SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

Dynaaminen raportointialusta ForestPro-toiminnanohjausjärjestelmään

TEKIJÄ: Perttu Leppänen

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä Perttu Leppänen	
Työn nimi Dynaaminen raportointialusta ForestPro-toiminnanohjausjärjestelmään	
Päiväys 9.3.2021	Sivumäärä/Liitteet 38
Toimeksiantaja/Yhteistyökumppani PiiMega Oy	

Tiivistelmä

Tässä opinnäytetyössä luotiin raportointialusta PiiMega ForestPro-toiminnanohjausjärjestelmään. Alustan tarkoitus oli sisältää dynaamiset ryhmittely- ja tallennusvalinnat.

Kehitystyö toteutettiin pääosin C#-ohjelmointikielellä.
Opinnäytetyössä tarkasteltiin kehitystyön ohessa pintapuolisesti ohjelmointiparadigman vaikutusta työhön ja lopputulokseen.

Lopputulokseksi saatiin raportointialusta, joka päätyi tuotantokäyttöön useammalle asiakkaalle. Kuitenkin jatkokehitys- ja korjaustarvetta jäi vielä paljon.

Avainsanat

C#, Visual Basic, .NET, Ohjelmointi, Raportointi

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author Perttu Leppänen	
Title of Thesis Dynamic Reporting Tool for the ForestPro ERP System	
Date 9 March 2021	Pages/Appendices 38
Client Organisation /Partner PiiMega Oy	

Abstract

The subject of this thesis was to create reporting tool for the PiiMega ForestPro ERP software. A Purpose was to include a possibility to save aggregates, grouping and date filters dynamically.

The development process was mostly done by the C# programming language. Faced problems and the results were also described. In addition, the effects of programming paradigm were considered.

The result of the thesis was a reporting tool which ended up in production usage with multiple customers.

Keywords

C#, Visual Basic, .NET, Programming, Reporting

ESIPUHE

Tässä opinnäytetyössä kuvatun kehitystyön syntymiseen johtaneet tekijät saivat alkunsa huhtikuussa 2019, kun sain työharjoittelupaikan oululaisesta PiiMega Oy -ohjelmistotalosta. Työskenneltyäni vuoden osin kokopäiväisesti osin osa-aikaisesti opintojeni ohessa etätöinä, annettiin minulle työtehtävä osana käyttöönottoprojektia, jossa uusi asiakas ottaisi ForestPro-järjestelmän käyttöön. Olin saanut ammattikorkeakouluopintoihini sisältyvät pakolliset kurssit ja harjoittelut juuri suoritettua loppuun ja lähiesimieheni tarjosi tilaisuutta kirjoittaa opinnäytetyö tarjotusta työtehtävästä. Heti alussa oli selvää, että toteutettava ominaisuus ylittäisi laajuudessaan insinöörityön vaatimukset moninkertaisesti, joten päätin lohkaista sopivan suuruisen ja yhtenäisen segmentin opinnäytetyötäni varten. Useamman viikon kehitystyön jälkeen minulle hahmottui riittävän selvästi tulevaan opinnäytetyöhöni sisällytettäväksi sopiva kokonaisuus ja prosessi sai alkunsa. Haluan kiittää työnantajayritystäni PiiMega Oy:tä, joka tarjosi mielekkään työtehtävän, tarvittavan avun ongelmatilanteiden selvittämiseksi sekä omaa oppimista tukevat puitteet opinnäytetyön suorittamiseksi.

Oulussa 9.3.2021
Perttu Leppänen

SISÄLTÖ

KUVALUETTELO

SANASTO

.NET	Microsoftin kehittämä ohjelmistokehitysalusta (.NET)
API	Application programming interface eli ohjelmointirajapinta (Avoinrajapinta.fi, 2014)
Base class	Koodiluokka, joka on luotu hallitsemaan yhtä tietokantataulua (Base Class, 2011)
C#	Microsoft .NET ympäristöön kuuluva ohjelmointikieli (Sivonen, 2004)
LINQ	.NET ympäristöön kuuluva kyselykieli tietorakenteiden hallintaan (Microsoft, 2017)
React	JavaScript-ohjelmointikirjasto
Pivot-taulukko	Taulukkokomponentti, joka laskee tietorakenteesta arvoja ryhmittelyjen perusteella
PWA	Mobiililaitteelle natiivisovelluksen tavoin skaalautuva web-sovellus
Serialisointi	Tietorakenteen muuttaminen tallennuskelpoiseen formaattiin
SQL	Tietokantakyselyjen muodostamiseen käytetty kieli

JOHDANTO

Tässä raportissa kuvaan insinöörin tutkintoon kuuluvan opinnäytetyöni ja sen toteuttamisen. Tämän opinnäytetyön aihe on kehitystyö, joka liittyy suurempaan kokonaisuuteen toteuttaa raportointiuudistus PiiMega Oy:n ForestPro-ohjelmistoon, joka on metsäliiketoiminnan tarkoituksiin toteutettu toiminnanohjausjärjestelmä. Kehitystyö kokonaisuudessaan ylitti laajuudeltaan insinöörityön vaatimukset, joten opinnäytetyöni aiheeksi paketoin segmentin, jonka aihe on luoda raportointi moottori, valita työkalut tiedon visualisointiin, toteuttaa tiedon visualisointi ja haku.

Ohjelmistoteknisesti tärkeimmäksi asiaksi nousi skaalautuva arkkitehtuurinen rakenne. ForestPro-ohjelmaa käytetään pääosin paikallisena Windows työpöytäsovelluksena tai pilvessä, mutta osa toiminnallisuuksista löytyy myös ForestPortal web-palvelusta, jonne oli tarkoitus toteuttaa käyttöliittymä raporttien tarkasteluun. Toteutuksessa oli otettava huomioon myös käyttäjien oikeudet tarkastella asiakasyrityksen liiketoiminnan kannalta merkityksellistä tietoa.

Eräs raportointiuudistuksen osista oli toteuttaa ForestPortal web-ympäristöön käyttöliittymä raporttien tarkasteluun. Tässä opinnäytetyössä ei kuvata web-osuuden toteutusta itsestään, mutta vaatimus ohjelmistolla generoitavien raporttien tarkasteluun web-käyttöliittymän kautta vaikutti tässä opinnäytetyössä kuvattavaan raportointialustan toteutukseen siinä määrin, että sitä jouduttiin sivuamaan muutamassa kohdassa raporttia.

Kehitystyö noudatti vahvasti olio-ohjelmointiparadigmaa. Tiedon hakuun hyödynnettiin mahdollisimman paljon valmiita olioita, mutta myös tiedon hakua, yhdistelyä ja tietokantakyselyitä oli toteutettava itse. Raportissa tarkastelenkin opinnäytetyötä paljon ohjelmistosuunnittelun ja oliopohjaisuuden näkökulmista. Pienessä mittakaavassa sivuan myös olio-ohjelmointiparadigman etuja ja haittoja vertauskohtana funktionaalinen ohjelmointiparadigma, sekä näiden kahden paradigman yhteneväisyyksiä. Nimi ”Dynaaminen raportointi” juontaa juurensa alustan toiminnallisuudesta, joka sallii käyttäjän valita ja vaihtaa tietojoukon ryhmittelyjä dynaamisesti ajamatta raporttia uudelleen. Lisäksi toteutettu tallennustoiminnallisuus tallentaa myös päivämäärärajaukset dynaamisesti ja tämä vaikutti osaltaan nimeämiseen.

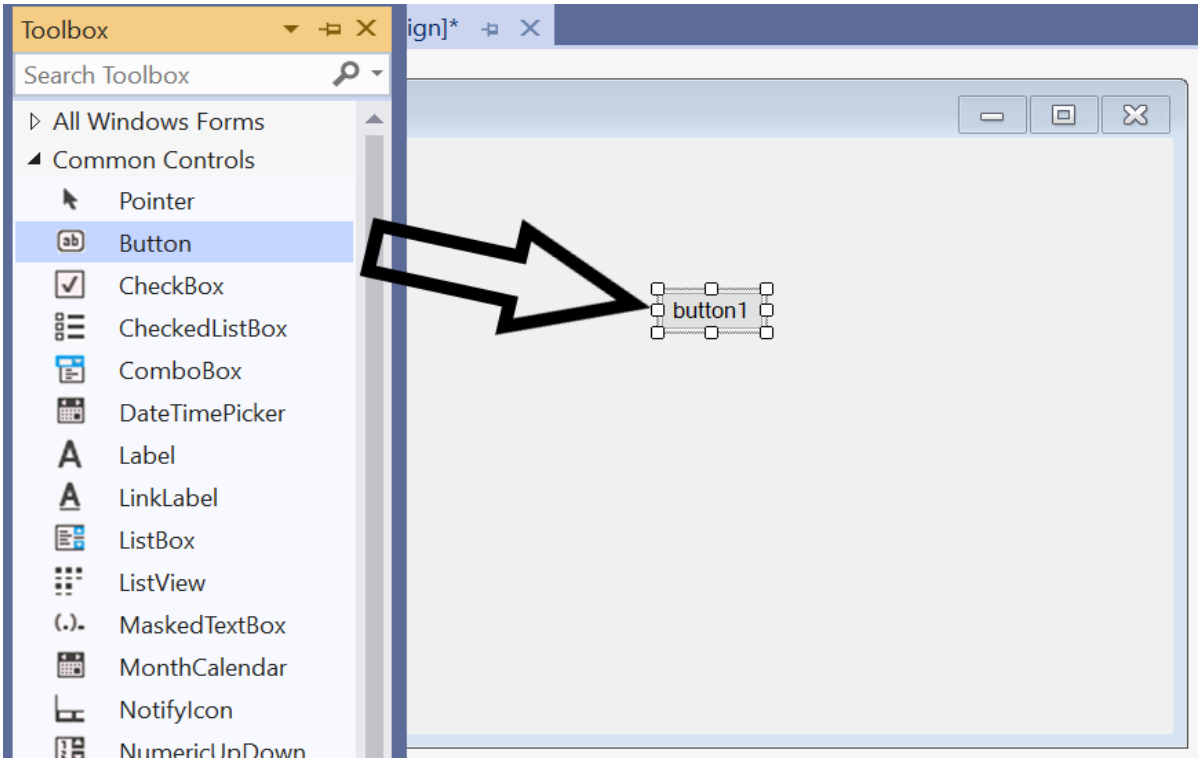
KÄYTETYT MENETELMÄT JA TEKNOLOGIAT

PiiMega ForestPro käytti kehityshetkellä .NET Frameworkia, joka on Microsoftin luoma ohjelmistokomponenttikirjasto. Suurin osa kappaleessa kaksi kuvatuista menetelmistä kuuluu .NET Frameworkiin.

3.1 Windows Forms

Windows Forms (WinForms) on Microsoftin alun perin 13.08.2002 julkaisema ilmainen avoimen lähdekoodin graafinen luokkakirjasto, jonka avulla voidaan rakentaa nopeasti ohjelmistoja Windows alustoille.

WinForms tarjoaa suunnittelua varten työkalun, jonka avulla käyttöliittymiä voidaan luoda niin kutsutulla drag and drop -periaatteella. Käyttöliittymän yhdistäminen ohjelmistologiikkaan on hyvin helppoa. Visual Studio -ohjelmointiympäristön WinForms designer tarjoaa kehittäjälle työkalupakin, josta voidaan hiirellä raahata kontrollin lomakkeen päälle. Alla oleva kuva (Kuva 1) havainnollistaa suunnittelutyökalun helppokäyttöisyyttä. ForestPro-toiminnanohjausjärjestelmä käytti käytettävyyden vuoksi alla olevasta kuvasta poiketen pääosin kaupallisen palveluntarjoajan tarjoamia kontrolleja sekä niiden päälle itse rakentamiaan laajennuksia.



Kuva 1. Visual Studion WinForms designer (Leppänen, 2020)

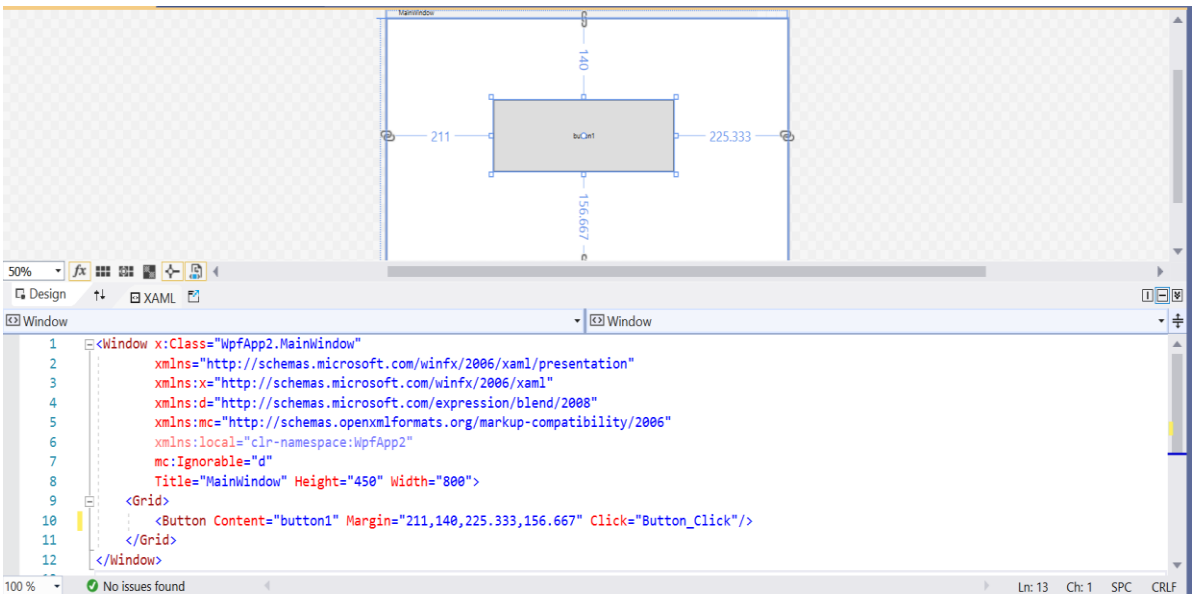
Käyttöliittymään lisätyn napin voi yhdistää koodiin esimerkiksi tuplaklikkaamalla nappia, jolloin ohjelmointiympäristö osaa generoida koodin, joka ajetaan ohjelman käyttäjän klikatessa käyttöliittymässä näkyvää nappia. Generoinnin seurauksena syntyvään tapahtumaan voidaan kirjoittaa ohjelma koodia. Koodin ylläpidettävyyden takia kontrollien tapahtumiin ei yleensä kirjoiteta juurikaan logiikkaa, vaan niiden sisältä kutsutaan varsinaiset toiminnallisuudet hoitavia funktioita. Tätä havainnollistan alla olevalla kuvalla (Kuva 2), jossa ylempi funktio on napin painalluksen jälkeen ajettava ”button1_click” -funktio ja alempi tapahtuman sisältä kutsuttava logiikan sisältävä funktio. Tämä mahdollistaa saman logiikkafunktion käyttämistä useammassa eri paikassa, myös silloin, kun kutsu ei tapahdu nappia painamalla.

