Jyri Paussu

MOBIILI-INVENTOINTISOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Tieto- ja viestintätekniikan koulutusohjelma
2020

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

MOBIILI-INVENTOINTISOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Paussu, Jyri
Satakunnan ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutusohjelma
Helmikuu 2020
Ohjaaja: Nuutinen, Petri
Sivumäärä: 30
Liitteitä: 0

Asiasanat: Ohjelmistokehitys, Mobiiliohjelmointi, REST

Tämän opinnäytetyön aiheena oli mobiili-inventointisovelluksen suunnittelu ja toteutus. Tehtävä sovellus luotiin työkaluksi SONET Premium sovellusta varten CGI Suomi yritykselle. Työhön kuului sekä mobiilisovelluksen luonti, että rajapintojen luonti itse SONET Premium sovelluksen sisälle.

Mobiilisovelluksen tavoitteena oli helpottaa isojen varastojen inventointia. Ilman mobiilisovellusta inventointiluvut oli pakko kirjata ensin jollekin välialustalle, josta luvut päivitettiin tietokoneella pyörivälle SONET Premium sovellukselle.

Sovelluskokonaisuuden tekemisessä käytettiin useita eri tekniikoita ja teknologioita, jotka käyn työssäni läpi. Projektia hallittiin ketterien ohjelmistokehysten, kuten Scrum:n ja SAFe:n avulla.

Työn teoriaosuudessa käydään läpi, mitä kaikkia työkaluja ja tekniikoita työhön käytettiin, jonka jälkeen perehdytään ohjelmistokehityksen eri vaiheisiin. Käyn myös läpi, kuinka projektia hallittiin.

Käytännön osiossa esittelen, kuinka sovellus toteutettiin, mitä haasteita kohtasin ja miltä lopputulos sitten näytti.

DESIGNING AND IMPLEMENTING A MOBILE INVENTORY APPLICATION

Paussu, Jyri
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information and Communications Technology
February 2020
Supervisor: Nuutinen, Petri
Number of pages: 30
Appendices: 0

Keywords: Software development, Mobile programming, REST

The subject of this thesis was to design and implement a mobile inventory application. The application was created as a tool for the SONET Premium application for CGI Finland. The work included both creating a mobile application and creating interfaces within the SONET Premium application itself.

The purpose of the mobile application was to facilitate the inventory of large warehouses. Without the mobile application, inventory numbers had to be recorded somewhere else and then manually updated to a computer running the SONET Premium application.

Many different techniques and technologies were used to create the application, which I will go through in this work. The project was managed with agile software frameworks such as Scrum and SAFe.

The theoretical part of the thesis discusses all the tools and techniques used in the work, and then familiarizes with the different stages of software development. I will also go through how the project was managed.

In the practical section I will show how the application was implemented, what challenges I faced and what the result looked like.

SISÄLLYS

LYHENTEET JA TERMISTÖ

.NET	Microsoftin kehittämä ohjelmistokehys
API	Ohjelmointirajapinta (Application Programming Interface)
CLR	Common Language Runtime (.NET:n hyödyntämä käyttöympäristö, jossa .NET-ohjelmia suoritetaan)
IDE	Ohjelmointiympäristö (Integrated Development Environment)
Natiivi	Tietyn alustan tai ohjelman oman muodon mukainen
PWA	Progressive Web Application (Progressiivinen web-sovellus)
REST	HTTP-protokallaan perustava arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen (Representational State Transfer)
SAFe	Skaalautuva ketterän kehittämisen viitekehys (Scaled Agile Framework)
Scrum	Ketterä projektinhallinnan viitekehys
TFS	Team Foundation Server (Ohjelmistokehityksen projektinhallintaympäristö)

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli luoda Windows-pohjaisille viivakoodinlukulaitteille varastoinventointisovellus, joka toimisi suoraan yhteydessä Sonet Premium sovelluksen kanssa. Varastoinventointisovelluksen tarkoituksena oli helpottaa suurten varastojen inventointia mobiililaitteen avulla. Opinnäytetyön perustana toimivat osallistuminen dokumentaatioon, suunnitteluun ja toteutukseen niiltä osin kuin itse olin kyseisissä toiminnoissa mukana. Sonet Premium on ohjelmistokokonaisuus, jolla voi hoitaa toiminnanohjausta, henkilöstönhallintaa sekä taloudenohjausta. Opinnäytetyöni sisältö keskittyy lähinnä ohjelman toiminnanohjauspuoleen, tarkemmin ottaen sen varastointitoiminnallisuuteen.

Tässä opinnäytetyössä tutustaan ohjelmistokehitykseen, perinteiseen sovelluskehitykseen ja mobiilikehitykseen. Kehityksen lisäksi tutustaan myös hieman kuinka ohjelmistoprojekteja hallinnoidaan ketterästi Scrum-mallin avulla, sekä miten Scrum-malli saadaan skaalautumaan isoihin ohjelmistoprojekteihin SAFe-mallin avustuksella. Sen jälkeen käydään läpi projektin osat, niissä käytetyt teknologiat ja kuinka ne toteutettiin.

Projekti sisältyy kahdesta osasta. Itse inventointisovelluksesta, joka tuotetaan C#:lla ja Prism-ohjelmistokehyksellä mobiililaitteelle, sekä Sonet Premiumiin X-kielellä tuotetusta REST-rajapinnasta. Rajapinnan kautta inventointisovellus saa yhteyden pääsovellukseen.

2 SOVELLUSKEHITYS

Moderni maailmamme ei pyörisi ilman ohjelmistoja, lähes kaikissa nykyajan yhteiskunnan toiminnoissa on jokin tietokoneohjelma auttamassa tai suorittamassa asioita. Teollisuus pyörii tietokoneiden ohjaamana ja suurimmassa osassa sähköisiä laitteita on jokin piiri, sekä sitä hallitseva ohjelmisto sisällä. Myös ihmisten kuluttama viihde on hyvin pitkälti nykypäivänä tietokoneella luotua. Ohjelmistokehitys on siis osaltaan luomassa perustaa nykyisen yhteiskunnan ylläpitämiseen.

Ohjelmistot ovat abstrakteja ja aineettomia, niitä eivät rajoita mitkään fysiikan lait, materiaalien ominaisuudet tai valmistusprosessit. Tämä rajattomuus yksinkertaistaa ohjelmistokehitystä, sillä ohjelmistojen potentiaalille ei ole mitään luonnollisia rajoituksia. On olemassa monia erilaisia ohjelmistojärjestelmiä, yksinkertaisista sulautetuista järjestelmistä monimutkaisiin maailmanlaajuisiin tietojärjestelmiin. Ei ole siis mielekästä määritellä universaaleja menetelmiä tai tekniikoita, jotka toimisivat kaikissa ohjelmistosuunnittelun tapauksissa, koska eri ohjelmistotyypit vaativat erilaisia lähestymistapoja. Siksi onkin tärkeää käsitellä jokaista ohjelmistoa tapauskohtaisesti. (Ian Sommerville 2011, 4.)

Organisaatiotietojärjestelmän kehittäminen on täysin erilaista kuin vaikka tieteellisen instrumentin ohjaimen kehittäminen. Kummallakaan näistä ei ole paljoa yhteistä grafiikkaintensiivisen tietokonepelin kanssa. Vaikka kaikki nämä sovellukset tarvitsisivat ohjelmistotekniikkaa, ne eivät kaikki kuitenkaan tarvitse samaa ohjelmistotekniikkaa. (Ian Sommerville 2011, 4.)

2.1 Sovellusten ketterä kehitys

Sovellusten ketterä kehittäminen (agile software development) on kehitysmalli, joka auttaa ohjelmistojen kehittämistä nopeampaan toisin sanoen ketterämpään tahtiin. Ketterä kehitys syntyi 90-luvulla, kun ohjelmistokehittäjät kyllästyivät vanhoihin hitaisiin ja raskaisiin malleihin, joissa kului enemmän aikaa ohjelmistojen suunnitteluun kuin niiden valmistamiseen ja testaamiseen. Vanhassa vesiputousmallissa, projektin kulku nähtiin etenevän perinteisesti yhtenä rintamana, lineaarisena kokonaisuutena. Vaikka

vesiputousmalli sisäistettiin eri tavalla kuin sen alkuperäinen kehittäjä sen suunnitteli, tämä työ keskittyy nykyaikaisiin ketteriin menetelmiin. (Ian Sommerville 2011, 57-60.)

