Tero Hakola

Reaaliaikainen tuotannon seurantajärjestelmä

Case: Skaala OY

Opinnäytetyö

Kevät 2015

Tekniikan yksikkö

Tietotekniikan koulutusohjelma

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Koulutusohjelma: Tietotekniikan koulutusohjelma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Tero Hakola

Työn nimi: Reaaliaikainen tuotannon seurantajärjestelmä

Ohjaaja: Markku Lahti

Vuosi: 2015 Sivumäärä: 45 Liitteiden lukumäärä: 0

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa järjestelmä, jolla voidaan seurata Skaalan päivittäisiä tuotantomääriä reaaliaikaisesti. Järjestelmän tuli valmistaa helposti luettavia raportteja tuotantomääristä, jolloin tuotannon mahdollisiin ongelmatilanteisiin reagoiminen tapahtuisi nopeammin.

Työssä rakennettiin ohjelmistotuotannon prototyyppimallin mukaisesti itsenäisesti toimiva ohjelmisto asiakasvaatimusten pohjalta.

Ohjelmiston rakentamisessa hyödynnettiin Microsoftin Visual Studio -kehitystyökalua ja Microsoftin SQL Server Management Studiota. Ohjelmointikielinä käytettiin SQL-, HTML- ja C#-kieltä.

Työn lopputuloksena syntyi itsenäisesti toimiva tietokantasovellus, joka rakentaa tuotannon määristä raportteja yrityksen sisäverkkoon HTML-muodossa.

Avainsanat: Ohjelmistotuotanto, Skaala, Tietokantasovellus, SQL, C#

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Tero Hakola

Title of thesis: Real time production observation system

Supervisor: Markku Lahti

Year: 2015 Number of pages: 45 Number of appendices: 0

The aim of this thesis was to plan and develop an independent system for real time production observation. The system was created to visualize daily production values so reacting to problematic situations would become faster.

The system was developed with Microsoft Visual Studio and Microsoft SQL Server Management Studio. The programming languages used were SQL, HTLM and C#.

The end product was an independent database application which creates database reports in the HTML form. The reports are created in the company's IIS server.

Keywords: software engineering, Skaala, Database application, SQL, C#

SISÄLTÖ

Opinnäytetyön tiivistelmä....2
Thesis abstract....3
SISÄLTÖ....4
Kuvio- ja taulukkoluettelo....6
Käytetyt termit ja lyhenteet....7
1 JOHDANTO....8
1.1 Työn tausta....8
1.2 Työn tavoite....8
1.3 Työn rakenne....9
1.4 Yritysesittely....9
2 OHJELMISTOKEHITYS....10
2.1 Ohjelmiston elinkaari....10
2.2 Prototyyppimalli....11
3 KÄYTETYT TEKNIIKAT JA TYÖKALUT....13
3.1 Tietokanta....13
3.1.1 Tietokannan hallintajärjestelmä....13
3.1.2 SQL-kyselykieli....14
3.2 Microsoft Visual Studio....15
3.3 C#-ohjelmointikieli....15
3.4 .NET Framework....16
3.5 Windows Service Application....16
4 SKAALA OY:N TOIMINNANOHJAUSJÄRJESTELMÄ JA KUITTAUKSET....18
4.1 Tuotannon vaiheet....18
4.2 Viivakoodien käyttö tuotannossa....20
5 TIETOKANTASOVELLUKSEN SUUNNITTELU JA TOTEUTUS....22
5.1 Ensimmäinen versio....22
5.1.1 Vaatimukset ja toimintamallin määrittely....22
5.1.2 Ensimmäisen version toteutus....23

5.2 Toinen versio ja dynaaminen lähestymistapa 28
5.2.1 Toisen version vaatimukset ja toimintamalli 28
5.2.2 Toisen version toteutus 29
5.3 Muunto palvelinversioksi 34
5.3.1 Palvelinversion vaatimukset 34
5.3.2 Dynaamisen version muunto palvelinversioksi 34
5.3.3 Palvelinversion asennus ja käynnistys 36
5.4 Lukijan rakentaminen 36
5.4.1 Lukijan vaatimukset ja toimintamalli 37
5.4.2 Lukijan toteutus 37
6 YHTEENVETO 42
LÄHTEET 43

Kuvio- ja taulukkoluettelo

Kuvio 1. Prototyyppimallin elinkaari (perustuu Immonen 2002) ... 12
Kuvio 2. Microsoft SQL Server Management Studio ... 14
Kuvio 3. Esimerkki SQL-kyselystä ... 15
Kuvio 4. Microsoft Visual Studio Express 2013 ... 15
Kuvio 5. Esimerkki C#-ohjelmointikielestä ... 16
Kuvio 6. Tuotannon vaiheet ... 19
Kuvio 7. Tuotetarra ja viivakoodi ... 21
Kuvio 8. Psionin Workabout pro3 -viivakoodinlukija ... 21
Kuvio 9. Ensimmäisen version toimintamalli ... 23
Kuvio 10. Esimerkki tietokantayhteyden määrittelystä ... 24
Kuvio 11. Esimerkki tietokantakyselyn määrittelystä ... 24
Kuvio 12. Tuotantolinja-luokan toteutus ... 25
Kuvio 13. Tuotantolinja-luokkien erittely tuotantopaikka-listoihin ... 25
Kuvio 14. HtmlGen-luokan toteutus ensimmäisessä versiossa ... 27
Kuvio 15. Ensimmäisellä versiolla rakennettu sivu ... 28
Kuvio 16. Dynaamisen version toimintamalli ... 29
Kuvio 17. App.config-asetustiedoston rakenne ... 30
Kuvio 18. Yhteystietojen lukeminen app.config-asetustiedostosta ... 31
Kuvio 19. LuoIndeksi-metodin linkkien rakentaminen ... 31
Kuvio 20. Sivuista tehty hakemisto ... 32
Kuvio 21. Ajastimen määrittely ... 32
Kuvio 22. Tim_tick-metodin toteutus ... 33
Kuvio 23. Palvelun lokitiedostojen määrittely ... 35
Kuvio 24. OnStart-metodin toteutus ... 35
Kuvio 25. Ohjelman rakentamia sivuja ... 36
Kuvio 26. Lukijan toimintamalli ... 37
Kuvio 27. Lukijan app.config-asetustiedosto ... 38
Kuvio 28. Lukijan sisäiset asetukset ... 39
Kuvio 29. Form1_Load-metodin toteutus ... 40
Kuvio 30. Refreshtimer_Tick-metodin toteutus ... 40
Kuvio 31. IdlecheckTimer_Tickin toteutus ... 41

Käytetyt termit ja lyhenteet

.NET Framework Luokkakirjasto, jota Visual Studiolla kehitetyt ohjelmat käyttävät.

IIS Microsoftin www-palvelinohjelmisto (Internet Information Services).

Luokka Ohjelmakomponentin muotti, joka sisältää tarvittavat muuttujat ja toiminnot.

Olio Luokasta tehty ilmentymä.

Prototyyppimalli Ohjelmistokehityksen malli, jossa ohjelma valmistetaan lyhyissä sykleissä, lisäten joka kierros jotain uutta.

SQL Kyselykieli (Structured Query Language).

Toiminnanohjausjärjestelmä
Yriykseen eri toiminnat yhdistävä ohjelmistokokonaisuus.

1 JOHDANTO

1.1 Työn tausta

Yrityksen toiminnassa käytetään jatkuvasti tietokantaa. Päivittäin tietokantaan tehdään päivityksiä niin tuotannon, kuin työsuunnittelunkin toimesta. Tuotantomäärien seuranta vuorojen aikana on kuitenkin ollut melko hankalaa ja usein määrät on laskettu käsin vasta vuoron vaihtuessa. Tästä johtuen haluttiin ohjelma, joka suorittaisi määrien haun ja laskemisen automaattisesti, sekä esittäisi tuloksen yksinkertaisessa muodossa. Pitkän tähtäimen visiona on tuotantoon sijoitetut näytöt, jotka esittävät kuluvan päivän tuotantomäärää. Näiden avulla sekä työnjohto että tuotantohenkilöstö voisivat sopeuttaa työtahtia vastaamaan päivittäisiä tavoitteita. Tämän lisäksi mahdollisiin ongelmatilanteisiin pystyttäisiin puuttumaan nopeammin.

Automaattisen seurantajärjestelmän rakentamista auttaa se, että yritys käyttää viivakoodeja ja viivakoodinlukijoita. Tämä poistaa tietokantapäivityksistä inhimillisiä virheitä ja mahdollistaa reaaliaikaisen määrien seurannan.

1.2 Työn tavoite

Tuotannossa on nähty ongelmana se, ettei henkilöstöllä ole tietoa paljonko vuoron tavoitteesta on saavutettu. Tästä voi muodostua ongelmia myös työnjohdolle, jos tavoitteista jäämistä ei huomata ennen vuoron päättymistä.

Työn tavoitteena on suunnitella ja toteuttaa työkalu, jonka avulla pystytään esittämään tuotantomäärät yksinkertaisessa muodossa mahdollisimman reaaliaikaisesti. Tällöin sekä tuotantohenkilöstöllä että työnjohdolla on mahdollisuus vaikuttaa asioihin nopeammin.

1.3 Työn rakenne

Johdannon jälkeen työssä kerrotaan ohjelmistokehityksestä yleisesti luvussa kaksi. Tämän jälkeen luvussa kolme esitellään käytetyt tekniikat ja työkalut. Teoriaosuuden jälkeen luvussa neljä esitellään pääpiirteittäin, miten yrityksen toiminnanohjausjärjestelmä ja tuotanto toimii. Luvussa viisi käydään läpi työn suunnittelu ja toteutus. Lopuksi luvussa kuusi käydään läpi yhteenveto, jossa pohditaan työn kehitystä ja lopputulosta.

Työssä sovelletaan Microsoftin .NET Framework -ohjelmistoarkkitehtuurin luokkia. Luokkien käyttö perustuu Microsoftin omaan .NET dokumentaatioon. (Microsoft 2014i.)

1.4 Yritysesittely

Skaala Oy on Ylihärmästä lähtöisin oleva ikkunoiden, ovien ja lasien palvelutoimittaja. Yritys on perustettu vuonna 1956. Yrityksen liikevaihto vuonna 2014 oli 95 M€ ja yritys työllistää yli 500 henkilöä. Ylihärmän lisäksi yrityksellä on tehtaita Alahärmässä, Karvialla, Kuortaneella, Kurikassa, Vetelissä, Lahdessa ja Pietarissa. (Skaala Oy 2015.)

