

RESPONSIIVISEN WEB-SOVELLUKSEN OHJELMOINTI .NET YMPÄRISTÖSSÄ

Case: Martten Finland Oy

Tiivistelmä

Tekijä(t) Latvaniemi, Ville	Julkaisun laji Opinnäytetyö, AMK	Valmistumisaika Syksy 2020
	Sivumäärä 37	
Työn nimi Responsiivisen web-sovelluksen ohjelmointi .Net ympäristössä Case: Martten Finland Oy		
Tutkinto Tieto- ja viestintätekniikan insinööri (AMK)		
Tiivistelmä <p>Opinnäytetyön tavoitteena oli toteuttaa responsiivinen web-sovellus laitoshuoltoalan yrityksen käyttöön. Sovelluksen päätarkoituksena olivat työntekijöiden ohjeistus, työn edistymisen seuranta ja asiakaspalautteen kerääminen. Työn toimeksiantajana oli Martten Finland Oy.</p> <p>Työn tavoitteeksi asetetun sovelluksen taustajärjestelmä toteutettiin käyttämällä Microsoftin .NET kehitysympäristön sisältämiä työkaluja, kuten Entity Frameworkia. Re-laatiotietokannan hallintajärjestelmänä käytettiin Microsoft SQL Server alustaa ja sille tarkoitettua SQL Server Management Studio hallintatyökalua. Sovelluksen käyttöliittymien ja niiden responsiivisuuden toteuttamisessa käytettiin avoimeen lähdekoodiin perustuvaa Bootstrap Frameworkia.</p> <p>Lopputuloksena saatiin aikaan sovellus, joka vastaa hyvin sille asetettuja vaatimuksia. Työ saatiin valmiiksi asiakkaan hyväksymällä aikataululla ja saatiin otettua käyttöön ilman ongelmia.</p>		
Asiasanat .NET, Entity Framework, Bootstrap		

Abstract

Author(s) Latvaniemi, Ville	Type of publication Bachelor's thesis	Published Spring 2020
	Number of pages 37	
Title of publication Programming responsive web application in .Net environment Case: Martten Finland Oy		
Name of Degree Information and Communications Technology, Software Engineering		
<p>Abstract</p> <p>The objective of this thesis was to implement a responsive web application for the use of a customer company from cleaning industry. The main purposes of the application were to instruct employees, monitor work progress and collect customer feedback. The work was commissioned by Martten Finland Oy.</p> <p>Application's back-end system was implemented using tools included in Microsoft .NET development environment, such as the Entity Framework. Microsoft SQL Server was used as a relational database platform and SQL Server Management Studio was used as a database management tool. An open source Bootstrap Framework was used to implement responsive interfaces for the application.</p> <p>The result was an application that meets well the requirements set for it. The work was completed within a timeframe approved by the customer and was put in use without problems.</p>		
Keywords .NET, Entity Framework, Bootstrap		

SISÄLLYS

1	JOHDANTO	1
2	.NET FRAMEWORK	2
2.1	ASP.NET	3
2.2	Entity Framework	3
2.2.1	Olioiden mappaus	4
2.2.2	Datan käsittely	5
2.2.3	EF Tools ja EF Runtime	5
2.2.4	Työjärjestykset (workflows)	7
2.3	LINQ	13
2.3.1	Tietolähteen määrittely	14
2.3.2	Suodattaminen	14
2.3.3	Järjestely	15
2.3.4	Ryhmittely	15
2.3.5	Liitokset	17
3	RESPONSIIVISUUS	18
3.1	Mediakyselyt	19
3.2	Viewport meta tag	21
3.3	Suhteelliset yksiköt	23
3.4	Bootstrap	24
3.4.1	Grid system	24
3.4.2	Components	26
4	CASE: MARTTEN FINLAND OY	27
4.1	Määrittely	27
4.2	Asiakkaan vaatimukset	27
4.3	Toteutus	28
4.3.1	Entity Frameworkin asennus	30
4.3.2	Tietorakenteiden luonti (Data Models)	30
4.3.3	Tietokantakonteksti (Database context)	31
4.3.4	Käyttöliittymien toteutus	31
5	YHTEENVETO	34
	LÄHTEET	35

1 JOHDANTO

Web-sovellusten käyttö työelämässä yleistyy ja niitä hyödynnetään jo lähes jokaisella alalla. Käyttäjinä toimivat usein toimistotyöntekijät perinteisillä tietokoneilla, mutta yhä useammin samoja web-sovelluksia käyttävät myös työntekijät, jotka liikkuvat työpäivän aikana useissa kohteissa. He käyttävät sovelluksia yleensä mobiililaitteilla työtehtävien suorittamisen ohessa. Tästä syystä responsiivisten käyttöliittymien suunnittelu on erityisen tärkeää.

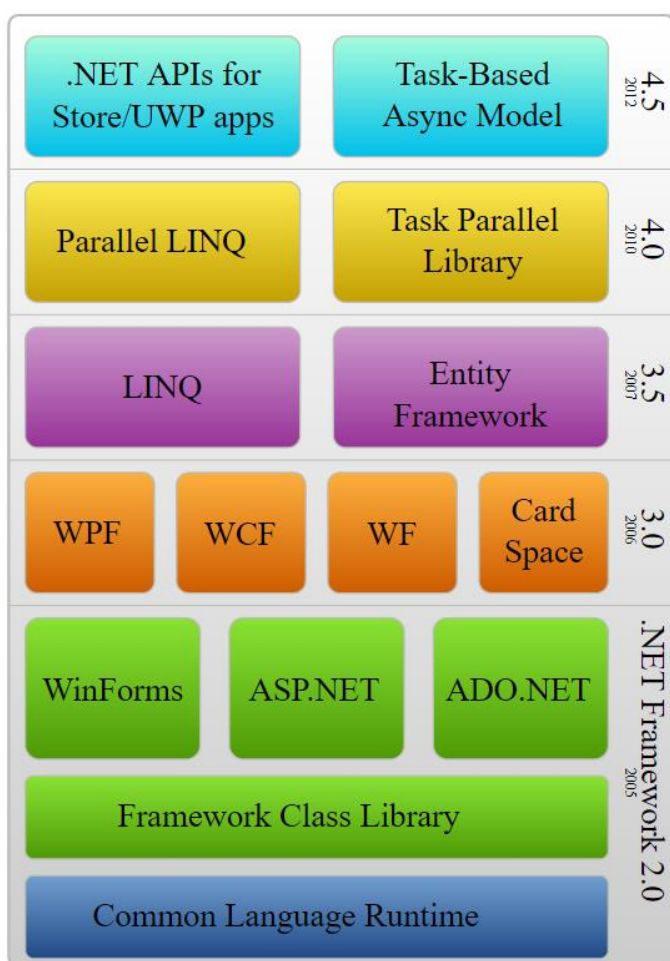
Työn toimeksiantajana toimii vuonna 2004 perustettu Martten Finland Oy, joka on suomalainen tulostimia, tulostusmateriaaleja sekä tulostamiseen liittyviä ohjelmistoja tuottava yritys. Sen ydinosaamista ovat erilaiset tulostettavat tarrat ja niissä esitettävä informaatio. Yritys tuottaa myös yksilöllisesti räätälöityjä tulostinohjelmistoratkaisuja eri alojen tarpeisiin. Näitä palveluita yritys myy myös ulkomaille Suomessa, Ruotsissa ja Virossa toimivien myyntiverkostojen kautta. Yrityksen pääkonttori sijaitsee Lahdessa ja siellä työskentelee tällä hetkellä noin 26 työntekijää.

Tämän opinnäytetyön tarkoitus on tuottaa laitoshuoltoalan tarpeisiin räätälöity toiminnanohjausjärjestelmä, joka hyödyntää QR-kooditarroja. Sovellusta tullaan käyttämään pääosin mobiililaitteilla, mutta myös tavallisilla tietokoneilla. Käyttöliittymien toteutuksessa on tärkeää, että ne skaalautuvat eri kokoisille näytöille. Samalla niiden on oltava selkeitä ja yksinkertaisia, koska sovellusta pitää pystyä käyttämään laitoshuollon työtehtävien suorittamisen yhteydessä vaivattomasti.

Opinnäytetyön alussa kerrotaan lyhyesti .NET kehitysympäristön eri osista ja niiden käyttötarkoituksista. Entity Frameworkiin ja sen toimintaperiaatteisiin tutustutaan syvemmällä tasolla. Työssä käydään läpi, miten Entity Frameworkissa sovelluksen olioiden ja tietokantataulujen välinen yhteys toimii ja miten datan käsittely siinä käytännössä tapahtuu. Lisäksi tutkitaan mitä Entity Frameworkin osat tekevät, miten ne lisätään sovellusprojektiin ja esitellään erilaisia tapoja käyttää Entity Frameworkia. Opinnäytetyössä tutkitaan myös mihin responsiivinen web-suunnittelu käytännössä perustuu ja lopuksi esitellään Bootstrap frameworkin keskeisimpiä ominaisuuksia.

2 .NET FRAMEWORK

.NET Framework (Kuva 1) on Microsoftin julkaisema kehitysalusta Windows-käyttöjärjestelmille. Se on tarkoitettu verkkosovellusten, sekä Windows, Windows Phone, Windows Server ja Microsoft Azure sovellusten kehittämiseen. Se koostuu pääasiassa kahdesta osasta: Common Language Runtime (CLR) ajoympäristöstä, joka vastaa ohjelmien suorittamisesta ja esimerkiksi muistinhallinnasta ja .NET Framework luokkakirjastosta, joka sisältää valmiita luokkia ja käyttöliittymiä yleisimpien perustason operaatioiden suorittamiseen. (Microsoft 2019.)



Kuva 1. .NET Framework stack (Commons Wikimedia 2007)

.NET Framework mahdollistaa myös eri ohjelmointikielten yhteiskäytön kääntämällä ne ensin Common Intermediate Language (CIL) välikielimuotoon, jonka jälkeen CLR kääntää ne ajon aikana käyttöjärjestelmän luettavaan binäärimuotoon. .NET Frameworkia tukevia kieliä ovat mm. C#, F# ja Visual Basic. (Microsoft 2020.)

.NET Framework sisältää myös eri käyttötarkoituksiin erikoistuneita kirjastoja, kuten ASP.NET web-sovelluksille, ADO.NET datan käsittelyyn, Windows Communication Foundation (WCF) service-tyyppisille sovelluksille ja Windows Presentation Foundation (WPF) työpöytäsovelluksille. (Microsoft 2019.)

2.1 ASP.NET

Web-sovellusten kehittämiseen tarkoitettu ASP.NET-kirjasto tarjoaa yhteensä kolme eri sovelluskehystä: Web Forms, ASP.NET MVC ja Web Pages. Jokaisella niistä voi hyödyntää kaikkia ASP.NET:in tarjoamia ominaisuuksia ja niitä kaikkia voi käyttää samassa sovelluksessa. Ne ovat kuitenkin kehitystyyliltään erilaisia ja painottavat sovelluskehityksen eri osa-alueita. Valinta niiden välillä voidaan tehdä tuottavuusperustein tai kehittäjän aikaisemman osaamisen pohjalta.

Web Forms:lla voidaan luoda nopeasti siistejä ja toimivia käyttöliittymiä datan käsittelyyn ja esittämiseen. Sillä voidaan esimerkiksi generoida datamallin pohjalta automaattisesti tarvittavat HTML, Javascript ja CSS koodit käyttöliittymälle, jossa on perus CRUD-toiminnot. Se soveltuu siis hyvin esimerkiksi erilaisten järjestelmän hallintapuolen käyttöliittymien luomiseen.

ASP.NET MVC nimensä mukaisesti perustuu MVC-arkkitehtuuriin, eli se erottaa selkeästi sovelluksen mallit, käyttöliittymän ja toiminnallisuuden toisistaan, sekä antaa kehittäjälle täyden hallinnan näiden osa-alueiden muokkaamiseen. ASP.NET:in kolmesta kehiksestä se soveltuu parhaiten laajojen ja monimutkaisten sovellusten kehittämiseen. Se antaa myös parhaat valmiudet testivetoiseen kehitykseen.