Ketterissä menetelmissä ohjelmistojen määrittely, kehitys ja toimitus tapahtuvat asteittain lyhyissä iteraatioissa. Ketterät menetelmät soveltuvat parhaiten sovelluskehitykseen, jossa järjestelmän vaatimukset muuttuvat yleensä nopeasti kehitystyön aikana. Niiden tarkoituksena on toimittaa nopeasti ohjelmisto asiakkaille, jotka voivat ehdottaa uusia ja muuttuneita vaatimuksia. Uudet vaatimukset sisällytetään kehityksen myöhempiin iteraatioihin. Niillä pyritään vähentämään prosessien byrokratiaa ja työtä, jonka pitkän aikavälin arvo on kyseenalainen ja poistetaan dokumentaatioita, joita ei todennäköisesti käytetä koskaan. (Ian Sommerville 2011, 57-60.)

2.2 Mobiilikehitys

Mobiililaitteiden ohjelmistojen kehitys on yhtä kirjavaa, kuin mitä on eri mobiililaitteita ja mobiilialustojakin. Eri alustoille on tarjolla eri ratkaisuja, Microsoft tarjoaa Windows-laitteille omansa, Google Androidille ja Apple iOS-laitteille. Koska mobiililaitteita on montaa erilaista, sekä niitä käytetään täysin eri tavalla kuin tietokoneilla pyöriviä sovelluksia, mobiilisovellusten kehityksessä pitää ottaa monta asiaa huomioon:

- **Käytettävyys:** Sovelluksen pitää olla käytettävyydeltään tarpeeksi selkeä ja yksinkertainen.
- **Suorituskyky:** Mobiililaitteilla on vähemmän resursseja käytettävänä, joten sovellusten pitää olla tarpeeksi kevyitä.
- **Ulkoasu:** Koska mobiililaitteet ovat varustettu monella eri näytön resoluutiolla, niin ulkoasun pitää olla responsiivinen, eli sen täytyy mukautua kaikille resoluutioille.

Mobiilisovellusten kehittämiseen on olemassa neljä pääasiallista kehitystapaa:

- **Natiivit mobiilisovellukset**
- **Alustariippumattomat mobiilisovellukset**
- **Hybridimobiilisovellukset**

- **Progressiiviset web-sovellukset**

(Amazon Web Services 2019)

Jokaisella näistä lähestymistavoista on omat edut ja haitat. Valittaessa oikeaa kehitysmenetelmää projektille kehittäjät ottavat huomioon halutun käyttökokemuksen, sovelluksen edellyttämät laskentaresurssit ja natiiviominaisuudet, kehittämisbudjetin, aikatavoitteet ja sovelluksen ylläpitämiseen käytettävissä oleva resurssit.
(Amazon Web Services 2019.)

2.2.1 Natiivit mobiilisovellukset

Natiivit mobiilisovellukset on kirjoitettu mobiilialustojen omistajien tarjoamalla ohjelmointikielillä- ja kehyksillä, joten ne toimivat suoraan käyttöjärjestelmätasolla. Natiivien mobiilisovellusten etuna on laiteläheisyys, sen tuoma nopeus ja suorituskyky. Haittapuolina ovat alemman tason ohjelmointikielien tuoma sovelluskehityksen hitaus ja vaikeus, sekä alustariippuvuus. Tämän takia sovellus jouduttaisiin kehittämään joka alustalle erikseen. (Amazon Web Services 2019.)

2.2.2 Alustariippumattomat mobiilisovellukset

Alustariippumattomat mobiilisovellukset voidaan kirjoittaa useilla erilaisilla ohjelmointikielillä ja –kehyksillä. Ne kootaan natiiviksi sovellukseksi, joka toimii käyttöjärjestelmätasolla. Alustariippumaton on suosituin tapa kehittää mobiilisovelluksia, sillä niiden kehitykseen käytetyt ylemmän tason ohjelmointikielet tekevät ohjelmien kehityksestä ja ylläpidosta helppoa. Sama koodi toimii joka alustalla.

Alustariippumattomuus tuo myös haittapuolensa. Jotta ohjelmakoodi voi toimia joka alustalla, se suoritetaan ensin ohjelmointitulkin läpi ennen kuin se voi toimia käyttöjärjestelmätasolla. Näin ollen menetetään suorituskykyä ja ohjelman koko kasvaa.
(Amazon Web Services 2019.)

2.2.3 Hybridi-mobiilisovellukset

Hybridi-mobiilisovelluksia kehitetään käyttäen tavallisia web-ohjelmoinnin teknologioita kuten JavaScript, CSS ja HTML. Selaimen sijaan ne pyörivät itse mobiililaitteessa niin sanotussa verkkosäiliössä, joka tarjoaa oman selainkäyttöliittymän ja sillan laitteen omille rajapinnoille. Web-ohjelmoinnin teknologioiden käyttö tekee ohjelman ja sen käyttöliittymän kehittämisestä varsinkin web-kehittäjille helppoa, sillä nämä ovat heille tuttuja ympäristöjä. Tämän lisäksi samaa ohjelmakantaa voidaan käyttää myös web-sovelluksen luomiseen. Kuitenkin verrattuna natiivisovelluksiin, hybridisovellusten suorituskyky on huono ja tuki laitteen ominaisuuksiin on puutteellinen. (Amazon Web Services 2019.)

2.2.4 Progressiiviset web-sovellukset

Progressiiviset web-sovellukset, Progressive Web Applications, eli PWA:t tarjoavat vaihtoehtoisen lähestymistavan perinteiseen mobiilisovellusten kehittämiseen ohittamalla sovelluskaupan toimitukset ja sovellusten asennukset. PWA:t ovat web-sovelluksia, jotka pyörivät jokaisella alustalla, joka on varustettu standardin mukaisella selaimella. Näin ollen ne ovat myös alustariippumattomia. PWA:t pyörivät palvelimilla, joten niillä on muihin mobiilisovelluksiin verrattuna suurin suoritusteho. Näillä sovelluksilla on kuitenkin rajoitettu tuki laitteen ominaisuuksiin, joten ne eivät sovellu kaikkeen. (Amazon Web Services 2019)

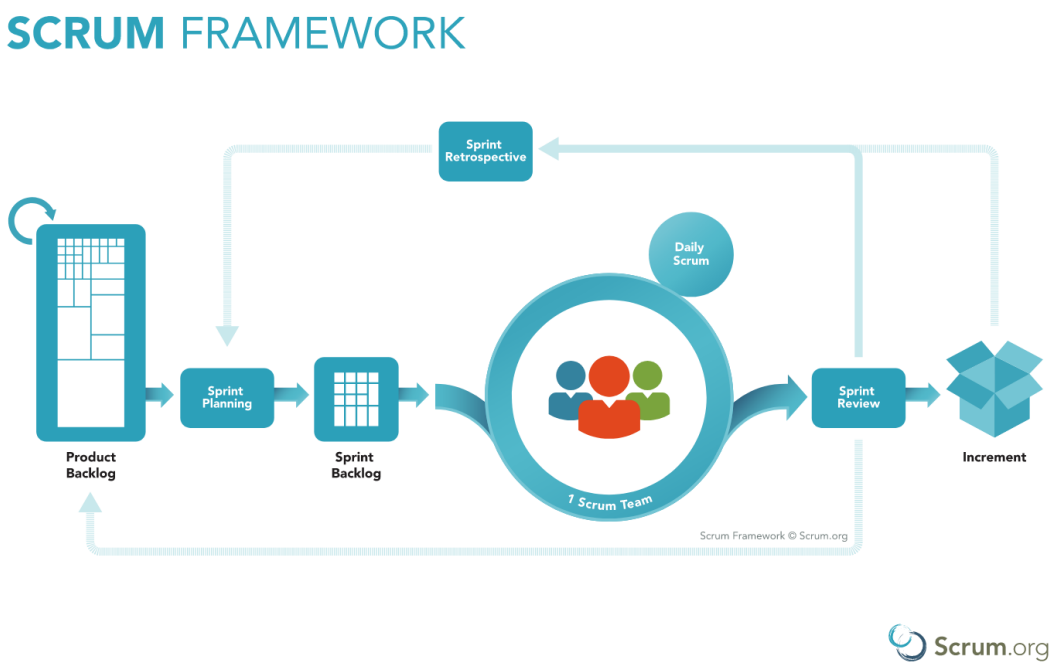
3 PROJEKTI

Mobiilisovelluksen määrittely alkoi noin kuukauden kuluttua työsuhteeni alettua toukokuussa 2018. Mobiili-inventointisovellusta oli toivottu asiakkaan puolelta jo pidempään ja työ toteutettiin asiakastilauksena. Asiakkaan toiminnassa varastokirjanpito oli oleellista. Tämän kaltaisessa ympäristössä mobiililaitteen käyttö oli perusteltua sen keveyden ja liikuteltavuuden vuoksi varastotiloissa. Toiveet ohjelman toiminnasta ja ulkoasusta sovittiin asiakkaan kanssa.