Yritys on erikoistunut ovien ja ikkunoiden valmistukseen. Yrityksen tiloissa toimii myös eristyslasitehdas. Tämän lisäksi yritys on laajentanut toimintaansa myös parveke- ja terassilasivalmistukseen. (Skaala Oy 2015.)

Yrityksen suurin markkina-alue on Suomi. Vientituotantoa on Ruotsiin, Iso-Britanniaan ja Venäjälle. Yritys tunnetaan energiatehokkaasta tuotevalikoimasta ja onkin energiatehokkuuden edelläkävijöitä. (Skaala Oy 2015.)

2 OHJELMISTOKEHITYS

Ohjelmistotuotantoa on ollut niin kauan kuin tietokoneitakin. Ohjelmistokehitys käsittää koko ohjelman tuotannon aina asiakastarpeesta lähtien valmiiseen ohjelmaan asti. Ohjelmistotuotannon yleistyessä ajauduttiin tilanteeseen, jossa kasvavat vaatimukset aiheuttivat valtavasti virheitä. On syntynyt tarve kehittää erilaisia kehitysmalleja, jotka selkeyttävät ja tukevat ohjelmiston kehitystä. (Immonen 2002.)

Yleisimpiä malleja ovat vesiputousmalli, prototyyppimalli ja nykyisin yleistyneet ketterät menetelmät (Lappalainen 2011).

Vesiputousmalli on näistä perinteisin. Siinä ohjelma rakennetaan pala kerrallaan jolloin sen kehitys muistuttaa vesiputousta. Vesiputousmallissa jokainen rakennettu pala antaa aina lähtökohdat seuraavalle. Näin paloista muodostuu lopulta kokonainen ohjelma. Ongelmana voidaan nähdä se, että lopputulosta on vaikea nähdä alkuvaiheessa. (Immonen 2002.)

Prototyyppimallissa rakennetaan tuotteen käyttöliittymä ja kierrätetään sitä asiakkaalla lisäten toimintoja joka kierroksella. Etuna tässä on se, että asiakkaalla on mahdollisuus muuttaa vaatimuksia pitkin kehityskaarta. (Immonen 2002.)

Ketterät menetelmät ovat yleistyneet johtuen nopeasti muuttuvista ympäristöistä. Ketterissä menetelmissä ohjelmistoa rakennetaan pienissä osissa nopealla syklillä. Näin mahdolliset virheet voidaan korjata nopeasti. Muuttuviin tilanteisiin pystytään reagoimaan nopeasti. (Lappalainen 2011.)

2.1 Ohjelmiston elinkaari

Ohjelmiston elinkaari sisältää koko ohjelman kehityksen asiakastarpeesta ylläpitoon. Elinkaaren yleisimmät vaiheet ovat määrittely, suunnittelu, toteutus, käyttöönotto ja ylläpito. Vaiheet voivat kuitenkin olla erilaiset eri projekteilla. (Helsingin Yliopisto 2009.)

Määrittelyvaiheessa asiakkaalla on ongelma tai tarve, joka vaatii ratkaisua. Asiakas selvittää ohjelmiston kehittäjän kanssa vaatimukset ja mahdolliset rajoitteet toiminalle. Tavoitteena on kattava dokumentointi asiakasvaatimuksista. (Helsingin Yliopisto 2009.)

Suunnitteluvaiheessa suunnitellaan ohjelmiston sisäinen arkkitehtuuri. Tämä sisältää ohjelman eri osat ja niiden väliset rajapinnat. Suunnitteluvaiheen tulosta verrataan määrittelyvaiheen tuotokseen ja pyritään täyttämään sen vaatimukset. (Helsingin Yliopisto 2009.)

Toteutusvaihe sisältää itse ohjelmoinnin ja ohjelman testauksen. Toteutuksen rakentaminen on melko suoraviivaista, jos edelliset vaiheet ovat huolellisesti tehtyjä. (Helsingin Yliopisto 2009.)

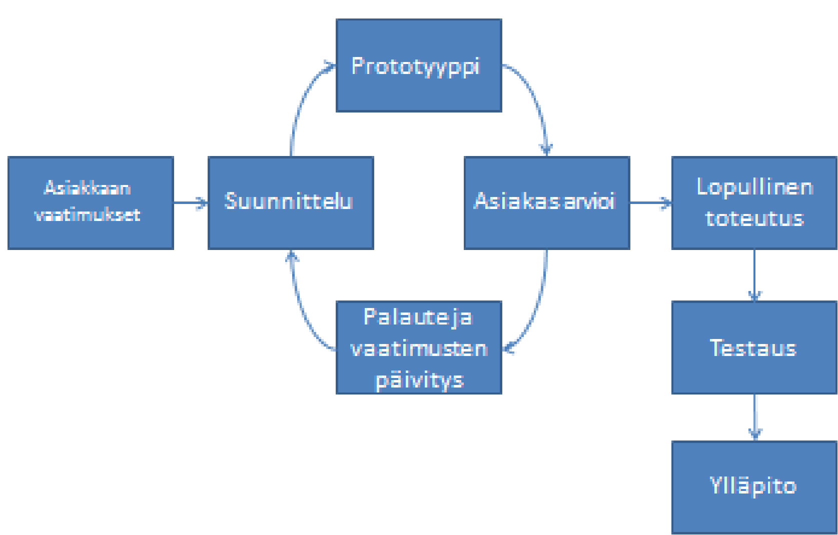
Käyttöönottovaiheessa ohjelma otetaan käyttöön sen oikeassa toimintaympäristössä. Mahdollisuuksien mukaan ohjelman käyttöönottoa voidaan myös kokeilla testiympäristössä. Käyttöönottovaiheessa myös huolehditaan siitä, että asiakas osaa käyttää ohjelmaa. Tätä tukee hyvin dokumentoidut käyttöohjeet. (Helsingin Yliopisto 2009.)

Ylläpitovaihe kattaa koko ohjelman jäljellä olevan elinkaaren ja on suurin vaiheista. Se sisältää ohjelman ylläpitämisen muuttuvassa ympäristössä ja tästä johtuvien virheiden korjauksen. Ylläpitovaiheessa ohjelmaa voidaan myös parantaa entisestään lisäämällä siihen toimintoja. (Helsingin Yliopisto 2009.)

2.2 Prototyyppimalli

Prototyyppimallin perusideana on valmistaa nopeasti toimiva prototyyppi ja rakentaa sen pohjalta uusia toimintoja lisäillen toimiva tuote. Prototyyppimallin elinkaari alkaa normaalisti asiakasmäärittelyllä, joka voi olla hyvinkin summittainen. Tästä rakennetaan nopea suunnitelma ja toteutetaan prototyyppi. Valmista prototyyppiä esitetään asiakkaalle ja arvioidaan, kuinka hyvin prototyyppi vastaa asiakkaan toiveita. Tässä vaiheessa asiakas päättää halutaanko prototyyppiin lisätä toimintoja, vai onko se valmis toteutettavaksi. Jos prototyyppiin halutaan lisätä toimintoja, se lähtee uudelle kierrokselle. Jokaisen kierroksen

jälkeen tuote hyväksytetään uudelleen asiakkaalla. Tätä spiraalimaista kiertoa jatketaan, kunnes tuote on asiakkaan mielestä valmis. Tämän jälkeen siirrytään lopulliseen toteutusvaiheeseen, jonka lopputuloksena syntyy toimiva tuote. Prototyyppimallin elinkaari on kuvattu kuviossa 1. (Taina 2000.)



Kuvio 1. Prototyyppimallin elinkaari (perustuu Immonen 2002)

3 KÄYTETYT TEKNIIKAT JA TYÖKALUT

3.1 Tietokanta

Tietokannaksi luetaan kokoelma toisiinsa yhteydessä olevia tietoja. Nykyisin lähes kaikki tietokannaksi luokiteltavat tietovarastot ovat sähköisiä ja niitä ylläpidetään erilaisilla hallintajärjestelmillä. (Techterms.com 2009.)

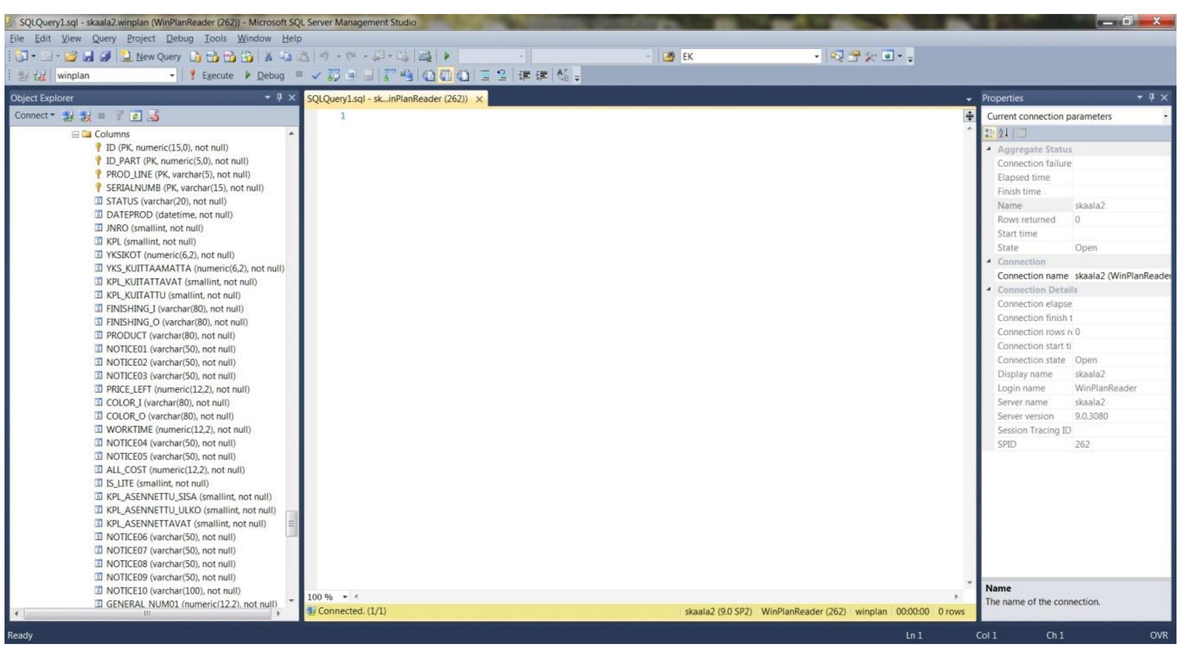
Yleisin käytössä oleva tietokantamalli on relaatiotietokanta. Relaatiotietokannaksi määritellään tietokanta, jonka taulut on yhdistetty toisiinsa avaimilla. Yleisesti avaimena toimii jokin tunniste, jonka avulla taulun tietueet voidaan erottaa toisistaan. Avaimen avulla voidaan toisiinsa liittyvää tietoa hakea useasta taulusta eri parametreilla. (Sarja 2006.)