Web Pages:lla voidaan luoda helposti ja nopeasti dynaamista dataa sisältäviä sivuja. Siinä HTML ja itse koodi kirjoitetaan samaan tiedostoon ASP.NET Razor-syntaksia käyttäen. Razor luo kaavoihin perustuvaa HTML:ää käyttäen C# tai VB.NET-kielellä kirjoitettua logiikkaa. Web Pages on ASP.NET kehiksestä kevyin ja yksinkertaisin. (Microsoft 2019.)

2.2 Entity Framework

Entity Framework (EF) on ns. ”object-relational mapper” (ORM), joka on suunniteltu .NET ympäristöön. Sillä pyritään vähentämään relaatiotietokannassa olevien tietojen käsittelyyn vaadittavan koodin kirjoittamista olio-ohjelmoinnissa. Entity Framework on tehty tukemaan erityisesti data orientoituneiden sovellusten kehitystä. Tavoitteena on mahdollistaa datan luominen ja ylläpitäminen käsitetason olioilla välittämättä taustalla olevasta tietokantarakenteesta.

Kun sovellusta tai palvelua suunnitellaan, se jaetaan usein kolmeen osaan: domain-, looginen- ja fyysinen malli. Domain malli määrittelee entiteetit ja niiden väliset suhteet järjestelmässä. Looginen malli määrittelee niitä vastaavan taulut tietokannassa ja niiden viiteavaimet. Fyysinen malli sisältää käytettävän tietokantamoottorin ominaisuudet määrittelemällä varaston yksityiskohdat kuten osiointiin ja indeksointiin liittyvät tiedot.

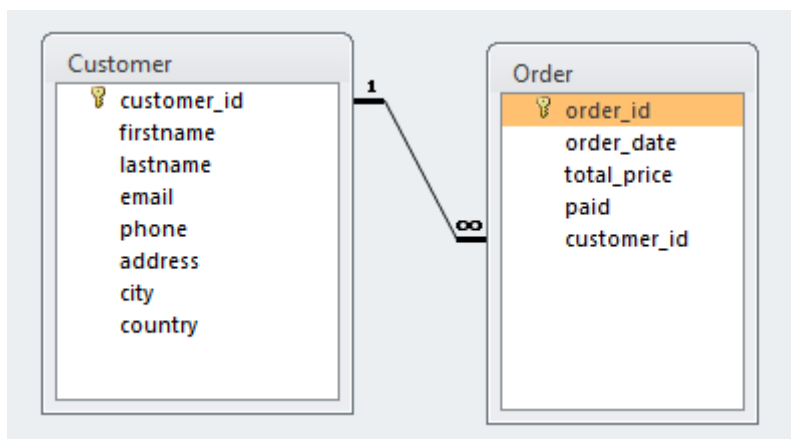
Fyysisen mallin valmistaa usein tietokannan ylläpitäjä, mutta ohjelmakoodia kirjoittavat ohjelmoijat työskentelevät pääsääntöisesti loogisen mallin parissa. Sitä varten joudutaan kirjoittamaan SQL-kyselyitä ja kutsumaan valmiiksi tallennettuja proseduureja. Domain mallia käytetään usein vain projektin alkuvaiheessa sovelluksen vaatimusten määrittelyyn diagrammien muodossa. Tai sitten domain mallia ei luoda ollenkaan, vaan siirrytään suoraan määrittelemään tauluja, sarakkeita ja avaimia relaatiotietokannassa.

Entity Framework mahdollistaa kehittäjille kyselyjen luomisen domain mallissa, eli käsitetasolla ja Entity Framework kääntää ne tietolähdekohtaisiksi komennoiksi. Näin sovelluksissa päästään eroon kovakoodatuista tiettyyn tietolähteeseen sidotuista kyselyistä.

2.2.1 Olioiden mappaus

Olio-ohjelmoinnin yksi haasteista on tiedontallennusjärjestelmiin tallennetun tiedon käsittely. Vaikka sovelluksen luokat usein vastaavat tietokannan tauluja, yksi luokka voi sisältää tietoa useammasta tietokantataulusta ja niiden välisten relaatioiden esitystapa voi poiketa toisistaan.

Esimerkkinä tilanne (Kuva 2), jossa pitää esittää tietyn myyntitilauksen asiakas. Myyntitilaukselle on luotu Order-luokka, jonka ominaisuutena on viittaus Customer-luokan instanssiin, kun taas Order-tilauksessa tietokannassa on yksi tai useampi sarakke viiteavaimille, jotka vastaavat pääavainta Customer-tilauksessa. Customer-luokalla taas voi olla ominaisuus, joka sisältää kokoelman Order-luokan instansseja eikä Customer-tietokantataulussa ole vastaavaa saraketta.



Kuva 2. Customer ja Order -luokkakaavio (Stackoverflow 2014)

Entity Framework tarjoaa kehittäjille joustavan tavan toteuttaa edellä mainitun relaatioiden esitystavan. Perinteiset ratkaisut mappaavat vain luokat ja niiden ominaisuudet niitä vastaaviin tietokantatauluihin ja sarakkeisiin, mutta Entity Framework mappaa loogisen mallin taulut, sarakkeet ja viiteavaimet erillisiksi käsitetason entiteeteiksi esittäen niiden väliset relaatiot aikaisemmassa esimerkissä kuvaillulla tavalla.

2.2.2 Datat käsittely

Olio-relaatio mappaus ratkaisun lisäksi Entity Frameworkin pohjimmainen tarkoitus on mahdollistaa sovellusten käyttämän tiedon käsittely käsitetason entiteetteinä ja niiden välisinä suhteina. EF kääntää käsitetason entiteetteihin kohdistuneet kyselyt tietolähdekohtaiseen muotoon käyttäen model ja mapping tiedostoissa olevia määrittelyjä. Kyselyjen tulokset esitetään vastaavasti käsitetason olioina.

Entity Frameworkissa on kaksi tapaa esittää kyselyitä käsitetason malleja vasten:

- LINQ to Entities, joka tarjoaa tuen LINQ (Language-Integrated Query) muotoisten kyselyiden kirjoittamisen käsitetason malleille.
- Entity SQL, joka on eräänlainen geneerinen SQL:ää muistuttava kieli. Entity Framework kääntää sillä kirjoitetut kyselyt tietojärjestelmäkohtaiseen muotoon.

(Microsoft 2018.)

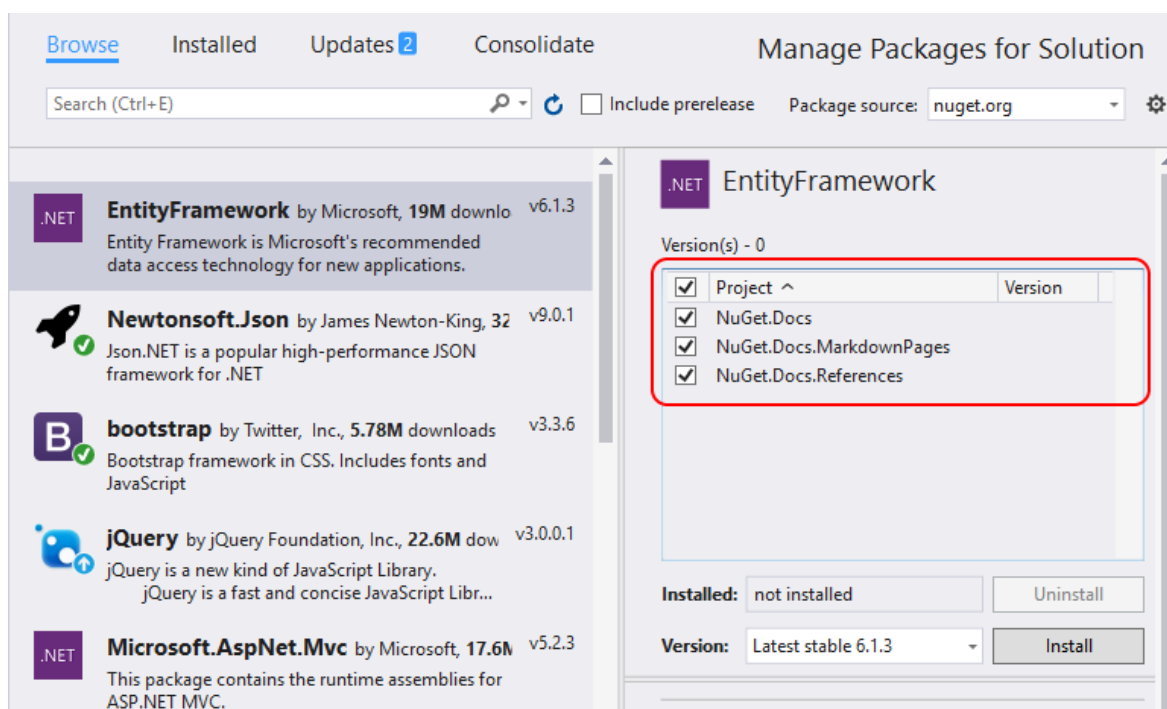
2.2.3 EF Tools ja EF Runtime

Entity Framework koostuu kahdesta eri osasta, jotka ovat EF Tools ja EF Runtime. Entity Frameworkia voi käyttää myös ilman EF Toolsia. Se sisältää EF Designer ja EF Model

Wizard työkalut, jotka ovat Visual Studioon integroituja käyttöliittymiä. Niitä tarvitaan, jotta Entity Frameworkin Model First ja Database First työjärjestyksiä (workflow) voidaan hyödyntää. Näitä työjärjestyksiä käsitellään tarkemmin tulevissa kappaleissa.

EF Tools on mukana kaikkien viimeisimpien Visual Studio versioiden vakioasennuksessa aina Visual Studio 2010:een asti. EF Tools on myös päivitettävissä todella pitkän ajan takaa, sillä vanhojen Visual Studio versioiden sisältämä EF Tools voidaan erillisillä asennuspaketeilla päivittää uusimpaan versioon jopa Visual Studio 2012:sta asti. Vasta Visual Studio 2010 version sisältämää EF Toolsia ei pysty mitenkään päivittämään yhteensopivaksi Entity Framework 6 Runtimein kanssa. Tämä johtuu siitä, että tässä versiossa EF Tools toimii yhdessä Entity Framework 4:n kanssa, joka eroaa uudemmista EF versioista siten, että sen koodin generointi perustuu EntityObject luokkaan, kun taas Entity Framework 5:stä eteenpäin koodin generointi perustuu DbContext luokkaan. EF 4:n koodin generointi voidaan kuitenkin manuaalisesti päivittää perustumaan DbContext luokkaan, mutta silloin menetetään Visual Studioon integroidut EF käyttöliittymät eli Model First ja Database First työjärjestysten käyttäminen ei ole mahdollista. Tässäkin tapauksessa Code First työjärjestystä voidaan kuitenkin vielä käyttää, koska se ei tarvitse EF Tools käyttöliittymiä. EF Tools on joka tapauksessa kaiken kaikkiaan täysin valinnainen Entity Frameworkin osa ja se voidaan jättää myös uudemmissa Visual Studio versiossa kokonaan asentamatta suorittamalla mukautettu Visual Studio asennus.

EF Runtimein eli varsinaisen Entity Frameworkin voi asentaa NuGet pakettina. NuGet on eräänlainen .NET kehitysympäristöä varten tehty kirjastojen jakamistyökalu. NuGet työkalut mahdollistavat .NET kirjastojen luomisen ja jakamisen ns. ”NuGet paketteina (NuGet packages)”. (Microsoft 2020) Visual Studio sisältää oman NuGet pakettien hallintatyökalun NuGet Package Managerin. Sitä käyttämällä EF Runtimein voi asentaa joko graafisella käyttöliittymällä (Kuva 3) tai konsolikomennolla 'Install-Package EntityFramework'. EF Runtime on julkaistu 4.1 versiosta eteenpäin erillisenä NuGet pakettina. Sitä vanhempia versioita ei voi asentaa erikseen vaan ne ovat tulleet osana .NET Framework kokonaisuutta. (Microsoft 2016.)



