoneuvo. Sovellus siis muistaa edellisellä käyttökerralla käytössä olleen ajoneuvon.



KUVA 12. Sisään kirjautuminen, kaluston valinta, tervehdysviesti ja viestit

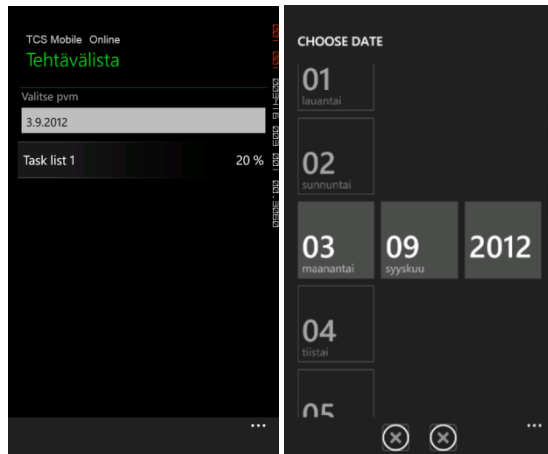
Ensimmäistä kertaa sisään kirjautuessa PIN-koodi tarkistetaan palvelimelta. Onnistuneen sisään kirjautumisen jälkeen käyttäjän tiedot tallennetaan myös puhelimen tietokantaan. Näin käyttäjä voi jatkossa kirjautua sisään sovellukseen myös ilman verkkoyhteyttä. Jos sovelluksen ei ole tallennettu viimeksi käytettyä ajoneuvoa tai käyttäjä haluaa vaihtaa edellisellä käyttökerralla käytettyä ajoneuvoa valitsemalla Vaihda ajoneuvo -valintalaatikon, näytetään lista mahdollisista ajoneuvoista.

Kalustot-näytöllä näytetään lista mahdollisista ajoneuvoista. Hakutoiminto tulee näkyviin näytön alareunan suurennuslasi-kuvakkeesta. Kun käyttäjä on valinnut ajoneuvon, näytetään Tervehdysviesti-näyttö. Tervehdysviesti on yhteinen yrityksen kaikille käyttäjille. Tätä näyttöä ei näytetä, jos tervehdysviestiä ei ole asetettu palvelimella.

Viestit-näytöllä näytetään lista henkilökohtaisista viesteistä. Aikaisemmin luetut viestit näytetään harmaalla tekstillä. Käyttäjä voi avata viestin koko näytölle koskettamalla viestin otsikkoa.

## 6.2 Tehtävälisan valinta

Tehtävälisan valinta -näytöllä (kuva 13) käyttäjä valitsee käytettävän tehtävälisan. Listalla näkyy tehtävälisan otsikko sekä prosenttimäärä, joka kuvaa kuinka suuri osa listan tehtävistä on valmis-tilassa. Näytöllä näytetään oletuksena kaikki kyseisen päivän tehtävälisat. Jos käyttäjälle on asetettu oletustehtävälisa, tätä näyttöä ei näytetä ja siirrytään suoraan kyseiselle tehtävälisalle.

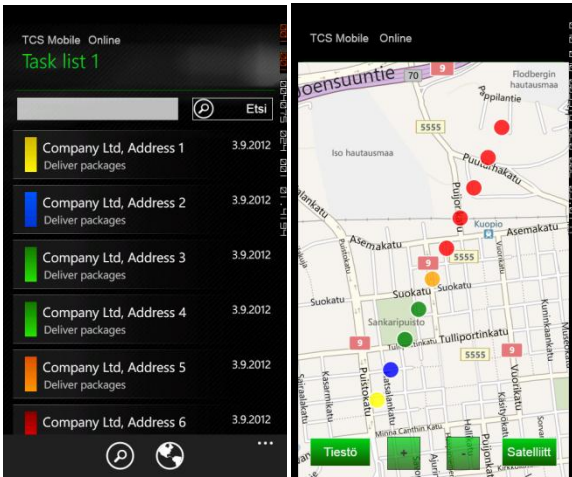


KUVA 13. Tehtävälisan valinta -näyttö

Käyttäjä voi vaihtaa käytettävää päivämäärää valitsemalla näytön yläreunassa olevan tekstikentän, josta aukeaa päivämäärän valinta -näyttö. Kun käyttäjä on valinnut päivämäärän, palataan takaisin Tehtävälisan valinta -näytölle, jossa näytetään valitun päivämäärän tehtävälisat.

## 6.3 Tehtävälisan näyttö

Tehtävälisa-näytöllä (kuva 14) näytetään kaikki tehtävät. Tehtävät on merkitty eri väreillä käytön helpottamiseksi. Eri värit kuvaavat tehtävän nykyistä tilaa. Tehtävien tilat ovat: aloittamaton (punainen), aloitettu (oranssi), keskeytetty (keltainen), valmis (vihreä) ja ei voi suorittaa (sininen).

