VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Niko Peräkorpi

XAMARIN CROSS PLATFORM

Mobiilisovelluksen kehitys Windows- ja Android-alustoille

VAASAN AMMATTIKORKEAKOULU

Tekniikka ja liikenne
2015

Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Niko Peräkorpi
Opinnäytetyön nimi	Xamarin Cross Platform
Vuosi	2015
Kieli	suomi
Sivumäärä	45
Ohjaaja	Timo Kankaanpää

Opinnäytetyössä luodaan mobiilisovellus Windows- ja Android-käyttöliittymille. Sovelluksesta on pois käytöstä otettu alkuperäinen Javascript versio, jonka pohjalta sovellus rakennetaan. Sovellus tehdään aluksi Windows-puhelimille, jonka jälkeen se portataan Androidille käyttäen Xamarin-kehitystyökalua, joka on laajalti käytetty maksullinen työkalu jolla porttaus Windows-, iOS- ja Android-käyttöliittymien välillä onnistuu pienellä työmäärällä.

Sovelluksen toteutuksessa tärkeimpänä osana on rakentaa Windows-versio sovelluksesta niin, että sen logiikka ei käytä kirjastoja, jotka ovat yksinomaisesti Windows-käyttöliittymälle. Logiikkakoodi, eli metodit, jotka hakevat dataa tietokannasta ja hallitsevat ohjelman datankulun ja logiikan, kirjoitetaan erilliseen projektiin. Tällöin Windows- ja Android-puhelimille tehdyillä projekteilla voidaan viitata logiikka-projektiin, ja käyttää sen metodeja. Kaikki logiikkakoodi tehdään C#-koodikielellä, joka on Windows-puhelimen natiivikieli, sillä Xamarinin avulla se pystytään kääntämään myös Android- ja iOS-alustoille. Kaikkein optimaalisimmassa tilanteessa Android-sovellukselle täytyy vain rakentaa ulkoasu joka kutsuu metodeja erillisestä, Windows-sovelluksen luomisen yhteydessä tehdystä projektista.

Opinnäytetyön tavoitteet saatiin Android-version loppuunsaattamista lukuunottamatta toteutettua. Vaikeimpia asioita työssä oli MVVM-rakenteen kehittäminen järkevästi, joka on myös sovelluksen tärkeimpiä osa-alueita. Sovellus saatiin rakennettua MVVM-mallisesti, vaikka rakennetta voisi varmasti parantaa.

Kaiken kaikkiaan opinnäytetyö oli erittäin mielenkiintoinen ja haastava. Ensimmäistä kertaa monialustaisen mobiilisovelluksen luominen ei ole helppoa, mutta työ onnistui hyvin ja lopputulokseen voi olla tyytyväinen.

Avainsanat Xamarin, tehokkuus, opetusmenetelmät

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Niko Peräkorpi
Title	Xamarin Cross Platform
Year	2015
Language	Finnish
Pages	45
Name of Supervisor	Timo Kankaanpää

This thesis is about implementing a mobile app for Windows and Android platforms. The app is based on an outdated, deprecated Javascript application. The application will be first developed on the Windows platform, after which it will be ported to Android phones using Xamarin cross-platform development software. Xamarin is a widely used tool which makes porting applications between Windows, iOS and Android simple and easy.

The most important part of the implementation is to build the original Windows version so that it will be easy to port to other platforms. Therefore, the logic of the program needs to be written to be independent from the platform. The logic code will contain methods that control the data flow and logic of the application. The logic will be built in it's own separate project from which we can call the methods to complement the user-interface code. All of the logic will be written in C# which is the native language for Windows platform, because with Xamarin it can be compiled to run in Android and iOS platforms. In the most optimal situation, the Android application will only need a completely new user-interface code which uses pre-created methods from the logic project.

Other than completely finishing and polishing the Android version, the requirements of the thesis were achieved. One of the most difficult tasks of the thesis was implementing the MVVM architecture well and logically. The program was successfully built using the MVVM architecture, but it might be possible to still optimize the architecture.

Overall the thesis was very interesting and challenging. Developing a multiplatform mobile program is not easy, but it went well and the outcome was good.

Keywords Xamarin, cross-platform, mobile

KÄYTETYT LYHENTEET

JSON	JavaScript-olioiden esitysmuoto tekstinä
XML	Merkintäkieli, joka sisältää tiedon ja sen merkityksen
XAML	Microsoftin kehittelemä XML-pohjainen merkintäkieli
AXML	Web-palveluihin kykenevä XML-pohjainen merkintäkieli
MVVM	Arkkitehtoninen malli ohjelmistokehitykseen
HTTP	Web-palvelimen ja –selaimen välinen keskusteluprotokolla
URI	Tiedon paikan sisältävä merkkijono
URL	Internetissä olevan tiedon paikan sisältävä merkkijono

SISÄLLYSLUETTELO

KUVA- JA TAULUKKOLUETTELO

1 JOHDANTO

1.1 Toimeksiantajaorganisaatio

Tämän opinnäytetyön toimeksiantajana toimii vaasalainen tietotekniikan yritys Devatus. Devatus on vuonna 2010 perustettu sovelluskehitykseen suuntautunut yritys, joka kehittää enimmäkseen mobiili- ja websovelluksia. Devatuksella on henkilökuntaa noin 10, mutta vaikka yritys on pienehkö, se on onnistunut saamaan suuria paikallisia yrityksiä asiakkaikseen. Yritys antoi opinnäytetyöaiheen työharjoitteluni yhteydessä.

1.2 Työn aihe ja sisältö

Tämän opinnäytetyön aiheena on mobiilisovellus, joka kehitetään sekä Windows- että Android alustalle. Opinnäytetyöhön kuuluu suunnitteluvaihe, jossa mietitään sovelluksen arkkitehtuuri ja analysoidaan työmäärä sekä työhön tarvittavat teknologiat, toteutusvaihe jossa kehitetään sovellus alusta loppuun, sekä testausvaihe jossa selvitetään onko sovelluksen toiminta vaatimuksienmukainen.