Kuva 3. NuGet Package Manager käyttöliittymä (Microsoft 2019)

2.2.4 Työjärjestykset (workflows)

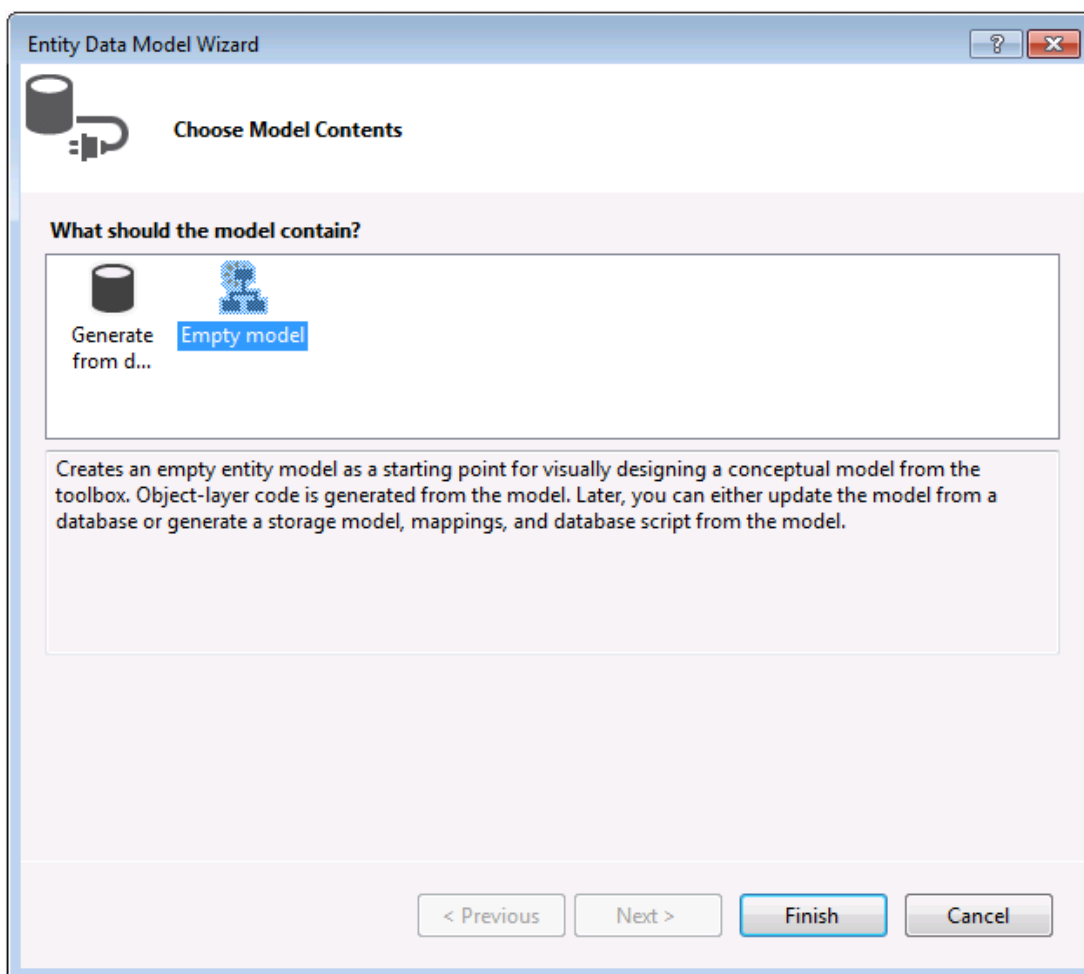
Entity Framework luo mallin, joka määrittelee miten sovelluksen luokat ja niiden ominaisuudet ovat yhteydessä tietokannan tauluihin ja sarakkeisiin. Tämän mallin luonti jakautuu neljään eri lähestymistapaan tai työjärjestykseen (workflow), jotka ottavat suurimman roolin lähinnä projektin perustamisvaiheessa. Valintaan näiden välillä vaikuttaa ensinnäkin se, halutaanko käyttää mallin luontiin EF Toolsin tarjoamia Visual Studioon integroituja käyttöliittymiä vai määritellä ne kirjoittamalla vain suoraan koodia. Lisäksi valintaan vaikuttaa myös se, onko sovellusta varten olemassa jo ennestään tietokanta vai pitääkö se luoda.

Entity Frameworkin mahdollisia työjärjestyksiä ovat:

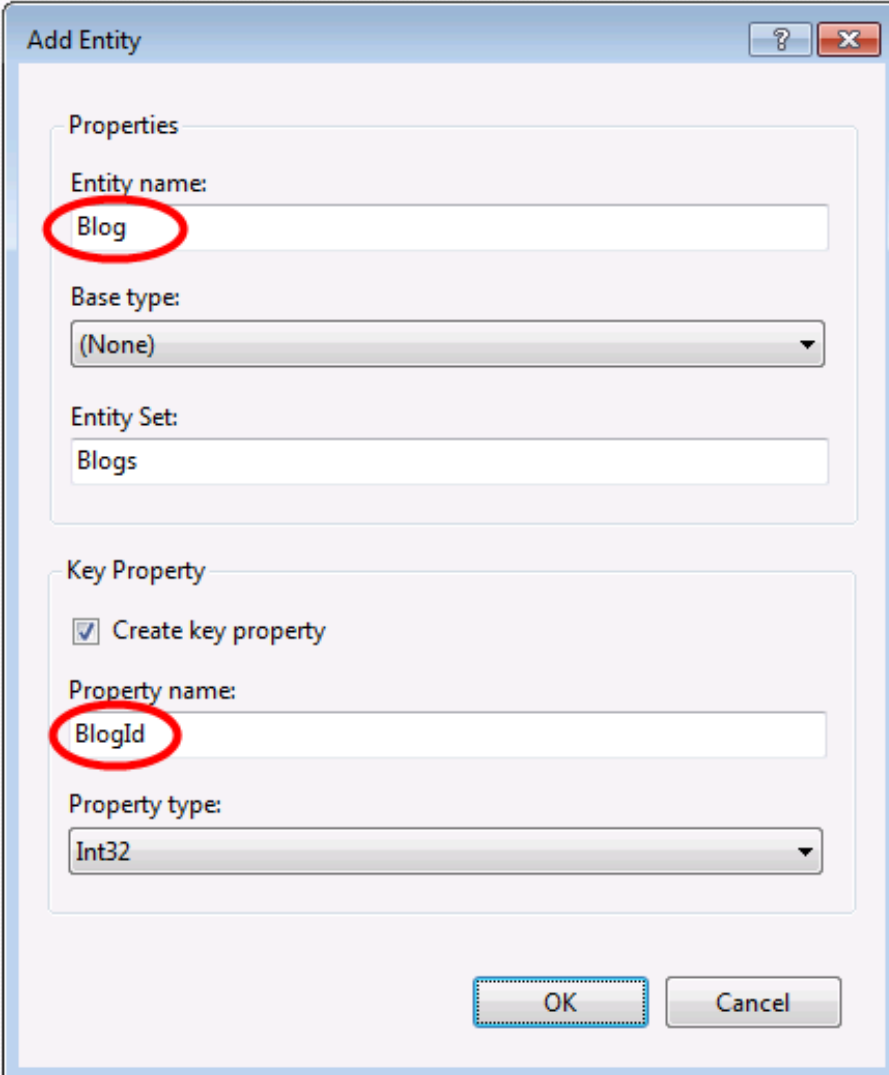
- Code First uudella tietokannalla
- Code First olemassa olevalla tietokannalla
- Model First
- Database First

Model First ja Database First tapojen käyttöön vaaditaan EF Tools. Näiden välinen ero on se, että Model First ei vaadi olemassa olevaa tietokantaa, kun taas Database First vaatii. Nämä ovat siis Visual Studioon integroituja käyttöliittymiä hyödyntäviä lähestymistapoja. Code First tapaa taas voidaan käyttää joko hyödyntämällä olemassa olevaa tietokantaa tai vastaavasti luomalla tietokanta sovellukseen koodattujen luokkien pohjalta. (Microsoft 2018.)

Model First tavassa määritellään sovelluksen entiteetit ja niiden väliset assosiaatiot hie-
man tavallisen Windows-ohjelman asennusvelhoa muistuttavalla Entity Data Model Wi-
zard käyttöliittymällä (Kuva 4 ja Kuva 5).



Kuva 4. Entity Data Model Wizard aloitus - Empty model (Microsoft 2016)



The image shows a screenshot of the "Add Entity" dialog box from the Microsoft Entity Data Model Wizard. The dialog is titled "Add Entity" and has a question mark icon and a close button in the top right corner. It is divided into two main sections: "Properties" and "Key Property".

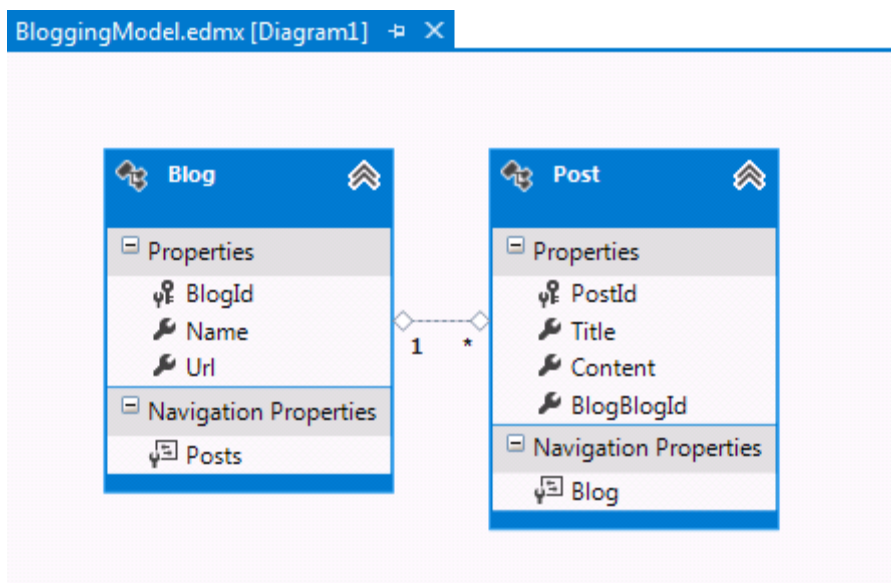
In the "Properties" section, the "Entity name:" field contains the text "Blog", which is circled in red. Below it, the "Base type:" dropdown menu is set to "(None)". The "Entity Set:" field contains the text "Blogs".

In the "Key Property" section, the "Create key property" checkbox is checked. The "Property name:" field contains the text "BlogId", which is also circled in red. The "Property type:" dropdown menu is set to "Int32".