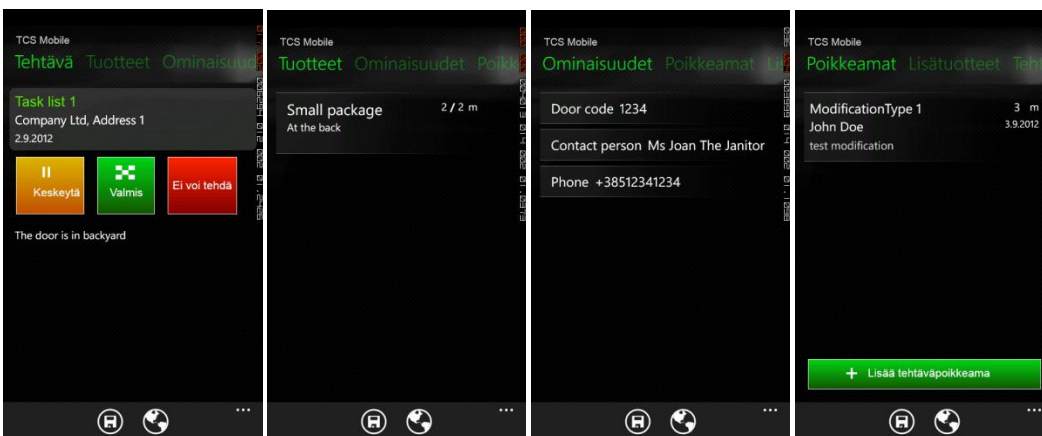


KUVA 14. Tehtävien lista- ja karttanäyttö

Tehtävälialta voidaan etsiä tehtäviä Haku-toiminnolla, joka saadaan näkyviin alareunan suurennuslasi-kuvakkeella. Maapallo-kuvakkeella avataan karttanäyttö, jossa voidaan tarkastella tehtävien sijainteja kartalla.

#### 6.4 Tehtävä

Tehtävä-näyttö on toteutettu PivotPage-komponentilla, jossa on kuvassa (kuva 15) olevat neljä näkymää. Näiden välillä voidaan siirtyä koskettamalla ylhäällä olevia otsikoita tai pyyhkäisemällä näyttöä sivulle. Tehtävä-näyttö koostuu tehtävän etusivusta, Tuotteet-, Ominaisuudet- ja Poikkeamat-listoista.



KUVA 15. Tehtävä-näyttö

Tehtävä-näytölle on toteutettu peruustustoiminto, jolla tehtävän tila ja kaikki sen tiedot voidaan palauttaa edellisen tallennuksen aikaiseen tilaan. Kun näytöltä poistutaan edelliseen näyttöön, kysytään käyttäjältä, tallennetaanko vai perutaanko kaikki muutokset.

Tehtävän etusivulla näytetään yleisiä tietoja tehtävästä, esimerkiksi tehtävälistan nimi, tehtävän osoite, kuvaus ja päivämäärä. Lisäksi etusivulla on painikkeet, joilla voidaan valita tehtävän tila. Painikkeista on näkyvissä vain ne, jotka voidaan valita kyseisellä hetkellä. Jos tehtävän tila on aloittamaton tai valmis, mitään muutoksia ei voida tehdä tehtävän tietoihin ja silloin kaikki tehtävän muokkaamisvalinnat on piilotettu käyttäjältä.

Tuotteet-näkymässä näkyy lista tehtävään liittyvistä tuotteista. Tuotteista näytetään tuotteen nimi ja kuvaus sekä alkuperäinen ja nykyinen määrä. Tuoteriviä koskettamalla avautuu uusi näyttö, jossa voidaan muokata tuotteen tietoja.

Ominaisuudet-näkymässä on lista tuotteelle kuuluvia lisätietoja. Nämä tiedot ovat vain luettavia eikä niitä siis voi muokata.

Poikkeamat-näkymässä on lista mahdollisesti tehtävälle lisätyistä poikkeamista. Tehtävälle voidaan lisätä poikkeama Lisää tehtäväpoikkeama -painikkeella. Poikkeama lisätään jos halutaan tehdä muutoksia tai huomautuksia tehtävään. Lisää tehtäväpoikkeama -painike avaa uuden näytön, josta valitaan poikkeamaryhmä ja -tyyppi ja sen jälkeen syötetään poikkeamatyypissä määritellyt tiedot. Myös yksittäiselle tuotteelle voidaan lisätä poikkeama, jos halutaan sen kohdistuvan vain sille tuotteelle.