Projekti aloitettiin tekemällä selvitystyötä siitä, mitä alustaa asiakkaan viivakoodinlukulaite käyttää. Sen jälkeen pystyin työkaverini avustuksella valitsemaan sovelluksen kehittämiseen käytettävät teknologiat. Alustaksi asiakkaan laitteissa osoittautui Windows 10 -käyttöjärjestelmän mobiiliversio. Lopuksi tutkimme mitä pyyntöjä mobiilisovellus tarvitsee toimiakseen oikein ja toteutimme tarvittavat rajapinnat Sonet Premium -sovellukseen, jotta mobiilisovellus saatiin kommunikoimaan sen kanssa.

3.1 Projektinhallinta

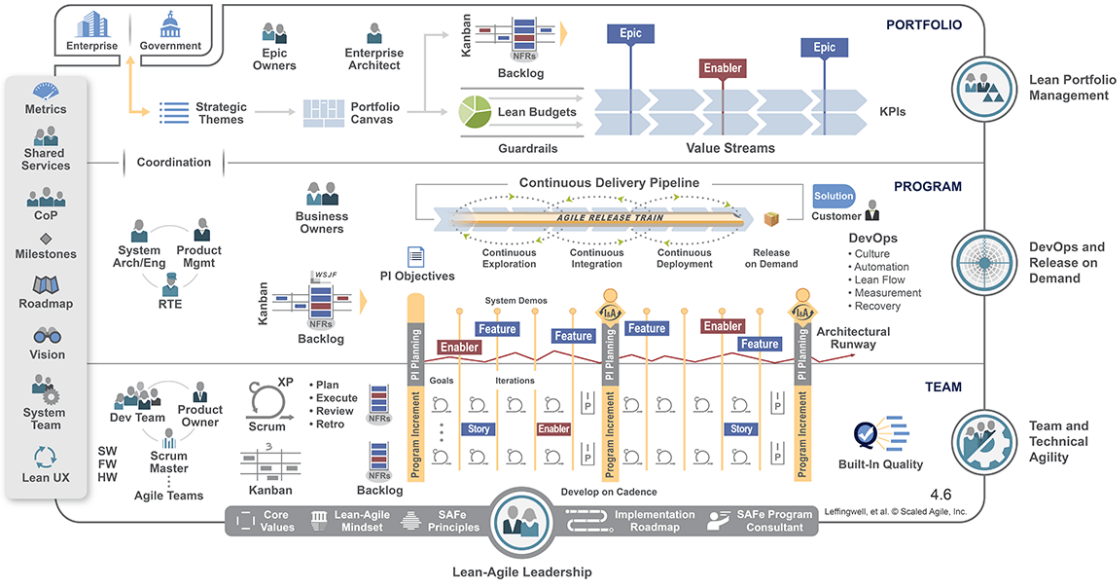
Mobiilisovelluksen kehityksessä hyödynnettiin ketterään ohjelmistokehitykseen pohjautuvia projektinhallintamenetelmiä. Pääasiallisena menetelmänä toimi Scrum-malli, jota hallittiin Team Foundation Server -projektinhallintatyökalun kautta. Scrum-mallissa kehitys tapahtuu niin sanotuissa sprinteissä eli kehitysjaksoissa. Jokaisen jakson aikana pyritään saamaan aikaiseksi julkaisukelpoinen tuote, jossa on toteutettuna kaikki kehitysjaksosuunnittelussa suunnitellut toiminnallisuudet. (Scrum.org 2019.)



Kuva 1. Scrum-malli kuvattuna (Scrum.org 2019.)

Scrum-mallin lisäksi toimittiin SAFe-mallin mukaisesti. SAFe (Scaled Agile Framework) on skaalautuva ketterän kehittämisen viitekehys, joka auttaa ketterässä kehityksessä, kun halutaan tehdä suurempi kokonaisuus. SAFe edistää ryhmittäytymistä, yhteistyötä ja toimitusta suurten ketterien kehitystiimien välillä. (Scaled Agile 2019.)

SAFen kehitysjaksoja eli iteraatioita kutsutaan nimellä Program Increment (PI). PI on aikaväli jolloin niin sanottu toimitusjuna eli Agile Release Train (ART) tuottaa lisäarvoa toimivien ja testattujen ohjelmistojen ja järjestelmien muodossa. Iteraation pituus on tyypillisesti 8-12 viikkoa, jonka aikana pyritään saavuttamaan PI-suunnittelun asettamat tavoitteet. (Scaled Agile 2019.)



Kuva 2. SAFe-mallin havainnekuva (Scaled Agile 2019.)

3.2 Aikataulutus

Sovelluksen kehitykselle oli laadittu tarkka aikataulu, jotta asiakas saisi sovelluksen käyttöönsä. Aikataulutuksessa käytettiin hyväksi ketterien menetelmien jaksotusta projektille. Mobiilisovelluksen eri työkokonaisuudet jaettiin toteutettavaksi yhden iteraation ja useamman eri kehitysjakson aikana. Yhden kehitysjakson pituus oli kaksi viikkoa.

Aikataulu kuitenkin venyi lähinnä oppimiseen kuluneen ajan, sekä projektin huonon ajoituksen takia. Projektin työstäminen kohdistui kesän lomakaudelle, jolloin kehittäjiltä tai asiakkaalta ei välttämättä voitu kysyä tarkentavia kysymyksiä. Työ saatiin kuitenkin valmiiksi lähes aikatauluun mennessä.

3.3 Dokumentaatio

Projektityön dokumentaatio oli lähinnä vanhemman sovellussuunnittelijan vastuulla. Sovellussuunnittelija kirjasi TFS-projektinhallintatyökaluun työhön kuuluvat työosiot ja dokumentoi muun muassa vaatimusmäärittelyn vaatimuksia ja toiminnallisuuksien tavoitteita. Vaatimusmäärittelyssä en ollut mukana, mutta esittelen sen yleisiltä osin seuraavassa pääkappaleessa.

Dokumentaatiosta omalle vastuulle jäi luoda asiakkaalle toimivat ja selkeät käyttöohjeet sovelluksen käyttöön. Vaikka ketterissä menetelmissä dokumentaation merkitys on vähäisempi, asiakaskohtaisissa töissä määrittelyn ja ohjeistuksen teko on tärkeää asiakastyytyväisyyden takaamiseksi.

4 MÄÄRITTELY JA SUUNNITTELU

4.1 Sovelluksen vaatimusmäärittely

Ohjelmistovaatimusmääritykset muodostavat perustan asiakkaiden ja urakoitsijoiden tai toimittajien väliselle sopimukselle. Sopimuksessa kerrotaan mitä ohjelmistotuotteen on tarkoitus tehdä ja mitä sen ei odoteta tekevän. Ohjelmistovaatimusmäärittely mahdollistaa vaatimusten tarkan arvioinnin ennen suunnittelun alkamista ja vähentää myöhempää uudelleensuunnittelua. Sen tulisi myös tarjota realistinen perusta tuotteen kustannusten, riskien ja aikataulujen arvioimiseksi.

Organisaatiot voivat käyttää ohjelmistotarpeiden erittelyasiakirjaa tehokkaan todentamis- ja validointisuunnitelmien kehittämisen perustana. Ohjelmistovaatimusmäärittely

tarjoaa myös hyvät lähtötiedot ohjelmistotuotteen siirtämiselle uusille käyttäjille tai ohjelmistoalustoille. Lopulta se voi tulevaisuudessa tarjota hyvän perustan ohjelmiston parannuksille. (IEEE Computer Society 2014.)