At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

Kuva 5. Entity Data Model Wizard uuden entiteetin luonti käyttöliittymä (Microsoft 2016)

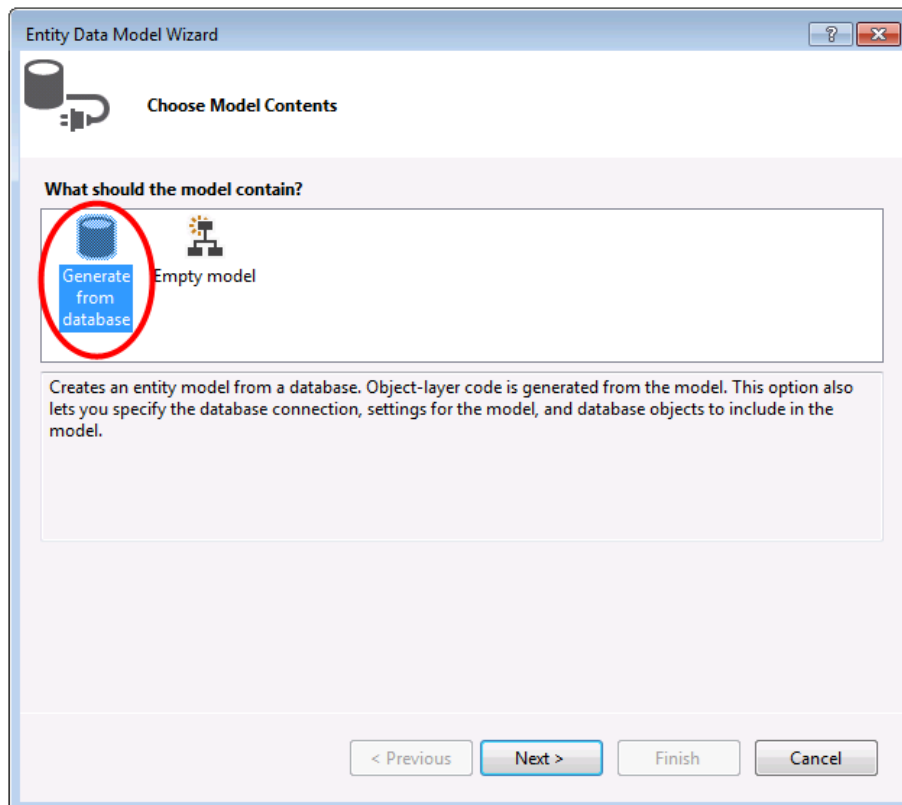
Model First tavalla määritellyt entiteetit esitetään EF Designerissä eräänlaisena diagrammina. Käytännössä tämä diagrammi koostuu laatikoista, jotka sisältävät tietokantataulujen kentät ja laatikoiden välille vedetyistä viivoista, jotka kuvastavat tietokantataulujen välisiä yhteyksiä (Kuva 6).



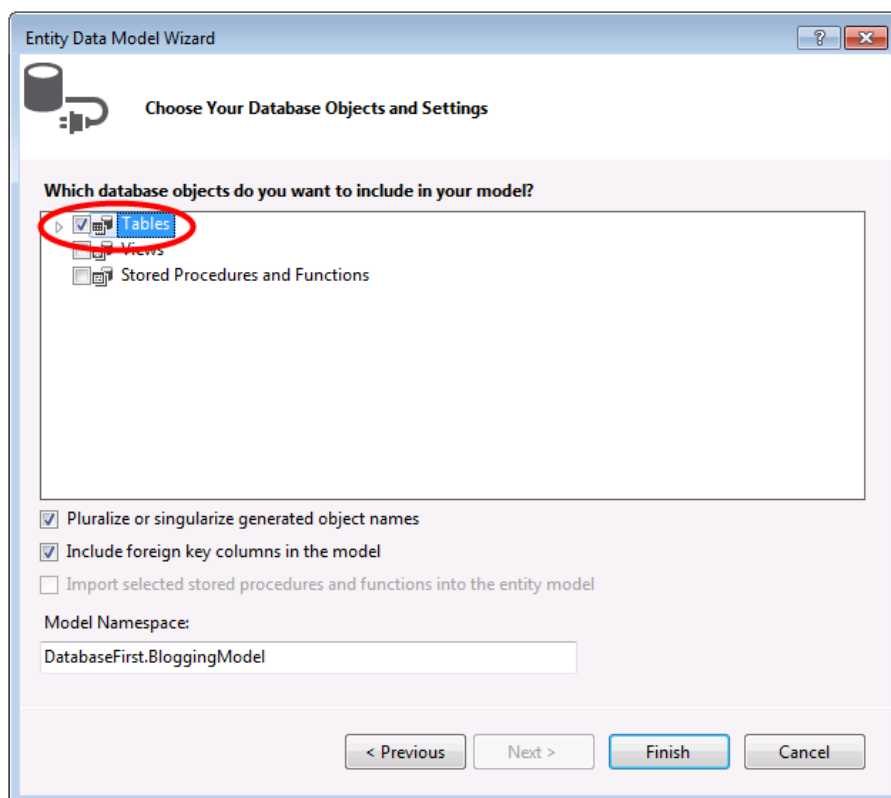
Kuva 6. EF Designer diagrammi (Microsoft 2016)

Entiteettien määrittelyn jälkeen Entity Framework luo automaattisesti tämän diagrammin pohjalta tietokannan ja lopuksi tietokantaa hyödyntämällä EF generoi myös automaattisesti objekteja vastaavat luokat, joita sovelluksen koodissa käsitellään.

Database First lähestymistapa on oikeastaan melkein sama kuin Model First, mutta käänteisessä järjestyksessä. Käyttöliitymässä valitaan tyhjän "Empty model" vaihtoehdon sijaan toinen vaihtoehto "Generate from database" eli generoidaan tietokannasta (Kuva 7). Tämän jälkeen tarvitsee vain valita tietokantayhteyden määrittävä olemassa oleva .dbo-tiedosto tai luoda sellainen. Halutessaan myös eritellä mitkä tietokannan tauluista halutaan sisällyttää malliin. Lisäksi voidaan valita, muutetaanko mallin objektien nimet yksiköön tai monikkoon riippuen kannassa olevasta muodosta tai sisällytetäänkö myös viiteavain sarakkeet malliin (Kuva 8). Sen jälkeen Entity Data Model Wizard pystyy automaattisesti luomaan EF mallin tietokannan perusteella.



Kuva 7. Entity Data Model Wizard aloitus - Generate from database (Microsoft 2016)



Kuva 8. Database First taulut ja lisäasetukset (Microsoft 2016)

Code First lähestymistapaa käyttämällä on mahdollista antaa Entity Frameworkin luoda sovellusta varten automaattisesti uusi tietokanta tai lisätä olemassa olevaan tauluja koodin perusteella. Sen myötä tietokantaan saadaan myös Entity Frameworkin migraatioihin perustuva versiohallinta, joka generoi tietokantamuutoksiin tarvittavat koodit automaattisesti seuraamalla sovelluksen objektiluokkiin tulevia muutoksia.

Code First tavassa pitää ensin luoda objektiluokat, joista muodostetaan myöhemmin tietokantataulut (Kuva 9). Objekteille voidaan myös määrittää navigaatio-ominaisuuksia, jotka ovat tietyn viiteavaimen perusteella viitteitä toisiin objekteihin. Esimerkiksi kuvan Blog luokan Posts on navigaatio-ominaisuus, joka on lista Post objekteja, joiden viiteavaimena toimii Post luokan BlogId arvo. Tällaisissa navigaatio-ominaisuuksissa on nopeampien latausajkojen kannalta järkevää käyttää Entity Frameworkin Lazy Loading ominaisuutta. Sen myötä navigaatio-ominaisuuksien tiedot ladataan tietokannasta vasta siinä vaiheessa, kun niitä halutaan käsitellä sovelluksessa. Lazy Loading astuu voimaan, kun ominaisuuden edessä käytetään *virtual* avainsanaa.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public virtual Blog Blog { get; set; }
}
```

Kuva 9. Blog ja Post luokkien määrittely (Microsoft 2016)

Jotta näiden luokkien instansseja voidaan tallentaa tietokantaan ja suorittaa kyselyjä niitä vasten, tarvitaan Entity Frameworkin DbContext luokasta johdettu kontekstiluokka (Kuva 10). Se kuvastaa sessiota, jossa tietokannan tietoja käsitellään ja sitä käytetään tavallisesti *using* lausekkeessa. Näin varmistetaan, että kontekstin käyttämät resurssit

vapautuvat jälleen, kun tietokantaa ei enää tarvita. Kontekstiluokkaan määritellään jo-
kaista aiemmin määriteltyä objektityyppiä vastaava DbSet ominaisuus, jotka mahdollista-
vat kyselyjen suorittamisen ja tietojen tallentamisen näille tyypeille.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

Kuva 10. Tietokantakonteksti-luokka (Microsoft 2016)

Tietokantakontekstin määrittelyn jälkeen Entity Framework luo automaattisesti uuden sitä
vastaavan tietokannan, kun kontekstin sisältämien objektien tietoja käsitellään. Olemassa
olevaan kantaan muutokset tehdään edellä mainittujen migraatioiden avulla, kunhan kan-
tayhteys on ensin määritelty. (Microsoft 2016.)

2.3 LINQ

LINQ nimitys tulee sanoista Language-Integrated Query. Se on .NET Framework kompo-
nenti, joka koostuu monesta teknologiasta, joiden periaatteena on kyselyominaisuuksien
integrointi suoraan C# -kieleen. Perinteiset kyselyt koostuvat yksinkertaisista merkkijo-
noista, joille ei tehdä käännösvaiheessa tyyppitarkastusta, eikä niille ole myöskään Intelli-
Sense tukea. Lisäksi jokaisella tietolähteellä, kuten SQL-tietokannoilla, XML-dokumen-
teilla ja eri verkkopalveluilla on yleensä oma kyselykielensä, jotka täytyy opetella erikseen.
LINQ:n tapauksessa, kyselyt muodostuvat yhdestä yhteisestä C# -kieleen perustuvasta
kielirakenteesta. Kyselyt kirjoitetaan vahvasti tyyhitettyjä objektikokoelmia vasten käyttäen
C# -kielen avainsanoja ja operaattoreita. LINQ teknologiaperhe mahdollistaa yksittäisen
yhtenäisen kyselykielen objekteille (LINQ to Objects), relaatiotietokannoille (LINQ to SQL)
ja XML-dokumenteille (LINQ to XML).

Kyselyjä kirjoittavalle ohjelmoijalle LINQ:n selkeimmin näkyvissä olevia kieleen integroituja
osia ovat kyselylausekkeet. Kielilausekkeet kirjoitetaan deklarativisella kyselysyntaksilla,
jolla voidaan suorittaa esimerkiksi suodattamis-, järjestely- ja ryhmittelyoperaatioita. Kyse-
lyjä voidaan kirjoittaa useisiin eri tietolähteisiin huomattavasti pienemmällä määrällä koo-
dia, kuin perinteisillä kyselyillä. Käytännössä LINQ kyselyitä voidaan kirjoittaa mitä ta-
hansa objektikokoelmaa vasten, joka tukee IEnumerable- tai geneeristä IEnumerable<T>

käyttöliittymää. LINQ tukea tarjoavat myös useat kolmannen osapuolen verkkopalvelut ja tietokantaimplementaatiot. (Microsoft 2017.)

Seuraavissa alakappaleissa esitellään muutamia yleisimpiä operaatioita, joita LINQ kyselyssä voidaan suorittaa. Ennen varsinaisia kyselyoperaatioita kerrotaan ensin, miten määritellään tietolähde, jota kyselyssä halutaan käsitellä.

2.3.1 Tietolähteen määrittely

C# -kielessä, kuten useimmissa muissakin ohjelmointikielissä, muuttujat on esiteltävä ennen kuin niitä voidaan käyttää. LINQ-kyselyn *from* lauseella esitellään esimerkissä (Kuva 11) tietolähde *customer* ja ns. välimuuttuja (range variable) *cust*.

```
//queryAllCustomers is an IEnumerable<Customer>  
var queryAllCustomers = from cust in customers  
                        select cust;
```

Kuva 11. LINQ tietolähteen määrittely (Microsoft 2015)

Välimuuttuja vastaa *foreach*-silmukan iterointimuuttujaa, mutta siitä poiketen varsinaista iterointia ei tapahdu kyselylausekkeessa. Vasta kun kysely suoritetaan, kääntäjä käyttää välimuuttujaa referenssinä jokaiselle kokoelman *customers* sisältämälle yksittäiselle elementille. Koska kääntäjä osaa päätellä välimuuttujan *cust* tyyppin, sitä ei tarvitse määritellä erikseen. Välimuuttuja voidaan myös esitellä useampia käyttämällä *let*-lauseetta.

2.3.2 Suodattaminen

Yleinen varsinainen kyselyoperaatio on suodatin, joka asetetaan kyselyyn totuusarvon määritelmän muodossa. Suodatin vaikuttaa kyselyyn siten, että se palauttaa vain sellaiset elementit, joiden kohdalla määritelmä on tosi. Suodattimella voidaan siis määritellä, mitkä lähdedatan elementit otetaan mukaan kyselyn tulokseen. Tulos tuotetaan käyttämällä *where*-lauseetta. Esimerkkitapauksessa () palautetaan vain ne asiakkaat, joilla on osoite Lontoossa.