Tietokannan ytimenä toimii tietokantamoottori. Tietokantamoottori on ohjelmistokomponentti, jonka ympärille ja ehdoilla tietokanta rakennetaan. (Microsoft 2015.)

3.1.1 Tietokannan hallintajärjestelmä

Tietokannan hallintajärjestelmä on työkalu tietokannan ohjaukseen. Erilaisia hallintajärjestelmiä on useita ja näistä yksi suosituimmista on Microsoftin SQL Server Management Studio. Hallintajärjestelmä toimii rajapintana ylläpitäjän ja tietokantamoottorin välillä. (Tutorialspoint 2014.) Kuviossa 2 on kuvattu Microsoftin SQL Server Management Studion käyttöliittymä.

Tietokannan hallintajärjestelmän tehtävänä on luoda ja hallita varsinaisen tietokantamoottorin ympärille rakennettua tietokantaa. Hallintajärjestelmän avulla pystytään muokkaamaan tietokannan rakennetta ja tietoja. Tämän lisäksi hallintajärjestelmällä pystytään ohjaamaan tai tarkkailemaan itse tietokantamoottorin toimintaa. Tämä on usein tärkeää optimoitaessa tietokannan käyttöä ja resurssien kulutusta. (Tutorialspoint 2014.)



Kuvio 2. Microsoft SQL Server Management Studio

3.1.2 SQL-kyselykieli

Tietokannan hallinnassa käytetään omaa kyselykieltä. Tässä työssä kyselykielenä toimi IBM:n kehittämä standardoitu SQL-kyselykieli (Structured Query Language). Sen tehtävänä on määrittää tapahtuma, joka tietokantaan halutaan suorittaa. Tapahtuma voi olla esimerkiksi kysely, tiedon lisääminen, tiedon muuttaminen tai tiedon poistaminen. Kysely syötetään hallintajärjestelmään, joka suorittaa haun tietokantaan kyselyn sisältämillä parametreilla ja näyttää vastausjoukon käyttäjälle. (W3schools 2014.)

SQL-kieli sisältää peruskomennot tietokantaobjektien ja näiden sisältämän tiedon hallintaan. Tärkeimmät operaatiot ovat Select, Insert, Delete ja Update. Tämän lisäksi kielen avulla pystytään myös määrittelemään muuttujia ja yksinkertaisia ehtolauseita, jolloin pystytään rakentamaan monimutkaisiakin hakuja. (W3schools 2014.) Kuviossa 3 on kuvattu SQL-kielellä rakennettu tietokantakysely.

```sql
SELECT t.prod_line, sum(t.kpl * r.units_row)
FROM tj_kuittaukset t, rows r
WHERE t.pvm>(select convert(varchar(10),getdate(),120)) and type=0
and (t.prod_line='KOTIM' or t.prod_line='EK' or t.prod_line='VINO' or t.prod_line like'OVI%')
and r.id = t.id and r.id_part = t.id_part and r.id_row = t.id_row
GROUP BY prod_line
```

Kuvio 3. Esimerkki SQL-kyselystä

3.2 Microsoft Visual Studio

Visual Studio on Microsoftin ohjelmankehitysympäristö. Se mahdollistaa useiden eri ohjelmointikielten käytön ja helpon graafisten ohjelmistojen valmistamisen. Visual Studio tukee .NET-ohjelmistokehystä. Ohjelma tarjoaa kehittäjille monenlaisia valmiita pohjia eri ohjelmien rakentamiseen. (Microsoft 2014a.) Kuviossa 4 on kuvattu Visual Studion käyttöliittymä.

Kuvio 4. Microsoft Visual Studio Express 2013

3.3 C#-ohjelmointikieli

C# eli C-sharp on Microsoftin kehittämä oliopohjainen ohjelmointikieli. Se julkaistiin heinäkuussa 2000. Kielen tavoitteena on olla yksinkertainen, nykyaikainen

oliopohjainen kieli, joka mukailee C- ja C++-kieltä. (Ecma international 2006.) Kuviossa 5 on esimerkki C#-kielestä.

```
try
{
    for (int i = 0; i < section.Yhteyslista.Count; i++)
    {
        Yhteystieto y = new Yhteystieto();
        y.connection = section.Yhteyslista[i].Yhteysstring;
        string filename = section.Yhteyslista[i].Sqlstring;
        using (StreamReader sr = new StreamReader(filename))
        {
            y.sqlstring = sr.ReadToEnd();
        }
        y.pagename = section.Yhteyslista[i].Sivunnimi;
        y.otsikko = section.Yhteyslista[i].Otsikko;
        tietolista.Add(y);
        HtmlGen indeksi = new HtmlGen();
        indeksi.LuoIndeksi(tietolista);
    }
}
catch (Exception ex)
{
    eventLogSPM.WriteEntry(ex.ToString());
}
```

Kuvio 5. Esimerkki C#-ohjelmointikielestä

3.4 .NET Framework

.NET on Microsoftin kehittämä ohjelmistoarkkitehtuuri. Se koostuu kahdesta eri osasta, luokkakirjastosta ja virtuaalisesta ajoympäristöstä. Luokkakirjasto (Framework Class Library, FCL) sisältää tarvittavat kirjastot. Virtuaalinen ajoympäristö (Common Language Runtime, CLR) toimii ohjelmien ajamisen virtuaaliympäristönä ja tarjoaa näin paremman tietoturvan, muistinhallinnan ja virheidenhallinnan. (Microsoft 2014b.)

3.5 Windows Service Application

Windows Service Application tarkoittaa palvelua. Se on ohjelma, jota ajetaan taustalla omana istuntona. Palvelut ovat yleisiä ohjelmia palvelinympäristössä,

jossa ohjelmien täytyy toimia ilman käyttäjän jatkuvaa kirjautumista. (Microsoft 2014c.)

Palvelu eroaa hieman normaalista ohjelmasta. Suurin ero on siinä, että palvelua ei pysty käynnistämään normaalin ohjelman tapaan käynnistystiedostolla, vaan se on erikseen asennettava järjestelmään. Asennukseen on olemassa erilaisia ohjelmia esimerkiksi Microsoftin InstallUtil.exe (Microsoft 2014h). Asennuksen jälkeen palvelu voidaan käynnistää palvelimen hallintatyökalun avulla.

Toinen ero on palveluna rakennetun ohjelman rakenteessa. Palvelun pääohjelmassa on ainoastaan sen käynnistys. Käynnistys kutsuu itse suoritettavaa ohjelmaa, joka sisältää metodit käynnistykselle, lopetukselle, pysäytykselle ja jatkamiselle. Näihin metodeihin sisällytetään toiminnot, jotka suoritetaan kutsuttaessa. (Microsoft 2014c.)

4 SKAALA OY:N TOIMINNANOHJAUSJÄRJESTELMÄ JA KUITTAUKSET

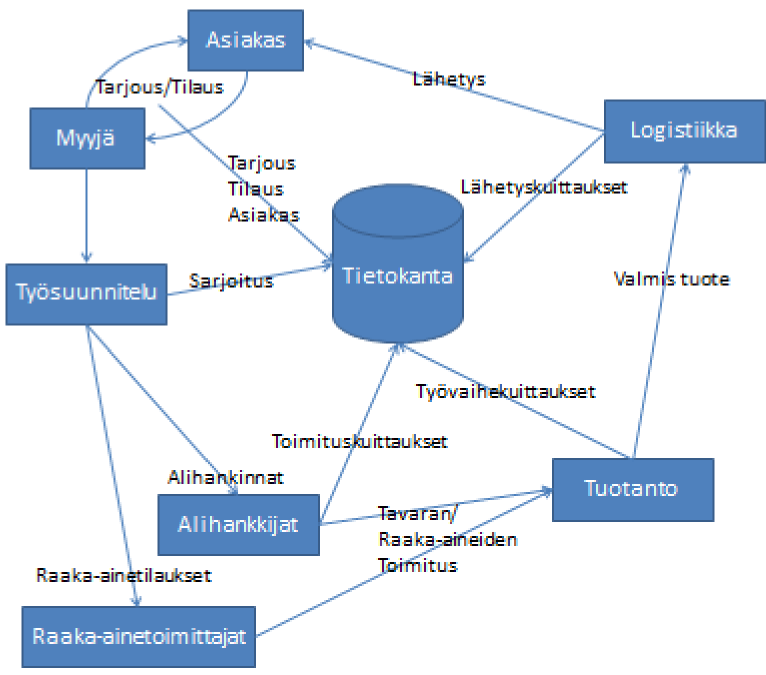
Yrityksen toiminnanohjausjärjestelmänä toimii Winplan. Se on ikkuna- ja ovitehtaille suunniteltu ohjelmisto, joka sisältää lähes koko toiminnan. (DB-Manager 2014.)

Winplan tallentaa tietokantaan kaikki tuotteiden ja tilausten tiedot. Tietovarastona toimii Microsoft SQL Server 2005 -tietokanta. Järjestelmään tallennettujen tuoterakenteiden tiedoilla myyjät tekevät tilauksia. Myös tilausten tiedot tallennetaan tietokantaan heti tarjouksen tekohetkestä alkaen. Yksityiskohtainen kuvaus yrityksen tuotannon toimintamallista on liitteessä 1.

Tuotannossa lähes kaikki työvaiheet kuitataan tietokantaan. Tämän lisäksi myös alihankkijat tekevät tietokantakuittauksia aina, kun toimitetaan tuotteiden osia tuotantoon. Näin erilaisilla raportointityökaluilla pystytään seuraamaan ja ohjaamaan tuotannon toimintaa.

4.1 Tuotannon vaiheet

Tuotannon vaiheet on kuvattu pääpiirteittäin kuviossa 6.