4.2 Inventointisovelluksen vaatimusmäärittely

Projektille oli luotu valmiiksi kattavat määritykset ja dokumentit, jossa kuvailtiin ohjelman haluttu toiminta, sekä ulkoasutoiveet. Myös työhön käytettävät ratkaisut olivat hyvin määritelty. Vaatimusmäärittelydokumentissa kerrotaan tarkasti, miten sovellus toimisi REST-rajapintoja hyödyntäen, kuvataan kyseisten rajapintojen toiminta, sekä näytetään minkä näköisiä sovelluksen eri käyttöruutujen tulisi olla. Näiden toiminnallisuuksien ja visualisoinnin avulla voidaan saavuttaa asiakkaan haluamia käyttötapauksia.

Määrittelyssä kerrotaan myös mitä keinoa sovellus käyttää käyttäjien todentamiseen, sekä mitä tietoformaattia käytetään tiedonvälitykseen. Määrittelyssä myös ehdotetaan sopivaa laitetta mobiilisovelluksen käyttämiseen. Luottamuksellisuussopimuksen vuoksi vaatimusten tarkempia parametreja ei tässä työssä esitellä.

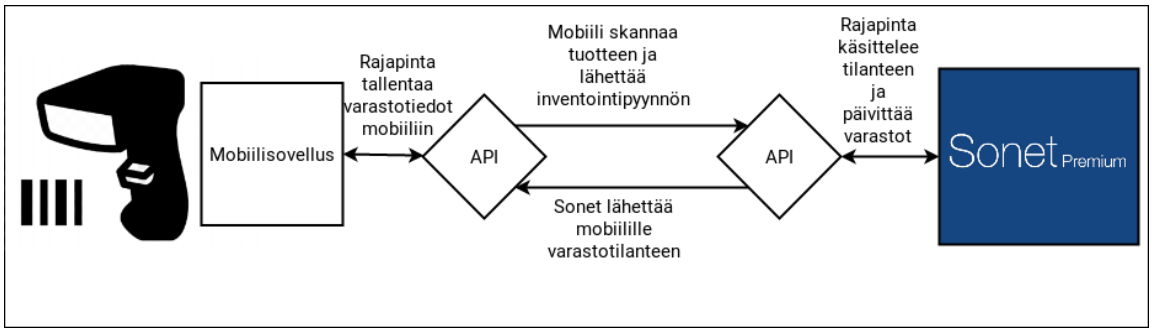
4.3 Mobiilisovelluksen suunnittelu

Sovelluksen määrittely ja suunnittelu tapahtuivat ripeään tahtiin, kuten yleensä ketteriin menetelmiin kuuluu. Sovellusta aloitettiin suunnittelemaan heti määrittelyvaiheen jälkeen. Suunnittelun hoidin yhdessä muiden sovelluskehittäjien kanssa.

Kehitysmenetelmäksi valittiin natiivi mobiilisovellus. Mobiilisovelluksen kehitykseen valittavien tekniikoiden valintaan vaikutti enimmäkseen mobiililaitteen alusta, Windows 10 Mobile. UWP ja C# sopivat siihen täydellisesti ja niissä yhdistyvät sekä ohjelmoinnin keveys, että tehokkuus. Ohjelmistokehys Prism tarjosi valmiin rungon sovellukselle käyttäen MVVM-sovellusarkkitehtuuria, joka helpottaa ulkoasun ja käyttöliittymän luomista. Aiheista Prism ja MVVM kerron myöhemmin luvussa 5.

Mobiilisovelluksen ulkoasu ja käyttöliittymä suunniteltiin vaatimusmäärittelyssä esitettyjen toiveiden mukaisesti. Mobiilisovellus oli myös tarkoitus saada vastaamaan käyttöliittymältään Sonet Premiumia. Ulkoasussa otettiin myös huomioon CGI:n omat käytännöt värien ja fonttien suhteen.

Kun mobiilisovelluksen rajapinnat olivat määritelty, aloitimme suunnittelemaan Sonet Premiumin omia rajapintoja. Rajapintojen tulisi lähettää ja vastaanottaa tietoa JSON-muodossa. Sonet Premium lähettäisi mobiilisovellukselle tiedot varastoista ja sen nimikkeistä, sekä varastosaldoista ja mobiilisovellus lähettäisi takaisin päivitetyt varastosaldot. Nämä rajapinnat tulisivat käyttämään X-kieltä ja aiemmin luotuja Java-luokkia.



Kuva 3. Sovelluksen suunniteltu toiminta.

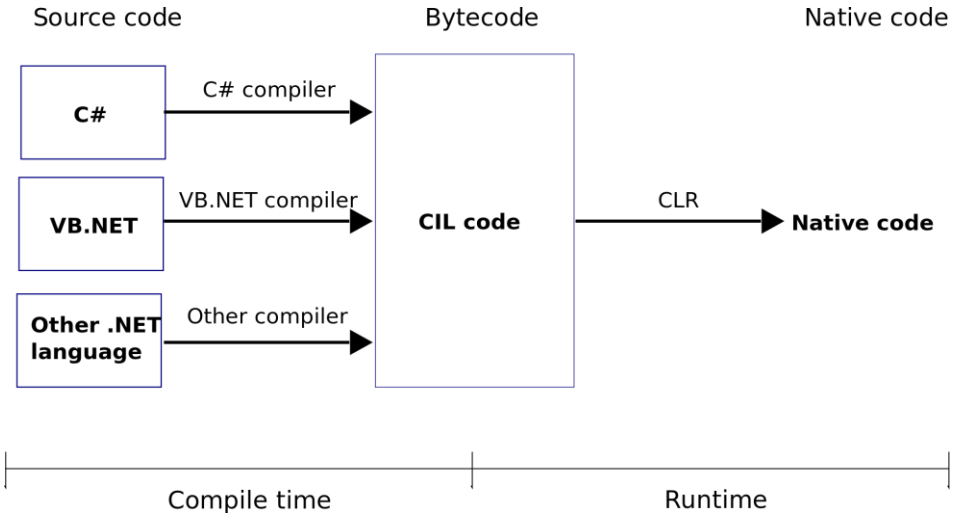
5 KÄYTETYT TEKNIIKAT JA TYÖKALUT

5.1 C#-ohjelmointikieli

C# (C sharp) on Microsoftin .NET-alustalleen kehittämä korkeamman asteen tulkattava olio-ohjelmointikieli, jonka tavoitteena on tehdä sovellusten kehityksestä mahdollisimman tehokasta. Saavuttaakseen ohjelmoinnin tehokkuuden, kieli painottaa yksinkertaisuutta, ilmaisukykyä ja suorituskykyä. Kieli on alustariippumaton, mutta se on luotu parhaiten toimivaksi .NET-alustan kanssa. C# toteuttaa olio-ohjelmointipara-

digman, joka sisältää kapseloinnin, periytyvyyden ja polymorfismin. Näiden abstraktiokeinojen avulla olioita voidaan muun muassa muokata ja käyttää monimuotoisesti. (O'Reilly 2012, 19-21.)