```
var queryLondonCustomers = from cust in customers  
                          where cust.City == "London"  
                          select cust;
```

Kuva 12. LINQ where-lause (Microsoft 2015)

Suodattimen määritelmässä voidaan tarvittaessa käyttää myös C# -kielen omia *and* ja *or* operaattoreita. Jos esimerkiksi halutaan palauttaa Lontoossa asuvista asiakkaista vain ne, joiden nimi on Devon, kyselyä laajennettaisiin kuten käyttämällä *and*-operaattoria (Kuva 13).

```
where cust.City == "London" && cust.Name == "Devon"
```

Kuva 13. LINQ suodattimen määrittely (Microsoft 2015)

2.3.3 Järjestely

Usein on käytännöllisyyden nimissä kannattavaa asettaa palautettava data tiettyyn järjestykseen. Tämä tapahtuu käyttämällä *orderby*-lausetta. Sen seurauksena palautettavan kokoelman elementit järjestetään järjesteltävän tyyppin oletusvertailijan mukaan. Esimerkiksi aiemmassa esimerkissä esitetyn Lontoo-kyselyn tulokset voidaan järjestellä aakkosjärjestykseen *cust* elementtien *Name* ominaisuuden perusteella (Kuva 14).

```
var queryLondonCustomers3 =  
    from cust in customers  
    where cust.City == "London"  
    orderby cust.Name ascending  
    select cust;
```

Kuva 14. LINQ orderby-lause (Microsoft 2015)

2.3.4 Ryhmittely

Kyselyn tuloksia voidaan ryhmitellä jonkun tietyn määriteltävän avainarvon perusteella käyttämällä *group*-lausetta. Ryhmittelyä voidaan tehdä myös keskellä kyselyä. Tässä tapauksessa kyselyä jatketaan käyttämällä *into* avainsanaa, mutta sen jälkeen kysely on lopulta päätettävä joko *select*-lauseella tai uudella *group*-lauseella. Esimerkissä (Kuva 15) on ensin ryhmitelty *students* kokoelman *student* elementit sukunimen ensimmäisen

kirjaimen mukaan ja sen jälkeen järjestelty tästä muodostuneet ryhmät aakkosjärjestykseen ryhmän avainarvon, eli sukunimen ensimmäisen kirjaimen mukaan.

```
var studentQuery2 =  
    from student in students  
    group student by student.Last[0] into g  
    orderby g.Key  
    select g;
```

Kuva 15. LINQ group-lause (Microsoft 2015)

Jos kysely päätetään *group*-lauseeseen, tuloksena saadaan lista ryhmiteltyjä elementtejä. Listassa jokainen elementti on objekti, jolla on *group*-lauseessa määritelty avain *Key* nimisenä ominaisuutena ja lisäksi avainarvon perusteella ryhmitellyt elementit alkuperäisestä lähdekokoelmasta. Jos *group*-lauseeseen päätetyn kyselyn tuloksia iteroidaan, se on tehtävä käyttämällä kahta sisäkkäistä *foreach*-silmukkaa. Ulommassa silmukassa käydään läpi ryhmittelyssä muodostuneita ryhmiä ja sisemmässä silmukassa jokaisen ryhmän yksittäiset elementit (Kuva 16).

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>  
var queryCustomersByCity =  
    from cust in customers  
    group cust by cust.City;  
  
// customerGroup is an IGrouping<string, Customer>  
foreach (var customerGroup in queryCustomersByCity)  
{  
    Console.WriteLine(customerGroup.Key);  
    foreach (Customer customer in customerGroup)  
    {  
        Console.WriteLine("    {0}", customer.Name);  
    }  
}
```

Kuva 16. Ryhmiteltyjen tulosten iterointi (Microsoft 2015)

2.3.5 Liitokset

Liitosoperaatiolla voidaan luoda kahden kokoelman välille sellaisia yhteyksiä, joita ei ole ennestään suoraan mallinnettu kokoelmien tietolähteissä. Tämä tehdään käyttämällä *join*-lausetta, jolla on esimerkki tapauksessa (Kuva 17) yhdistetty *customers* ja *distributors* lähdekokoelmien elementit kummastakin löytyvän *City* ominaisuuden perusteella. Näistä on luotu uusi kokoelma, jossa jokainen elementti sisältää aina asiakkaan *cust* ja jakelijan *dist* nimet, jotka sijaitsevat samassa kaupungissa. (Microsoft 2015.)

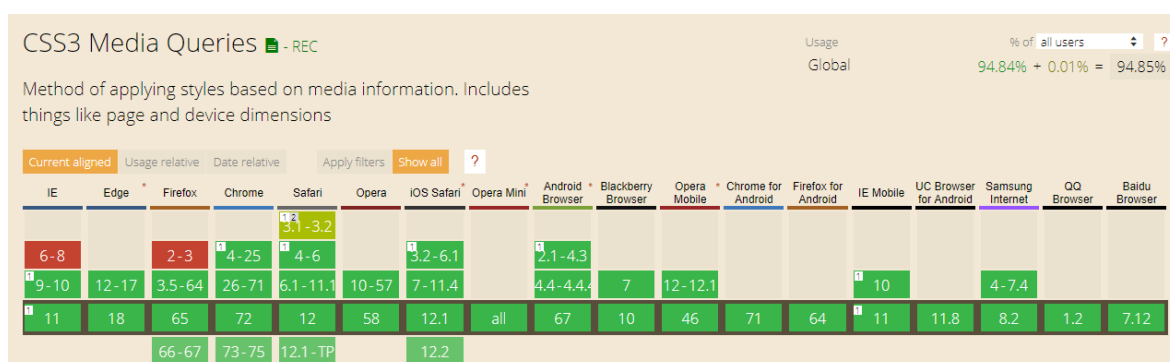
```
var innerJoinQuery =  
    from cust in customers  
    join dist in distributors on cust.City equals dist.City  
    select new { CustomerName = cust.Name, DistributorName = dist.Name };
```

Kuva 17. LINQ join-lause (Microsoft 2015)

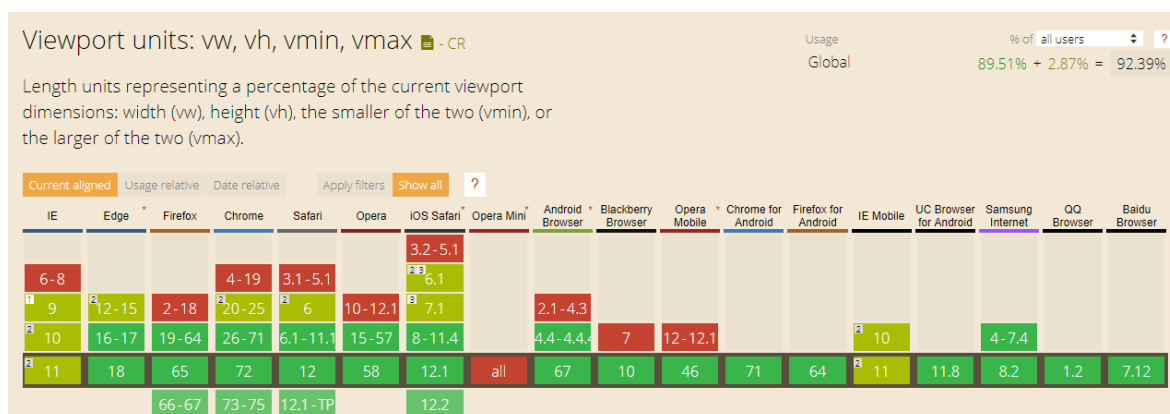
3 RESPONSIIIVISUUS

Responsiivisella suunnittelulla web-käyttöliittymä tehdään mukautumaan jokaiselle käyttö-laitteelle sopivaksi. Niin saadaan yksi yhtenäinen käyttöliittymä, joka on käytettävä ja luet-tava laitteesta ja sen näyttökoosta riippumatta. Käytännössä tämä saavutetaan käyttä-mällä sivun tyylimäärityksessä suhteellisia arvoja kiinteiden sijaan.

Responsiivisen web-suunnittelun mahdollisti HTML5:n ja CSS3:n mukana tulleet mediaky-selyt, viewport-metatagi ja uudet mittayksiköt. Niitä tukevat lähes kaikki yleisimmät selai-met (Kuva 18 ja Kuva 19). (Körner, Christoph 2016, 2)



Kuva 18. CSS3 mediakyselyjä tukevat selaimet (Can I use 2019)



Kuva 19. Viewport-yksiköitä tukevat selaimet (Can I use 2019)

3.1 Mediakyselyt

CSS2 standardi toi uutena ominaisuutena mediatyypit (media type). Ne mahdollistivat sivun ulkoasun määrittelyn mediatyyppin perusteella, eli käytännössä sen perusteella esitetäänkö sivu esimerkiksi tulostettuna paperilla, kannettavalla laitteella, televisiossa vai tietokoneen ruudulla. Esimerkissä (Kuva 20) on määritelty eri fonttikoko ruudulla esitetylle ja tulostetulle sivulle.

```
<style tyle = "text/css">
  <!--
    @media print {
      body { font-size: 10pt }
    }

    @media screen {
      body { font-size: 12pt }
    }
    @media screen, print {
      body { line-height: 1.2 }
    }
  -->
</style>
```

Kuva 20. Mediatyyppi määrittely (TutorialsPoint 2019)

CSS2:ssa esiteltyt mediatyypit ovat:

- all
- braille
- embossed
- handheld
- print
- projection
- screen
- speech
- tty
- tv

Mediatyypit saivat huonosti tukea laitevalmistajilta (W3Schools 2019). Media Queries 4 standardin jälkeen suurin osa näistä poistui käytöstä ja jäljelle jäivät tyypit *all*, *print*, *screen* ja *speech*. (W3C 2018.)

Mediakyselyt (media query) tulivat CSS3:n mukana mediatyyppin laajennuksena. Niillä voidaan tarkastella mediatyyppin lisäksi laitteen ominaisuuksia, jotka vaikuttavat olennaisesti kuvan näyttämiseen. Näitä ovat esimerkiksi ruudun koko, resoluutio ja laitteen asento. Mediakysely on periaatteessa ehtolause, joka sisältää tyylimäärittelyjä (Kuva 21). Lauseessa määritellään mediatyyppi ja lisäksi ominaisuus (media feature), jota halutaan tarkastella. Aaltosulkeiden sisällä olevat tyylimäärittelyt ovat voimassa, jos laitteen ominaisuus vastaa lausekkeessa määriteltyä arvoa.

```
@media [Media Type [and|not|only]] (Media Feature) {
    ...
}
```

Kuva 21. Mediakyselyn perusmuoto (W3Schools 2019)

Responsiivisessä suunnittelussa tarkoitus on saada sivun sisältö esitettyä mahdollisimman sujuvasti eri kokoisilla näytöillä. Sivun tyylimäärittelyt jaetaan tyyppillisesti tiloihin *phone*, *tablet*, *desktop* ja *large*. Tilojen välille määritellään tietyt pisteet (breakpoint), joiden kohdalla siirrytään tilasta toiseen. Nämä pisteet toteutetaan käytännössä mediakyselyillä, joka tarkastelee ruudun kokoa pikseleinä (Kuva 22).



Kuva 22. Column-luokan tyylimäärittely (W3Schools 2019)

3.2 Viewport meta tag

Selaimen viewportilla tarkoitetaan aluetta, jota selain käyttää verkkosivun sisällön esittämiseen käyttäjälle. Se ei ole sama asia kuin itse sivun koko, vaan on usein pienempi. Silloin selain tuo esiin vierityspalkit, joilla käyttäjä voi vierittää sivua vaaka- ja pystysuunnassa nähdäkseen sivun koko sisällön. (Mozilla 2019.)