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    RunReport(ID);
}
1 reference
private void RunReport(int iID)
{
    // Business logic
}
```

Kuva 2. Tapahtumien käyttö (Leppänen, 2020)

3.2 Microsoft Presentation Foundation

WPF on Microsoftin alun perin vuonna 21.11.2006 julkaisema graafinen luokkakirjasto ohjelmistojen luontiin Windows alustoille. WPF suunniteltiin korvaamaan Windows Forms. WPF käyttöliittymät luodaan XAML-kuvauskielellä. Myös WPF tarjoaa designerin ja mahdollisuuden yhdistää käyttöliittymä helposti ohjelmakoodiin, esimerkiksi tuplaklikkaamalla nappia suunnittelunäkymässä. WPF on WinFormsia uudempi teknologia ja sen käyttöä onkin Microsoftin toimesta suositeltu näistä kahdesta ensisijaisena vaihtoehtona, mutta myös WPF:lle on kehitetty uudempia vaihtoehtoja, kuten Universal Windows App ja alustariippumaton Xamarin. WPF:än oppiminen vie yleensä hieman enemmän aikaa WinFormsiin verrattuna. Alla olevassa kuvassa (Kuva 3) designerin yläosassa näkyy graafinen näkymä ja alaosassa XAML-kuvauskielellä kirjotetut määritykset.



Kuva 3. Visual Studio ohjelmointiympäristön WPF designer. (Leppänen, 2020)

3.3 Ohjelmointikielet

3.3.1 C#

C# on Microsoftin alun perin kesäkuussa 2000 .NET-ympäristöä varten julkaisema ohjelmointikieli. Se kuuluu käännettäviin ohjelmointikieliin ja on vahvasti tyypitetty, eli muuttujat saavat tyypin esim. kokonaisluku tai merkkijono. (Sivonen, 2004.) Tässä opinnäytetyössä C# oli kaikkein keskeisin ohjelmointikieli, sillä kaikki ForestPro:n uusi logiikka ohjelmoitiin sillä. Kuvassa (Kuva 4) on esimerkki C#-ohjelmointikielestä.

```
using System;

namespace HelloWorld
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Kuva 4. Hello World -esimerkki C# -ohjelmointikielellä. (Leppänen, 2020)

3.3.2 VB.NET

Visual Basic on Microsoftin alun perin vuonna 1991 julkaisema ohjelmointikieli. Vuonna 2002 Microsoft julkaisi huomattavasti uudistuneen .NET ympäristöön kuuluvan VB.NET:tin. Vaikka raportointi uudistuksen pääasiallinen ohjelmointikieli oli C#, käytettiin joissain tilanteissa ForestPro:n vanhoja VB.NET:llä ohjelmoituja olioita. Visual Basic on C#:in tavoin vahvasti tyypitetty kieli. Nämä kaksi ohjelmointikieltä ovat täysin yhteensopivia toistensa kanssa. Visual Basicilla kirjoitettuja funktioita voi kutsua C# koodista ja päinvastoin. Kuvassa (Kuva 5) on esimerkki Visual Basic ohjelmointikielestä.

```
Imports System

0 references
Module Program
    0 references
    Sub Main(args As String())
        Console.WriteLine("Hello World!")
    End Sub
End Module
```

Kuva 5. Hello World -esimerkki VB.NET -ohjelmointikielellä. (Leppänen, 2020)

3.3.3 ActiveReports

ActiveReports on GrapeCityn omistama .NET kehittäjille suunnattu työkalu tulosteiden ja pdf-dokumenttien generointiin. Sillä voidaan suunnitella raportti graafisella työkalulla ja antaa raportille ohjelmallisesti tiedot, josta rakennetaan tuloste tai pdf-dokumentti. (ActiveReports, 2020.)

OHJELMOINTIPARADIGMAT

Ohjelmointiparadigma on ohjelmointikielen tapa jäsentää ohjelman rakenne ja suoritusmalli. Ohjelmointi paradigmoja on useita ja niitä voidaan jaotella aliparadigmoihin. (Ohjelmointiparadigmat, 2019.)
Tässä opinnäytetyössä kuvaan lyhyesti olio- ja funktionaalista paradigmaa, sillä tiedon hakuun ja laskentaan pohjautuvassa raportointityökalussa valittu ohjelmointiparadigma vaikuttaa suorituskykyyn ja koodin ylläpidettävyyteen. ForestPro on laaja järjestelmä, jossa samat tieto-oliot ovat monessa paikkaa järjestelmää käytössä, joten ylläpidettävyys ja vakaus vaatimukset korostuvat

4.1 Olio-ohjelmointi

Oliopohjaisessa ohjelmoinnissa ohjelma jaetaan olioihin, jotka sisältävät funktioita ja muuttujia. Jokaisen olion on tarkoitus hoitaa omia tehtäviään. Kun ohjelma jaetaan onnistuneesti olioihin, voidaan saavuttaa korkea suoritusvarmuus ja laaja skaalautuvuus. Kun yksi olio ohjelmoidaan hyvin suorittamaan tiettyä tehtävää, voidaan tätä samaa olioita käyttää aina uudelleen samaan tarkoitukseen ja näin vältetään moninkertaista koodia, sekä tilanteita, joissa sama asia on yritetty tehdä kahdessa eri

paikassa. (OOP, 2015.) C#-ohjelmointikielessä olio määritellään avain sanalla ”class”. Olioita käytetään myös tietorakenteiden tyypittämiseen, jolloin olio ei välttämättä sisällä yhtään funktiota, vaan pelkkiä ominaisuuksia, jotka ovat luokan sisäisiä muuttujia arvojen tallentamiseen. Tässä raportointiuudistuksessa hyödynnettiin paljon valmiita olioita, joiden tehtävä oli palauttaa valmista dataa. Lisäksi tietokyselyt tyypitettiin olioiden avulla. Alla olevassa kuvassa (Kuva 6) havainnollistetaan olion käyttämistä tyypitykseen. Esimerkin ”InvoiceRow” -luokka ei sisällä yhtään funktiota, vaan kertoo minkä nimisiä ja tyyppisiä muuttujia tietorakenne voi sisältää.

```
private class InvoiceRow
{
    0 references
    private int ProductID { get; set; }
    0 references
    private string Description { get; set; }
    0 references
    private decimal Quantity { get; set; }
    0 references
    private decimal Price { get; set; }
}
```

Kuva 6. Tyypitys luokka. (Leppänen, 2020)

4.1.1 Funktionaalinen ohjelmointi

Funktionaalisessa ohjelmoinnissa ohjelman suoritus jaetaan funktioihin, jotka ovat toimintatavaltaan samanlaisia, kuin funktiot olio-ohjelmoinnissa. Funktionaalisessa ohjelmoinnissa ei kuitenkaan ole olioita eikä näin ollen periytyminen ole mahdollista. Funktionaalisissa ohjelmointikielissä muuttujia voidaan niputtaa useamman funktion käyttöön nimiavaruuden avulla. Funktio voi vastaanottaa parametreja ja palauttaa arvon. (Heinonen, 1996.) Logiikan pilkkominen pieniin osiin helpottaa koodin luettavuutta ja ylläpidettävyyttä.

Myös testaaminen on helpompaa, kun toisiinsa liittymätöntä logiikkaa ei ole kirjoitettu saman funktion sisään. Opinnäytetyön aiheena olevassa raportointiuudistuksessa käyttöliittymä, raporttimoottori ja raporttifiltteri ovat jokainen oma olionsa, mutta varsinkin käyttöliittymäluokan sisäinen logiikka muistuttaa funktionaalista ohjelmointia. Alla olevassa kuvassa (Kuva 7) on esimerkkiluokka, jonka sisäinen logiikka on hyvin funktionaalista. Kuitenkin käytettäessä funktiota olion sisällä, on parametrien käytön tarve pieni, sillä luokan sisäiset funktiot näkevät oman olionsa jäsenmuuttujat eikä niitä näin ollen tarvitse liikutella funktioiden välillä parametreina. Funktio kutsujen edessä oleva avainsana ”this” viittaa olioon, jonka sisälle logiikka on kirjoitettu. Kuva sisältää esimerkki koodia havainnollistamaan paradigmojen päällekkäisyyttä.

```
3 references
public partial class Form1 : Form {
    1 reference
    private ReportEngine oReportEngine { get; set; } // Fieldi. Kaikki olion sisällä olevat funktiot näkevät tämän.
    1 reference
    public Form1() {
        InitializeComponent();
        this.InitUserControls();  // Kutsutaan funktiota, joka alustaa UserControllit
        this.InitForm();          // Kutsutaan funktiota, joka suorittaa itse määrittelemiäni käyttöliittymän alustuksia
        this.InitReportEngine();
    }
    #region UI functions
    1 reference
    private void InitUserControls() // Funktio UserControllien alustamiseen. UserControllien alustaminen on jaettu alifunktioihin
    {
        this.InitReportmanagementUserControl();
        this.InitGroupingUserControl();
    }
    1 reference
    private void InitForm()
    {
        this.InitDateRange(); // Kutsutaan funktiota, joka alustaa formilla olevat päivämäärä valitsimet
    }
    #endregion
    #region Functions
    1 reference
    private void InitReportEngine() // Funktio, joka lähettää raportointi moottorille
    {
        oReportEngine.FetchCoreData();
```

Kuva 7 Funktioiden käyttö olion sisällä. (Leppänen, 2020)

4.1.2 Paradigmojen vaikutus opinnäytetyöhön

Raportointiuudistuksessa olio-ohjelmointi helpottaa ylläpitoa, sillä virheellisen tiedon noustessa raportille, voidaan virhe korjata vain yhdestä tiedonhaku-oliosta, eikä jokaista raporttia ole tarpeen korjata yksitellen. Funktionaalisella ohjelmoinnilla voidaan tavoitella samaa ohjelmoimalla mahdollisimman stabiileja tiedonhakufunktioita, mutta tällöin tullaan tilanteeseen, joissa joudutaan käyttämään kuormituksia ja optionaalisia parametrejä.

Toisaalta olio-ohjelmoinnissa täytyy oliosta luoda ilmentymä, jotta olion sisältämiä metodeja voidaan kutsua. Olion alustaminen voi vaatia paljon resursseja, jos konstruktori sisältää tietokantakutsuja kenttien alustamiseen. Tällaisessa tilanteessa funktionaalisella ohjelmoinnilla voidaan saavuttaa parempi suorituskyky. Jos olio alustetaan tietokannasta, tehdään alustus vaiheessa yksi kysely ja muokkausvaiheessa joudutaan tekemään uusi muokkaus kysely. Kuitenkin olio voi tallentaa muuttujia toisin kuin funktio ja on näin ollen helpommin hallittavissa oleva kokonaisuus. Laajassa järjestelmässä olio-ohjelmointia pitää koodin ylläpidettävänä ja skaalautuvana. Asiakaspalautteen mukaan teollisuuden käyttöön suunnatuissa ohjelmistoissa vakaus ja tiedon oikeellisuus ovat asiakkaille tärkeimmät asiat, joten suorituskykyä ei aina voida optimoida näiden kustannuksella.

KEHITYSTYÖN TOTEUTTAMINEN

ForestPro järjestelmä sisälsi valmiiksi laajaa raportointia, joka ei kuitenkaan käytettävyydeltään vastannut sitä tasoa, jota PiiMega Oy sekä ohjelmistojen käyttäjät siltä edellyttivät. Suurin ongelma vanhassa raportoinnissa oli sen jakautuminen moneen eri paikkaan. Vanhaa raportointia ei kuitenkaan lähdetty täysin korvaamaan uudella dynaamisella alustalla, vaan tarkoitus oli tarjota uusi työkalu vanhojen työkalujen lisäksi. Ennestään olemassa olevan raportoinnin käyttämää tiedonhakua pystyttiin hyödyntämään huomattavasti kehitystyössä. Muutenkin laskentalogiikkaa oli paljon valmiina monia liiketoiminnan kannalta merkityksellisiä tilastoja koskevia summia ja lukumääriä varten, lähes sellaisinaan hyödynnettävissä. Johdannossa mainittiin tarkoituksesta luoda myöhemmin web-käyttöliittymä raporttien tarkasteluun ja tämän aiheuttamista skaalautuvuus vaatimuksista.