Kuvio 6. Tuotannon vaiheet

Tilauksen elinkaari alkaa, kun myyjä tekee asiakkaalle tarjouksen ja samalla syöttää toiminnanohjausjärjestelmään kuvat tuotteista sekä kaikki tarvittavat tiedot. Asiakkaan tehdessä ostopäätöksen tarjous lukitaan ja status muuttuu tarjouksesta tilaukseksi. Tiedot kopioidaan tietokannan toiseen tauluun. Tämän jälkeen muutoksia tilaukseen ei voi enää tehdä.

Seuraavassa vaiheessa työsuunnittelu käsittelee tilauksen. Työsuunnittelu optimoi tilauksen materiaalitarpeet, tekee raaka-ainetilaukset ja tilaukset alihankkijoille. Tämän jälkeen tilaus sarjoitetaan. Sarjoituksessa tilaus saa yhden tai useamman sarjanumeron. Sarjanumero määräytyy tuotantoyksikön mukaan. Tuotantopäivän täytyy olla sellainen päivä, jolloin kaikki materiaalit ovat saapuneet tehtaalle ja tilaus sopii päivittäiseen kapasiteettiin. Tämän jälkeen tilaus on valmis tuotannolle.

Tuotantoprosessissa lähes jokainen tuotantovaihe suorittaa työvaiheen kuittauksen tietokantaan. Esimerkkinä mainittakoon kotimaan ikkunoiden valmistus. Ikkunoiden puuosat höylätään muotoonsa käyttäen työsuunnittelun tekemää optimointia raaka-ainehukan minimoimiseksi, jonka jälkeen tietokantaan tehdään kuittaus. Puuosat koneistetaan tarpeen mukaan, siirretään maalaamoon pintakäsittelyyn ja suoritetaan työvaiheen kuittaus. Pintakäsittelyn jälkeen osiin

koneistetaan saranat ja lukot, jonka jälkeen ne tiivistetään, kootaan ja kuitataan kyseiset vaiheet. Lasitusvaiheessa koottuihin ikkunapokiin asennetaan lasielementit ja erä kuitataan valmiiksi. Sovituslinjalla koottuihin karmeihin asennetaan lasitetut pokat. Tässä vaiheessa alumiiniosat on saatu toisesta hallista ja ne liitetään myös tuotteisiin. Tuotteisiin asennetaan lisävarusteet ja ne pakataan lavoille kuljetusta varten. Pakkaaja suorittaa kuittauksen tietokantaan, kun erä on valmis, ja siirtää erän ulos odottamaan toimitusta.

Logistiikka hoitaa toimituksen ja tekee tarvittavat kuittaukset tietokantaan. Logistiikka myös seuraa pakkauskuittauksia ja toimii pakkaajien ohjaajana, jos kyseessä on kiireellisiä eriä tai useiden eri tuotantolinjojen välisiä yhteispakkauksia.

Tiedot, jotka voidaan palauttaa laskennallisesti, poistetaan tietokannasta 28—90 päivän kuluessa. Muut tiedot jäävät pysyvästi tietokantaan.

4.2 Viivakoodien käyttö tuotannossa

Vuoden 2012 viimeisellä neljänneksellä yrityksessä siirryttiin käyttämään Upcode-viivakoodijärjestelmää. Järjestelmän tavoitteena on antaa reaaliaikainen kuva tilauksen valmistumisesta. Viivakoodit tulostetaan karmien koonnin yhteydessä niihin kiinnitettäviin karmitarroihin. Näin ollen ne pystytään pakkauksen yhteydessä lukemaan viivakoodinlukijalla, jolloin järjestelmä kuittaa tuotteen valmiiksi. Koko tilauksen valmistuttua järjestelmä kuittaa sen kokonaan valmiiksi. Kuviossa 7 on esimerkki tuotteeseen kiinnitettävästä karmitarrasta ja kuviossa 8 on yrityksen käyttämä viivakoodinlukija.

Kuvio 7. Tuotetarra ja viivakoodi

Kuvio 8. Psionin Workabout pro3 -viivakoodinlukija

5 TIETOKANTASOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Tietokantasovelluksen rakentamisessa päätettiin käyttää prototyyppimallia, koska haluttiin rakentaa eräänlainen testiversio prototyyppinä ja kehittää sen pohjalta toimiva ohjelma. Testiversion vaatimuksena oli myös toimia pelkästään paikallisena konsoliversiona. Tällä tavoin ohjelman käyttämiä SQL-kyselyitä pystytään testaamaan ennen niiden ajamista ohjelman todellisessa ympäristössä.

Tietokantasovelluksen toteutus tapahtui Microsoftin Visual Studio 2012 -ohjelmalla ja ohjelmointikielenä toimi C#. Ohjelmisto tehtiin käyttäen Microsoftin .NET 4.0-kehystä.

Apuna toteutuksessa käytettiin Microsoftin SQL Server Management Studio -ohjelmaa. Sen avulla rakennettiin sovelluksen käyttämät kyselyt ja tarkastettiin niiden toimivuus.

5.1 Ensimmäinen versio

Prototyyppimallin ensimmäisellä kierroksella rakennettiin ohjelman perusominaisuudet sisältävä prototyyppi.

5.1.1 Vaatimukset ja toimintamallin määrittely

Ohjelman ensimmäisen version vaatimuksena oli paikalliselta koneelta tehtävä tietokantakysely ja kyselyn tuloksista muokattu esitys. Esitys oli tarkoitus rakentaa automaattisesti HTML-muotoon paikalliselle koneelle.

Ensimmäisen version toimintamalli oli yksinkertainen. Tietokantayhteyden tiedot ja tietokantakysely oli kovakoodattu ohjelmaan. Jos käyttäjä halusi vaihtaa esimerkiksi tietokantaa tai kyselyä, oli ohjelma käännettävä uudestaan muutosten jälkeen. Samoin tiedot rakennettavan sivun tiedoista ja sijainnista löytyvät kovakoodattuna ohjelmasta. Ensimmäisen version toimintamalli on kuvattu kuviossa 9.

Kuvio 9. Ensimmäisen version toimintamalli

5.1.2 Ensimmäisen version toteutus

Toteutus ensimmäisessä versiossa oli jaettu neljään osaan. Ensimmäinen osa oli tietokantayhteyden muodostaminen. Toinen ja kolmas osa muodostivat tietokantakyselyn suorittamisen ja sen tulosjoukon vastaanottamisen. Neljäntenä oli esityksen muuntaminen HTML-muotoon.

Yhteyden luonti: Ohjelman toiminta alkaa tietokantayhteyden määrittelyllä. Yhteyden määrittelyssä käytetään System.Data.SqlClient.SqlConnection-luokkaa. Luokasta tehdään olio, jonka parametreiksi annetaan tietokantayhteyden tiedot. Kuviossa 10 on esimerkki tietokantayhteyden määrittelystä.

```
SqlConnection yhteys = new SqlConnection("Data Source=10.10.28.20; Initial Catalog=tietokannan_nimi; User ID=käyttäjänimi; Pwd=salasana; Connection timeout = 30");
```

Kuvio 10. Esimerkki tietokantayhteyden määrittelystä

SQL-kysely: Yhteyden lisäksi määritellään käytettävä tietokantakysely. Tietokantakysely määritellään System.Data.SqlClient.SqlCommand-luokan avulla.

Kyselyssä käytetään yhtenä hakuparametrina kuluvaa päivämäärää. Ensimmäisessä versiossa päivämäärä liitetään kyselyyn muuttujan avulla ennen kyselyn lähettämistä tietokantapalvelimelle. Päivämäärä muokataan ennen liittämistä tietokantapalvelimen hyväksymään muotoon yyyy-mm-dd. Kuviossa 11 on esimerkki tietokantakyselyn määrittelystä.

```
SqlCommand cmd = new SqlCommand("select t.prod_line, sum(t.kpl * r.units_row) from tj_kuittaukset t, rows r where t.pvm>'" + tanaanpvm + "' and type=0 and t.prod_line<>'' and r.id = t.id and r.id_part = t.id_part and r.id_row = t.id_row and t.id_part<1000 group by prod_line", yhteys);
```

Kuvio 11. Esimerkki tietokantakyselyn määrittelystä

Kyselyn tulokset: Yhteysparametrien ja kyselyn määrittelyiden jälkeen ohjelma suorittaa yhteyden avaamisen ja kyselyn ajamisen. Kyselyn lukeminen suoritetaan System.Data.SqlClient.SqlDataReader-luokan avulla. Luokalle annetaan parametrina edellä syötetty kysely ja se saa paluuarvona tietokannalta tulosjoukon. Ensimmäisen version tapauksessa tulosjoukko on tuotantolinjojen nimiä ja niiden tuotantomääriä kuluneen päivän ajalta.

Tulosjoukon käsittelyä varten on rakennettu oma tuotantolinja-luokka. Se sisältää kaksi muuttujaa nimi ja määrä, sekä tulostusmetodin joka palauttaa muuttujat välilyönnillä erotettuna. Tuotantolinja-luokan toteutus on esitetty kuviossa 12.

```
class Tuotantolinja
{
    public string nimi;
    public string maara;

    public Tuotantolinja(string nimi, string maara)
    {
        this.nimi = nimi;
        this.maara = maara;
    }

    public override string ToString()
    {
        string s = nimi + " " + maara;
        return s;
    }
}
```

Kuvio 12. Tuotantolinja-luokan toteutus

Lukuvaiheen alussa ennen yhteyden avaamista tuotantolinja-luokista tehdään System.Collections.Generic.List-luokan avulla listat eri tuotantopaikoille.

Tulosjoukkoa luettaessa tehdään uusia tuotantolinja-luokan olioita jokaiselle yksittäiselle tuotantolinjalle. Jokainen näistä käydään läpi ehto-lauseessa, jossa kukin tuotantolinja-olio tallennetaan oikean tuotantopaikan listaan. Ehto-lauseen toteutus on kuvattu kuviossa 13.

```
while (lukija.Read())
{
    Tuotantolinja t = new Tuotantolinja(lukija[0].ToString(), lukija[1].ToString());
    if (t.nimi == "EK") { yliharma.Add(t); }
    if (t.nimi == "KOTIM") { yliharma.Add(t); }
    if (t.nimi == "OVI") { yliharma.Add(t); }
    if (t.nimi == "OVIA") { yliharma.Add(t); }
    if (t.nimi == "OVICE") { yliharma.Add(t); }
    if (t.nimi == "VINO") { yliharma.Add(t); }
    if (t.nimi == "KORIS") { yliharma.Add(t); }

    if (t.nimi == "VIENT") { alaharma.Add(t); }
    if (t.nimi == "VIEN2") { alaharma.Add(t); }

    if (t.nimi == "KARVI") { karvia.Add(t); }
    if (t.nimi == "KARVP") { karvia.Add(t); }

    if (t.nimi == "PRKLA") { lahti.Add(t); }
}
```

Kuvio 13. Tuotantolinja-luokkien erittely tuotantopaikka-listoihin

Yhteyden avaus ja tulosjoukon haku on upotettu try-catch-rakenteen sisään (Microsoft 2014f). Rakenteen tarkoituksena on käsitellä mahdolliset virhetilanteet tietokantayhteyden avaamisessa ja tulosjoukon haussa siten, ettei ohjelman suoritus keskeydy virhetilanteisiin, vaan se jatkuu siitä huolimatta.