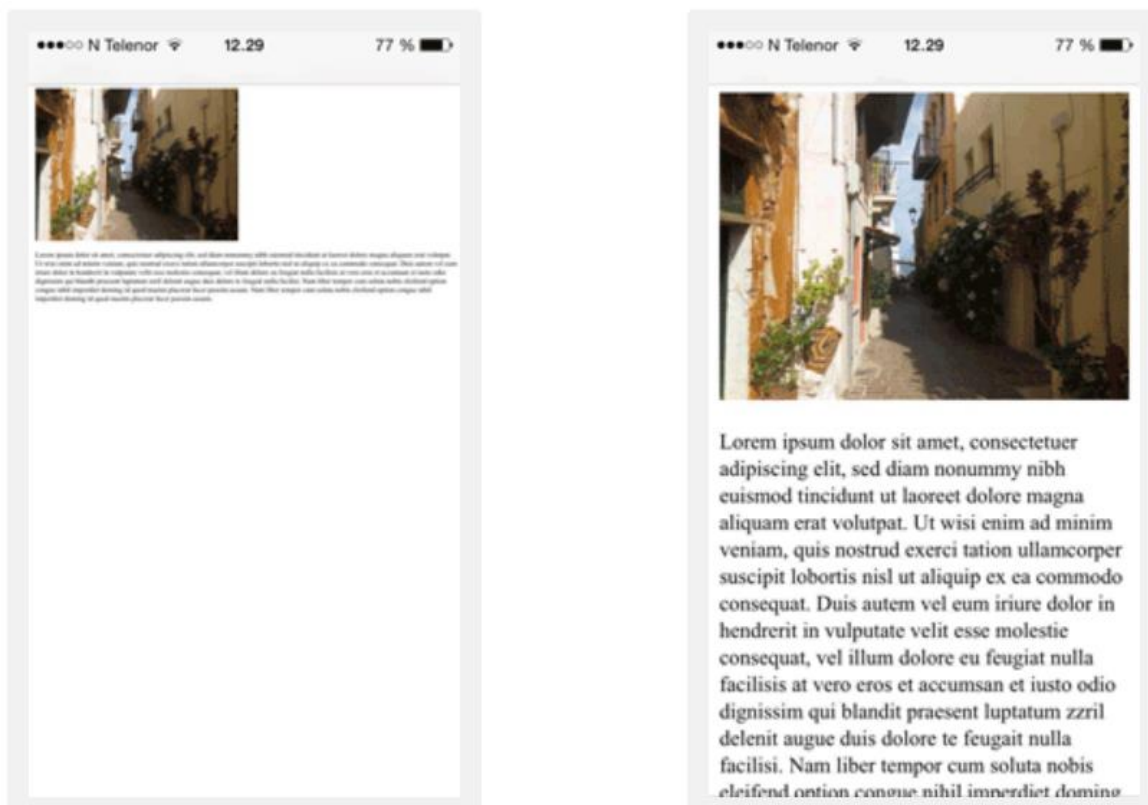
Ennen kuin mobiililaitteet yleistyivät, verkkoa selattiin pääasiassa tietokoneilla. Verkkosivut oli suunniteltu staattisiksi ja tietyn kokoisiksi. Kun internetiä alettiin käyttää mobiililaitteilla, aluksi staattiset sivut vain pienennettiin sopimaan kapeille näytöille. Tällöin suurten sivujen sisällöt näkyivät todella pienenä mobiililaitteiden näytöillä ja käyttäjät joutuivat suurentamaan ja siirtämään näkymää sivun eri osiin nähdäkseen sen sisällön kunnolla. (W3Schools 2020.)

Ratkaisuksi tähän ongelmaan kehitettiin viewport meta tag, jonka Apple esitteli ensimmäisenä iOS:n Safari selaimelle. Sen käyttö on sittemmin yleistynyt ja nykyään monet mobiiliselaimet tukevat sitä. Suurin osa mobiililaitteille optimoiduista verkkosivuista käyttää viewport meta tagia (Kuva 23).

```
1 | <meta name="viewport" content="width=device-width, initial-scale=1">
```

Kuva 23. Viewport meta tag (W3Schools 2019)

Viewport meta tagin width-ominaisuus määrittää viewportin koon. Siihen voidaan syöttää numeerisesti suoraan tietty pikselimäärä tai erikoisarvo "device-width", joka on kulloinkin käytössä olevan laitteen näytön leveys CSS pikseleinä. Vastaavasti on olemassa myös harvemmin käytetty height-ominaisuus ja "device-height"-erikoisarvo. Näiden arvojen avulla verkkosivun sisältö muovautuu viewportin koon mukaan ja näyttyy käyttäjälle selkeämmin (Kuva 24). (Mozilla Developer Network 2019.)



Kuva 24. Viewport meta tag vertailu (W3Schools 2019)

3.3 Suhteelliset yksiköt

CSS-tyylimäärittelyissä voidaan käyttää erilaisia pituuden mittayksiköitä. Osa niistä on ns. absoluuttisia (absolute) ja osa suhteellisia (relative).

Unit	Name
cm	Centimeters
mm	Millimeters
Q	Quarter-millimeters
in	Inches
pc	Picas
pt	Points
px	Pixels

Kuva 25. Absoluuttiset pituuden mittayksiköt (W3Schools 2019)

Unit	Relative to
em	Font size of the parent, in the case of typographical properties like <code>font-size</code> , and font size of the element itself, in the case of other properties like <code>width</code> .
ex	x-height of the element's font.
ch	The advance measure (width) of the glyph "0" of the element's font.
rem	Font size of the root element.
lh	Line height of the element.
vw	1% of the viewport's width.
vh	1% of the viewport's height.
vmin	1% of the viewport's smaller dimension.
vmax	1% of the viewport's larger dimension.

Kuva 26. Suhteelliset pituuden mittayksiköt (W3Schools 2019)

Absoluuttiset yksiköt (Kuva 25) ovat kiinteään kokoisia eivätkä ole suhteessa mihinkään muuhun mittaan. Absoluuttisia yksiköitä ei juurikaan suositella käytettäväksi verkkosivun

ruudulla esitettävän sisällön koon määrittämisessä. Ne ovat kuitenkin hyödyllisiä sellaisissa tilanteissa, missä sisällön koko määräytyy jonkin tunnetun standardin mukaan. Esimerkiksi tulostettavan sivun sisällön kokoa määrittäessä.

Suhteelliset yksiköt (Kuva 26) tukevat paremmin responsiivista web suunnittelua ja soveltuvat paremmin verkkosivulla esitetyn sisällön koon määrittämiseen. Niiden koko määräytyy aina suhteessa johonkin, esimerkiksi ylemmän tason elementin fontin mukaan tai viewportin kokoon. (Mozilla Developer Network 2020.) Käyttämällä suhteellisia yksiköitä saadaan esimerkiksi tekstin tai muiden elementtien koko muuttumaan suhteessa muuhun sisältöön niin, että sisältö pysyy koko ajan viewportin sisällä ja vierityspalkkeja ei tarvita (Google Developers 2020).

3.4 Bootstrap

Bootstrap on maailman käytetyin responsiivinen front-end framework. Se on HTML, CSS ja JavaScript pohjainen ja perustuu nykyään avoimeen lähdekoodiin. Alun perin sen kehitti kaksi Twitterin työntekijää Mark Otto ja Jacob Thornton vuonna 2010. Sen alkuperäinen tarkoitus oli toimia tyylioppaana organisaation sisäisten työkalujen kehittämiseen. Se ehti olla yrityksen omassa käytössä noin vuoden kunnes 2011 syksyllä lähdekoodi julkaistiin yleiseen käyttöön. (Bootstrap 2019.)

Teknisesti Bootstrap koostuu responsiivisista CSS ja LESS määrittelyistä, sekä niitä tukevista HTML ja JavaScript komponenteista (Körner, Christoph 2016, 7).

3.4.1 Grid system

Bootstrapin yksi keskeisimmistä ominaisuuksista on ns. ”grid” eli ruudukko, joka helpottaa responsiivisen verkkosivun layoutin tekemistä. Grid käyttää ”container”, ”row” ja ”column” CSS-luokkia, jotka perustuvat ”CSS Flexible Box Layout”-moduuliin. Flexbox on käyttöliittymien tekemiseen optimoitu versio CSS:n normaalista laatikkomallista. Se käsittelee sisällään olevia elementtejä yksilotteisesti alkaen mistä tahansa suunnasta ja muokkaa niiden leveyttä ja korkeutta käytössä olevan tilan mukaan. (Mozilla Developer Network 2020.)

Bootstrapin ruudukko jakaa näytön 12 sarakkeeseen, jonka mukaan sivun elementtien koko ja sijainti määräytyy. Container-luokalla määritetään ruudukon ylimmän tason elementti. Se keskittää sivun sisällön ja lisää sivun reunoille marginaalit. Container pitää sisällään yhden tai useamman rivin eli row-luokan elementin ja sen sisältämät column elementit.

Ruudukossa on lisäksi viisi tasoa (Kuva 27), joilla voidaan määrittää sivun elementtien sijoittuminen eri kokoisilla näytöillä. Ne ovat kaikki column-luokan eri variaatiota.

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-

Kuva 27. Bootstrap column-luokat (W3Schools 2019)

Column-luokalle voidaan määrittää myös leveys välillä 1 – 12, joka perustuu ruudukon sarakkeisiin. Esimerkiksi elementti, jolla on luokka "col-6" varaa itselleen puolet sivun leveydestä ja "col-4" kolmasosan. Eri column-luokan variaatioita voidaan myös yhdistellä kuten esimerkissä (Kuva 28) on tehty. Yhdistämällä vakio col-luokka sen näytökoko riippuvaisen variaation kanssa, saadaan määritettyä kuinka paljon elementit varaavat tilaa, jos variaatioluokkien määrittelyt eivät ole voimassa. Esimerkiksi "col-6 col-md-4" määrittely varaa vakiona elementille puolet näytön leveydestä, kunnes näytön koko on suurempi tai yhtä suuri kuin "col-md" luokan määräämä 720 pikseliä. Rajan ylittyessä elementti varaa vain kolmasosan näytön leveydestä.

.col-12 .col-md-8		.col-6 .col-md-4
.col-6 .col-md-4	.col-6 .col-md-4	.col-6 .col-md-4
.col-6		.col-6

```

<!-- Stack the columns on mobile by making one full-width and the other half-width -->
<div class="row">
  <div class="col-12 col-md-8">.col-12 .col-md-8</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-6">.col-6</div>
  <div class="col-6">.col-6</div>
</div>

```

Kuva 28. Column-luokkamäärittelyt (Bootstrap 2020)

3.4.2 Components

Bootstrap sisältää paljon erilaisia uudelleenkäytettäviä ja vapaasti muokattavia komponentteja, joita verkkosivuilla usein esiintyy. Näitä ovat esimerkiksi erilaiset painikkeet (Kuva 29), navigaatiopalkit, pudotusvalikot, kuvakarusellit, lomakkeet, latauspalkit ja animaatiot, popupit, tooltipit jne. Käytännössä komponentteja käytetään lisäämällä elementeille niitä vastaavat CSS luokat. Usein komponentit käyttävät CSS-tyylimäärittelyiden lisäksi niille kuuluvia osia Bootstrapin jQuery koodista. (Bootstrap 2020.)



Kuva 29. Bootstrapin esimääritetyt painiketyyli (Bootstrap 2020)

4 CASE: MARTTEN FINLAND OY

Martten Finland Oy on suomalainen kansainvälistyvä yritys, joka myy erilaisia tarratulostimia ja niiden kanssa yhdessä toimivia ohjelmistoja. Se tarjoaa yksilöllisesti räätälöityjä tulostinohjelmistoratkaisuja eri alojen tarpeisiin. Yrityksen vahvinta osaamista ovat erilaiset tulostettavat tarrat ja niissä esitettävä informaatio. (Martten Finland Oy 2019.)