## 7 YHTEENVETO

Opinnäytetyön vaatimuksena oli toteuttaa Windows Phone 7 -puhelimelle tehtävienhallinnan päätelaitesovellus ja sen palvelinrajapinta käyttämällä CSLA-sovelluskehystä. Sovelluksen täytyi pystyä toimimaan myös ilman verkkoyhteyttä. Opinnäytetyö vastaa näitä vaatimuksia melko hyvin.

Sovelluksella voidaan ladata palvelimelta olemassa olevia tehtävälisteroja ja tehtäviä sekä tehdä niihin muutoksia. Tehdyt muutokset tallentuvat takaisin palvelimelle. Kaikki tieto, jota käsitellään sovelluksessa, tallennetaan myös puhelimen tietokantaan. Sovellusta voidaan myös käyttää ilman verkkoyhteyttä. Kuitenkaan siirtyminen verkkoyhteydellisestä tilasta yhteydettömään välillä ei toimi saumattomasti.

Opinnäytetyö osoittautui melko haastavaksi. CSLA-sovelluskehys oli tuntematon minulle ennen opinnäytetyön aloittamista ja suuri osa ajasta menikin tutkiessa CSLA:n rakennetta ja opetellessa sen toimintaa. Varsinkin Windows Phone -sovelluksesta tehtävät palvelimen tietokantakerroksut osoittautuivat alussa haasteellisiksi eikä ollut aina selvää, tulevatko virheet Windows Phone -sovelluksesta vai palvelimen WCF-palvelusta. Bisnesluokkiin tehtiin testit tätä tarkoitusta varten jälkikäteen, mutta ehkä olisi ollut järkevää tehdä järjestelmän kehittäminen kokonaan testivetoisesti.

Jälkikäteen ajatellen CSLA:sta oli kuitenkin suurin hyöty sovelluksen ja palvelinrajapinnan toteutuksessa. Varsinkin MVVM-suunnittelumallia käyttävän sovelluksen kanssa CSLA:n käyttäminen on järkevää, sillä CSLA:n avulla tehdyt bisnesluokat voidaan suoraan liittää käyttöliittymään DataBinding-tekniikkaa käyttäen. Bisnesluokkien ja tietokantakerroksen tekeminen CSLA:n avulla on melko suoraviivaista ja ohjelmoija voi keskittyä enemmän sovelluksen logiikan ja käyttöliittymän tekemiseen. CSLA:kaan ei ratkaise kaikkia ongelmia mitä verkon yli tietoa siirtävän tekemisessä tulee vastaan. Suurimpana haasteena oli tiedonsiirron suorituskyky suuria bisnesoliotoita siirrettäessä.

Opinnäytetyön tekeminen opetti paljon uutta ohjelmoinnista. Varsinkin ongelmien ratkaisukyky ja virheiden etsiminen kehittyivät melkoisesti. Myös kerrosarkkitehtuurin edut alkoivat hahmottua paremmin projektin edetessä. Opinnäytetyön tekemiseen meni vähän arvioitua enemmän aikaa, mutta lopputulokseen olen tyytyväinen.

Jatkoin järjestelmän kehittämistä opinnäytetyön jälkeen työharjoitteluna ja nykyisin sitä kehittää useampi henkilö Ecomond Oy:ssä. Järjestelmä tulee Ecomond Oy:n asiakkaiden käyttöön lähiaikoina.

## LÄHTEET

CodePlex *Basic Xaml Framework*. [verkkodokumentti] [viitattu 25.9.2012] Saatavissa: <http://bxf.codeplex.com/>

Ecomond 2012. [verkkodokumentti] [viitattu 20.10.2012] Saatavissa: <http://www.ecomond.com/>

Lecrenski, N., Watson, K, Fonseca-Ensor, R. 2011. *Windows Phone 7 Application Development*. Indianapolis: Wiley Publishing Inc.

Lhotka, R. 2009. *Expert C# 2008 Business Objects*. Berkeley: APress.

Lhotka, R. 2011. *Using CSLA .NET 4: Data Portal Configuration*. [PDF] Marimer LLC [viitattu 13.9.2012] Saatavissa: <http://lhotka.net/cslanet>

Lhotka, R. 2012. *Using CSLA .NET 4: Windows Phone*. [PDF] Marimer LLC [viitattu 13.9.2012] Saatavissa: <http://lhotka.net/cslanet>

Microsoft. *LINQ: .NET Language-Integrated Query* 2007. [verkkodokumentti]. [viitattu 25.9.2012] Saatavissa: <http://msdn.microsoft.com/en-us/library/bb308959.aspx>

Microsoft. *What Is Windows Communication Framework* 2012. [verkkodokumentti] [viitattu 25.9.2012] Saatavissa: <http://msdn.microsoft.com/en-us/library/ms731082.aspx>