Ohjelman suoritettua tietokantakyselyn yhteys suljetaan ja siirrytään muodostamaan HTML-sivut.

Esityksen luonti: Sivujen tekemiseen on ohjelmaan tehty oma luokka nimeltä HtmlGen. Se saa parametrina listan tuotantolinja-olioita, sekä kuluvan päivänmäärän joka aiemmin upotettiin kyselyyn.

HtmlGen-luokassa käytetään System.Text.StringBuilder-luokan System.Text.StringBuilder,AppendLine-metodia kokoamaan HTML-tiedosto valmiiksi kirjoitetuista HTML-komennoista, joihin upotetaan parametrina saadun listan sisältämät tiedot.

Ensimmäisessä versiossa generoitiin sivu vain Ylihärmän tuotantopaikalle, joten kyseisen sivun nimi oli kovakoodattu luokkaan. Kuviossa 14 on listattu HtmlGen-luokan toteutus.

```
class HtmlGen
{
    public void LuoSivu(List<Tuotantolinja> lista, string pvm)
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("<!DOCTYPE html PUBLIC " + (char)34 + "-//W3C//DTD XHTML 1.0 Transitional//EN"
            + (char)34 + " " + (char)34 + "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" + (char)34 + ">");
        sb.AppendLine("<head>");
        sb.AppendLine("<meta http-equiv=" + (char)34 + "refresh" + (char)34
            + " content=" + (char)34 + "360" + (char)34 + " >");
        sb.AppendLine("<meta http-equiv=" + (char)34 + "Content-Type" + (char)34
            + "content=" + (char)34 + " text/html; charset=utf-8" + (char)34 + "/>");
        sb.AppendLine("<title>Skaala - valmistuneet " + pvm + "</title>");
        sb.AppendLine("<style type=" + (char)34 + "text/css" + (char)34 + ">");
        sb.AppendLine("<!--");
        sb.AppendLine(".style1 {font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 30px}");
        sb.AppendLine(".style4 {font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 36px}");
        sb.AppendLine("");
        sb.AppendLine("-->");
        sb.AppendLine("</style>");
        sb.AppendLine("</head>");
        sb.AppendLine("<body>");
        sb.AppendLine("<p align=" + (char)34 + "center" + (char)34 + "><img src=" + (char)34 + "logo.jpg" + (char)34
            + " width=" + (char)34 + "1024" + (char)34 + " height=" + (char)34 + "183" + (char)34 + " alt=" + (char)34
            + "logo" + (char)34 + " /></p>");
        sb.AppendLine("<table border=" + (char)34 + "0" + (char)34 + " align=" + (char)34 + "center" + (char)34 + ">");
        sb.AppendLine("<tr>");
        sb.AppendLine("<td colspan=" + (char)34 + "2" + (char)34 + "><div align=" + (char)34 + "center" + (char)34
            + "><span class=" + (char)34 + "style4" + (char)34 + ">Valmistuneet tuotteet linjoittain " + pvm
            + "</span></div></td>");
        sb.AppendLine("</tr>");

        foreach (var t in lista)
        {
            sb.AppendLine("<td class=" + (char)34 + "style1" + (char)34 + "><div align="
                + (char)34 + "center" + (char)34 + " >" + t.nimi + "</td>");
            sb.AppendLine("<td class=" + (char)34 + "style1" + (char)34 + "><div align="
                + (char)34 + "left" + (char)34 + " >" + t.maara + "</td>");
            sb.AppendLine("</tr>");
        }
        StreamWriter sw = new StreamWriter("C:\\tmp\\Yliharma.html", false);
        sw.Flush();
        sw.Write(sb);
        sw.Close();
    }
}
```

Kuvio 14. HtmlGen-luokan toteutus ensimmäisessä versiossa

Sivun rakenteeseen upotetaan myös metakoodi, joka käskee selainta päivittämään sivun annetun ajan välein. Näin varmistetaan, että järjestelmän generoimat uudet sivut päivittyvät automaattisesti selaimen ollessa sivulla.

Html-sivun rakentamisen jälkeen ohjelma laitetaan lepotilaan käyttäen System.Threading.Thread.Sleep-metodia. Metodin parametreihin määritellyn ajan jälkeen ohjelman suoritus aloitetaan uudelleen yhteyden luonnilla. Kuviossa 15 on esimerkki ohjelman rakentamasta sivusta.

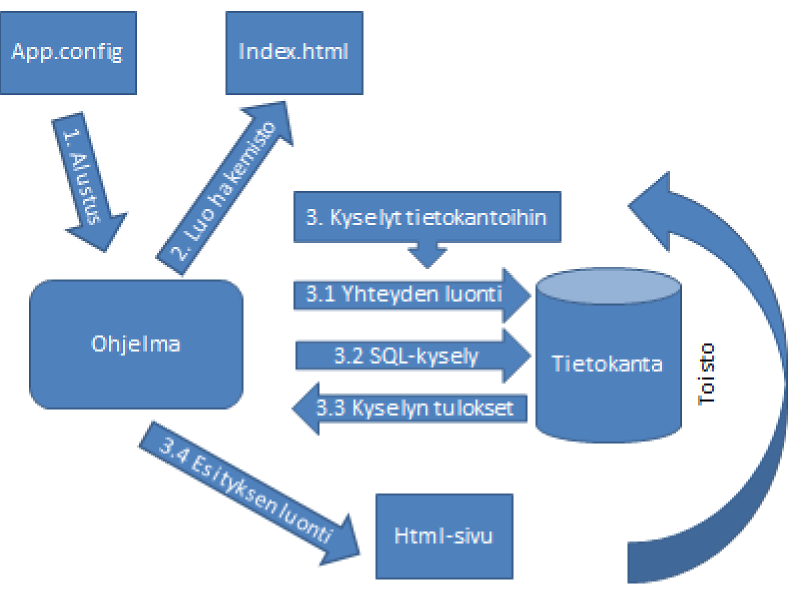
Kuvio 15. Ensimmäisellä versiolla rakennettu sivu

5.2 Toinen versio ja dynaaminen lähestymistapa

Prototyyppimallin toisella kierroksella täydennettiin asiakasvaatimuksia ohjelmiston suhteen.

5.2.1 Toisen version vaatimukset ja toimintamalli

Toimintamallin toinen versio on kehitetty ensimmäisen version pohjalta. Pääperiaate on sama eli haetaan haluttu tieto tietokannasta ja tehdään siitä esitys. Kokonaan uutena ominaisuutena haluttiin mahdollisuus tuoda tietokantakyselyt ohjelman ulkopuolelta. Tämän lisäksi ohjelman haluttiin tekevän hakemisto ohjelman rakentamista sivuista, jolloin navigointi niille helpottuisi. Toisen version toimintamalli generoinnista on kuvattu kuviossa 16.



Kuvio 16. Dynaamisen version toimintamalli

5.2.2 Toisen version toteutus

Alustus: Toisen version suurin muutos tapahtuu käynnistyksen yhteydessä lukemalla app.config-asetustiedostosta alustustiedot. Koska normaalisti app.config-asetustiedostoon pystyy tallentamaan vain yksittäisiä avain-arvo -pareja, jouduttiin ohjelmassa käyttämään custom sectioneita (Microsoft 2014d). Custom sectionit mahdollistavat monimutkaisempien tietorakenteiden sisällyttämisen app.config-tiedostoon. Niiden lukemiseen ohjelmaan rakennettiin 3 erillistä luokkaa, jotka hoitavat tiedon hakemisen app.config-tiedostosta ja sen tallentamisen ohjelmaan (Fazekas 2007). Näiden lisäksi rakennettiin oma luokka yhteystietojen tallentamiseen. Esimerkki asetustiedostosta on kuvattu kuviossa 17.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="Yhteystiedot" type ="ConReader.YhteystietoConfig, ConReader"/>
  </configSections>
  <appSettings>
    <!-- Aika kuinka usein kyselyt tehdään kannasta -->
    <add key="queryinterval" value="20"/>
    <!-- Tulostuskansio .html-tiedostoille -->
    <add key="outputpath" value="C:\\tmp\\"/>
  </appSettings>
  <Yhteystiedot>
    <Yhteydet>
      <!-- Yhteystiedot!! -->
      <!-- yhteysnro=juokseva numero joka toimii indeksinä (ei saa olla kahta samaa) -->
      <!-- yhteysstring="Data Source=tietokanta; Initial Catalog=kannan nimi; User ID=käyttäjä; Pwd=salasana; Connection timeout=arvo timeoutille -->
      <!-- sqlstring=tekstimudossa oleva tiedosto, jossa on TOIMIVA sql-lause, joka haetaan ko. tietokannasta -->
      <!-- sivunnimi=nimi sivulle jonka ohjelma luo (ohjelma lisää automaattisesti .html-päätteen) -->
      <!-- otsikko=sivulle tuleva otsikko joka näkyy tietojen päällä ja sivun välilehdessä selaimen yläosassa -->
      <add yhteysnro="1" yhteysstring="Data Source=10.10.28.20; Initial Catalog=winplan; User ID=user; Pwd=password;
          Connection timeout = 30" sqlstring="Rasti.txt" sivunnimi="rasti" otsikko="Tuotantopäivärästi yksiköittäin"/>
    </Yhteydet>
  </Yhteystiedot>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

Kuvio 17. App.config-asetustiedoston rakenne

Yleisiksi parametreiksi määriteltiin seuraavat muuttujat:

Queryinterval: Aika sekunteina, kuinka usein haut tehdään tietokannasta.

Outputpath: Hakemisto, johon sivut ja hakemisto muodostetaan.

Yksittäisen yhteystiedon parametreiksi määriteltiin seuraavat muuttujat:

Yhteysnro: Juokseva numero, joka täytyy olla uniikki jokaiselle erilliselle haulle.