4.1 Määrittely

Projektin tarkoitus on luoda ERP-järjestelmän kaltainen web-sovellus Martten Finland Oy:n laitoshuoltoalan asiakasyrityksen tarpeisiin. Sovellusta käytetään mobiililaitteilla ja tietokoneella, joten käyttöliittymä on täysin responsiivinen. Sovelluksen toiminnot mukautuvat käyttäjän roolin mukaan. Esimies-roolin käyttäjälle keskeisimpinä osina ovat työntekijöiden ohjeistuksen syöttäminen ja työn edistymisen seuranta. Työntekijän on pystyttävä lukemaan työtehtäviin liittyvää ohjeistusta ja asiakaspalautetta, sekä kirjaamaan tehtyjä työtehtäviä valmiiksi. Kirjautumaton käyttäjä voi arvioida työn laatua ja antaa vapaamuotoista asiakaspalautetta.

4.2 Asiakkaan vaatimukset

Sovelluksessa on näytettävä erilaista sisältöä eri roolissa oleville käyttäjälle. Lisäksi sisältö määräytyy kirjautuneelle käyttäjälle määriteltyjen roolien mukaan. Tarvittavia rooleja määrittelyvaiheessa olivat: järjestelmävalvoja, esimies ja työntekijä. Myöhemmin mukaan tuli myös rooliin kuulumattomalle käyttäjälle näytettävä sisältö.

Sovelluksessa on asiakasorganisaatioita, joilla on useita kohteita ja kiinteistöjä. Näissä laitoshuollon työntekijät suorittavat määrättyjä työtehtäviä. Kohteissa on useita sijainteja, jotka on kategorioitu. Myös alakategoriat ovat mahdollisia.

Esimerkki kohteen kategorioista:

- WC
- Toimisto
- Aula
- Neuvottelu

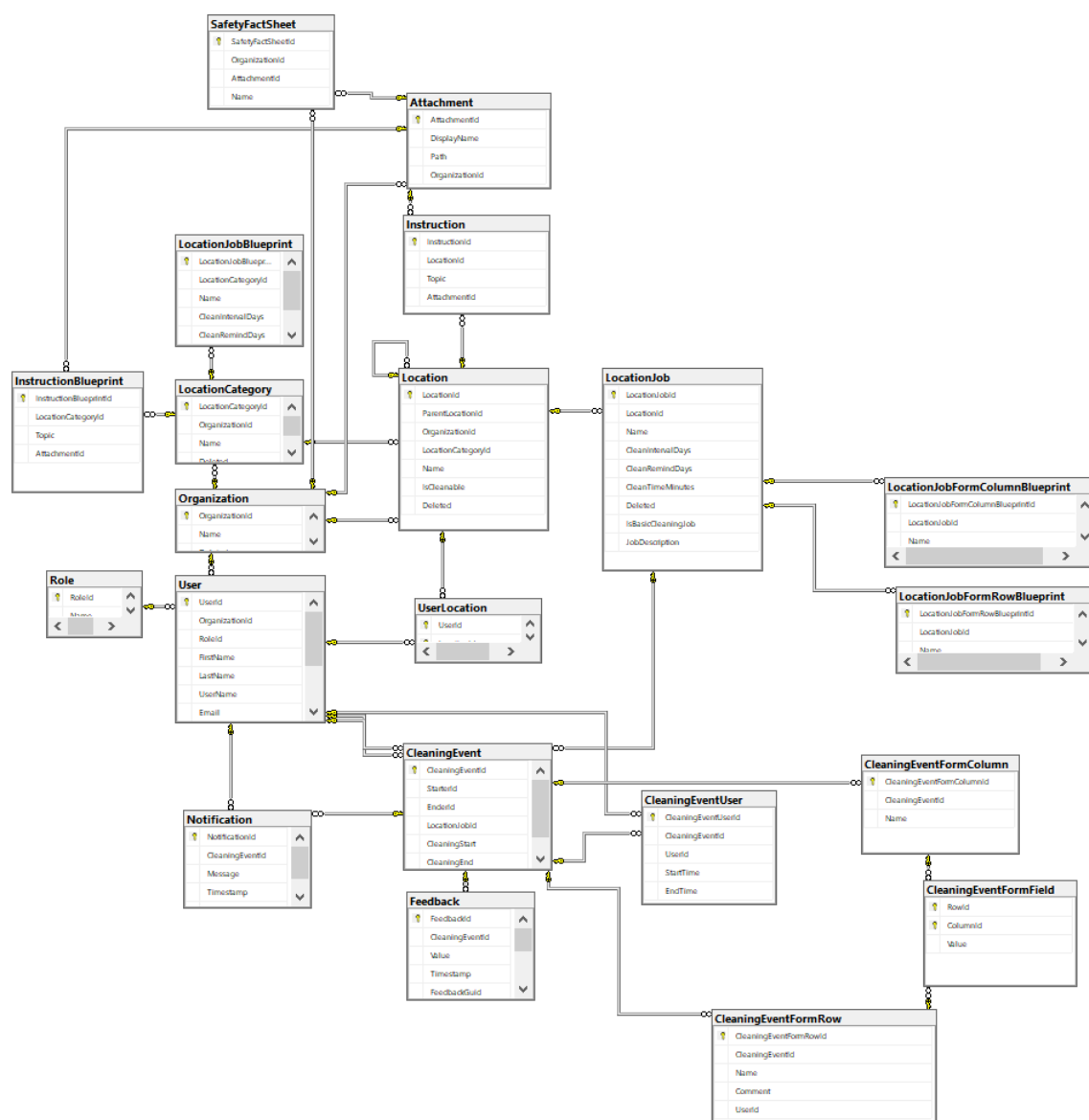
(Liite: Siivouspalvelu.pptx)

Kun käyttäjä lukee sijainnissa olevan QR-kooditarran, sovellus avaa käyttäjäroolin huomioiden sijainnin päänäköymän. Esimies roolin käyttäjä voi määrittää sijainnille työtehtäviä

sekä kirjoittaa työohjeen. Työtehtäville määritetään suoritusaikaväli sekä suunniteltu suoritamiseen kuluva aika. Työntekijä roolin käyttäjä näkee esimiehen kirjoittamat työohjeet. Hän voi kirjata työsuorituksia ja halutessaan niihin liittyviä huomioita. Työhön kulunut aika tallennetaan automaattisesti. Kirjautumaton käyttäjä näkee työsuoritukset ja voi antaa niistä palautetta. Käyttäjä voi kirjoittaa työtehtäviin tekstimuotoisen palautteen ja jättää 1-5 arvostuksen.

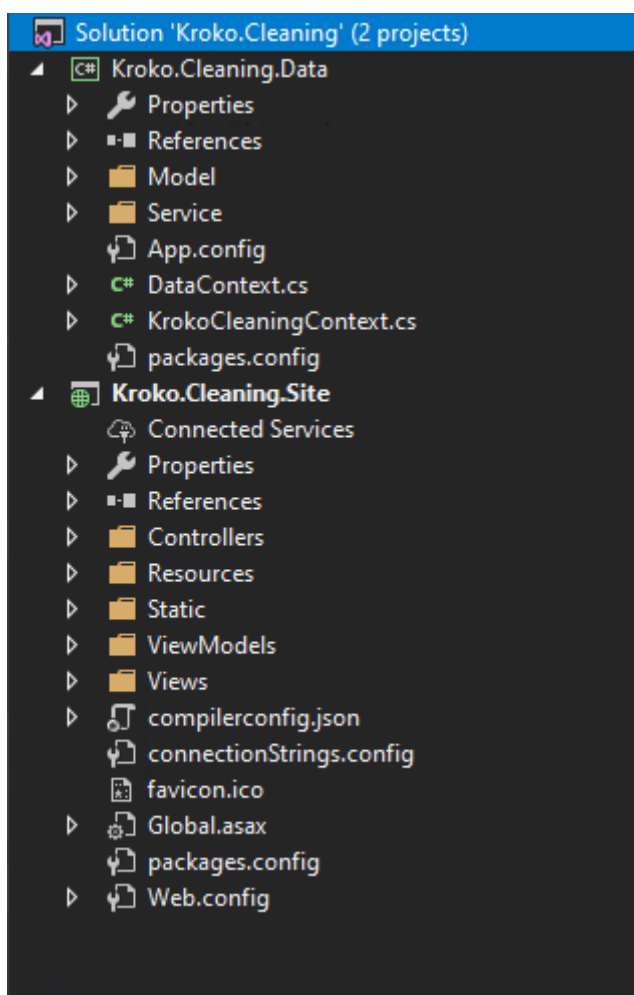
4.3 Toteutus

Sovelluksen varsinainen toteutus aloitettiin luomalla sille ensin uusi tietokanta ja tarvittavat tietokantataulut relaatioineen SQL Server Management Studiolla (Kuva 30). Aikaisemmin mainittuja sijaintikategorioita varten kantaan luotiin LocationCategory-taulu. Siihen liitettiin one-to-many suhteella taulu Location, joka pitää sisällään varsinaisia siivouskohteita. Sijaintikategoria sisältää luonnollisesti useita sijainteja. Location taululle luotiin linkitys itseensä, jotta vaatimuksissa määritettyjä alasijainteja pystytään luomaan ja tällöin niistä voidaan myös muodostaa puurakenteita tarpeen mukaan. Sijainneille määrättyjen siivoustehtävien tietoja pidetään LocationJob taulussa, jolla on jälleen one-to-many suhde Location tauluun, koska niitäkin voi olla useita yhdessä sijainnissa. Suoritetut tehtävät kirjataan CleaningEvent tauluun. Niiden tietoja voidaan verrata sovelluksessa LocationJob taulun tietoihin, kun halutaan tarkastella, onko työn suorittaminen mennyt suunnitellusti. Nämä taulut muodostivat pääasiassa tietokantarakenteen lähtökohdat, joka oli aikaa vievin osuus koko kannan suunnittelussa. Kun niiden sisältämä tieto oli määritelty ja ne saatiin liitettyä järkevästi toisiinsa, loput taulut saatiin sen jälkeen helposti lisättyä niille loogisiin paikkoihin.



Kuva 30. Sovelluksen tietokanta

Tietokannan jälkeen luotiin Visual Studiassa uusi tyhjä työtila (solution) ja sen alle erilliset projektit sovelluksen taustajärjestelmälle ja varsinaiselle sivustolle. Data projekti luotiin käyttäen .NET Framework Class Library projektipohjaa ja sivusto käyttäen tyhjää ASP.NET Web Application pohjaa. Data-projektiin lisättiin omat kansiot data malleille (models) ja niiden sisältämää dataa tarjoaville palveluille (services). Sivusto-projektiin luotiin MVC arkkitehtuuria mukaillen ViewModels, Views ja Controllers kansiot, sekä Resources ja Static lokalisaatioresursseille, tyylikirjastoille, scripteille ja tyyli tiedostoille (Kuva 31).



Kuva 31. Sovelluksen projektirakenne

4.3.1 Entity Frameworkin asennus

Entity Framework on saatavilla Nuget-pakettina, joten se asennettiin käyttäen Visual Studiosta löytyvää Nuget-pakettien hallintatyökalua NuGet Package Manageria. Tällä tavalla Entity Framework saatiin nopeasti mukaan molempiin projekteihin. Periaatteessa pelkästään EF Runtimein asentaminen olisi tässä tapauksessa riittänyt, koska paketissa mukana tullutta EF Toolsia ei tarvittu. Kuitenkin asennuksen yksinkertaisuuden takia lisättiin Entity Framework Nuget-paketti kokonaan.

4.3.2 Tietorakenteiden luonti (Data Models)

Data projektiin luotiin tietokannan tauluja vastaavat data model luokat, joita Entity Framework käyttää käsitelleessään tietokannassa olevaa dataa. Ne voisi luoda myös automaattisesti Visual Studioon saatavilla olevilla Entity Framework työkaluilla, mutta tässä

projektissa ne haluttiin luoda manuaalisesti. Osin koulutusmielessä, mutta myös koska haluttiin käyttää .NET:in "DataAnnotation"-attribuutteja validointiehtojen ja viiteavaimien määrittämiseen.