Suunnitteluvaiheessa perehdytään MaintBox web-sovellukseen ja sille luotuun JavaScript mobiilisovellukseen, jotka toimivat tämän uuden monialustaisen mobiilisovelluksen pohjana. Sen lisäksi suunnitellaan itse ohjelmistokoodin rakenne. Tässä työssä käytämme Model View Viewmodel (MVVM) rakennetta koodille, joka mahdollistaa ohjelman logiikan ulkoistamisen sovelluksen ulkoasusta. Koska ohjelma on monialustainen, täytyy koodirakenteen suunnittelussa ottaa myös huomioon, että Windows-pohjaiset kirjastot eivät ole käyttökelpoisia Android-alustalla ja toisinpäin, eli koodi, joka jaetaan Windows- ja Android alustoille ei saa sisältää mitään alustakohtaisia kirjastoja tai referenssejä. Suunnittelussa mietitään myös, kuinka kauan aikaa ohjelman eri osien toteuttamiseen kuluu.

Toteutusvaiheessa rakennetaan itse sovellus. Työn ensimmäisessä vaiheessa luodaan sovellus Windows-puhelimelle. Ensimmäisessä vaiheessa luodaan sovelluksen ulkoasun lisäksi myös ohjelman koko logiikkakoodi. Tämän jälkeen aloitetaan ohjelman porttaus Android-puhelimelle käyttäen Xamarin-työkalua.

Xamarin mahdollistaa C# koodin kääntämisen Windowsin lisäksi myös Android- ja iOS alustoille. Android-puhelimelle luodaan siis täysin uusi ulkoasu, sillä Xamarin ei pysty kääntämään graafista käyttöliittymää alustalta toiselle. Kaikista optimaalisimmassa tapauksessa logiikkakoodiin ei tarvitse koskea ollenkaan, mutta tässä työssä siihen tehdään hieman muutoksia Android version toteutuksen yhteydessä, sillä jotain asioita jäi ottamatta huomioon kun logiikkakoodi alunperin luotiin.

Sovellusta testataan sisäisesti yhtenäisesti sen kehityksen ajan, mutta varsinainen testausvaihe suoritetaan kaksi kertaa. Ensimmäisen kerran, kun Windows-alustan sovellus on saatu valmiiksi ja käyttökelpoiseksi. Näin löydetään ja voidaan korjata kaikki mahdolliset ongelmat, mitä ohjelman logiikkaan on voinut jäädä, ja mahdolliset ongelmat Android-version toteutuksessa jäävät minimiin. Testausvaiheessa sovellus annetaan asiakasyritykselle testikäyttöön, jolloin selvitetään soveltuuko ohjelma käytännön olosuhteisiin. Viimeinen testivaihe suoritetaan, kun Android-sovellus on saatu täysin valmiiksi ja toimii identtisesti Windows-version kanssa.

2 TAUSTATEKNIIKAT

2.1 Visual Studio

Opinnäytetyössä käytetään kehitysympäristönä Microsoftin Visual Studio 2013 versiota. Visual Studio tarjoaa hyvän graafisen käyttöliittymän ohjelmakoodin kirjoittamiseen, sekä Intellisense-teknologian joka on ns. 'älykäs koodin täydennys' -teknologia. Se tarjoaa käyttäjälle koodia kirjoittaessa vaihtoehtoja esim. muuttujista ja luokista, ja täydentää rivin lopun automaattisesti käyttäjän komennosta. Visual Studio tukee monella eri ohjelmointikielellä ohjelmoimista, joista tässä opinnäytetyössä käytämme C#, XAML sekä AXML kieliä.

2.1.1 Visual Studio Designer

Visual Studio 2013 tarjoaa käyttäjälle helpon keinon luoda sovelluksille näyttäviä ulkoasuja. Kun Visual Studiolla luodaan sovellus, jossa on myös graafinen puoli, se tarjoaa käyttäjälle mahdollisuuden käyttää Visual Studio Designeriä, joka visualisoi ja nopeuttaa ulkoasun luomisen. Designer on ikkuna, joka osoittaa miltä sovellus näyttää, kun se on käynnissä. Designeriin voi vetää ns. 'drag and drop' tekniikalla ulkoasun komponentteja, jolloin se lisää kyseisen komponentin ja sen parametrit automaattisesti ohjelman XAML koodiin.

2.2 Silverlight

Silverlight on Microsoftin sovelluskehys web- ja mobiilisovelluksille. Se tarjoaa sovelluksille graafisen systeemin ja integroi multimedian, grafiikat, animaatiot ja interaktiivisudeen yhteen RE:iin. Silverlight ei ole enää niin suuressa osassa sovelluskehitystä kuin ennen, sillä Microsoft on sanonut lopettavansa sen tukemisen lähitulevaisuudessa, mutta sillä saa edelleen luotua ulkoasultaan kauniita sovelluksia.

2.3 Xamarin

Xamarin on Xamarin-nimisen ohjelmistostudion luoma alusta, jonka avulla voidaan kehittää Windows, Android ja iOS sovelluksia C#:lla. Sen avulla kehittäjät voivat käyttää olemassaolevaa C# koodia muille alustoille, ja jakaa

suuren osan koodista. Tämän johdosta työmäärä, joka sovelluksen porttauksesta syntyy, vähenee suuresti. Xamarin tarjoaa myös Xamarin Studio kehitysympäristön, mutta tässä opinnäytetyössä käytämme Visual Studiota johon saa Xamarin-lisätoiminnot. Xamarinin käyttö ei ole ilmaista, vaan se vaatii maksullisen lisenssin.

Xamarin tarjoaa ominaisuuden nimeltä Xamarin.Forms, jolla myös ulkoasu voidaan portata helposti alustalta toiselle. Xamarin.Forms käyttää Windows-tyylistä XAML-C# paria näkymillään, jolloin myös databindaus onnistuu. Tämä ei kuitenkaan tarkoita sitä, että Windows-ulkoasukoodin voisi suoraan kopioida Xamarin.Formsiin vaan Xamarin.Forms sisältää omat ulkoasukomponenttinsa ja metodinsa. Tässä työssä ei käytetä Xamarin.Formsia vaan Windowsille luodaan oma Windows-ulkoasu käyttämällä XAML-tekniikkaa, ja Androidille luodaan oma ulkoasunsa käyttämällä AXML-tekniikkaa.