SUUNNITTELU JA TOTEUTUS

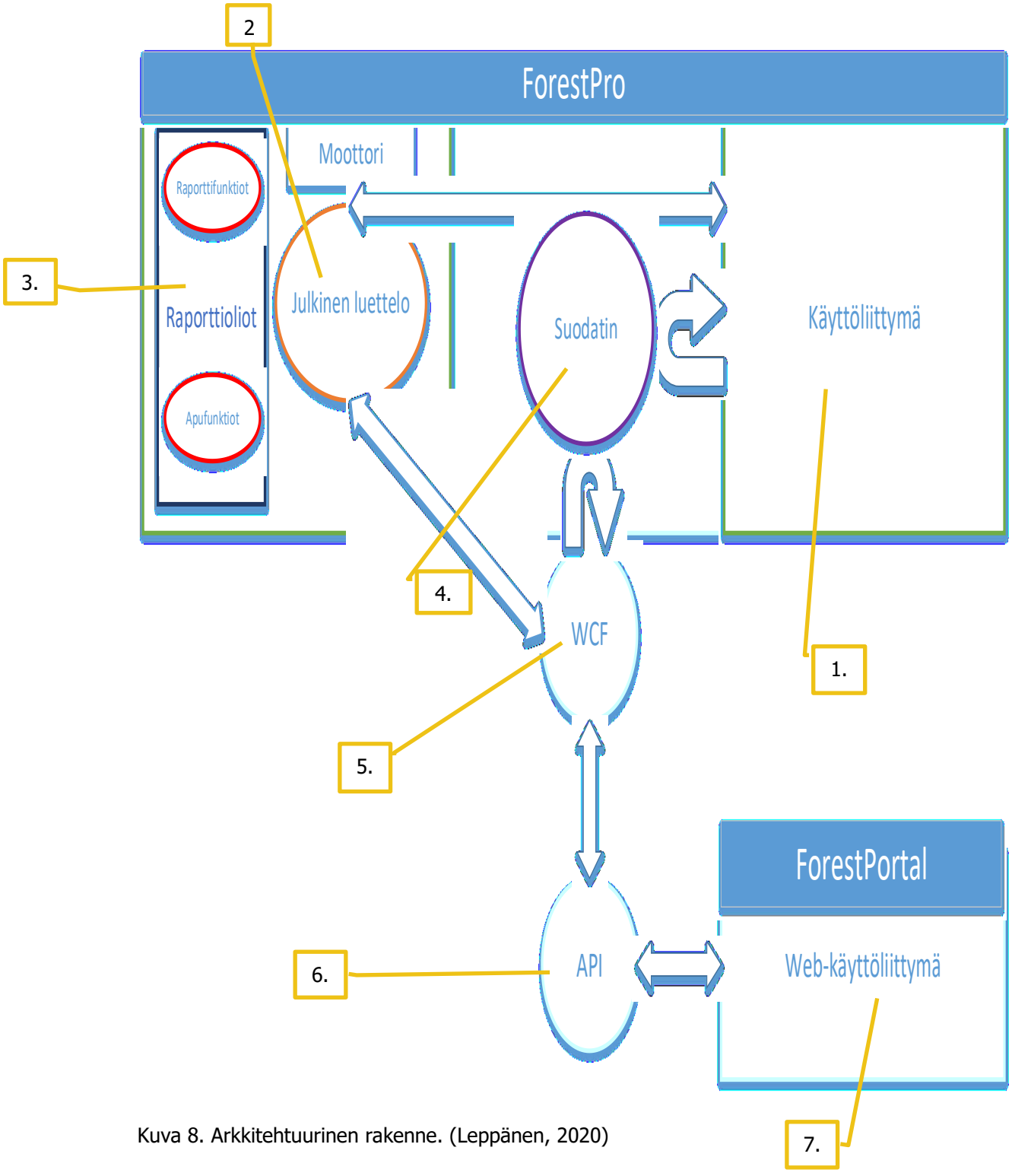
6.1 Suunnittelu

Työ alkoi suunnittelusta ja ensimmäinen lähtökohta oli luonnollisesti ohjelmistoarkkitehtuurinen rakenne. Ensimmäinen selvä asia oli se, että haluttiin käytettävyydeltään yhtenäinen kokonaisuus, jossa kaikki toteutettavat raportit tulisivat olemaan mahdollisimman samassa paikassa. Tämä siksi, että näin käyttäjä löytää käytössä olevat toiminnallisuudet ja oppii muistamaan ne nopeammin, joten palvelun tarjoajan puolelta koulutustarve vähenee. Käytännössä tämä tarkoitti työpöytäsovelluksen käyttöliittymä toteutuksen osalta ”Dynaaminen raportointi”-nimistä valikkoa, jonka painaminen avasi alivalikon, johon raportointikategoriat olivat listattu. Kategoriaa klikkaamalla avautui ikkuna, jonka sisältä löytyivät varsinaiset aihealueeseen kuuluvat raporttipainikkeet. Joihinkin raportteihin saattoi päästä käsiksi myös muualta ohjelmasta, jossa raporttiin liittyvää tietoa käsiteltiin.

Toinen suunnitteluvaihe oli työkalujen valinta. Tässä kehitystyössä se tarkoitti käytännössä komponenttia, jonka päällä tieto esitetään käyttäjälle, eli itse raporttia. Toteutukseen valittiin kaupallisen toimittajan pivot-taulukko johtuen sen tarjoamista valmiista tiedon ryhmittely ja esitys toiminnallisuuksista, räätälöinti mahdollisuuksista, sekä siitä, että kyseinen toimittaja oli toimittanut kaupallisia ohjelmistokomponentteja PiiMega Oy:lle jo useiden vuosien ajan. Web-käyttöliittymä oli toteutettu ReactJS:llä, jolle tämä komponentti ei ollut saatavilla. Tässä päädyttiin valitsemaan saman toimittajan JQuerylle toteuttaman pivot-taulukko-komponentin sovittaminen Reactille. Komponentintoimittaja tarjosi tämän muunnoksen suorittamiseen dokumentaation, joten valinta oli luonnollinen.

Kolmas suunnittelukohta oli skaalautuvuus. Idea oli, että raporttimoottori palauttaa käyttöliittymän pyynnön mukaisesti geneerisen joukon tietoa, jonka käyttöliittymä muodostaa taulukkoon. Tässä kohtaa kävi selväksi se, että täydellistä geneerisyyttä ei voida toteuttaa. Eri raportit tarjosivat eri

ryhmittelyvalintoja, joten käyttöliittymän piti luonnollisesti tietää pyydettävän raportin tyyppi. Lisäksi tietopyynnön piti kertoa raportointimoottorille, mitä raporttia ollaan käsittelemässä, joten päätettiin kehittää niin kutsuttu ”suodatin”-luokka, josta käyttöliittymä voisi luoda ilmentyminen. Lisäksi raporttikohtaiset ryhmittely valinnat täytyi tallentaa tietorakenteisiin käyttöliittymää varten. Näin päädyttiin käyttämään karkeasti alla olevan kuvan (Kuva 8) mukaista arkkitehtuuria.



Kuva 8. Arkkitehtuurinen rakenne. (Leppänen, 2020)

Yllä olevassa kuvassa (Kuva 8) on esitelty seuraavat kohdat:

1. ForestPro:n käyttöliittymä koostuu WinForms ja Wpf kontrolleista. Tämä on varsinainen raportointialusta, jonka asiakas näkee ja jossa raporttien hallinta, suorittaminen ja tarkastelu tapahtuu. Käyttöliittymä loi instanssin suodattimesta ja kutsui Moottorin ”julkisesta luettelosta” löytyviä raportin funktioita käyttäen suodatinta parametrina.

2. Raporttimoottoriolio sisältää julkisiksi asetetut ulospäin näkyvät raporttifunktiot, jotka hakevat ja palauttavat tiedon varsinaisista raporttifunktioista.

3. Raporttimoottoriolio sisältää varsinaiset raporttioliot. Näitä olioita toteutettiin yksi jokaista kategoriaa kohti. Tällaisia ovat varastoraportointiolio sekä ostoraportointiolio. Raporttioliot ovat raporttimoottoriolion sisällä ja lisäksi ne perivät moottoriolion. Raporttiolio sisältää varsinaiset raporttifunktiot, jotka ylikirjoittavat moottoriolion julkiset funktiot. Näiden sisällä kutsutaan ja yhdistellään järjestelmästä löytyvien tieto-olioiden dataa LINQ:lla, sekä haetaan jonkun verran uuttaa tietoa tietokannasta SQL:ää hyödyntäen. Eli varsinainen raportin muodostus ja laskeminen tapahtuu raporttimoottoriolioissa.

4. Suodatin on olio, johon tallennetaan tiedonhakusuodattimet. Käyttöliittymä luo ilmentymän suodattimesta ja tallentaa siihen esimerkiksi päivämäärärajaukset DateTime-tietotyyppisenä, varastotyypin enum-tyyppisenä. Lisärajaukset kuten puutavaralaji, myyjä, kunta ja sopimus tallennetaan suodattimeen kokonaislukulistana, jotka sisältävät ID arvoja. Suodatin olio lähetetään moottorille parametrina raporttifunktiota kutsuttaessa.

5. WCF-service välittää web-käyttöliittymästä tulevat raporttipyynnöt oikeaan asiakastietokantaan.

6. Api välittää web-käyttöliittymästä tulevat pyynnöt WCF-servicen kautta raporttimoottorille ja palauttaa raportin web-sovellukseen. Apiin luodaan kopio suodatin luokasta kyselyiden tyypittämiseksi.

7. ForestPortalissa sijaitsee raportoinnin web-käyttöliittymä. Sen toteutusta ei käsitellä tässä opinnäytetyössä, mutta se vaikutti ohjelmistoarkkitehtuuriin siinä määrin, että oli tarpeellista sisällyttää se omalle paikalleen arkkitehtuurin kuvaukseen.

6.2 Toteutus

6.2.1 Käyttöliittymä

Ohjelmointityö alkoi toteuttamalla yksinkertainen käyttöliittymä, jossa oli raportoinnissa käytettävä pivot-taulukko ja painonappi tiedon hakemiseen. Tiedonhaku toteutettiin tässä vaiheessa hakemalla satunnaisesti valitusta oliosta tietoa perehtymättä sen oikeellisuuteen tai rakenteeseen. Tämä tarkoituksena oli varmistaa, että valittu taulukkokomponentti soveltuu tarpeisiin. Se suoritettiin heti ensimmäiseksi siitä syystä, että mahdollisten väärästä komponentin valinnasta johtuvien muutostarpeiden tullessa jouduttaisiin tekemään mahdollisimman vähän työtä uusiksi. Tähän vaiheeseen kului useita täysiä työpäiviä ja komponentintoimittajaan jouduttiin olemaan useampaan kerran yhteydessä.

Kävi ilmi, että valittu komponentti ei yksistään olisi ominaisuuksiltaan riittävä kaikkien raporttien toteuttamiseen, vaan sen rinnalle valittiin vaihtoehdoksi saman kaupallisten komponenttien tarjoajan tuottama taulukkokomponentti. Alkuperäistä pivot-taulukkoa päädyttiin käyttämään niissä raporteissa, joissa numeerisen tiedon määrä oli suuri. Tavallista taulukkoa sen sijaan raporteissa, joissa haluttiin näyttää tekstipohjaista tietoa.

Kun käyttöliittymän malli oli hahmottunut, rakennettiin visuaalinen käyttöliittymä valmiiksi toiminnallisuuksien puolesta ilman lopullista käytettävyyden optimointia. Oli tärkeää tehdä käyttöliittymä jokseenkin valmiiksi aikaisessa vaiheessa, jotta tiedettiin, että kokonaisuus on mahdollista toteuttaa linjattujen suunnitelmien mukaan. Käytettävyyden hiominen tässä vaiheessa koettiin turhaksi. Käyttöliittymän toteutus kehitystyönaikana modernissa ohjelmistossa olisi vaatinut paljon työaikaa, ja asiakkaan mielipiteet voivat erota kehittäjän näkemyksistä, joten käytettävyyden lopullinen muotoilu tuotantokelpoiseksi jätettiin projektin loppupuolelle.