4.3.3 Tietokantakonteksti (Database context)

DataContext on Entity Frameworkin luokka, joka pitää sisällään joukon DbSet -kokoelmia (collection). Näihin kokoelmiin on liitetty tietokannan tauluja vastaavat data mallit, joista framework muodostaa datan käsittelyä helpottavia entiteettejä. DataContext luokan ilmentymän (instance) kautta käsitellään tietokannassa olevaa dataa ja suoritetaan muu-
tostransaktioita.

DataContextin sisällä voitaisiin määritellä myös entiteettien relaatiot, mutta näin ei tehty koska ne määriteltiin suoraan data model luokkiin data annotaatio attribuuttien avulla. Tässä projektissa DbContext oli hyvin yksinkertainen. Ainoana erikoisuutena lisättiin OnModelCreating yliajo metodi, jossa DbBuilder luokan sopivuussäännöistä (conventions) poistettiin "PluralizingTableNameConvention". Se saa Entity Frameworkin olettaamaan, että tietokantataulujen nimet ovat data mallin nimen monikoita, mikä ei tässä tapauksessa pitänyt paikkaansa.

4.3.4 Käyttöliittymien toteutus

Bootstrap-kirjaston voi lisätä sovellukseen kolmella eri tavalla:

- 1) Ladataan kirjaston valmiiksi käännetyt css ja js tiedostot Bootstrapin omalta sivustolta, tallennetaan ne projektihakemistoon ja lisätään sovelluksen layout-näkymään html link ja script tagit, jotka osoittavat css ja javascript tiedostojen sijainteihin.
- 2) Hyödynnetään BootstrapCDN:ää, eli sisällönjakeluverkkoa ja laitetaan html link ja script tagit osoittamaan sen verkko-osoitteisiin.
- 3) Asennetaan kirjasto käyttäen jotakin paketinhallintajärjestelmää. .NET ympäristössä esimerkiksi Visual Studion NuGet Package Managerilla. Tämä tapa vaatii lisäksi myös sass kääntäjän ja autoprefixerin.

Kuten yrityksen muissakin projekteissa, myös tässä käytettiin ensimmäistä tapaa. Näin sovelluksessa on aina valmiiksi käännetty kirjasto paikallisesti saatavilla. Sovellus ei ole siis ainakaan tältä osin riippuvainen ulkoisista verkkopalveluista eikä SASS kääntäjää ja autoprefixeriä tarvita.

Sovelluksen yksi tärkeimmistä näkymistä on kohteen perusnäkö, johon oli saatava näkyviin kaikki tarvittava tieto työtehtävien suorittamista ja työohjeiden kirjaamista varten. Lisäksi näkymässä on pystyttävä tarkastelemaan aikaisempia työsuorituksia ja niille kirjoitettuja huomioita. Kun työntekijä tulee kohteeseen suorittamaan sille asetettuja työtehtäviä, hän lukee kohteessa sijaitseva QR-kooditarran mobiililaitteellaan ja hänelle avautuu kyseisen kohteen perusnäkö mobiililaitteelle mukautetussa muodossa (Kuva 32). Samaan näkymään pääsee myös hallintakäyttöliittymässä olevan kohdelistauksen kautta, jota esimiehet pääsääntöisesti käyttävät, kun he tulevat syöttämään työohjeita kohteelle. Silloin näkö avautuu tietokoneen ruudulla näytettävässä muodossa (Kuva 33). Kohteen perusnäkö ja monien muiden sovelluksen näköjen toteutuksen osalta oleelliset osat Bootstrap-kirjasto olivat "col"-luokat, joilla esimerkiksi työtehtäviä kuvaavat elementit saatiin järjestettyä mobiilinäkömissä helposti allekkain.

Kohteet

Lahti > Avenla > 8. Kerros - Saunatilat -

Työkirjaukset

● Lauteiden puhdistus
(20 min)

Jatka työtä

● Perussiivous
(40 min)

Aloita työ

Viimeisimmät työt

Lauteiden puhdistus
22.05.2019 12:54



Tekijä
Siivousväli
Suunniteltu aika

Mika Rissanen
7
00:20:00

Huomiot

Lisää huomio

Lauteiden puhdistus
22.05.2019 12:50



Perussiivous
22.05.2019 12:48



Perussiivous
22.05.2019 12:48



Kuva 32. Kohteen perusnäkö mobiililaitteella

Kohteet

Lahti > Avenla > 8. Kerros - Saunatilat -

Työkirjaukset

● Lauteiden
puhdistus
(20 min)

Jatka työtä

● Perussiivous
(40 min)

Aloita työ

Viimeisimmät työt

Lauteiden puhdistus
22.05.2019 12:54



Tekijä Mika Rissanen
Siivousväli 7
Suunniteltu aika 00:20:00

Huomiot

Lisää huomio

Lauteiden puhdistus
22.05.2019 12:50



Perussiivous
22.05.2019 12:48



Perussiivous
22.05.2019 12:48



Lauteiden puhdistus
22.05.2019 12:44



Perussiivous
20.05.2019 15:32



Kohteen työohjeet

Lisää työohje

Työohje1

Avenlan Logo



Kuva 33. Kohteen perusnäkö tietokoneen ruudulla

5 YHTEENVETO

Tämän opinnäytetyön tarkoitus oli tuottaa Martten Finland Oy:lle laitoshuoltoalan tarpeisiin räätälöity toiminnanohjausjärjestelmä, joka hyödyntää Martten Finlandin tuottamia QR-kooditarroja. Työ toteutettiin opintoihin kuuluvan työharjoittelujakson aikana Avenla Oy yrityksessä. Tavoitteena oli tuottaa sovellus, jonka toiminnot ja näkymät mukautuvat käyttäjäröolien mukaan. Käyttöliittymissä tärkeässä osassa olivat helppokäyttöisyys ja responsiivisuus, koska sovellusta tullaan käyttämään pääasiassa mobiililaitteilla laitoshuoltoalan työtehtävien suorittamisen yhteydessä. Työn lopputuloksena tuotettu sovellus täytti asetetut vaatimukset hyvin eikä projektin toteuttamisen aikana kohdattu mitään suurempia haasteita tai ongelmia. Alkuun hieman haastetta tuotti tietokantarakenteen suunnittelu. Kuitenkin harkinnan jälkeen priorisoimalla sen oleelliset osat suunnittelun lähtökohdiksi, ongelmat saatiin ratkaistua.

Sovelluksen käyttökokemukset ovat olleet positiivisia ja käyttäjät ovat olleet tyytyväisiä käytön helppouteen ja selkeyteen. Käyttöönotto on tapahtunut vaivattomasti ja perehdytys sovelluksen käyttöön ei ole vienyt paljoa aikaa. Kaiken kaikkiaan käyttöliittymien suunnittelussa on onnistuttu erinomaisesti.

Ilman työssä hyödynnettyjä .NET ja Bootstrap teknologioita projektia ei olisi todennäköisesti voitu toteuttaa suunnitellun aikataulun puitteissa. Etenkin Entity Framework helpotti huomattavasti alkuun monimutkaisilta vaikuttaneiden datarakenteiden hahmottamista ja käsittelyä. Työtä tehdessä oli huomioitava paljon asioita relaatiotietokantarakenteen suunnittelussa, jotta sovelluksen ohjelmointi olisi mahdollisimman vaivatonta noudattaen olio-ohjelmoinnin periaatteita.

Sovellus on otettu käyttöön asiakasyrityksessä ja sen tuottanut Avenla Oy jatkaa sovelluksen ylläpitoa ja hostausta.

LÄHTEET

Elektroniset lähteet:

Mozilla Developer Network 2020. CSS Flexible Box Layout. [viitattu 6.5.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout

Bootstrap 2020. About [viitattu 6.5.2020]. Saatavissa: <https://getbootstrap.com/docs/4.0/layout/grid/>

LePage, Peter. Google Developers 2020. Responsive Web Design Basics [viitattu 17.4.2020]. Saatavissa: <https://developers.google.com/web/fundamentals/design-and-ux/responsive>

Mozilla Developer Network 2020. CSS values and units [viitattu 17.4.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units

W3Schools. 2020. Responsive Web Design - The Viewport [viitattu 17.4.2020]. Saatavissa: https://www.w3schools.com/css/css_rwd_viewport.asp

Mozilla Developer Network 2019. Using the viewport meta tag to control layout on mobile browsers [viitattu 17.4.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

Microsoft 2018. Entity Framework overview [viitattu 9.3.2020]. Saatavissa: <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview>

Microsoft 2019. .NET Framework Guide [viitattu 18.1.2020]. Saatavissa: <https://docs.microsoft.com/fi-fi/dotnet/framework/>

Commons Wikimedia. .NET Framework component stack [viitattu 3.11.2020]. Saatavissa: <https://commons.wikimedia.org/wiki/File:DotNet.svg>

Stackoverflow 2014. How do you know what relationship to use? [viitattu 3.11.2020]. Saatavissa: <https://stackoverflow.com/questions/25685670/how-do-you-know-what-relationship-to-use>

Microsoft 2017. Language Integrated Query [viitattu 27.9.2020]. Saatavissa: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>

Microsoft 2015. Basic LINQ Query Operations [viitattu 27.9.2020]. Saatavissa: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations>

Microsoft 2015. Group clause (C# Reference) [viitattu 30.9.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/group-clause>

Microsoft 2016. Get Entity Framework [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/fundamentals/install>

Microsoft 2018. Creating a Model [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/modeling/>

Microsoft 2016. Code First to a New Database [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>

Microsoft 2016. Code First to an Existing Database [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/existing-database>

Microsoft 2016. Model First [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/model-first>

Microsoft 2016. Database First [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first>

Microsoft 2020. NuGet documentation [viitattu 11.10.2020]. Saatavissa:
<https://docs.microsoft.com/en-us/nuget/>

Microsoft 2019. Install and manage packages in Visual Studio using the NuGet Package Manager [viitattu 11.10.2020]. Saatavissa: <https://docs.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio>

Microsoft 2020. .NET Programming Languages [26.1.2020] Saatavissa:
<https://dotnet.microsoft.com/languages>

Microsoft 2019. ASP.NET Overview [26.1.2020] Saatavissa:
<https://docs.microsoft.com/en-us/aspnet/overview>

Microsoft 2016. Entity Framework 6 [viitattu 10.2.2019]. Saatavissa:
<https://docs.microsoft.com/en-us/ef/ef6/>

Körner, Christoph. 2016. Learning Responsive Data Visualization [viitattu 2.3.2019].
 Saatavissa:
<http://search.ebscohost.com.aineistot.lamk.fi/login.aspx?direct=true&db=nlebk&AN=1213513&site=ehost-live>

Can I use. 2019. CSS3 Media Queries [viitattu 2.3.2019]. Saatavissa:

<https://caniuse.com/#feat=css-mediaqueries>

Can I use. 2019. Viewport units: vw, vh, vmin, vmax [viitattu 2.3.2019]. Saatavissa:

<https://caniuse.com/#feat=viewport-units>

TutorialsPoint. 2019. CSS – Media Types [viitattu 2.3.2019]. Saatavissa:

https://www.tutorialspoint.com/css/css_media_types.htm

W3Schools. 2019. CSS Media Queries [viitattu 2.3.2019]. Saatavissa:

https://www.w3schools.com/css/css3_mediaqueries.asp

W3C. 2018. Media Queries Level 4 [viitattu 2.3.2019]. Saatavissa:

<https://drafts.csswg.org/mediaqueries/#media-types>

Bootstrap 2019. About [viitattu 27.2.2019]. Saatavissa:

<https://getbootstrap.com/docs/4.2/about/overview/>

Martten Finland Oy 2019. Keitä olemme? [viitattu 10.2.2019]. Saatavissa:

<http://www.martten.fi/about-us/>