Xamarinia käytetään tässä opinnäytetyössä siten, että luodaan Visual Studioon Portable Class Library(PCL) -projekti, joka sisältää kaiken sen koodin, mikä ei ole alustasta riippuvainen. Lisäksi luodaan alustakohtaiset Android- ja Windows Phone-projektit. Alustakohtaisiin projekteihin lisätään referenssi PCL-projektiin, jolloin alustakohtaisissa projekteissa voidaan käyttää PCL-projektin metodeja ja luokkia.

PCL-projekti sisältää sovelluksen kontrollerit, viewmodelit sekä modelit. Koska tässä työssä ei käytetä Xamarin.Formsia, ViewModelit ovat vain pienessä osassa Android-alustalle kehitettävässä projektissa. ViewModelia ei voi sitoa Androidissa samalla tavalla kuin Windowsilla, mutta Viewmodelin metodeja voidaan kuitenkin hyödyntää.

2.4 Push-notifikaatiot

Internetiin pohjautuva kommunikaatiotyyli, jossa keskinäinen serveri hallitsee viestien kulkua. Asiakasohjelma liittyy serverin tarjoamiin informaatio-'kanavaan' ja kun kanavalla on uutta tietoa tarjottavana, se lähettää sen asiakasohjelmalle. Esimerkiksi paljon käytetty WhatsApp käyttää push-notifikaatioita viesti-

ilmoituksissa.

2.5 JSON

Sovelluksen tietojenkäsittely tapahtuu Http-kutsuina, jolloin web-palvelin palauttaa dataa tietokannasta. Tietokannan data muutetaan JSON (JavaScript Object Notation) muotoon ja palautetaan Http-kutsun vastauksena mobiilisovellukselle. JSON-merkkijono koostuu attribuuttiarvopareista, ja on toimiva vaihtoehto esim. XML:lle. JSON-merkkijono voidaan deserialisoida suoraan olioksi, jota voidaan sitten käyttää sovelluksessa. Sovelluksen kaikki data, joka haetaan Http-kutsuilla, tulee siis JSON-stringeinä jotka muutetaan olioiksi. Tässä sovelluksessa käytetään Newtonsoftin JSON.Net sovelluskehystä, joka löytyy ilmaisena Visual Studion NuGet-laajennuksesta.

2.6 NuGet

NuGet on ilmainen open source pakettimanageri Microsoftin kehitysalustalle. Sen avulla käyttäjä voi hakea tarvitsemiaan kirjastoja, ja ladata sekä asentaa niitä vaivattomasti. NuGet on ennalta-asennettu Visual Studio 2012 ja sitä uudemmissa Visual Studion versioissa.

2.7 HTTP POST & GET

HTTP rajapinta mahdollistaa WWW-protokollien hyödyntämisen ohjelmakoodissa. Rajapinta on itsenäinen, eli se ei vaadi tiettyä käyttöliittymää tai koodikieltä jotta sitä voidaan hyödyntää. HTTP rajapintaa tarvitaan tässä työssä datan hakemiseen ja lähettämiseen web-palvelimen välillä.

3 VAATIMUKSET

3.1 Vaatimusmäärittely

Tässä luvussa käydään läpi sovelluksen lähtökohtaiset vaatimukset. Vaatimukset tehtiin kehityssuunnitelman perusteella. Allaolevassa taulukossa listataan sovelluksen tärkeimmät vaatimukset ja niiden tärkeydet.

ID	Vaatimus	Prioriteetti
R1	Sovellus voi suorittaa HTTP-kutsuja web-palvelulle	1
R2	Navigointi sivulta toiselle kosketusnäyttöä pyyhkäisemällä	4
R3	Push-notifikaatioiden hyödyntäminen	2
R4	Miellyttävä käyttöliittymä	3
R5	Mahdollisuus käyttää logiikkakoodia eri alustoilla	1
R6	Käyttäjätietojen tallentaminen muistiin	2
R7	Lokalisaatio suomi ja englanti.	3

Taulukko 1. Vaatimukset

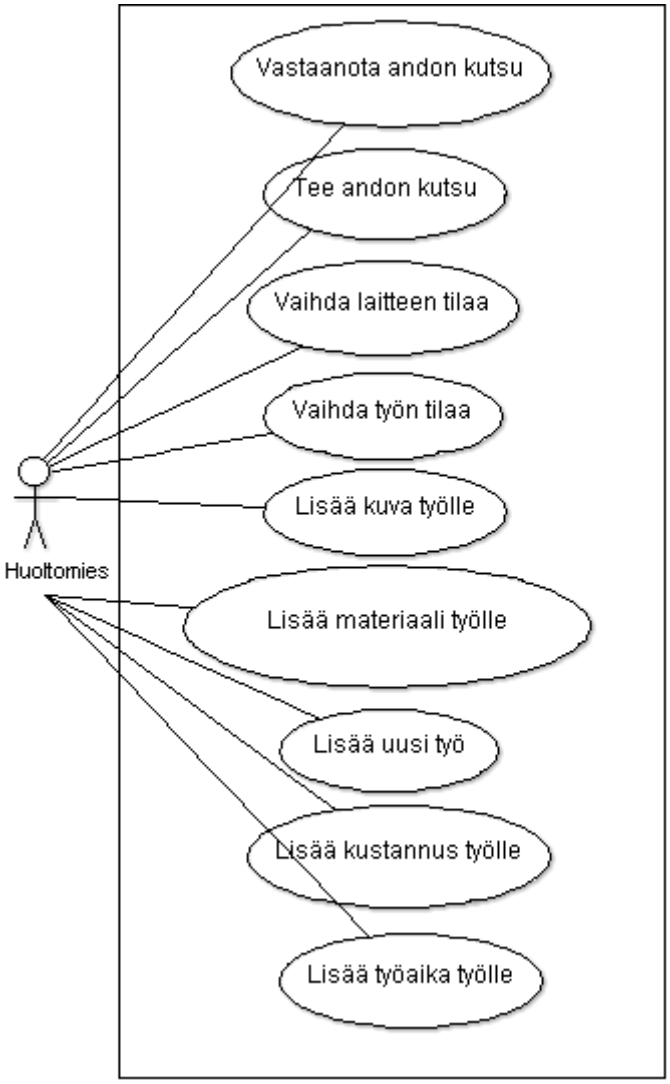
Tarkoituksena on luoda monialustainen englannin- ja suomenkielinen sovellus, jonka toiminta vaatii datan lähettämistä ja hakemista web-palvelusta. Sovellus kehitetään Windowsille ja Androidille. Koodin uudelleenkäyttäminen on erittäin tärkeää, jotta ohjelman porttaus uudelle alustalle saadaan tehtyä järkevässä ajassa. Portattavuus ja Internetin hyödyntäminen ovat työn tärkeimpiä asioita, ja niihin panostetaan eniten.