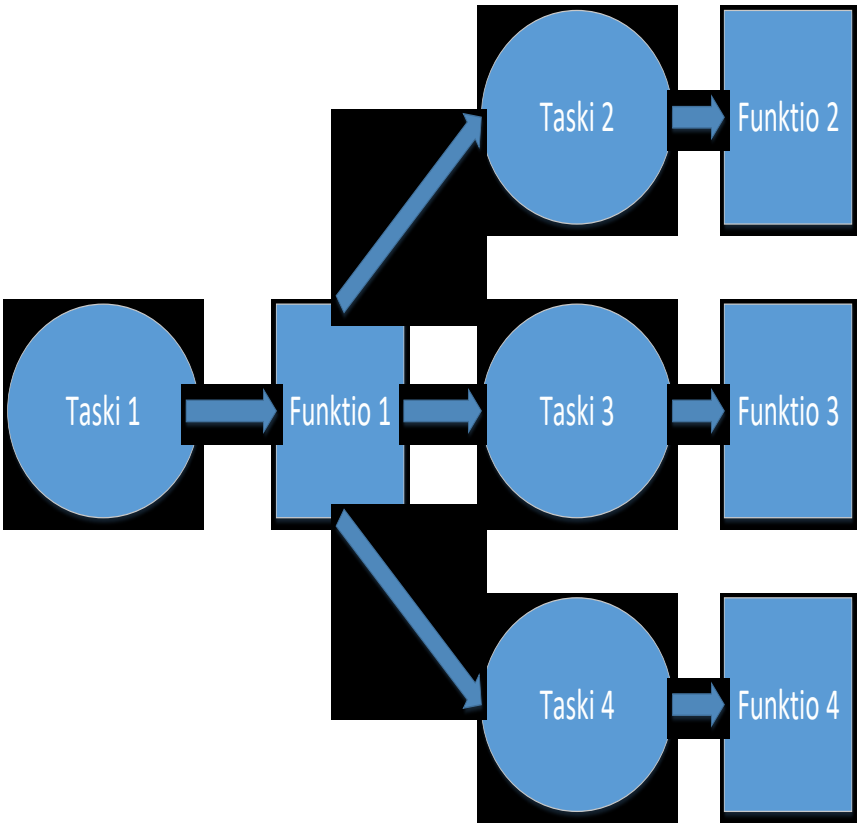
6.2.2 Tiedon haku

PiiMega ForestPro oli kehitystyön alkaessa ollut markkinoilla noin 9-vuotta. Lisäksi ohjelmisto sisälsi laajasti raportointia ennestään, joten tarvittavaa tietoa löytyi valmiiksi paljon. Kelvollisen tiedon löytäminen suuresta ohjelmistosta nousi ongelmaksi. Uusien tietokantakyselyiden kirjoittaminen ei ole mielekästä, jos tarvittava tieto on haettavissa hyödyntämällä olemassa olevia tieto-olioita. Jos tietoa ei ollut valmiina saatavilla, kirjoitettiin uusia SQL-kyselyjä sen hakemiseen tietokannasta. Tiedon yhdistelyssä ja osin myös ryhmittelyssä käytettiin LINQ:a, joka osoittautui käytettävyydeltään hyväksi tekniikaksi. Kävi ilmi, että LINQ on myös niin hyvin optimoitu, että omien toistorakenteiden ohjelmointi tietorakenteiden käsittelyyn on harvoin tarpeellista .NET ympäristössä.

Ohjelmiston eliniän takia osa toiminnallisuuksista oli muuttunut. Joko oli päädytty ohjelmoimaan vanhaa logiikkaa uusiksi uudemmilla tekniikoilla tai muutokset metsätalouden käytännöissä olivat

muuttaneet toimintamallia. Tiedon löytämisen lisäksi piti myös osata korvata vanhentuneet tekniset toiminnallisuudet uudemmilla ja muuttaa logiikkaa niiltä osin, joilta tarve vaati. Ohjelmointitaitojen lisäksi tässä nousi kuvaan metsätalouden toimintamallien tunteminen. Lisäksi se, että raportoinnissa käytettiin osin olemassa olevia olioita ja osin uusiksi luotuja aiheutti riskin huonoon ohjelmistoarkkitehtuuriseen suunnitteluun, jonka seurauksena koko ohjelmiston ylläpidettävyys olisi laskenut. Esimerkiksi tietoa hakeva olio kannatti toteuttaa siten, että sitä voidaan uudelleen käyttää muihin tarkoituksiin.

Tiedonhaussa tuli vastaan myös ongelmat suorituskyvyn ja muistin osalta, haettaessa puukaupan kustannuksia tietyillä hakuehdoilla yhdestä oliosta, varastomääriä toisesta oliosta ja rinnalle useampia oliolistoja, jotka sisälsivät tietoja puutavaralajeista, hakkuutavoista ja vastaavista tiedoista. Näitä tietojoukkoja yhdisteltiin, ryhmiteltiin ja suodatettiin raporttimoottoriluokassa ajetun raportin, käyttäjäoikeuksien ja käyttäjän asettamien hakusuodattimien mukaan. Joillain asiakkailla oli tietokannoissaan niin valtavat tietovarannot, että tiedon hakeminen synkronisesti saattoi kestää sekunneista jopa muutamaan minuuttiin.

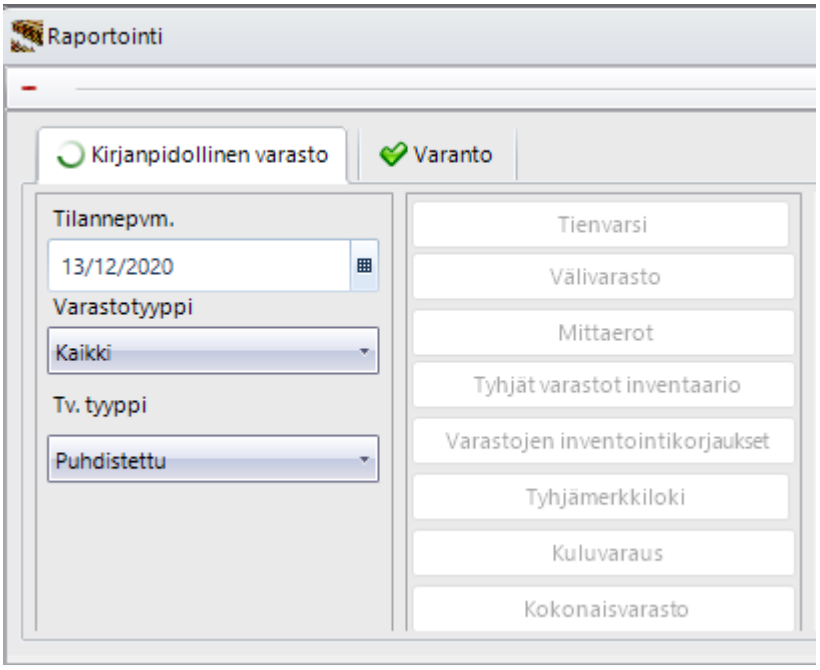


Kuva 9. Asynkronisuus tiedon haussa. (Leppänen, 2020)

Varastoraportointia varten kehitettiin optimointimalli (Kuva 9), jolla pyrittiin nopeuttamaan tiedon hakua.
Välittömästi, kun käyttäjä klikkaa raportointi –valikkokuvaketta, lähettää käyttöliittymä säikeestä erillinen asynkroninen tehtävän raportointimoottorille käskyn alkaa louhia ydintietoa, josta varsinaiset

raportit haetaan. Tämä käyttöliittymä luokassa alustettava tehtävä jakautuu raportointimoottoriluokassa kolmeen alitehtävään, joista esimerkiksi yksi lähtee louhimaan kuluja, toinen varastoarvoja ja kolmas lisätietoja, joita olivat esimerkiksi asiakkaan puutavaralajit ja hakkuutavat. Kun kaikki nämä tehtävät ovat valmiita, yhdistellään saaduista tietojoukoista ydindata. Tämä yhdistely on hyvin nopea toimenpide, vaikka tietoa olisi satoja tuhansia rivejä. Kun ydintieto on valmis, käyttöliittymässä olevat raporttinapit muuttuvat painettaviksi. Kun käyttäjä nyt ajaa raportin, palauttaa raporttimoottori hakuehtojen mukaan suodatetun tietojoukon hyvin nopeasti, sillä ydintieto on valmiina, eikä tässä vaiheessa tarvitse enää tehdä resursseja kuluttavaa tiedonhakua.

Alla (Kuva 10) näkyy osa ”Varasto & varanto” -raportointi-ikkunaa, jossa on kaksi välilehteä. Tässä tapauksessa käyttöliittymä lähettää kaksi asynkronista pyyntöä, joista toinen kehottaa raporttimoottoria louhimaan varastotietoa ja toinen varantotietoa. Kuitenkin nämä raportit käyttävät myös samaa tietoa, joten nämä tehtävät kutsuvat alitehtäviä ensimmäinen ehtii ensin periaatteella, jolloin yhteistä tietoa haettaessa toinen tehtävä odottaa ensin ehtineen valmistumista. Näin optimoidaan muistin käyttöä. Kuvasta (Kuva 10) käy ilmi, että varantotieto on ehtinyt valmiiksi, mutta varastotieto ei. Näin ollen myös varastovälilehdellä olevat raportti painikkeet ovat kuvan mukaisesti estettyinä.



Kuva 10. Asynkronisuus käyttöliittymässä. (Leppänen, 2020)

Arkkitehtuuri, jossa louhittiin aluksi massiivinen ydintieto, aiheutti kehitysvaiheessa muutaman kerran muistin ylivuoto-ongelman. Tämä pakotti eriyttämään raportointikategoriat toisistaan kokonaan. Lisäksi asynkronisuutta hyödynnettiin myös käyttöliittymän osien nopeaan lataamiseen, joten tässä jouduttiin ottamaan laiteresurssit huomioon.

Asynkroninen ohjelmointi vaatii laiteresurssien huomioimista, sillä tietokone ei pysty suorittamaan

tehtäviä asynkronisesti, jos niitä on liikaa. Siksi toteutettiin niin kutsuttu raporttimoottori, eli olio, jonka tarkoitus ensimmäisissä suunnitelmissa oli hakea ja ryhmitellä tietoa. Käyttöönotettu kaupallinen pivot-taulukko-komponentti osasi kuitenkin hoitaa tiedon ryhmittelyn niin tehokkaasti kehittäjältä piilossa, että suuri osa raportointimoottorin toteutettavaksi kaavailluista tehtävistä päätyikin käyttöliittymäluokan hoidettaviksi. Työpöytäsovelluksesta tämä ei aiheuttanut suorituskykyongelmia, sillä ohjelmakoodi suoritettaisiin molemmissa tapauksissa ohjelman käyttäjän koneella tai pilvessä ja käytössä olevan komponentin datan ryhmittely oli valmiiksi optimoitu tehokkaaksi. Web-ympäristö oli toteutusaikaan moderni PWA sovellus, joten voitiin olettaa, että laitteet, joita PiiMegan ympäristö tuki, soveltuivat myös tiedon ryhmittelyyn ilman suorituskykyongelmia.

6.2.3 Tiedon visualisointi ja ryhmittelyt

Käytössä oleva pivot –komponentti osasi luoda hallintapaneelin ryhmittelyille ja visualisoida itsekseen raporttimoottorin palauttaman tyypittämättömän tiedon. Asiakas halusi kuitenkin määrätä, mitä ryhmittelyjä missäkin raportissa on mahdollista asettaa, joten ryhmittely mahdollistettiin kahdella tavalla. Pivot-komponentin oman hallintapaneelin lisäksi toteutettiin "pikaryhmittelyt"-niminen paneeli, jonka avulla käyttäjä pystyy lisäämään rivi- ja sarakeryhmittelyjä sen mukaan, mitä vaihtoehtoja asiakas millekin raportille halusi.

6.2.4 Laskenta

Pivot-taulukko käyttää rivi- ja sarakeryhmittelyjä numeraalisen tiedon laskentaan. Käytössä ollut komponentti säästi paljon kehitys aikaa, sillä perinteisellä taulukkokomponentilla numeroiden laskulogiikka pitää ohjelmoida itse ryhmittelyjen muuttuminen huomioon ottaen. Kuitenkin raportointi uudistuksen vaikein vaihe liittyi juuri laskentaan, sillä käytössä olleeseen komponenttiin piti lisätä räätälöityjä laskenta malleja, jotka tukivat paremmin raportointialusta asiakkaiden metsäliiketoimintaa varten tarjoamia hyötyjä. Yksi tällainen erikoistapaus oli painotettu keskiarvo. Esimerkiksi, puutavaralajien kuutiokohtaisen keskiarvohinnan tarkastelu sopimuksen tasolla ei ole mielekästä, jos yhtä puutavaralajia on ostettu sopimuksella viisi kiintokuutiota ja toista yli sata kiintokuutiota. Tällaisissa tilanteissa perinteinen keskiarvo ei tarjoa puunostajalle eli raportointialustan käyttäjälle liiketoiminnan kannalta mielekästä informaatiota, kun taas määrillä painotettu keskiarvo tarjoaa.

Painotetun keskiarvon lisääminen käytössä olevaan komponenttiin tapahtui ylikirjoittamalla komponentin laskentafunktiota. Haasteena oli se, että käyttöoikeutta komponentin lähdekoodiin ei ollut ja ohjelmointi sekä laskennan oikeellisuuden varmistaminen olivat työläitä tehtäviä. Lisäksi oli tilanteita, joissa haluttiin pivot-taulukon summaavan dataa normaalisti, mutta yhden yksittäisen ryhmittelyvalinnan osalta haluttiinkin näyttää prosenttiarvo. Näiden ohjelmointi oli työlästä, sillä komponentin sisäisiä kehittäjiltä piilossa olevia funktioita jouduttiin ylikirjoittamaan.

6.2.5 Tallentaminen ja käyttöoikeudet

Vaatimustenmäärittelyyn kuului vaatimus raporttien tallentamisesta dynaamisilla päivämääräasetuksilla. Pivot taulukon osalta toteutin toiminnallisuuden, joka serialisoi ryhmittelyt ja aggregaatit xml formaattiin. Tähän komponentin toimittaja tarjosi valmiit koodi esimerkit. Lisäksi lisärajaukset esimerkiksi puutavaralaji tai myyjä serialisoitiin, ja parsittiin ne samaan xml tiedostoon. Päivämäärien osalta xml:ään lisättiin takauma ja aikaväli.