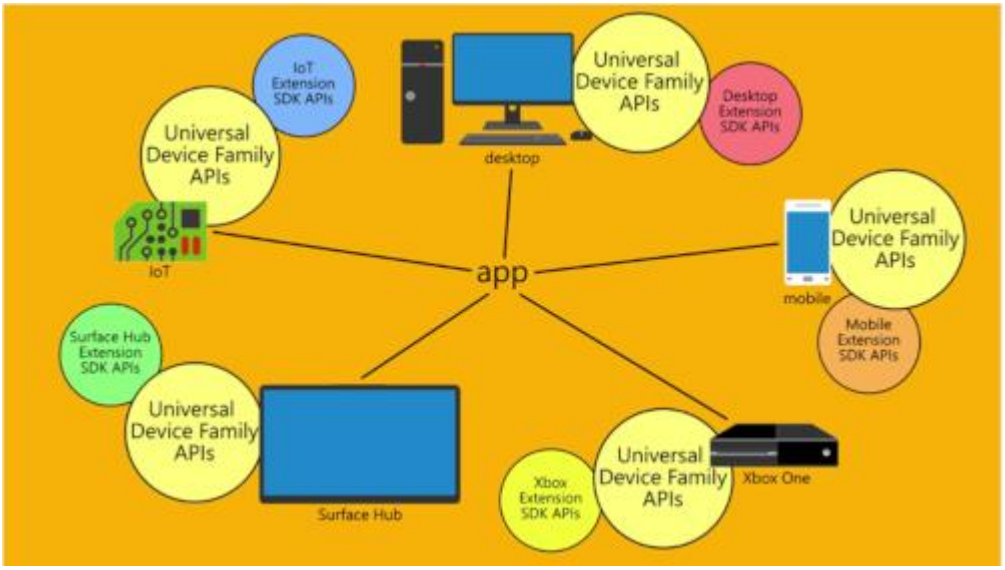
C# on yksi useista niin sanotuista tulkattavista kielistä, jotka käännetään tulkin läpi konekieleksi. Tulkattu koodi pakataan joko suoritettavan tiedoston (.exe) tai kirjaston (.dll) muotoon, yhdessä tyyppitietojen tai metatietojen kanssa. Tulkattu koodi on esitetty välikielellä eli CIL:llä.
Tämän välikoodin CLR kääntää ajonaikaisesti konekieleksi, jonka suoritin voi suorittaa.



Kuva 4. C#-sovelluksen käännös konekieleksi (Leif Arne Storset 2008.)

5.2 UWP

Windows 10 toi mukanaan uuden avoimen lähdekoodin ohjelmointirajapinnan, joka auttaa kehittämään universaaleja sovelluksia. Nämä sovellukset toimivat eri Windows 10-pohjaisilla laitteilla ilman tarvetta ohjelmoida sovellusta uudelleen. UWP:n eli Universal Windows Platformin ansiosta jokaisella laitteella on yhteinen rajapinta ja sama ohjelmakoodi pyörii kaikilla laitteilla samalla tavalla. (Microsoft 2019.)



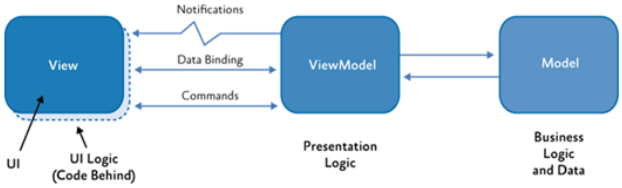
Kuva 5. UWP toimii jokaisella Windows 10 –pohjaisella laitteella yhteisen API:n kautta. (Microsoft 2018.)

5.3 XAML

XAML (Extensible Application Markup Language) on niin kutsuttu deklaratiivinen kieli, jota käytetään apuna graafisten käyttöliittymien luomiseen. Sitä käytetään lähes aina imperatiivisen ohjelmointikielen kanssa, kuten C#. (Buddy James, Lori Lalonde, 3.)

5.4 MVVM ja Prism

MVVM (Model-view-viewmodel) on sovellusarkkitehtuuri, joka erottaa graafisen käyttöliittymän ja sovelluksen päälogiikan kehittämisen erilleen toisistaan. Prism on ohjelmistokehys, joka käyttää hyväkseen MVVM-arkkitehtuuria ja sillä voidaan luoda XAML-sovelluksia (Prism 2017.)



Kuva 6. Model-view-viewmodel-arkkitehtuurin toimintaperiaate (Prism 2017.)

5.5 X-ohjelmointikieli

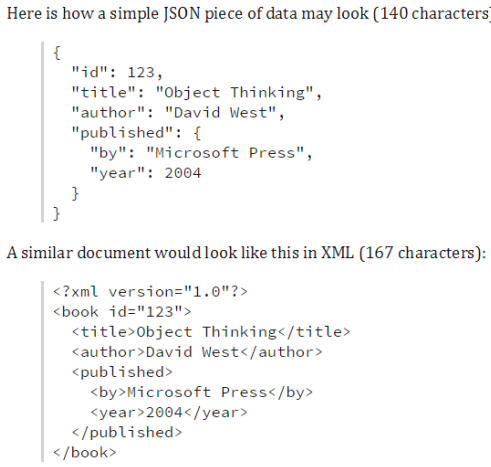
X-kieli on Sonetin sisäiseen kehitykseen luotu kieli, sovelluksen sisällä oleva Magda-niminen moottori kääntää kielen toimiviksi toiminnoiksi. Kieli on C-johdannainen kieli ja hyvin pelkistetty Sonetin tarpeisiin. Kun Sonetia tarvitsee uudistaa, ei tarvitse muuttaa kuin X-kieltä kääntävää Magdaa. 20 vuotta sitten kirjoitettu X-koodi toimii edelleen samalla lailla.

5.6 REST

REST eli Representational State Transfer on ohjelmistoarkkitehtuurimalli, joka tarjoaa standardin verkossa toimivien tietokonejärjestelmien välille. REST helpottaa järjestelmien kommunikointia keskenään. REST-yhteensopivia järjestelmiä kutsutaan usein RESTful-järjestelmiksi. RESTful-järjestelmille on tunnusomaista se, miten ne ovat tilattomia ja erottavat palvelimen ja asiakkaan tarpeet. (Roy Thomas Fielding 2000.)

5.7 JSON

JSON eli JavaScript Object Notation on kevyt tekstipohjainen avoin standardi, joka on suunniteltu tiedonsiirtoon, joka olisi myös ihmisen luettavissa. Se on täysin kieliriippumaton ja sitä on sekä ihmisten helppo lukea ja kirjoittaa, että tietokoneiden helppo jäsentää ja tuottaa. Perinteisempään tiedostomuotoon XML:ään verrattuna JSON on huomattavasti yksinkertaisempi. (Sanjay Patni 2017, 36-40.)


Here is how a simple JSON piece of data may look (140 characters):

```
{
  "id": 123,
  "title": "Object Thinking",
  "author": "David West",
  "published": {
    "by": "Microsoft Press",
    "year": 2004
  }
}
```

A similar document would look like this in XML (167 characters):