Kuten taulukosta 1 nähdään, sovelluksen on tarkoitus hyödyntää myös push-notifikaatioviestejä, jolloin käyttäjä saa ilmoituksen viesteistä. Push-notifikaatiot ovat olennainen osa sovellusta, mutta niiden puuttuminen ei tee sovellusta täysin käyttökelvottomaksi.

Taulukosta 1 nähdään myös, että sovelluksen tarkoitus on myös olla helppokäyttöinen ja sulava, jolloin käyttäjäkokemus on miellyttävä ja käyttötuntuma ammattimainen. Tämä tarkoittaa myös sitä, että käyttäjän tiedot tallennetaan muistiin, jolloin käyttäjän ei tarvitse aina sovellusta käynnistäessä kirjautua sisään. Käyttöliittymän miellyttävyys on tärkeää, mutta se ei ole sovelluksen toiminnan kannalta pakollinen seikka. Käyttöliittymää parannellaan, kun sovelluksen korkeamman prioriteetin työt ovat tehty.

3.2 Use-case-diagrammi

Use-case-diagrammia käytetään systeemin vaatimuksien keräämiseen. Nämä vaatimukset ovat enimmäkseen luonnoksen suunnitelua varten.



Kuva 1. Use-case-diagrammi sovelluksen toiminnasta

3.3 HTTP-kutsut ja web-palvelin

Kun sovellus käynnistetään, käyttäjälle näytetään enimmäisenä sisäänkirjautumissivu. Sisäänkirjautumissivulla käyttäjä asettaa oman tunnuksensa

ja salasanansa lisäksi osoitteen, johon sovellus lähettää kirjautumispyynnön. Osoite on web-palvelun osoite, joka vastaanottaa ja lähettää dataa.

Mobiilisovellus lähettää käyttäjän syöttämät tiedot HTTP POST-kutsulla serverille, joka tarkistaa vastaako käyttäjän tiedot tietokannan tietoja. Jos vastaa, niin web-palvelu palauttaa käyttäjälle JSON-merkkijonon, joka sisältää kaikki käyttäjän tiedot. Tästä kutsusta otetaan talteen eväste istunnolle, jota hyödynnetään muissa HTTP-kutsuissa. Tällöin web-palvelu tunnistaa käyttäjän ja ottaa vastaan käyttäjän kutsuja.

3.4 Käyttäjätietojen tallentaminen muistiin

Käyttäjäkokemuksen helpottamiseksi käyttäjän syöttämä käyttäjätunnus, salasana sekä osoite tallennetaan laitteen sisäiseen muistiin. Tällöin käyttäjän ei tarvitse asettaa kirjautumistietoja uudestaan ellei nimenomaan kirjaudu sovelluksesta ulos. Kun käyttäjä kirjautuu ulos, puhelimen muistista poistetaan käyttäjän tunnus ja salasana, mutta osoite pidetään tallessa. Osoitteen voi pyyhkiä muistista sisäänkirjautumissivulta löytyvästä osoitteen poistamispainikkeesta.

3.5 Sovelluksen ydinluokat

Sovelluksen sisältö koostuu kolmesta tekijästä; laite, andon ja työ. Sovellus näyttää käyttäjälle listat näistä luokista, ja antaa käyttäjän suorittaa tehtäviä ja tarkastella näiden lisätietoja.

3.5.1 Equipment

Equipment on luokka, joka sisältää kaiken datan, joka on fyysiselle teollisuuslaitteelle tarpeellista. Ohjelmassa tarkkaillaan laitteen toimintaa ja sen tiloja. Laitteen perustila on normaali. Jos laitteeseen tulee vika, sen tila voidaan muuttaa vialliseksi. Jos laite ei ole täysin viallinen mutta siinä on jotain häiriötä, sen tila voidaan asettaa hälytykseksi. Näiden tilojen manipulointi mahdollistetaan sovelluksen kautta Http-kutsujen avulla, joilla muutetaan laitteen dataa tietokannassa.

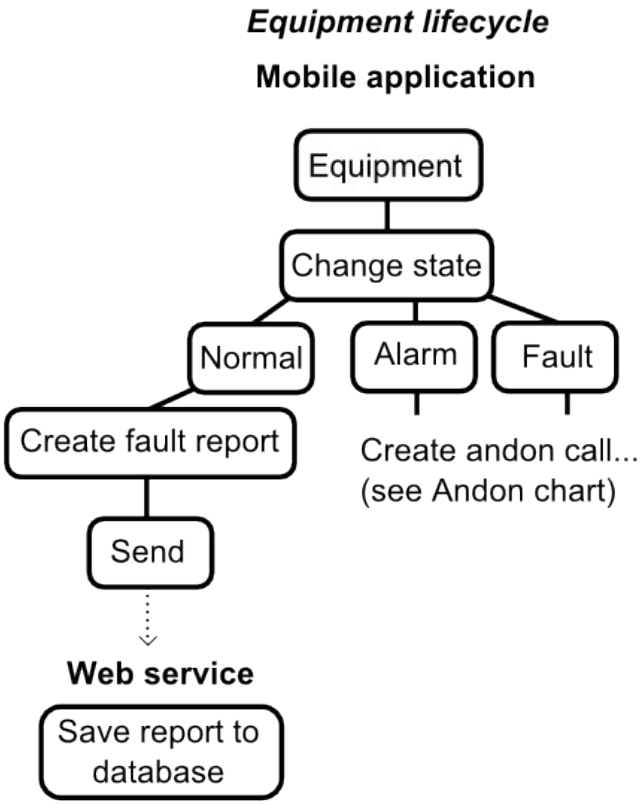
Kun laitteen tila muutetaan hälytyksestä tai viallisesta normaaliksi, pyydetään