Raporttimoottori vastasi tiedonhaun lisäksi myös tallennuksesta. Luotu xml-tiedosto lähetettiin parametrina raporttityypin- ja kategorian mukaan ja raporttimoottori tallensi nämä tiedot tietokantaan. Lisäksi tietokantaan tallentuivat myös raportin käyttöoikeudet. Raportin tallentamiseen tietokantaan löytyi valmis olio, mutta tallennus vaati kuitenkin ohjelmointityötä ja räätälöintiä uutta alustaa varten. ForestPro:ssa oli seuraavat datanäkyvyydet: hallinto, tiimiesimies ja hankintaesimies. Tiimiesimies sai nähdä vain omien hankintaesimiesten kauppoja koskevaa tietoa, hankintaesimiehet omat kauppansa ja hallintoroolin käyttäjille kaikki tieto oli näkyvillä.

6.2.6 Vienti

Raportointialustalle toteutettiin mahdollisuus viedä raportteja rajauksineen Excelillä luettavaan xlsx-formaattiin tai sähköisien lomakkeiden tallennuksessa yleisesti käytettyyn pdf-formaattiin. Käyttöliittymään lisättiin nappi, jota painamalla vienti tapahtui. Komponentin toimittajan dokumentaatiosta löytyi valmiit lähdekoodit viennin toteuttamiseen, joten ohjelmointityö jäi vähäiseksi. Raportit saattoivat olla horisontaalisesti pitkiä, joten pdf-dokumenttien ulkoasu saattoi olla luettavuudeltaan huono. Excel-documenttien tarkastelu oli helppoa, sillä Excel tarjoaa mahdollisuuden horisontaaliseen vieritykseen.

6.2.7 Lokalisointi

ForestPro oli mahdollista asettaa suomen, englannin, ruotsin, ranskan tai venäjän kielelle. Raportointiuudistuksessa lokalisointi toteutettiin vain suomen ja englannin kielelle. Raportointialustan kääntäminen kaikille kielille ei ollut ohjelmistokehittäjän vastuulla, vaan siitä huolehtivat eri tahot kielestä riippuen.

Lokalisoinnin tekninen toteutus hoidettiin Babylon-ohjelmistolla. Myös järjestelmän muu lokalisointi oli hoidettu samaa ohjelmistoa käyttäen, joten käännökset eivät vaatineet juuri muuta lisätyötä, kuin suomenkielisten tekstien kääntäminen englannin kielelle ja päinvastoin. Raportointialusta toteutettiin ensin suomenkielisenä, mutta toisaalta käytössä olevien komponenttien sisältämät tekstit olivat ensi-

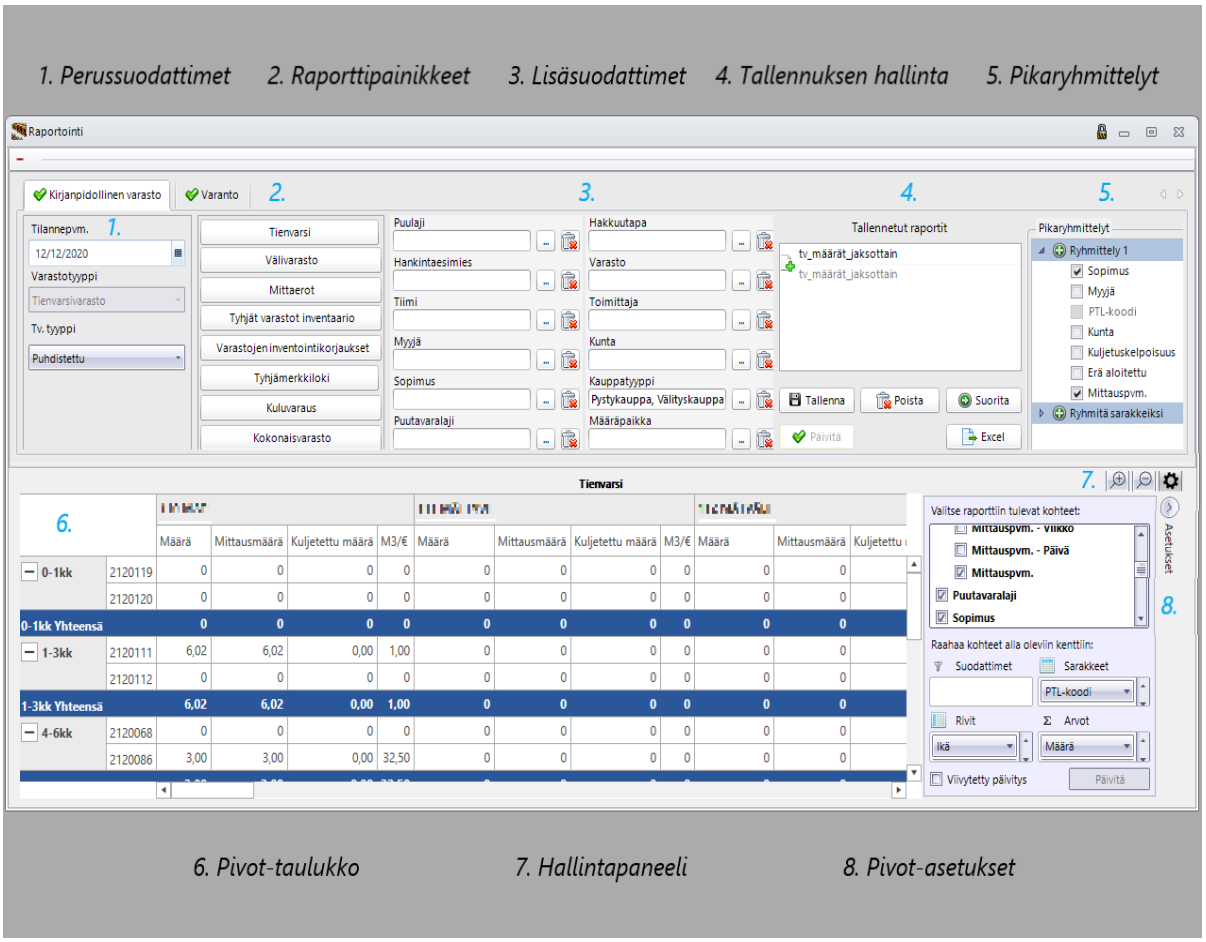
sijaisesti englanniksi, joten lokalisointi oli kaksisuuntaista. Suuri osa tarvittavista teksteistä käännöksineen löytyi valmiiksi järjestelmän käännösresursseista. Ohjelmointiteknisesti lokalisointi oli koko kehitystyön helpoin osuus. Alla (Kuva 11) havainnollistetaan napin lokalisointi.

```
btnRemove.Text = Properties.Resources.MyResources.Remove;
```

Kuva 11. Lokalisointi esimerkki. (Leppänen, 2020)

TOTEUTETUT NÄKYMÄT

Alla olevan kuvan (Kuva 12) mukaisesti ikkuna jaettiin vertikaalisesti kahteen osaan. Kokoon kutistettava yläosa sisältää suurimman osan hallinta työkaluista. Ikkuna voi sisältää välilehtiä kuten kuvan ”Varasto- ja varanto”-lomake. Näkymän alempi puolikas sisältää visualisoidun tiedon sekä pivot-pohjaisissa raporteissa tarkemmat hallintatyökalut.



Kuva 12. Varasto & varanto -raportointinäkymä. (Leppänen, 2020)

Varasto- ja varantoikkunan perustoiminnot numeroituna:

1. Perussuodattimet sisältävät päivämääräsuodattimen, joka on valitun raportin mukaan tilannepäivämäärä tai päivämääräväli. Lisäksi perussuodattimiin kuuluu raporttikohtaisia perussuodattimia. Tällaisia ovat esimerkiksi varastoraportoinnissa varastotyyppi.

2. Raporttipainikkeista suoritetaan haluttu raportti. Napin painaminen lähettää raportointimoottorille käskyn palauttaa valitun raportin tieto valituilla suodattimilla.
Lisäsuodattimilla raportointimoottorille voidaan antaa tietosisältöön kohdistuvia suodattimia. Esimerkiksi pyytää tienvarsivarastotietoa vain tietyn kunnan alueelta.

3. Tallennuksen hallinnassa hoidetaan sekä raporttien tallennus että tallennettujen raporttien poistaminen ja suorittaminen. Myös vientinappi on sijoitettu siihen. Tallennettu raportti sisältää tiedot raporttityypistä, ryhmittelyistä ja dynaamisesti tallennetusta päivämäärä rajauksesta.

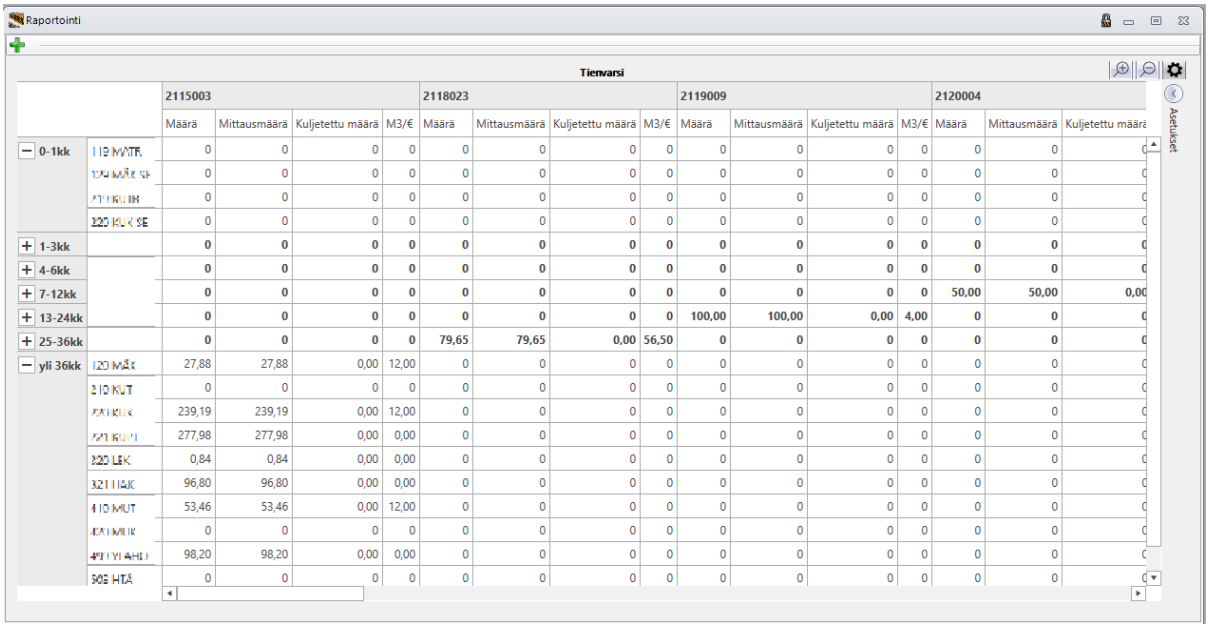
4. Pikaryhmittelyissä käyttäjä voi asettaa rivi ja sarakekohtaisia ryhmittelyvalintoja oman tahtonsa mukaan. Tämä joustava ryhmittely selittää osaltaan nimeä ”dynaaminen raportointi”.

5. Pivot-taulukko visualisoi tiedon ja laskee arvoja rivi- ja sarake-ryhmittelyjen mukaisesti.

6. Hallintapaneelista voidaan muuttaa taulukon ulkoasua sekä piilottaa kuvassa sinisellä värillä näkyvät välisummat.

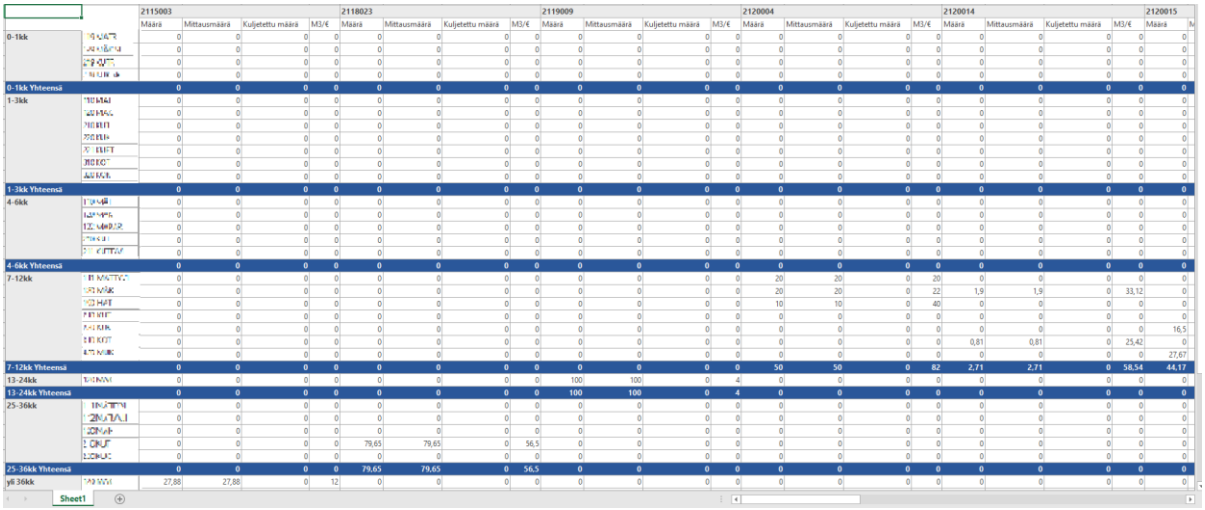
8. Pivot-asetuksista voidaan asettaa ryhmittelyjä kuten, pikaryhmittelyissä, mutta se tarjoaa myös tiedon suodatusvalintoja.