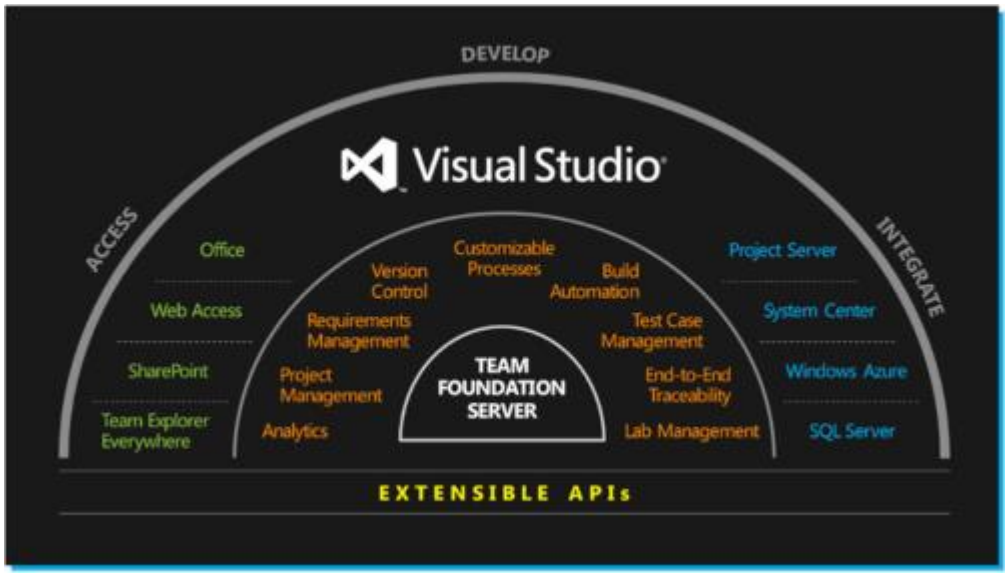
```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Kuva 7. Sama tieto sekä JSON-, että XML-muodossa (Dave White 2016.)

5.8 Team Foundation Server

Team Foundation Server on erillinen palvelintuote. Se on suunniteltu erityisesti ohjelmistojen suunnitteluryhmille kehittäjien, testaajien, arkkitehtien, projektipäälliköiden, liiketoiminta-analyytikoiden ja muiden ohjelmistojen kehityksessä mukana olevien kanssa. TFS kattaa kaikki ohjelmistotuotteen hallintaan liittyvät osa-alueet, kuten version-, projektin-, dokumentoinnin- ja vaatimustenhallinnan, sen koko elinkaaren ajan. (Jakob Ehn 2012, 16.)

TFS on myös suunniteltu toimimaan hyvin muiden työkalujen kanssa, esimerkiksi Visual Studio:ssa luodut ohjelmatiedostot voidaan suoraan lähettää TFS:n versionhallintaan. TFS:n laajennettavissa olevan ohjelmointirajapinnan avulla kuka tahansa voi käyttää ohjelmassaan TFS:n toimintoja. (Jakob Ehn 2012, 16.)

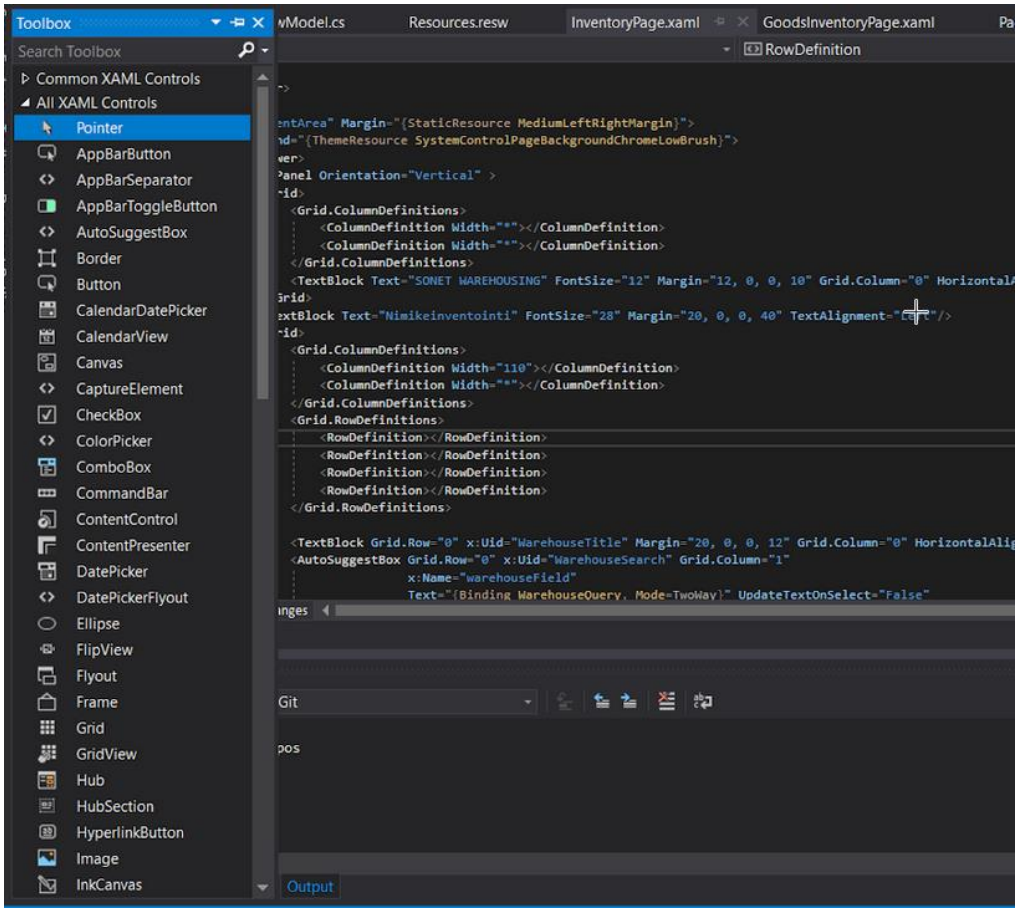


Kuva 8. TFS integroituu hyvin muihin työkaluihin, sekä on helposti laajennettavissa. (Jakob Ehn 2012, 16.)

6 TOTEUTUS

6.1 Mobiilisovellus

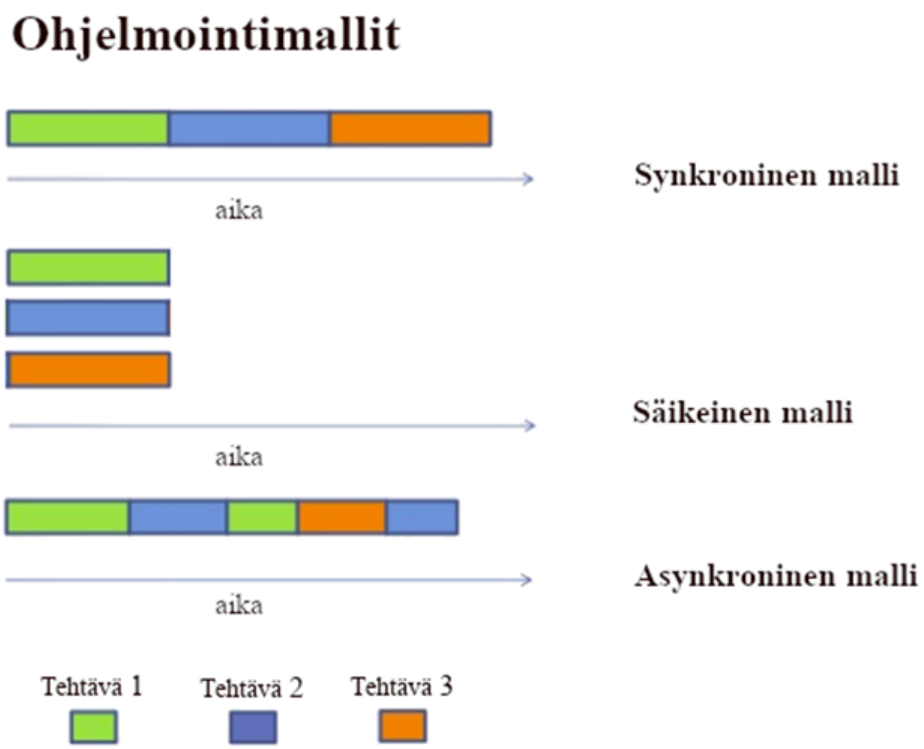
Mobiilisovelluksen kehitys koostui kolmesta eri osa-alueesta. Käyttöliittymän ja sen grafiikan luonnista, itse pääsovelluksen toteuttamisesta, sekä Sonetin ja mobiilisovelluksen välillä kommunikoivan rajapinnan luonnista. Ensin kehitys keskittyi käyttöliittymään. Vastuu käyttöliittymän luonnista oli minulla ja se osoittautui huomattavasti helpoimmaksi osuudeksi. UWP:n käyttämä XAML teki käyttöliittymän ja sen grafiikoiden luonnista äärimmäisen helppoa. Kieltä on helppo ymmärtää, sekä elementtejä voi lisätä drag ‘n drop -tyylillä, kuten alemmasta kuvasta näkee.



Kuva 9. Toolbox-valikosta voi poimia haluamansa valmiin elementin.

Itse pääsovellus luotiin C#:lla ja Prism-nimisellä ohjelmistokehyksellä. C# olikin minulle jo hyvinkin tuttu koulusta. MVVM-mallinen kehittäminen, asynkroninen ohjelmointi ja UWP oli minulle täysin uutta ja sen opettelussa meni paljon aikaa. Asynkronista ohjelmakoodia sovelluksessa käytettiin paljon, sillä muuten ohjelma jumittaisi

käyttöliittymän joka kerta kun ohjelma tekee jotain taustalla. Asynkroninen ohjelmakoodi jakaa sovelluksen toimintaa töihin, joiden valmistumista ei jäädä odottamaan. Näin ollen voidaan tehdä useampaa asiaa samaan aikaan, kuten alemmasta kuvasta näkee.



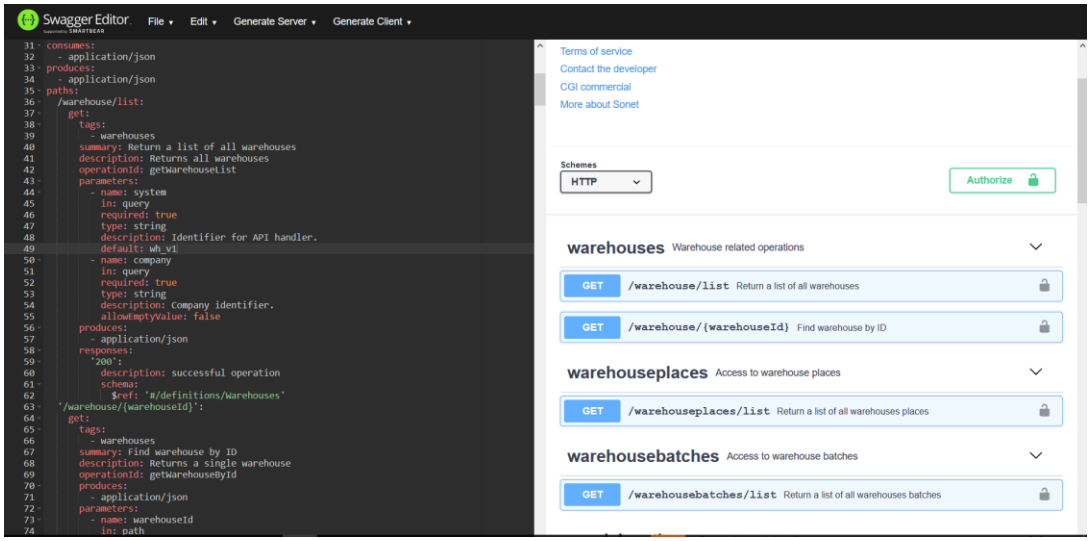
Kuva 10. Asynkroninen toimintapa verrattuna muihin.