käyttäjää täyttämään vikaraportti, josta selviää missä vika tai häiriö oli, mitä seuraksia siitä oli, ja kuinka kauan se kesti sekä kuka teki korjauksen. Vikaraportin tekijä alustetaan käyttäjän nimellä, sekä aika alustetaan vian alkamisen ja raportin tekemisen välisellä ajalla.

Kaikki laitteet, jotka käyttäjän organisaatiolla ovat käytössä, näkyvät sovelluksen pääsivulla laitelistassa. Laitelistassa näytetään jokaiselle laitteelle sen nimi, tila, organisaatio sekä laitteelle suunnattujen töiden määrä. Laitelistasta käyttäjä voi valita laitteen, jota halutaan tarkastella tarkemmin ja siirtyä laitteen lisätietonäkymälle.

3.5.1.1 Laitteen elinkaari

Equipment-luokan tehtävänä on hallita laitteen tilaa ja ilmoittaa siitä web-palvelulle. Tilanmuutoksen yhteydessä käyttäjää pyydetään luomaan vikaraportti tai lähettämään andon-kutsu jollekin käyttäjälle.



Kuva 2. Laitteen perustehtävä sovelluksessa

Kuvassa 2 nähdään, että laitteen aktiivinen toiminta sisältää vain sen tilan muuttamisen. Kun käyttäjä muuttaa laitteen tilan, luodaan joko vikaraportti tai Andon-kutsu.

3.5.2 Job

Job on luokka, joka sisältää kaiken datan, joka on työlle tarpeellista. Työt kohdistetaan laitteille, eli jokaisella työllä on oma laitteensa jolle se on määrätty.

Työn tarkoituksena sovelluksessa on näyttää käyttäjälle, mitä töitä organisaation tai tiimin sisällä on tehtävissä. Sovelluksessa on kaksi työlistaa; ne työt, jotka ovat avoimia eivätkä kenellekään tietylle henkilölle osoitettu, ja ne työt, jotka on osoitettu kirjatuneelle käyttäjälle.

Käyttäjän täytyy pystyä tarkastelemaan työn tietoja, eli käyttäjä voi valita työlistasta työn jolloin sovellus siirtyy näkymälle, jossa käyttäjä näkee työn lisätiedot. Työn lisätietonäkymän sisällä on kolme eri näkymää joiden välillä käyttäjä pystyy liikkumaan näyttöä pyyhkäisemällä. Näiden näkymien sisällä käyttäjä voi lisätä työlle työtunteja, työkustannuksia sekä materiaaleja, ja tiedot lähetetään tietokantaan Http-kutsulla. Työn lisätietosivulla käyttäjä voi myös suorittaa työn, eli merkitä sen suoritetuksi, jolloin työ poistuu avoimien töiden listasta.

Käyttäjä voi myös luoda uuden työn päävalikosta. Painamalla 'Uusi työ'-painiketta käyttäjä siirtyy näkymälle, jossa listataan kaikki organisaation laitteet. Käyttäjä valitsee laitteen, jolle haluaa työn suoritettavaksi. Kun käyttäjä valitsee laitteen, sovellus siirtyy näkymälle, jossa asetetaan työn tiedot, kuten prioriteetti ja kuvaus.

Johdannossa on yleensä työn taustatietoa. Johdannon tarkoitus on kertoa, miksi valittu aihe on tärkeä yleisestä näkökulmasta. Johdannossa esitellään lyhyesti mahdollinen toimeksiantajaorganisaatio ja se, miksi aihe on toimeksiantajalle tärkeä.

3.5.2.1 Työn dokumentit: työaika, kustannukset & materiaalit

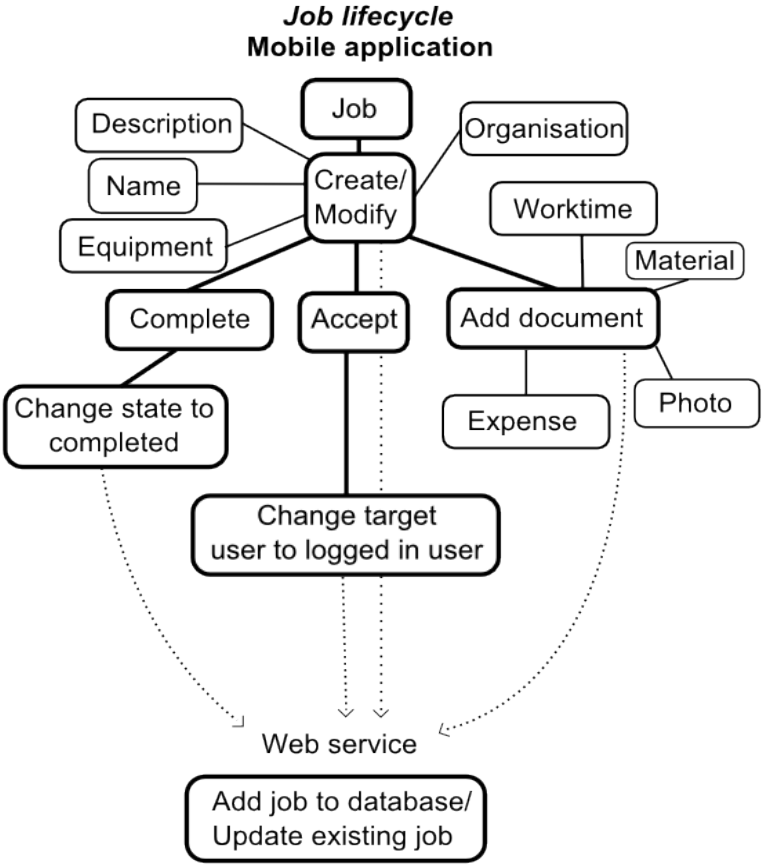
Työn lisätietonäkymässä voidaan lisätä työlle työaikoja, kustannuksia tai materiaaleja. Näille työn lisätiedoille tehdään omat luokkansa, sillä ne sisältävät jonkin verran dataa.