Tavallisessa käyttötilanteessa käyttäjä suorittaa valmiiksi tallennetun raportin, joten asetukset ja työkalut on mahdollista piilottaa varsinaista raportin tarkastelua varten. Tätä havainnollistetaan alla olevassa kuvassa (Kuva 13).



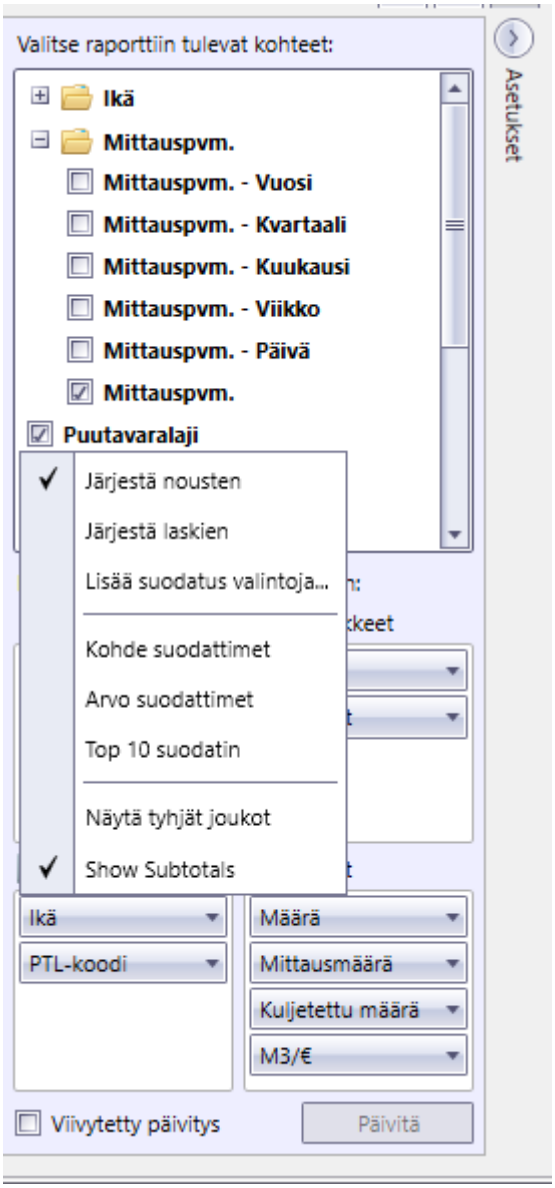
Kuva 13. Pivot-taulukko. (Leppänen, 2020)

Viennin avulla raportteja voidaan tallentaa myös järjestelmän ulkopuolelle. Alla olevasta kuvasta (Kuva 14) käy ilmi, että ulkoasu pysyy samana.



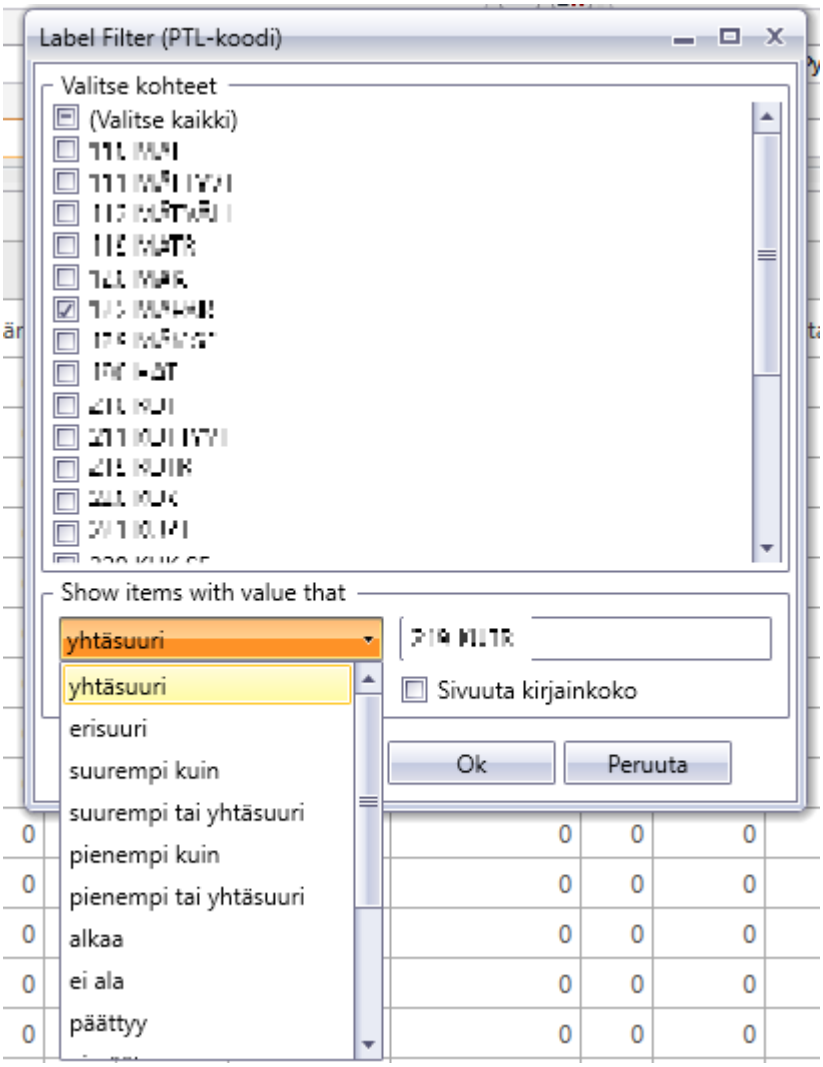
Kuva 14. Excel vienti. (Leppänen, 2020)

Pivot-komponentin toimittaja oli toteuttanut myös hallintapaneelin taulukolle, joka tarjosi hyödyllisiä työkaluja valmiiksi. Kuva (Kuva 15) näyttää, kuinka työkalu osaa poimia tietojoukosta DateTime-tyyppiset kentät ja luoda niille laajat ryhmittely valinnat. Vain lokalisointi, hallintapaneelin yhdistäminen pivot-taulukkoon ja räätälöityjen aggregaattien lisääminen jäivät kehittäjän tehtäviksi.



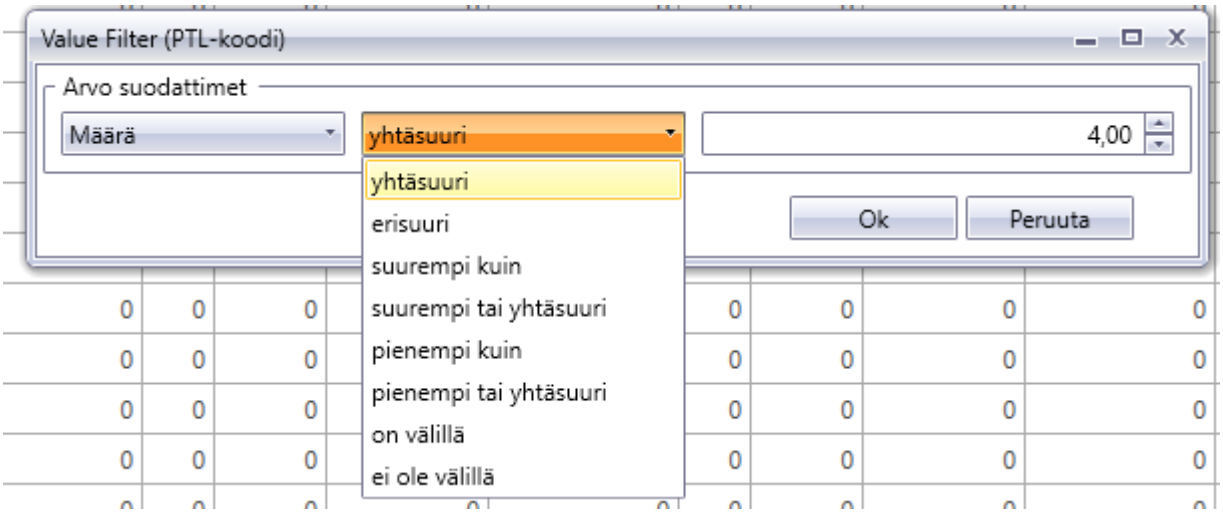
Kuva 15. Pivot-asetukset. (Leppänen, 2020)

Pivot asetukset tarjosivat myös valmiita työkaluja datan suodattamiseen (Kuva 16).



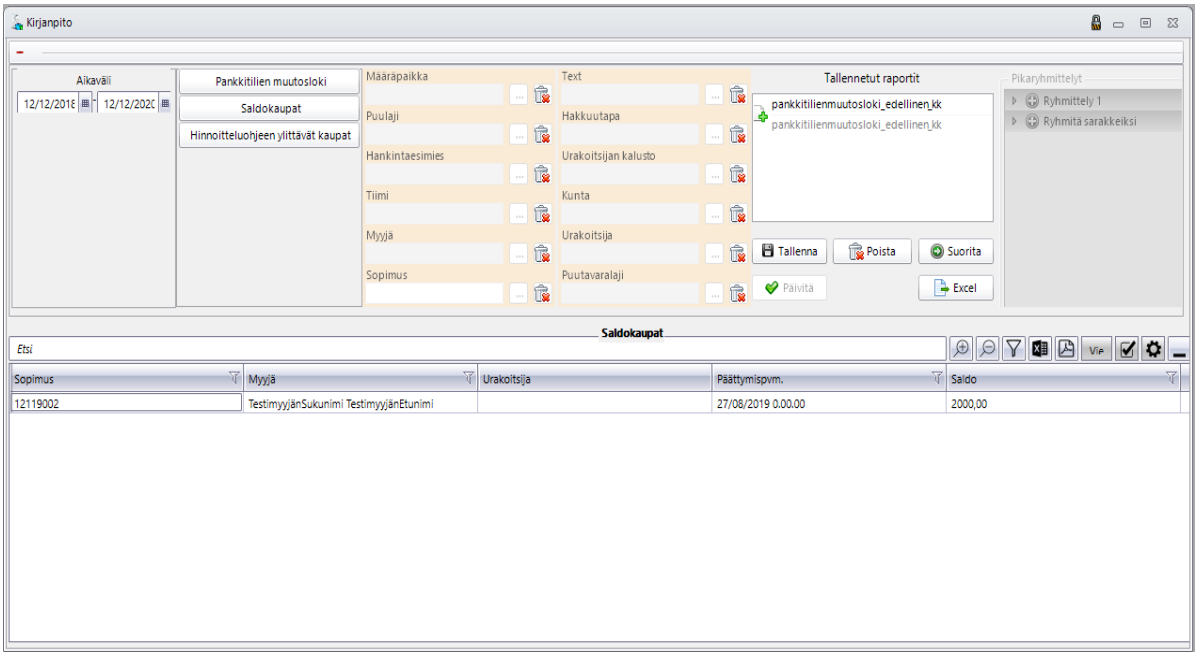
Kuva 16. Kohdesuodattimet. (Leppänen, 2020)

Tietoa pystyi suodattamaan ryhmittelykohteen ja arvon perusteella useampien ehtojen mukaan.



Kuva 17. arvo suodattimet. (Leppänen, 2020)

Kuten toteutusvaiheessa mainitaan, tekstipohjaiset raportit toteutettiin hyödyntäen pivot-taulukon sijasta tavallista taulukkonäkymää. Alla on kuvattuna (Kuva 17) Kirjanpito-ikkuna, jossa valittu ”Saldokaupat”-raportti käyttää tavallista taulukkoa. Lisäksi tilannepäivän sijaan käytössä on aikaväli-rajaus ja käyttö on estetty osaan valinnoista. Tämä toteutus vaati huolellisen suunnittelun, sillä samaan ikkunaan asetettiin useampia komponentteja ja valintoja, joiden käyttö oli eri ehtojen takana.



Kuva 17. Tekstipohjainen raportti ilman pivot-taulukkoa. (Leppänen, 2020)

KORJAUSTARPEET, JATKO- JA OHEISKEHITYS

Alustan kehitystä jatkettiin lisäämällä uusia raportteja sekä parantamalla vakautta ja käytettävyyttä. Dynaamisen raportointialustan ohessa järjestelmään toteutettiin uusia tulostemuotoisia raportteja ja lomakkeita.