```
protected async Task AddToBalanceCommandAction()
{
    int updatedBalance = _previousBalance + _inventoryAmount;
    // store updated value here as we call BalanceCommandAction which will use this value.
    InventoryAmount = updatedBalance.ToString();

    // update value
    await BalanceCommandAction();
}
protected async Task BalanceCommandAction()
{
    // update value
    if (ScannedObject != null)
    {
        ScannedObject.Balance = InventoryAmount;

        IDataStoreReturnCodes returnValue = _dataStore.UpdateScannedObject(ScannedObject, IDConstants.SCANNED_OPERATIONDETAILS_ID);
        if (returnValue != IDataStoreReturnCodes.NoErrors)
        {
            var message = new MessageDialog(ResourceLoader.GetForCurrentView().GetString("uptadeScannedObjectFailed"));
            await message.ShowAsync();
        }
        InventoryAmount = DEFAULT_INVENTORY_AMOUNT.ToString();
        ScannedTitle = string.Empty;
        ScannedObject = null;
    }
}
```

Kuva 11. Asynkronista ohjelmakoodia mobiilisovelluksesta

6.1.1 Rajapinta

Mobiilisovelluksen rajapinnan luomisen hoiti sovelluskehittäjä, joka käytti tehtävään automatisoitua koodigeneraattoria, Swaggeria. Swaggerillä luotu koodi pystyttiin suoraan kytkemään kiinni mobiilisovellukseen.



Kuva 12. Swaggerilla luodut määrittelyt mobiilirajapinnalle.

6.2 Sonet Premiumin rajapinnat

Swaggerilla toteutettu määrittely kertoi minkälaisia pyyntöjä mobiilisovellus Sonetille lähettäisi. Tämän jälkeen tehtäväksi jäi vielä selvittää, kuinka nuo pyynnöt käsiteltäisiin Sonetissa. Onnekseni Sonetiin oli valmiiksi jo luotu erilaisia testejä, jotka kokeilivat tämän kaltaisia pyyntöjä ja pystyin ottamaan mallia niistä. Tarvitsin kuitenkin sovelluskehittäjien apua joissain asioissa, kuten kuinka pyynnössä tulleet inventoinnit päivitetään Sonetissa varastoon.

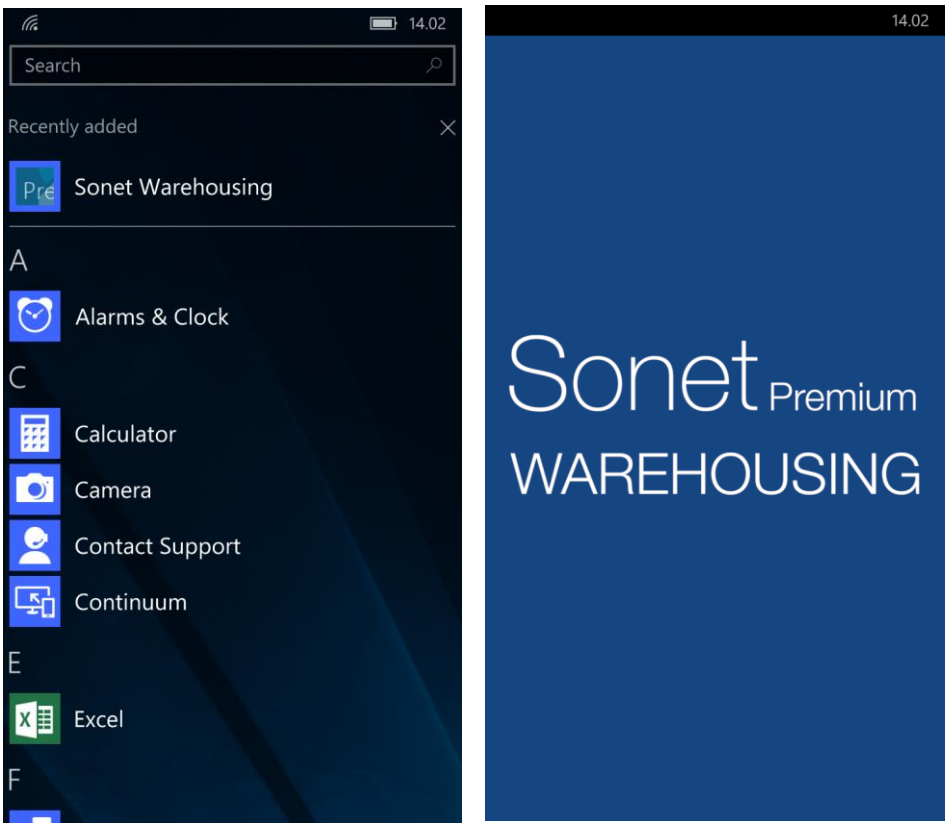
X-kieli tarjosi valmiiksi tarvittavat funktiot JSON-datan luontiin, sekä myös tietokantapyyntöjen luomista on helpotettu niin sanotuilla tietokantanäkymillä. Tietokantanäkymiin on helppo määritellä tarvittavat tietokantakentät ja kuinka niitä käsitellään. Kun rajapinta on valmis, taustalla pyörivä Magda-moottori tulkkaa ohjelmakoodin ja se on heti käytettävissä.

```
if(Empty(warehouseId))
{
olno_var1 = $MinStr; olno_var2 = $MaxStr;
}
else
{
olno_var1 = warehouseId; olno_var2 = warehouseId;
}

if(!OpenSet(VARASTOA)) return(404);

builder = CreateJsonBuilder();
AddJsonArray("warehouses",builder);

i = 0; // start of index
for( ; FetchNext(VARASTOA); )
{
  if(i != 0) NextJsonArrayElement(builder);

  AddJsonProperty("id",olno_var,builder);
  AddJsonProperty("companyid",yrtun,builder);
  apustr = GetLangText(-1,varnimi);
  AddJsonProperty("name",apustr,builder);
  AddJsonProperty("warehousetype",varkentta4[0:1],builder);
  i++;
}