Jokaiselle näistä luodaan näkymä, jossa asetetaan olion parametrit, ja lähetetään ne tietokantaan Http-kutsulla. Kaikki materiaali-, kustannus- ja työaikalisäykset näkyvät työn lisätietosivulla.

3.5.2.2 Työn elinkaari

Job-luokan tehtävänä on hallita laitteen tilaa ja ilmoittaa siitä web-palvelulle. Tilanmuutoksen yhteydessä käyttäjää pyydetään luomaan vikaraportti tai lähettämään andon-kutsu jollekin käyttäjälle.

Kuva 3. Työn perustoiminnot sovelluksessa



Kuvassa 3 nähdään, että työn aktiivinen toiminta on sovelluksessa paljon monipuolisempi kuin laitteen. Työtä luodessa sille annetaan attribuutit. Kun työ on luotu, sille voidaan luoda dokumentteja. Käyttäjä voi hyväksyä työn, jolloin se poistuu avoimet työt-listalta ja siirtyy käyttäjän jono-listaan. Kun työ on suoritettu, sen voi merkitä tehdyksi.

3.5.3 Andon

Andon tarkoittaa hälytystä. Tässä työssäAndon on luokka, joka sisältää kaiken datan, joka on hälytyskutsulle tarpeellista. Andon on sovelluksen kolmas pääluokka. Andon kutsut toimivat vähän kuten työt. Andonit näytetään kahdella listalla; listassa 'avoimet' näkyvät ne andonit, jotka on kohdistettu sisäänkirjautuneelle käyttäjälle tai sen tiimille tai ei kenellekään ja joita kukaan ei ole kuitannut itselleen. Listassa 'jono' näkyvät sisäänkirjautuneen käyttäjän kuittaamat andonit, jotka ovat vielä kesken. Andonin kohdekäyttäjä on se käyttäjä, joka sille on alunperin asetettu tai se käyttäjä, joka lähettää sille vastauksen ensimmäisenä.

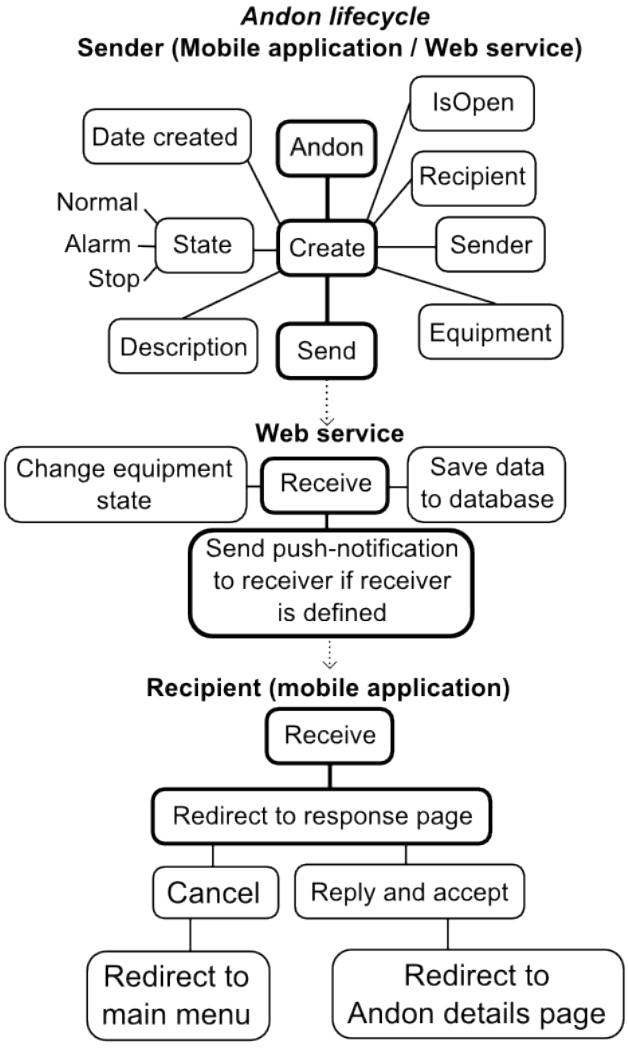
Andon on kesken silloin, kun sen tila ei ole normaali. Andon kohdistetaan tietylle laitteelle, ja se sisältää viestin sekä kohdekäyttäjän tai -tiimin. Andonilla on kolme tilaa, 'normaali', 'hälytys' ja 'vika'. Kun laitteen tila muutetaan esim. 'hälytys'-tilaan, sovellus pyytää tilan muuttajaa lähettämään andon-viestin jollekin käyttäjälle tai tiimille. Tämän andon-viestin tila on myös 'hälytys' eli andon-tila on sama kuin se, mihin tilaan laite muutetaan. Kun viesti on lähetetty, vastaanottaja saa push-notifikaation siitä mikäli push-notifikaatiokanava on auki, ja se ilmestyy käyttäjän 'avoimet' andonlistaan.