8.1 Korjaustarpeet

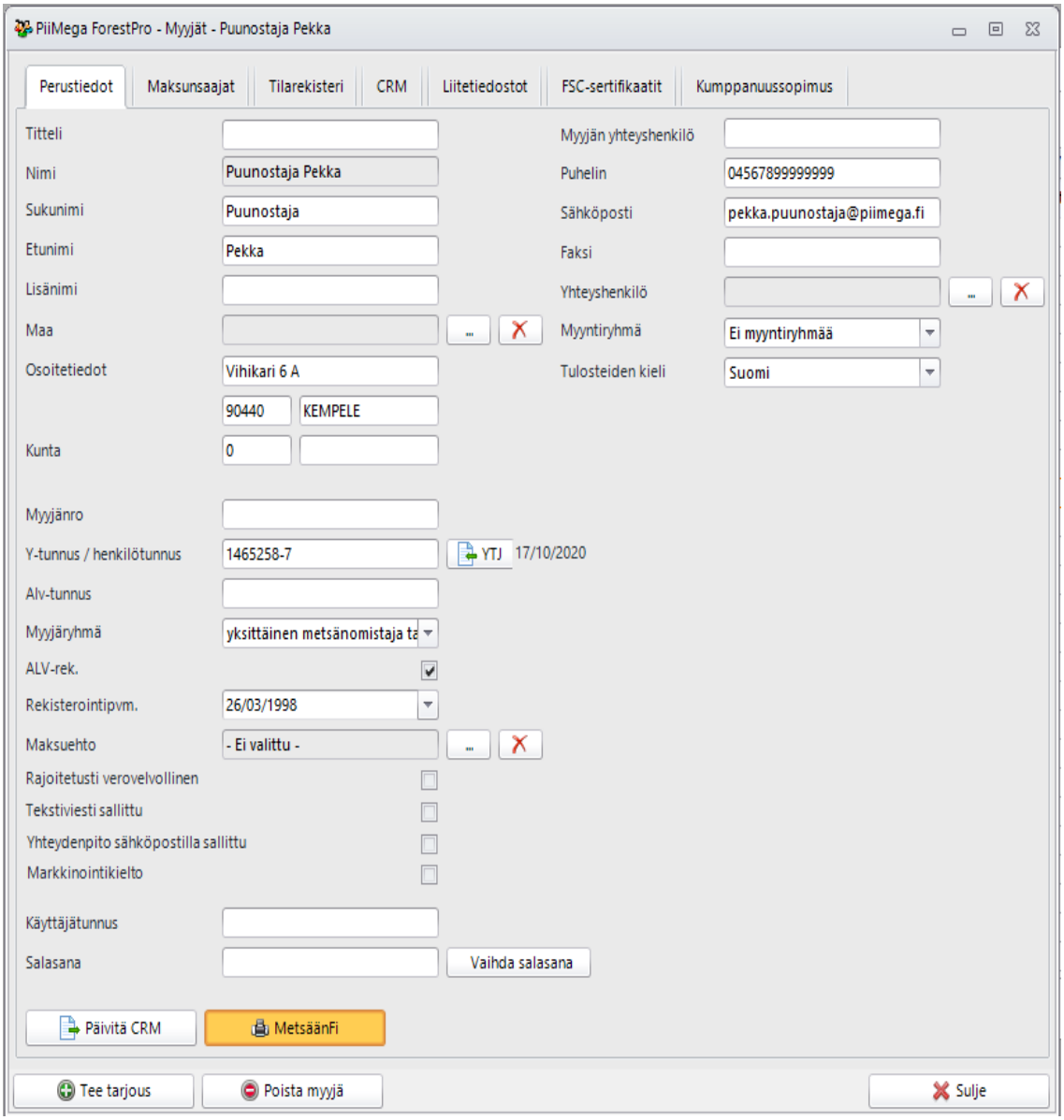
Alustan vakaus, käytettävyys ja tiedon oikeellisuuden varmistaminen aiheutti ongelmia laajassa järjestelmässä, jossa monien asioiden välillä on yhteyksiä. Vaatimustenmäärittely oli jaettu useampaan osaan kategorioiden ja yksittäisten raporttien osalta. Osan toteutus onnistui kerralla hyvin odotettua aikataulua nopeammin, mutta toisaalta vastaan tuli yksittäisiä raportteja, joihin täytyi palata useamman kerran. Eniten jo toteutetun raportin palaamisen johtavia tilanteita aiheuttivat käyttöliittymäongelmat, suurimpana raportin huono luettavuus. Toiseksi eniten ongelmia aiheuttivat ohjelman kaatumiset. Kaatumiset olivat selkeitä virhetilanteita, jotka johtuivat asiakkaiden suorittamista käyttöketjuista, jotka oli jätetty huomioimatta ja testaamatta. Tällaisten virheiden korjaaminen oli nopeaa, yleensä vain muutaman tunnin hommia. Muut korjaustarpeet koskivat tiedon oikeellisuutta ja saavutettavuutta. Näiden korjaaminen vei enemmän aikaa, sillä raportille nousevien väärien numeroiden etsiminen oli välillä työlästä. Saavutettavuuden osalta päädyttiin joitain raportteja niputtamaan yhdeksi raportiksi, jotta asiakkaalle olisi selkeää, mitä raporttia hän haluaa käyttää. Jatkokehitys tarpeeksi jäi luoda näiden tilalle käytettävyyden nimissä omat pikasuodattimet suoraan taulukon yläpuolelle, jotta käyttäjä voi kirjoittaa rajauksia nopeasti.

8.2 Jatkokehitys

Vaikka työkalu päätyi useamman asiakkaan käytettäväksi, oli se vielä monelta osin keskeneräinen. Käytettävyyden hiomista, lokalisointia ja graafisen ulkoasun parantamista jäi tehtäväksi, mutta myös varsinaista uutta kehitystä. Projekti jatkui työkalun jatkokehityksellä, jossa alustaa jatkokehitetään ja uusia raportteja sekä kategorioita lisätään. Varastoraportointia jatkokehitetään lisäämällä uusia raportteja ja yhdistelemällä toisiaan muistuttavia raportteja, jotta käytettävyys paranisi. Myös web-käyttöliittymän kehittämien jatkuu opinnäytetyön suorittamisen jälkeen. Alustalle lisätään myös kategoriat myynti, urakointi, osto ja viranomaisraportointi sekä vielä määrittelemättömiä kategorioita. ForestPro on SaaS-palvelu, eli usean asiakkaan jakama palvelu, jonka käytöstä asiakas maksaa säännöllistä maksua käytössä olevien lisensoitujen ominaisuuksien mukaan. Näin ollen yhden asiakkaan tilaama raportointityökalu siirtyi kaikille asiakkaille, joilla oli riittävän uusi versio ohjelmistosta. Tästä seurasikin lisäkehityspyyntöjä muiden asiakkaiden osalta, koskien muun muassa puun sertifiointiin liittyvää viranomaisraportointia.

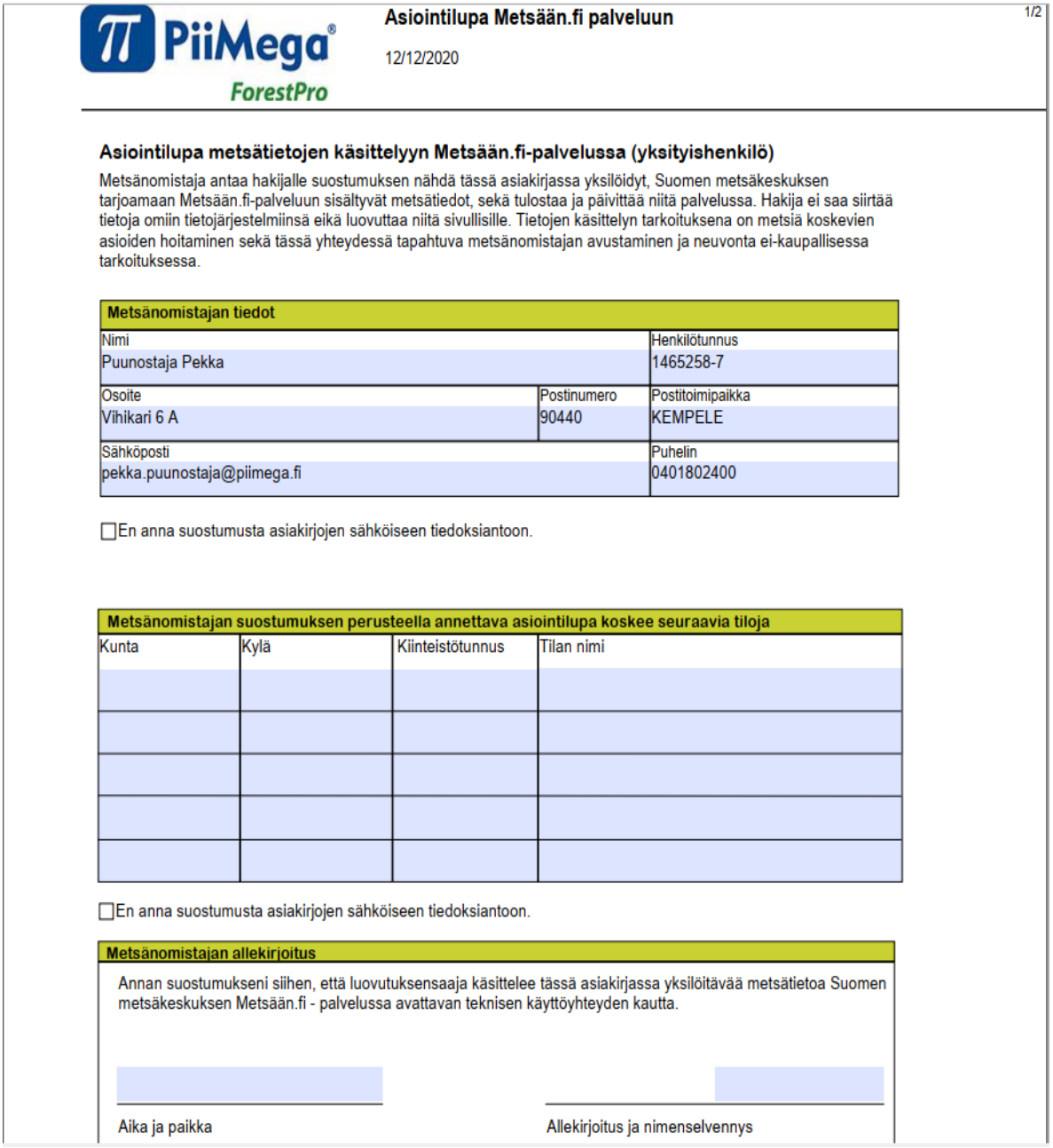
8.3 OHEISKEHITYS

Raportointiuudistuksen määrittelyyn sisältyi myös vaatimus toteuttaa myyjän tiedoilla esitäytetty Metsään.fi tietojen käsittelylupalomake. Metsään.fi palvelu tarjosi metsänomistajille ajantasaista metsävaratieto, jotta nämä voisivat tutustua tarkemmin omistuksiinsa. Metsänomistaja pystyi myöntämään toiselle henkilölle asiointiluvan, joka oikeutti luvan haltijan tekemään metsänkäyttöilmoituksia, Kemera-rahoitushakemuksia ja -toteutusilmoituksia sekä lähettää palveluilmoituksia. ForestPro:n myyjätietojen ylläpito lomakkeelle lisättiin nappi, jota painamalla järjestelmä tulostaa tulosteen, joka on esitäytetty tarkastelun alaisen myyjän tiedoilla. Tätä ei voitu luonnollisesti toteuttaa uudelle taulukkopohjaiselle raportointialustalle, sillä Metsään.fi asiointilupa täytyi lähettää lomakkeena. Tekniikaksi valikoitui ActiveReports, sillä ForestPro ohjelmiston tulosteet olivat toteutettu sillä ennestään. Metsään.fi-lomakkeen generointi tapahtuu klikkaamalla nappia valitun myyjän perustiedoissa alla olevan kuvan (Kuva 18) mukaisesti.



Kuva 18. MetsäänFi-nappi. (Leppänen, 2020)

Avautuvalle lomakkeelle (Kuva 19) saadaan tiedot suoraan järjestelmästä, eikä asiakkaan tarvitse täyttää niitä käsin. Lomakkeen luominen ei ollut tiedonhaun tai logiikan näkökulmasta haastava tehtävä, mutta kaikkineen vei kaksi täyttä työpäivää luoda graafinen malli sekä tiedonhaku.



PiiMega ForestPro

Asiointilupa Metsään.fi palveluun 1/2

12/12/2020

Asiointilupa metsätietojen käsittelyyn Metsään.fi-palvelussa (yksityishenkilö)

Metsänomistaja antaa hakijalle suostumuksen nähdä tässä asiakirjassa yksilöidyt, Suomen metsäkeskuksen tarjoamaan Metsään.fi-palveluun sisältyvät metsätiedot, sekä tulostaa ja päivittää niitä palvelussa. Hakija ei saa siirtää tietoja omiin tietojärjestelmiinsä eikä luovuttaa niitä sivullisille. Tietojen käsittelyn tarkoituksena on metsiä koskevien asioiden hoitaminen sekä tässä yhteydessä tapahtuva metsänomistajan avustaminen ja neuvonta ei-kaupallisessa tarkoituksessa.

Metsänomistajan tiedot		
Nimi Puunostaja Pekka		Henkilötunnus 1465258-7
Osoite Vihikari 6 A	Postinumero 90440	Postitoimipaikka KEMPELE
Sähköposti pekka.puunostaja@piimega.fi		Puhelin 0401802400

☐ En anna suostumusta asiakirjojen sähköiseen tiedoksiantoon.

Metsänomistajan suostumuksen perusteella annettava asiointilupa koskee seuraavia tiloja			
Kunta	Kylä	Kiinteistötunnus	Tilan nimi

☐ En anna suostumusta asiakirjojen sähköiseen tiedoksiantoon.

Metsänomistajan allekirjoitus

Annan suostumukseni siihen, että luovutuksensaaja käsittelee tässä asiakirjassa yksilöitävää metsätietoa Suomen metsäkeskuksen Metsään.fi - palvelussa avattavan teknisen käyttöyhteyden kautta.