M4J_response = GetJsonString(builder);

return(200);
```

Kuva 13. Esimerkkinä rajapinta, joka hakee varastojen perustiedot.

7 LOPPUTULOS

Tässä kappaleessa esittelen miltä valmis sovellus lopulta näytti ja näytän sovellusta toiminnassa. Ensimmäisessä kuvassa näkyy miltä sovellus näyttää sovellusvalikossa, pikakuvakkeen kuvaksi valittiin tietokoneversion nykyinen pikakuvakekuva. Toisessa kuvassa sovellus käynnistyy ja näyttää tekemäni käynnistysruudun. Sovelluksen käyttöruudut muistuttavat pitkälti vaatimusmäärittelyssä esitettyjä toiveita, sekä sovellus vastaa värimaailmaltaan ja käyttöliittymältään hyvin pitkälti Sonet Premiumia.

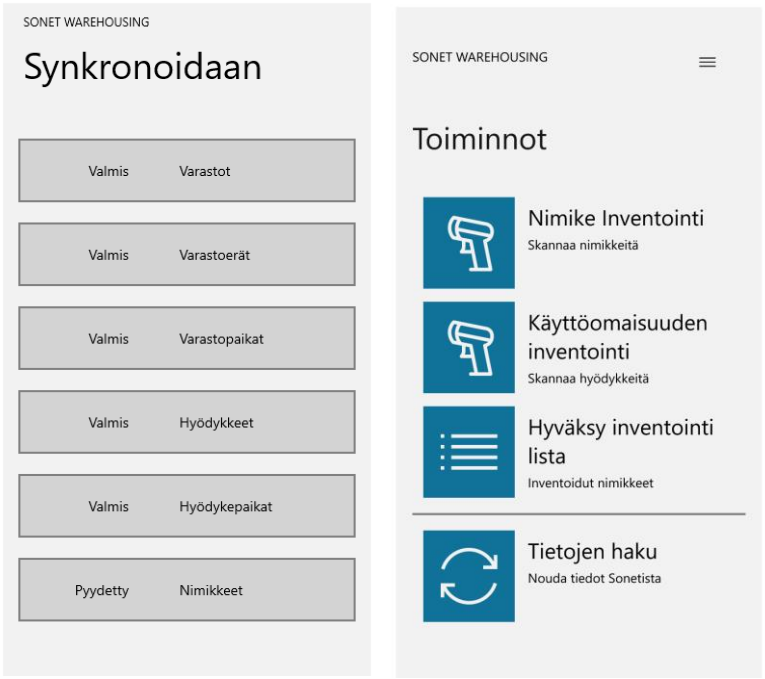


Kuva 14. Sovelluksen pikakuvake ja käynnistysruutu.

Sovellusta ensimmäistä kertaa käynnistäessä asetuksiin tarvitsee määrittää vähintään yritystunnus ja Sonet-palvelimen osoite. Palvelimelta mobiilisovellus hakee kaiken datansa. Käyttäjätunnuksina toimivat Sonetiin luodut tunnukset.

Kuva 15. Kirjautumisruutu ja asetukset.

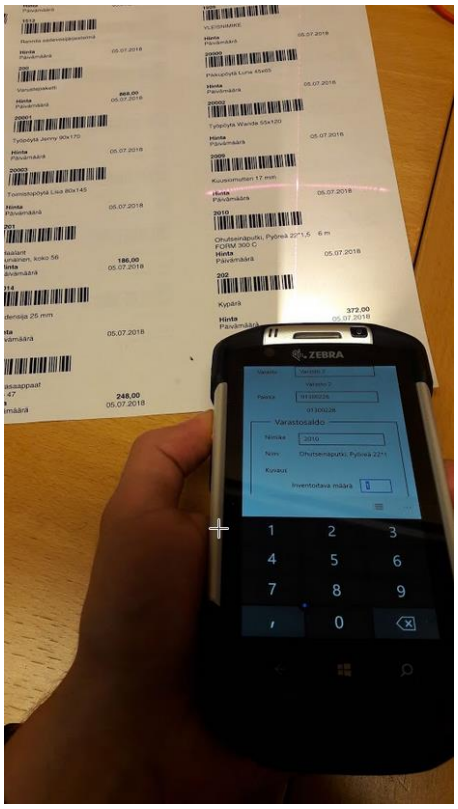
Oikeiden asetusten määrittämisen ja kirjautumisen jälkeen päästään sovelluksen päävalikkoon, jossa voidaan ensin käydä hakemassa Sonetin tiedot mobiiliin. Sovellus voi toimia ilman verkkoa ja näin ollen ylläpitää omaa paikallista tietokantaansa.



Kuva 16. Sovelluksen päävalikko ja tietojen haku.

Kun varastotiedot ovat saapuneet Sonetista, inventointi voidaan aloittaa. Ennen kuin nimikkeitä voidaan inventoida, pitää syöttää varaston ja varastopaikan tiedot. Tämän jälkeen nimike voidaan inventoida joko syöttämällä tiedot ennakoiviin tekstikenttiin tai skannaamalla viivakoodeja laitteella ja antamalla inventoitavan määrän. Nimikkeen inventoinnin jälkeen ohjelma siirtyy nimikelistalle, jossa tietoja voidaan vielä muokata. Kun inventointi on valmis, tiedot lähetetään nappia painamalla Sonetiin päivitettäväksi.

Kuva 17. Nimikkeen inventointi ja nimikelista. Sovellus toimi kuten pitikin ja viivakoodien salamannopea lukeminen yllätti kehittäjätkin.



Kuva 18. Inventointisovellus toiminnassa.

8 YHTEENVETO

Opinnäytetyössä suunniteltiin ja toteutettiin mobiili-inventointityökalu Sonet Premium –ohjelmistoa varten. Mobiilisovelluksen toteutus koostui kahdesta osasta. C#:lla tehtiin mobiilisovellus ja X-kielellä REST-rajapinta Sonetiin.

Opinnäytetyön aikana opin paljon uutta C#-kielestä ja sen edistyksellisemmistä ominaisuuksista, sekä huomattavan paljon Sonetista ja sen käyttämästä X-kielestä. Sain myös hieman käsitystä siitä, miten taustalla toimiva Magda-moottori hoitaa suurimman osan kehitystyöstä. Asioita kuitenkin raapaistiin vain pintapuolisesti, sillä Sonet Premium on massiivinen ohjelmistokokonaisuus ja pitää sisällään valtavan paljon opittavaa asiaa. Myös sovellusten ketterät kehitysmallit tulivat työn aikana tutuksi, kuten Scrum ja SAFe.

Lopputulokseksi saatiin yllättävän toimiva kokonaisuus, johon asiakaskin oli tyytyväinen. Jatkokehittämisen varaa tosin jäi ulkoasun osalta, sekä myös muiden kuin Windows-pohjaisten laitteiden tukemista voisi minun puolestani harkita tulevaisuudessa. Windows-pohjaiset mobiililaitteet ovat näillä näkymin siirtymässä historiaan, eikä asiakkaan käytössä ollutta skannauslaitettakaan enää valmisteta Windows-pohjaisena.

LÄHTEET

.NET Foundation. Prism Library. Viitattu 22.04.2019.
https://prismlibrary.github.io/docs/wpf/images/Ch5MvvmFig1.png

Amazon Web Services. What is Mobile Application Development?
Viitattu 02.10.2019.
https://aws.amazon.com/mobile/mobile-application-development/

Buddy James Lori Lalonde 2012. Pro XAML with C#: From Design to Deployment on WPF, Windows Store and Windows Phone.
Kanada, Yhdysvallat: Apress Berkley.

Ehn Jakob. 2012. Team Foundation Server 2012 Starter.
Iso-Britannia: Packt Publishing Limited

Fielding Roy Thomas 2000. Viitattu 22.04.2019.
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

IEEE Computer Society 2014. Guide to the Software Engineering Body of Knowledge.
IEEE Computer Society Press.

Microsoft. What's a Universal Windows Platform (UWP) app? Viitattu 22.04.2019.
https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

O'Reilly. 2015. C# 6.0 in a Nutshell 6th edition.
Yhdysvallat: O'Reilly Media Inc.

Patni Sanjay. 2017. Pro RESTful APIs_ Design, Build and Integrate with REST, JSON, XML and JAX-RS. Yhdysvallat: Apress Berkley.

Scaled Agile. Scaled Agile Framework. Viitattu 02.10.2019.
https://www.scaledagileframework.com

Scrum.org. The Scrum Framework Poster. Viitattu 02.10.2019.

https://www.scrum.org/resources/scrum-framework-poster

Sommerville Ian 2011. Software Engineering (9th Edition)

Yhdysvallat: Pearson Education, Inc.

Storset Leif Arne. Common Language Runtime diagram. Viitattu 22.04.2019.
https://commons.wikimedia.org/wiki/File:CLR_diag.svg

White Dave. XML vs JSON. Viitattu 02.10.2019
https://blog.quark.com/2016/08/xml-vs-json/amp/