Yhteysstring: Sisältää tietokantayhteyden avaamiseen tarvittavat tiedot. Näihin kuuluu tietokannan ip-osoite, käytettävän tietokannan nimi, käyttäjätunnus, salasana ja aikakatkaisun arvo sekunteina.

Sqlstring: Viittaus tekstitiedostoon, joka sisältää itse tietokantakyselyn. Tähän ratkaisuun päädyttiin, koska itse kyselyt voivat kasvaa monen rivin kokoisiksi ja konfiguraatio-tiedoston ulkoasu haluttiin pitää helposti luettavissa.

Sivunnimi: Kertoo rakennettavan sivun tiedostonimen.

Otsikko: Kuvaa sivun otsikon, joka näkyy rakennetulla sivulla ja hakemistossa.

Ohjelma lukee asetustiedostosta kaikki yhteystiedot ja rakentaa jokaisesta yhteysnro-muuttujalla alkavasta elementistä yhteystieto-luokan olion. Tietokantayhteys, sivunnimi ja otsikko luetaan suoraan muuttujiin. Sql-kysely sen

sijaan luetaan käyttämällä System.IO.StreamReader-luokkaa. Yhteystietojen lukeminen on esitetty kuviossa 18.

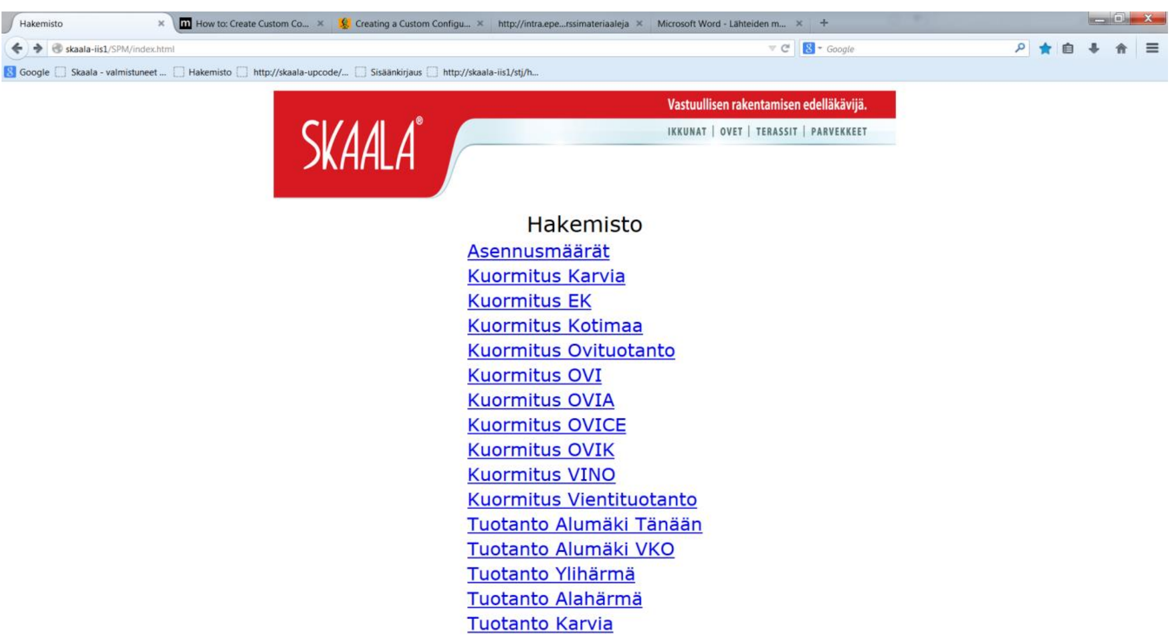
```
try
{
    for (int i = 0; i < section.Yhteyslista.Count; i++)
    {
        Yhteystieto y = new Yhteystieto();
        y.connection = section.Yhteyslista[i].Yhteysstring;
        string filename = section.Yhteyslista[i].Sqlstring;
        using (StreamReader sr = new StreamReader(filename))
        {
            y.sqlstring = sr.ReadToEnd();
        }
        y.pagename = section.Yhteyslista[i].Sivunnimi;
        y.otsikko = section.Yhteyslista[i].Otsikko;
        tietolista.Add(y);
    }
    HtmlGen indeksi = new HtmlGen();
    indeksi.LuoIndeksi(tietolista);
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
```

Kuvio 18. Yhteystietojen lukeminen app.config-asetustiedostosta

Luo hakemisto: Seuraava merkittävä muutos ensimmäiseen versioon tehtiin HtmlGen-luokkaan, johon lisättiin LuoIndeksi-niminen metodi hakemiston generointiin. Kuviossa 18 nähtävässä ohjelman osassa metodia kutsutaan ja sille annetaan parametrina lista yhteystiedoista. LuoIndeksi rakentaa normaalia sivua vastaavan pohjan ja lisää siihen jokaisen listan sisältämän alkion tiedot kuvion 19 mukaan. Tiedoista muodostuu suora hyperlinkki jokaiseen ohjelman rakentamaan sivuun. Kuviossa 20 on esitetty LuoIndeksi-luokan rakentama hakemisto.

```
foreach (var y in lista)
{
    sb2.AppendLine("<td class=" + (char)34 + "style1" + (char)34 + "><div align=" + (char)34 + "left"
        + (char)34 + " ><a href=" + (char)34 + y.pagename + ".html" + (char)34 + ">" + y.otsikko + "</td>");
    sb2.AppendLine("</tr>");
}
```

Kuvio 19. LuoIndeksi-metodin linkkien rakentaminen



Kuvio 20. Sivuista tehty hakemisto

Kyselyt tietokantoihin: Kolmas muutos tehtiin ohjelman suorittamaan silmukkaan. Silmukan toteutus toisessa versiossa tapahtui käyttämällä System.Timers.Timer-luokan ajastinta. Luokan avulla tehtiin olio, joka aktivoi ohjelman haku-osan aina määrätyn ajan kuluttua. Kuviossa 21 on esitetty ajastimen alustus.

```
int time = (int.Parse)(ConfigurationManager.AppSettings["queryinterval"]);
Timer tim = new Timer();
tim.Elapsed += new ElapsedEventHandler(tim_tick);
tim.Interval = (1000) * (time);
tim.Enabled = true;
```

Kuvio 21. Ajastimen määrittely

Ajastin määrittää tietokantahakujen suorituksen välisen tauon. Aika noudetaan app.config-asetustiedostosta queryinterval-muuttujasta. Ajastimen saavuttaessa määritellyn arvon se kutsuu tim_tick-metodia, joka suorittaa tietokantakyselyt. Tim_tick-metodin rakenne on kuvattu kuviossa 22.

```
private static void tim_tick(object sender, ElapsedEventArgs e)
{
    foreach (var y in tietolista)
    {
        SqlConnection con = new SqlConnection(y.connection);
        DataTable dt = new DataTable();
        con.Close();
        try
        {
            con.Open();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        try
        {
            SqlCommand cmd = new SqlCommand(y.sqlstring, con);
            dt.Load(cmd.ExecuteReader());
            HtmlGen gen = new HtmlGen();
            gen.LuoSivu(dt, y.pagename, y.otsikko);
            dt.Dispose();
        }
        catch (Exception ex)
        {

            Console.WriteLine(ex.ToString());
        }

        //Yhteyden sulkeminen
        try
        {
            con.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }



    }
}
```

Kuvio 22. Tim_tick-metodin toteutus

Ohjelman suorittama tietokantakysely tapahtuu lähes samalla tavalla kuin ensimmäisessä versiossa. Erona on että hakutulos talletetaan System.Data.Datatable-luokasta tehtyyn olioon. Periaate kuitenkin on sama. Avataan tietokantayhteys, suoritetaan haku ja rakennetaan tuloksista sivu HtmlGen-luokan avulla. Jokainen erillinen haku käydään läpi ohjelman yhden toistokerran aikana.

Tämän jälkeen ohjelma odottaa ajastimeen määritellyn ajan ennen uutta tietokantayhteyden avaamista.

5.3 Muunto palvelinversioksi

Prototyyppimallin kolmannella kierroksella ohjelma toteutettiin Windows Service Applicationina.

5.3.1 Palvelinversion vaatimukset

Seuraava vaatimus ohjelmalle oli toimia itsenäisesti yrityksen IIS-palvelimella. Tätä varten päätettiin muuntaa ohjelma toimimaan Windows Service Applicationina, jolloin ohjelma käynnistyy aina palvelimen käynnistymisen yhteydessä ja se toimii itsenäisesti ympäri vuorokauden.

5.3.2 Dynaamisen version muunto palvelinversioksi

Muunto Windows Service Applicationiksi tapahtui avaamalla Visual Studiolla tyhjä Windows Service -malli, joka sisältää valmiiksi kaikki tarvittavat metodit ja asennuksen vaatimat luokat (Microsoft 2014g). Toisen version sisältämät luokat siirrettiin sellaisenaan uuteen versioon.

Palvelu alkaa lokitiedoston määrittelyllä. Lokitiedostoon kerätään kaikki ajon aikana tapahtuneet virhetilanteet, jolloin niiden seuranta myöhemmin on helppoa. Koska toisessa versiossa käytettiin paljon try-catch-rakenteita, muunnettiin ne kirjoittamaan virhetilanteet palvelinversion lokitiedostoon. Lokitiedostojen määrittely on kuvattu kuviossa 23.

```
public Service1()
{
    //Logitiedoston määrittely mihin esim virhetilanteet kirjataan
    InitializeComponent();
    if (!System.Diagnostics.EventLog.SourceExists("MySource"))
    {
        System.Diagnostics.EventLog.CreateEventSource(
            "MySource", "MyNewLog");
    }
    eventLogSPM.Source = "MySource";
    eventLogSPM.Log = "MyNewLog";
}
```

Kuvio 23. Palvelun lokitiedostojen määrittely

Lokitiedostojen määrittelyn jälkeen palvelu jaetaan neljään eri metodiin:

OnStart: Sisältää kaiken mitä tapahtuu palvelun käynnistyessä. Metodissa tehdään uusi säie toiselle metodille **WorkerFunction** ja käynnistetään se. OnStart-metodin toteutus on kuvattu kuviossa 24.

```
protected override void OnStart(string[] args)
{
    //Uus threadi ja startataan OnStart menee läpi käynnistyksessä
    ThreadStart st = new ThreadStart(WorkerFunction);
    workerThread = new Thread(st);
    serviceStarted = true;
    workerThread.Start();
}
```

Kuvio 24. OnStart-metodin toteutus

OnStop: Sisältää tulostuksen lokitiedostoon palvelun pysäyttämisen yhteydessä.