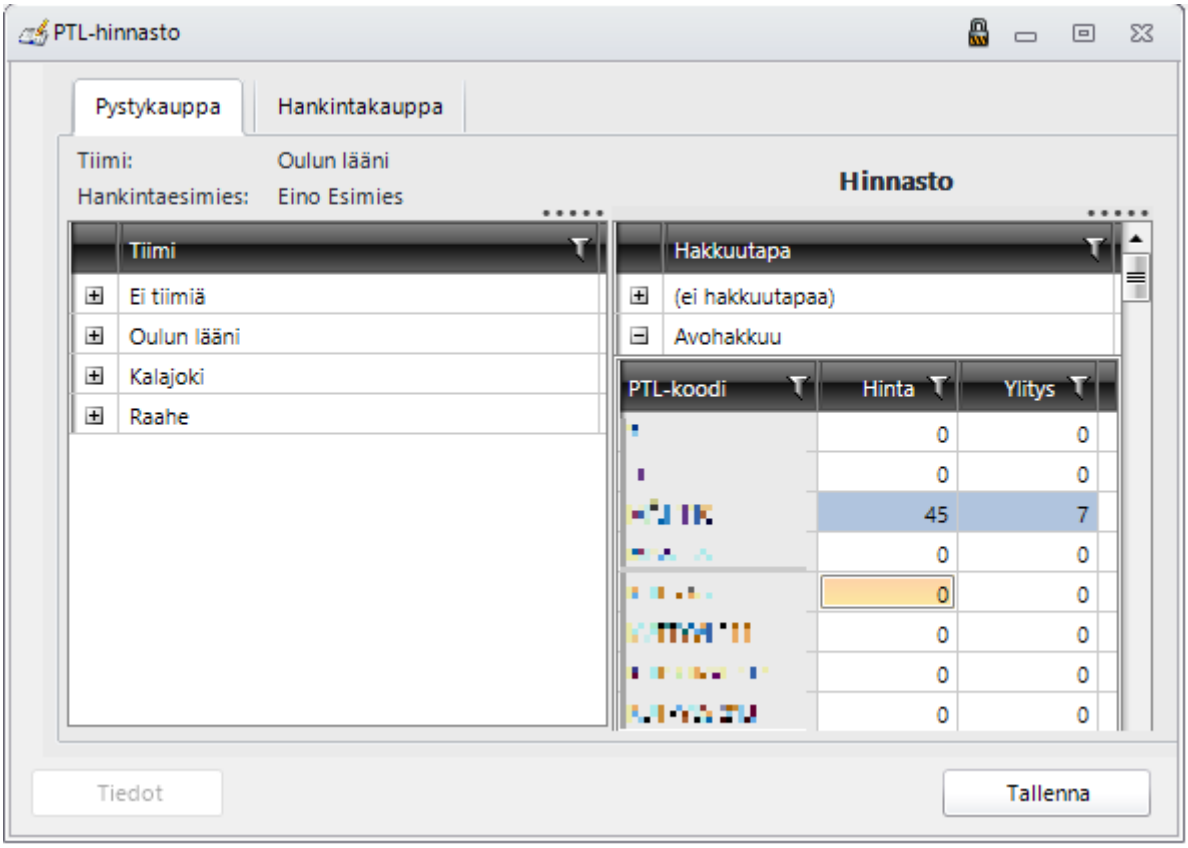
Aika ja paikka

Allekirjoitus ja nimenselvennys

Kuva 19. Metsään.fi lomakkeen alkuosa. (Leppänen, 2020)

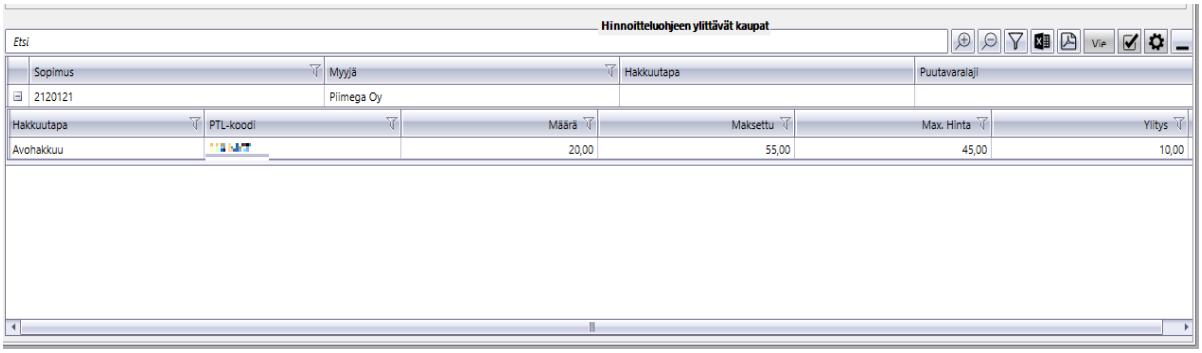
Eräs alustan kirjanpito osioon lisätty raportti sisälsi kaupat, joissa oli ostettu jotain puutavaralajia hinnoitteluohjeita korkeammalla hinnalla. Tätä varten tuli toteuttaa myös suositushinnasto (Kuva 20), jossa käyttäjä pystyi syöttämään laatukohtaisia suositushintoja ja suurimman sallitun ylityksen tälle hinnalle. Jos hankintaesimies ostaa puuta hinnalla, joka ylittää suositushinnan sekä suurimman sallitun ylityksen summan, nousevat tällaiset sopimukset raportille. Suositushinnat annetaan puutavaralaji, kauppatyyppi, hakkuutapa ja tiimi/hankintaesimies yhdistelmälle. Tilanteessa, jossa sekä hankintaesimiehelle että tiimille, johon hän kuuluu, on annettu samalla kauppatyyppi ja hakkuutapa yhdistelmällä suositushinta, hankintaesimiehen henkilökohtainen suositushinta määrää suosituksen. Ohjelmistotekninen haaste oli ottaa kaikki kohdistusvalinnat huomioon. Toteutuksessa päädyttiin

käyttämään niin kutsuttua base classia, joka olio-ohjelmointiparadigman mukaisesti hoitaa suositushinta tietojen tallentamisen, hakemisen sekä muokkaamisen. Tässä olio-ohjelmoinnin hyödyt nousevat hyvin esille. Kehittäjän täytyy ottaa monta asiaa huomioon, mutta kun olion tekee huolella valmiiksi, toimii se aina samalla tavalla. Logiikka ikään kuin piilotetaan konepellin alle ja luotetaan siihen, että tarkoitusta varten huolella ohjelmoitu olio hoitaa tehtävänsä.



Kuva 20. puutavaralajien suositushinnat -ikkuna. (Leppänen, 2020)

Alla olevalle raportille (Kuva 21) on noussut kauppa, jossa hankintaesimies on ylittänyt hänelle ylemmässä kuvassa (Kuva 20) asetetun suositus hinnan mäntytukille. Kuvan raportti on erikoisraportti, jossa on käytetty pivot-taulukon sijasta perinteistä taulukkoa datan visualisointiin.



Kuva 21 Hinnoitteluohjeen ylittävät kaupat. (Leppänen, 2020)

YHTEENVETO

Opinnäytetyön suorittamisen yhteydessä toteutettiin dynaaminen raportointityökalu, joka meni useammalle asiakkaalle tuotantokäyttöön. Lisää asiakkaita oli odotettavissa, sillä ForestPro oli versioitu ohjelmisto ja asiakkaiden versiopäivitykset toivat uusia käyttäjiä. Työn alkaessa osaamiseni ja järjestelmän tuntemukseni eivät olleet riittävällä tasolla, mutta koen kehittyneeni nopealla tahdilla työn edetessä.

POHDINTA

Opinnäytetyöhön sisällytettyä kehitystyötä toteuttaessani ohjelmistosuunnittelutaitoni kehittyivät huomattavasti. Löysin itseni useamman kerran ohjelmoimasta liian nopeasti uusia toiminnallisuuksia sen sijaan, että olisin keskittynyt viimeistelemään työn alla oleva osa-aluetta loppuun. Tekniseltä puolelta LINQ ja SQL taitoni kehittyivät eniten. Opinnäytetyö myös opetti minua eläytymään asiakkaan asemaan hahmottaakseni asiakastarpeita paremmin. Opin paljon myös asiakaspalautteen tärkeydestä.

Kehitystyö vaati myös ForestPro-järjestelmän sekä metsäliiketoiminnan tuntemista. Varsinkin järjestelmän puutteellinen tunteminen rajoitti työskentelyäni työn alkuvaiheessa. WPF tekniikka tuli minulle uutena kesken kehitystyön ja jouduin omaksumaan sen hyvin nopealla aikataululla. Tässä tein jälkeenpäin mietittynä myös virheitä. Hukkasin aikaa yrittäessäni uudelleen kirjoittaa WinForms:lla luomiani toiminnallisuuksia WPF:llä liian kunnianhimoisesti.

Oma osaamiseni ja varsinkin järjestelmän tunteminen kasvoi niin paljon kehitystyönaikana, että pystyisin nyt tekemään ja suunnittelemaan vastaavan raportointityökalun huomattavasti tehokkaammin ja kivuttomammin. Koenkin, että opinnäytetyö ei ollut minulle pelkästään velvollisuus osoittaa oma kypsyyteni oppilaitokselle vaan myös tilaisuus kehittyä ammattilaiseksi. En olisi uskonut tämän prosessin kehittävän minua näin paljon suhteellisen lyhyessä ajassa, sillä ohjelmointikokemusta oli jo useampi vuosi. Ajattelen yhä, että ohjelmointitaitojeni puolesta olin osin jo varsin edistynyt alkaessani toteuttamaan työtä, mutta yllätyin siitä, kuinka monta asiaa suhteellisen selkeässä kehitystyössä on otettava huomioon, kun mukaan luetaan tuotantokelpoisen sovelluksen toimittaminen asiakaskäyttöön. Huomasin myös, että asioiden pitäminen mahdollisimman yksinkertaisena on monesti etu. Ymmärsin, että olen joissain määrin pyrkinyt ohjelmoimaan liian hienosti tai turhia asioita optimoiden yksinkertaisia toiminnallisuuksia. Työssäni olen tavannut paljon itseäni kokeneempia ohjelmistokehittäjiä, joilla on taito toimittaa asiakkaalle toimivia kokonaisuuksia, vaikkei tietoa välttämättä löytyisikään kaikista pikkutarkoista yksityiskohdista. Nämä kokemukset yhdistettynä kehitystyöhön auttoivat minua löytämään kehitystarpeen omassa pyrkimyksessäni toteuttaa asioita liian erikoisella tavalla.

Lopuksi haluan todeta erityishuomiona, että oli mukava toteuttaa natiivisovellus, jossa käytettiin muun muassa WinForms:a, joka ei ollut opinnäytetyön suorittamisen aikana enää moderni teknologia. Vaikka oma taustani on hyvin vahvasti web-painotteinen ja alustariippumattomuus on syystäkin kova sana, uskon, että työpöytäsovelluksilla tulee olemaan pitkä tulevaisuus teollisuusohjelmistoissa. Tuntui mielekkäältä ohjelmoida hyötysovellusta, jonka suuret asiakasmäärät ovat seurausta järjestelmän tarjoamasta hyödystä asiakkaan liiketoiminnalle, eikä vain hienon näköisissä web ja mobiilikäyttöliittymissä. Metsäteollisuus on keskeinen tekijä Suomen hyvinvoinnissa ja ohjelmistokehitykseen liittyy teknisen puolen lisäksi tarkoitus, jota varten työtä tehdään. Kouluprojekteissa tämä asia jäi teknisen puolen varjoon.

LÄHTEET

ActiveReports. .NET Reporting Made Easy. Verkkojulkaisu. Saatavissa: https://www.grapecity.com/activereportsnet. Viitattu 13.12.2020.

Avoinrajapinta. Avoimen rajapinnan määritelmä. Verkkojulkaisu. Saatavissa: http://avoinrajapinta.fi. Viitattu 12.13.2020.

Base Class. Base Class. Verkkojulkaisu. Saatavissa: https://www.techopedia.com/definition/26896/base-class. Viitattu 13.12.2020.

Heinonen, Matti 1996. Funktionaalinen ohjelmointi. Verkkojulkaisu. Saatavissa: http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/funktion. Viitattu 12.13.2020.

Microsoft Oy. LINQ. Verkkojulkaisu. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq. Viitattu 13.12.2020.

Ohjelmointiparadigmat 2019. Ohjelmointiparadigmoja. Verkkojulkaisu. Saatavissa: https://ohjelmointi-19.mooc.fi/osa-7/1-ohjelmointiparadigmoja. Viitattu 13.12.2020.

OOP. Introduction to Object Oriented Programming Concepts (OOP) and More. Verkkojulkaisu. Saatavissa: https://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#OOP. Viitattu 12.13.2020.

Sivonen, Veli-Matti. C#. Verkkojulkaisu. Saatavissa: https://www.cs.helsinki.fi/u/pohjalai/k04/ohpe/seminar/Sivonen-CSharp.pdf. Viitattu 13.12.2020.

SQL. SQL-Overview. Verkkojulkaisu. Saatavissa: https://www.tutorialspoint.com/sql/sql-overview.htm. Viitattu 13.12.2020.

Windows Forms. Verkkojulkaisu. Saatavissa: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0. Viitattu 13.12.2020.

WPF. Verkkojulkaisu. https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019. Viitattu 13.12.2020.

.NET. What is .NET? Verkkojulkaisu. Saatavissa: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet. Viitattu 13.12.2020.