Andon-listasta voi valita andonin napautamalla jotain listan andoneista, jolloin sovellus avaa andonin lisätietonäkymän. Andonin lisätietonäkymä koostuu kahdesta sivusta joiden välillä käyttäjä voi navigoida pyyhkäisemällä näyttöä. Ensimmäisellä sivulla 'yleistä' käyttäjä näkee andonin perustietoja, kuten lähettäjän nimen, andonkutsun luomisajan, kohdelaitteen nimen. Toisella sivulla 'lisätietoja' käyttäjä näkee saman laitteen, jolle andon on kohdistettu kaksi viimeisintä andonkutsua. Sen avulla käyttäjä voi katsella, mitä laitteelle on viimeksi tapahtunut ja tehty.sekä lähettäjän viestin. Lisätietosivulla on myös painike, jolla andonin tilan voi muuttaa normaaliksi. Sen painaminen avaa vikaraporttinäkymän, ja vikaraportin lähettäminen lähettää laitteelle uuden andonin, jonka tila on normaali. Laitteen andontila on se tila, joka on laitteelle kohdistetulla uusimmalla andonilla. Lisätietosivulla on myös painike, joka kuittaa

andonin käyttäjälle, joka sitä painaa. Tämä painike näkyy vain silloin, kun andonia ei ole vielä kuitattu. Kun käyttäjä painaa tätä painiketta, andon kuitataan käyttäjälle lähettämällä andonille vastaus. Tällöin andon poistuu 'avoimet'-listasta ja siirtyy käyttäjän 'jono'-listaan.

3.5.3.1 Andonin elinkaari

Andon-luokan tehtävänä on kommunikoida laitteiden tiloista ja niiden ongelmista sovelluksen käyttäjille. Kun laitteen tilaa muutetaan häiriölliseksi, pyydetään käyttäjää lähettämään andon-kutsu jollekin käyttäjälle. Sovellus lähettää push-notifikaation palvelimelle, joka lähettää sen eteenpäin kohdekäyttäjälle.



Kuva 4. Andonin perustoiminto sovelluksessa

Kuvassa 3 nähdään, että Andon luodaan, kun laitteen tila muutetaan Alarmiksi tai Faultiksi. Andonin luomisen yhteydess sille asetetaan kuvan 4 mukaiset attribuutit, ja lähetetään push-notifikaatioserverille. Kun viesti saapuu serverille, se lähettää viestin eteenpäin vastaanottajalle.

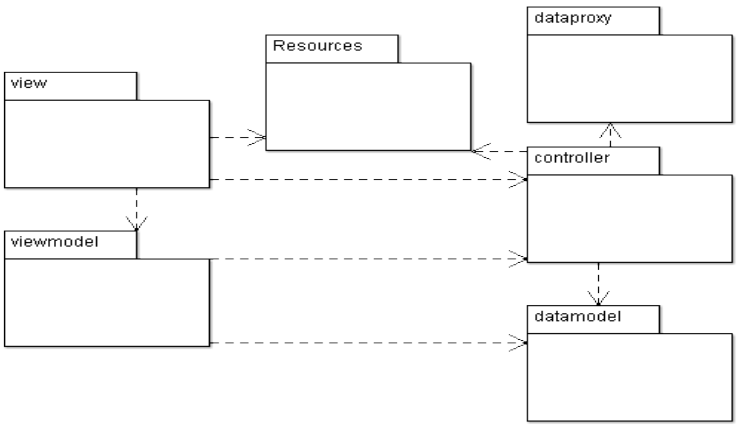
3.6 Push-notifikaatiot

Push-notifikaatioteknologia sallii 'popup' viestien lähettämisen ja vastaanottamisen mobiililla. Push-notifikaatioiden käyttöönotto vaatii eri metodit eri käyttöjärjestelmille, mutta periaate on kaikissa sama. Käyttäjän sovellus avaa notifikaatiokanavan, jolloin sovellus saa kanavan URIn (Uniform Resource Identifier). Tämän URIn avulla sovellus voi tehdä kutsuja oman pilvipalvelun ja käyttäjäsovelluksen välillä.

Tässä sovelluksesa käytetään push-notifikaatioita. Push-notifikaatiokanava avataan ja URI päivitetään aina sisäänkirjautumisen yhteydessä. Sen jälkeen käyttäjä voi vastaanottaa ja lähettää viestejä (tässä sovelluksessa viestejä käytetään Andon-kutsuissa; Käyttäjä saa push-notifikaatioita myös silloin, kun ohjelma ei ole käynnissä mikäli kanavaa ei suljeta.

3.7 Package-diagram

Package-diagram eli pakettidiagrammi visualisoi riippuvaisuudet pakettien välillä.



Kuva 5. Pakettidiagrammi sovelluksesta

Kuvassa 5 nähdään, että käyttäjä hallitsee kontrollereja GUIn (Graphical User Interface) kautta. Kontrollerit hakevat datan, joka visualisoidaan käyttäjälle GUIssa.

4 SUUNNITTELU

4.1 Ulkoasun suunnittelu ja perustelut

Ulkoasun suunnittelussa mietitään, miten saataisiin mahdollisimman käyttäjäystävällinen ja arkkitehtuuriltaan loogisin graafinen käyttöliittymä.

4.1.1 MVVM

Model View Viewmodel (MVVM) arkkitehtuurilla tarkoitetaan sovellusrakennetta, joka koostuu neljästä suuresta osasta; Model, View, Viewmodel ja Controller. Vaikka nimessä ei mainita Controlleria, se on silti suuressa osassa rakennetta. MVVM-malli perustuu siihen, että View näyttää datan, joka siihen on sidottu Viewmodelin kautta. Model sallii yksinkertaisempien näkymien sidonnan ilman Viewmodelia, ja toimii Viewmodelin datamallina. Suurin osa ohjelman logiikasta suoritetaan kontrollerin sisällä, ja viedään näkymälle suoraan kontrollerista. Näkymälle voidaan myös luoda näkymämalli kontrollerin sisällä, jolloin sille voidaan asettaa valmiiksi alustetut arvot.

4.1.1.1 Model

Model eli malli edustaa data-access kerroksen sisältöä. Käytännössä model on luokka, josta luodaan olio tai olioita sovelluksen datan pohjalta. Tässä opinnäytetyössä malleja on esim. Equipment, Job ja Andon. Jokainen näistä on tärkeässä roolissa ohjelman toiminnassa. Jokaiselle näistä tehdään oma luokkansa, ja annetaan tarvittavat muuttujat jotta oikea tieto voidaan näyttää käyttäjälle, ja jotta se olisi helppo muuttaa JSON-merkkijonosta olioksi.