OnContinue: Sisältää tulostuksen lokitiedostoon palvelun uudelleenkäynnistyksen yhteydessä pysäyttämisen jälkeen.

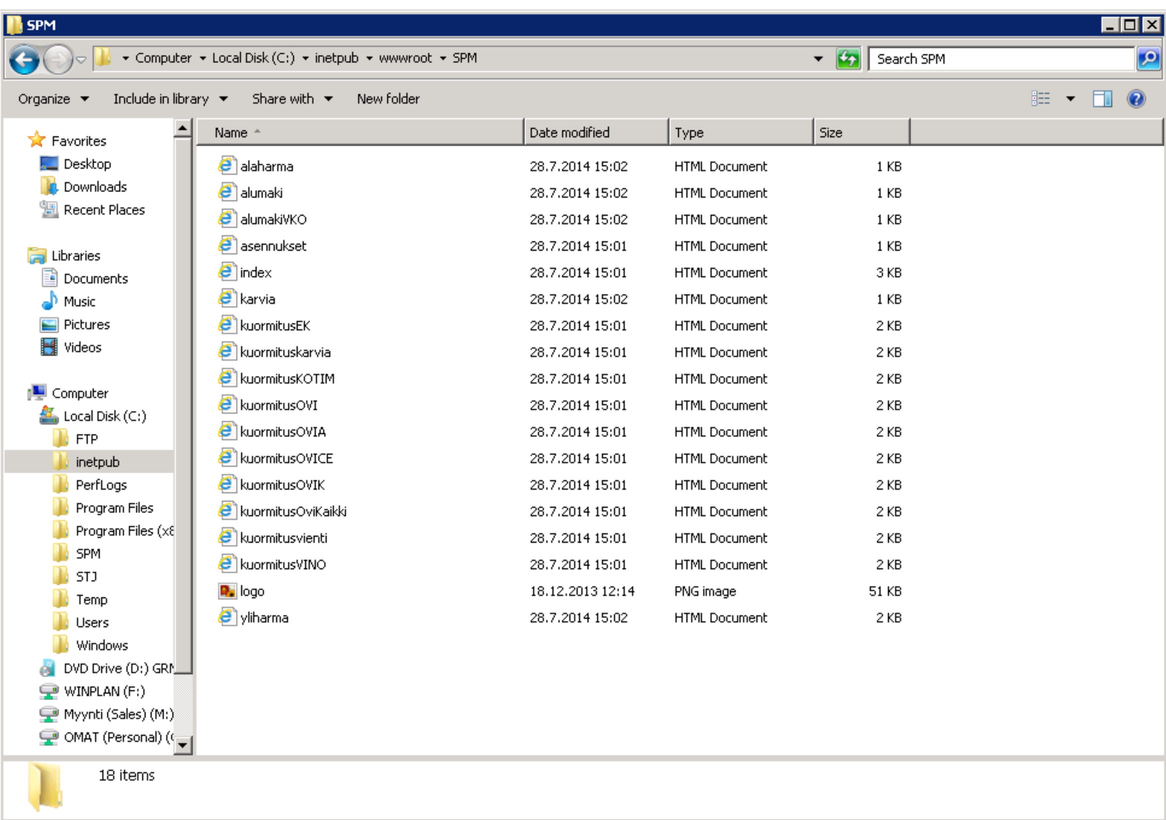
WorkerFunction: Sisältää toisen toteutusversion ohjelmakoodin eli toistettavan ohjelman toiminnan. Ainoa muutos toisen version toimintaan on virhetilanteiden hallinta. Virheet kirjataan nyt lokitiedostoon.

5.3.3 Palvelinversion asennus ja käynnistys

Palvelinversio asennettiin IIS-palvelimelle siirtämällä käynnistystiedostot omaan hakemistoon ja määrittelemällä rakennettavien sivujen hakemistoksi wwwroot-hakemisto, mikä näkyy yrityksen sisäverkossa.

Tämän jälkeen palvelu asennettiin käyttämällä Microsoftin Installutil-työkalua.

Asennuksen jälkeen palvelu käynnistettiin palvelimen hallintatyökaluista. Kuviossa 25 on kuvattu ohjelman käynnistyksen jälkeen rakentamat sivut. Hakemistoon on manuaalisesti lisätty logo.jpg-tiedosto, jota sivut käyttävät.



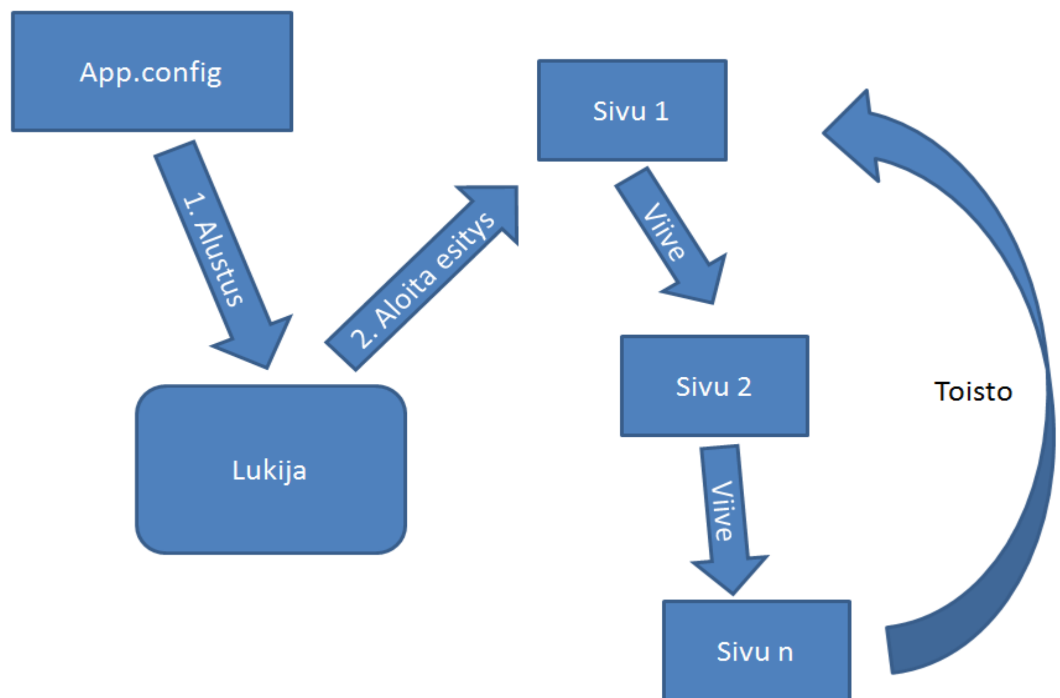
Kuvio 25. Ohjelman rakentamia sivuja

5.4 Lukijan rakentaminen

Pääohjelman lisäksi haluttiin sen rinnalle tapa esittää sen rakentamaa materiaalia automaattisesti. Tähän tarkoitukseen kehitettiin lukijaksi nimetty ohjelma.

5.4.1 Lukijan vaatimukset ja toimintamalli

Vaatimuksena oli, että lukijaa voidaan käyttää esittämään sivuja itsenäisesti. Lukija haluttiin rakentaa Windows Forms -ohjelmana (Microsoft 2014e). Sen tavoitteena oli www-sivujen esittäminen koko näytön kokoisena erikseen asetetulla viiveellä. Lisävaatimuksena haluttiin että lukija-ohjelmaa pystyy käyttämään myös näytönsäästäjänä normaaleilla työasemoilla, jolloin sen tulee olla toiminnassa jatkuvasti taustalla ja aktivoida esitys tietyn tyhjäkäyntiajan jälkeen. Lukijan toimintamalli on kuvattu kuviossa 26.



Kuvio 26. Lukijan toimintamalli

5.4.2 Lukijan toteutus

Alustus: Lukijan asetusten määrittely suoritetaan app.config-asetustiedostossa. Lukijan app.config-asetustiedoston muoto on kuvattu kuviossa 27.

```
<appSettings>
  <!-- Kauanko kone täytyy olla idle-tilassa kunnes ohjelma käynnistyy sekunteina -->
  <add key="idle" value="10"/>
  <!-- Arvo sekunteina minkä välein sivu vaihtuu -->
  <add key="refresh" value="60"/>
  <!-- Osoitteet puolipisteellä eroteltuna -->
  <add key="osoite" value="http://skaala-iis1/spm/yliharma.html"/>
</appSettings>
```

Kuvio 27. Lukijan app.config-asetustiedosto

Lukijan parametreiksi muodostuivat seuraavat muuttujat:

Idle: Tyhjäkäyntiaika sekunteina, jonka jälkeen ohjelma käynnistyy.

Refresh: Määrittelee sivujen päivitystiheyden sekunteina.

Osoite: Sisältää kaikki näytettävät sivut puolipisteellä eroteltuna.

Lukijan käynnistyessä se muodostaa itselleen System.Windows.Forms.NotifyIcon-luokan olion, joka toimii ohjelman ikonina ilmaisinalueella. Tehdylle oliolle annetaan kuva, teksti ja siihen liitetään System.Windows.Forms.ContextMenu-luokan olio, jossa määritellään hiiren oikean napin painalluksesta avautuvan valikon sisältö. Lukijan tapauksessa valikko sisältää ainoastaan Exit-komennon. Lukijan asetukset on kuvattu kuviossa 28.

```
static void Main()
{
    //Pääohjelma alkaa
    //Luodaan system trayhin ikoni
    NotifyIcon ikoni = new NotifyIcon();
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    //Ikonille menu, kuva, nimi jne
    //Menusa yksi itemi joka sammuttaa ohjelman
    //////////////////////////////////////////////////////
    NotifyIcon notifyIcon1 = new NotifyIcon();
    ContextMenu contextMenu1 = new ContextMenu();
    MenuItem menuItem1 = new MenuItem();
    contextMenu1.MenuItems.AddRange(new MenuItem[] { menuItem1 });
    menuItem1.Index = 0;
    menuItem1.Text = "Exit";
    menuItem1.Click += new EventHandler(menuItem1_Click);
    notifyIcon1.Icon = new Icon("Spm.ico");
    notifyIcon1.Text = "Skaala Performance Manager Reader";
    notifyIcon1.ContextMenu = contextMenu1;
    notifyIcon1.Visible = true;
    //////////////////////////////////////////////////////

    //Käynnistetään itse formi
    Application.Run(new Form1());
}

//Jos käyttäjä klikkaa system trayn ikonista exit niin heilutaan tänne ja suljetaan ohjelma
private static void menuItem1_Click(object Sender, EventArgs e)
{
    Application.Exit();
}
```

Kuvio 28. Lukijan sisäiset asetukset

Tämän jälkeen ohjelma muodostaa System.Windows.Forms.Form-luokan olion valmiiksi määritellystä pohjasta. Formin pohjaksi on määritelty pelkkä System.Windows.Forms.WebBrowser-luokan olio, josta on poistettu kaikki ohjauspalkit ja jäljelle on jätetty pelkkä näyttö-alue. Lopuksi ohjelma siirtyy muodostettuun Formiin.

Aloita esitys: Formiin siirtymisen yhteydessä ohjelma alustaa kaksi eri System.Timers.Timer-luokan oliota.

Ensimmäinen olio toimii ajastimena sivujen vaihteluvälille ja se saa arvonsa suoraan app.config-asetustiedostosta. Sitä kutsutaan joka kerta, kun määritelty aika on kulunut.

Toinen olio määrittää näppäimistösyötteen tarkistusvälin. Sitä kutsutaan yhden sekunnin välein. Tämä aika on asetettu vakioksi, eikä sitä voi muuttaa ohjelman ulkopuolelta.

Ajastimien lisäksi tehdään System.Collections.Generic.Queue-luokan olio. Siihen ohjelma tallentaa kaikki app.config-asetustiedostossa määritellyt www-sivut.

Ohjelman suoritus sisältää kolme metodia:

Form1_Load-metodi ajetaan kerran, kun ohjelma käynnistetään. Se lataa aiemmin rakennetun jonon, siirtyy jonon ensimmäiselle sivulle ja piilottaa ikkunan. Metodin toteutus on kuvattu kuviossa 29.

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Hide();
    string ekaUrl = osoiteJono.Dequeue().ToString();
    osoiteJono.Enqueue(ekaUrl);
    webBrowser1.Navigate(ekaUrl);
}
```

Kuvio 29. Form1_Load-metodin toteutus

Refreshtimer_Tick-metodi sisältää sivunvaihdon. Se hakee jonosta seuraavan osoitteen ja siirtyy sinne. Metodia kutsutaan ohjelman alussa määritetyn ajan välein. Metodin toteutus on kuvattu kuviossa 30.

```
private void refreshtimer_Tick(object sender, EventArgs e)
{
    string currentUrl = osoiteJono.Dequeue().ToString();
    osoiteJono.Enqueue(currentUrl);
    webBrowser1.Navigate(currentUrl + "?refreshToken=" + Guid.NewGuid().ToString());
}
```

Kuvio 30. Refreshtimer_Tick-metodin toteutus

Idlechecktimer_Tick-metodi suorittaa tyhjäkäyntiajan tarkistuksen. Metodia kutsutaan sekunnin välein. Metodi lataa app.config-asetustiedostossa määritellyn idle-muuttujan arvon ja vertaa nykyistä tyhjäkäyntiaikaa siihen. Jos nykyinen tyhjäkäyntiaika ylittää muuttujan arvon, metodi nostaa näytettävän sivun sisältävän ikkunan esille ja päällimmäiseksi, muussa tapauksessa se piilotetaan. Vertailussa käytetään apuna erillistä luokkaa, jonka IdleTimeFinder-metodi antaa viimeisimmästä syötteestä kuluneen ajan (Xavi23cr 2006). Idlechecktimer_Tick-metodin toteutus on kuvattu kuviossa 31.

```
private void idlechecktimer_Tick(object sender, EventArgs e)
{
    //Haetaan app.configista sekuntimäärä
    uint idleTrigger = (uint.Parse)(ConfigurationManager.AppSettings["idle"]);
    if (IdleTimeFinder.GetIdleTime() > (1000) * idleTrigger)
    {
        this.Show();
        this.TopMost = true;
        this.Update();
        this.BringToFront();
    }
    else
    {
        this.Hide();
    }
}
```

Kuvio 31. Idlechecktimer_Tickin toteutus

6 YHTEENVETO

Ohjelmisto saavutti kaikki sille asetetut vaatimukset ja enemmänkin. Alkuperäinen vaatimus oli esittää tuotannon valmistusmääriä, mutta lopullinen ohjelma päivittää jo yli kymmenen erilaista sivua. Ohjelman toiminta viimeisen puolen vuoden aikana on ollut moitteetonta. Lokitiedostoista näkyy, että ohjelma on joskus ajautunut lukkotilanteisiin tietokantahakujen aikana, mutta jatkanut silti toimintaa normaaliin tapaan. Muistinkäyttö on pysynyt reilussa 10 megatavussa.

Ohjelman suunnittelu ja toteutus oli kaiken kaikkiaan mielenkiintoinen projekti. Menetelmiä toteutukseen löytyy varmasti useampiakin, mutta käytetyt menetelmät osoittautuivat hyvin toimiviksi.

Haasteellisimmaksi työssä osoittautui asetustietojen hakeminen ohjelman ulkopuolelta halutulla tavalla, mutta ongelmaan löytyi toimiva ratkaisu. Toinen haasteellinen asia oli itse tietokantakyselyt johtuen yrityksen tietokannan rakenteesta. Näihin sain kuitenkin valtavasti apua työn valvojalta Ilkka Talaskiveltä.

LÄHTEET

DB-Manager. 2014. Ikkuna- ja ovitehtaan toiminnanohjausjärjestelmä. [www-dokumentti]. DB-Manager. [Viitattu 21.7.2014]. Saatavissa: http://www.dbmanager.fi/fi/winplan/winplan/

Ecma international. 2006. C# Language Specification. [Verkkojulkaisu]. Ecma international. [Viitattu 21.7.2014]. Saatavissa: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Fazekas, D. 2007. Creating a Custom Configuration Section in C#. [www-dokumentti]. CodeProject. [Viitattu 28.7.2014]. Saatavissa: http://www.codeproject.com/Articles/20548/Creating-a-Custom-Configuration-Section-in-C

Helsingin Yliopisto. 2009. Ohjelmistojen mallintaminen, Johdatus ohjelmistotuotantoon. [www-dokumentti]. Helsingin Yliopisto. [Viitattu 3.4.2015]. Saatavissa: http://www.cs.helsinki.fi/u/pohjalai/ke09/ohma/slides/ohma_01-ohjelmistotuotannosta.pdf

Immonen, J. 2002. Johdatus Ohjelmistotuotantoon. [www-dokumentti]. Joensuun Yliopisto. [Viitattu 3.4.2015]. Saatavissa: http://cs.joensuu.fi/~jimmonen/jot_moniste/jot_moniste_121.html

Lappalainen, V. 2011. Agile-menetelmät. [www-dokumentti]. Jyväskylän Yliopisto. [Viitattu 3.4.2015]. Saatavissa: http://users.jyu.fi/~vesal/kurssit/ohjelmointi2011/materiaali/agile.html

Microsoft. 2014a. Application Development. [www-dokumentti]. Microsoft. [Viitattu 18.7.2014]. Saatavissa: http://www.visualstudio.com/explore/application-development-vs

Microsoft. 2014b. Overview of the .NET Framework. [www-dokumentti]. Microsoft. [Viitattu 21.7.2014]. Saatavissa: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

Microsoft. 2014c. Introduction to Windows Service Applications. [www-dokumentti]. Microsoft. [Viitattu 21.7.2014]. Saatavissa: http://msdn.microsoft.com/en-us/library/d56de412%28v=vs.110%29.aspx

Microsoft. 2014d. How to: Create Custom Configuration Sections Using ConfigurationSection. [www-dokumentti]. Microsoft. [Viitattu 28.7.2014]. Saatavissa: http://msdn.microsoft.com/en-us/library/vstudio/2tw134k3%28v=vs.100%29.aspx

Microsoft. 2014e. Windows Forms. [www-dokumentti]. Microsoft. [Viitattu 29.7.2014]. Saatavissa: http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx

Microsoft. 2014f. Try-catch (C# Reference). [www-dokumentti]. Microsoft. [Viitattu 17.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/0yd65esw.aspx

Microsoft. 2014g. Windows Service Template. [www-dokumentti]. Microsoft. [Viitattu 22.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/1xzez001%28v=vs.80%29.aspx

Microsoft. 2014h. Installutil.exe (Installer Tool). [www-dokumentti]. Microsoft. [Viitattu 22.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/50614e95%28v=vs.110%29.aspx

Microsoft. 2014i. .NET Framework Class Library. [www-dokumentti]. Microsoft. [Viitattu 6.4.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/gg145045%28v=vs.110%29.aspx

Microsoft. 2015. SQL Server Database Engine. [www-dokumentti]. Microsoft. [Viitattu 26.4.2015]. Saatavissa: https://technet.microsoft.com/en-us/library/ms187875%28v=sql.120%29.aspx

Sarja, J. 2006. Relaatiotietokanta. [www-dokumentti]. Verkkopedagogi. [Viitattu 18.7.2014]. Saatavissa: http://verkkopedagogi.net/vanhat/fi/sisalto/materiaalit/access2003/luku0375c6.html?C:D=419702&selres=419702

Skaala Oy. 2015. [www-dokumentti]. Skaala ikkunat ja ovet Oy. [Viitattu 15.4.2015]. Saatavissa: http://www.skaala.com/

Taina, J. 2000. Ohjelmistotuotannon prosessimallit. [www-dokumentti]. Helsingin Yliopisto. [Viitattu 3.4.2015]. Saatavissa: http://www.cs.helsinki.fi/u/taina/ohtu/s-2000/luennot/prosessi/kaikki.html

Techterms.com. 2009. Database. [www-dokumentti]. Techterms.com. [Viitattu 18.7.2014]. Saatavissa: http://www.techterms.com/definition/database

Tutorialspoint. 2014. Database Management System [DBMS] Tutorial. [www-dokumentti]. Tutorialspoint. [Viitattu 18.7.2014]. Saatavissa: http://www.tutorialspoint.com/dbms/

W3schools. 2014. SQL Tutorial. [www-dokumentti]. W3schools. [Viitattu 20.3.2014]. Saatavissa: http://www.w3schools.com/sql/

Xavi23cr. 2006. Getting the user idle time with C#. [www-dokumentti]. CodeProject. [Viitattu 29.7.2014]. Saatavissa: http://www.codeproject.com/Articles/13384/Getting-the-user-idle-time-with-C