4.1.1.2 View

View eli näkymä on se osa rakennetta, jossa näytetään käyttäjälle graafinen ulkoasu ja data. Näkymä ei yleensä sisällä juuri ollenkaan logiikkaa, ja tässäkin työssä logiikkakoodi kirjoitetaan suurimmalta osalta kontrollereihin, sillä näkymät joudutaan rakentamaan täysin uudestaan Android-versioon. Näkymälle sidotaan näkymämalli, joka päivittää näkymällä näkyvän datan kontrollereiden avustuksella.

4.1.1.3 Viewmodel

Viewmodel eli näkymämalli hallitsee näkymän datan tuomisen näkymälle. Näkymämalli ei ole aina tarpeellinen, sillä näkymä saattaa kyetä näyttämään kaiken tarvitsemansa tiedon sitomalla vain pelkän mallin näkymään, mutta monimutkaisemmissa näkymissä näkymämalli on hyvin tarpeellinen. Pelkkää mallia voidaan käyttää näkymässä sidottuna silloin, kun näkymä näyttää vain yhden mallin tietoja ja dataa. Näkymämalli sallii monimutkaisemman ja monipuolisemman datan näyttämisen näkymällä, esimerkiksi jos halutaan näyttää lista laitteista, näkymälle sidotaan näkymämalli joka sisältää listan joka koostuu laite-malleista.

4.1.1.4 Controller

Controller eli kontrolleri on rakenteen taustalla näkymättömässä roolissa, ja se hoitaa ns. logiikkakoodin eli tässä työssä se suorittaa datan hakemisen ja viemisen sovelluksen ja tietokannan välillä sekä näkymämallien päivittämisen uudella datalla.

4.1.2 XAML-rakenne

Tässä opinnäytetyössä luodaan sovelluksen Windows-version ulkoasu käyttämällä XAML-tekniikkaa. XAML-tiedostot ovat sovelluksen näkymiä. XAML on ulkoasultaan kuin XML, eli sen rakenne on hierarkinen. Jokaisella XAML-tiedostolla on parina Page-luokka. Page-luokasta voidaan C# koodilla kontrolloida näkymän komponenttien ominaisuuksia. Kun Visual Studiolla luodaan XAML-näkymä, se luo näkymälle myös automaattisesti C#-tiedoston. Tärkeä ominaisuus XAML-käytöllä on se, että näkymälle voidaan asettaa näkymän 'Datacontext' eli model tai viewmodel johon ulkoasun datakomponentit sidotaan. Tätä kutsutaan databindaukseksi.

4.1.3 AXML-rakenne

AXML-tekniikka eroaa XAML:sta siten, että siinä ei käytetä databindausta. AXML-tekniikalla luodut näkymät ovat samalla tavalla linkitetty C#-koodiin,

josta näkymän komponenttien ominaisuuksia voidaan hallita. Näitä C# luokkia kutsutaan tilanteen mukaan joko Activityiksi tai Fragmenteiksi. Activityä käytetään silloin, kun tehdään yksittäinen sivu joka sisältää kaiken tarvittavan datan, ja josta mahdollisesti siirrytään toisille sivuille. Activityn ulkoasupohja luodaan AXML-tiedostossa, ja sille luodaan Activity-luokka jossa asetetaan näkymäksi sille luotu AXML-tiedosto.

Fragment on Activityssä luotu luokka, joka toimii kuten sivu. Yksi Activity voi sisältää monta Fragmentiä, joka tarkoittaa sitä, että yhdellä sivulla voi olla useampi 'alisivu'. Tätä käytetään silloin, kun halutaan mahdollistaa näkymästä toiselle navigoinnin ilman, että ladataan täysin uusi näkymä. Esimerkiksi tässä työssä käytetään Fragmenteja, kun halutaan vaihtaa näkymää pyyhkäisemällä mobiilinäyttöä. Pyyhkäiseminen vaihtaa näkymän Fragmentin poistamatta edellistä Fragmentiä muistista. Tällöin käyttäjä voi navigoida sivujen välillä ilman ulkoasun uudelleenlataamista. Fragment-sivujen pohjat luodaan samalla tavalla kuin Activityjen.

4.2 Logiikan ja ulkoasun yhteys

Ulkoasu on vain visuaalinen kuori datalle, jota ohjelmassa halutaan näyttää käyttäjälle. Tässä opinnäytetyössä data haetaan käyttämällä PCL-projektin metodeja. Tämän datan näyttäminen käyttäjälle onnistuu käyttämällä C#-tiedostoja joka on parina AXML-(Android) tai XAML(Windows) -ulkoasutiedostoille.

Ulkoasun C#-tiedostossa luodaan PCL-projektin kontrolleriluokasta olio, jonka avulla voidaan hakea dataa näytölle. Esimerkiksi päävalikossa näytetään käyttäjälle lista laitteista. Tällöin käytetään laitekontrolleria ja suoritetaan sillä verkkometodi, joka hakee käyttäjän organisaation laitteet listaan. Tämän listan objektit linkitetään ulkoasun listakomponenttiin, jolloin se näyttää listan objekteista.

Windows-alustan ominainen mahdollistaa suoran databindauksen ulkoasukomponentteihin, jolloin komponentteihin ei tarvitse manuaalisesti asettaa

dataa, vaan komponentin datasisältö päivittyy automaattisesti, kun dataan tehdään muutoksia. Tämä onnistuu Modelin ja ViewModelin avulla. XAML-tiedostossa määritellään minkä komponenttien sisältö halutaan bindata, C#-tiedostossa näkymän datacontextiksi asetetaan luotu Model tai ViewModel, ja kun tämän viewmodel- tai model-olion dataa muutetaan, ulkoasu päivittyy käyttäjälle. Tämä toimii System.Windows.Controls.NotifyPropertyChanged() metodilla.

4.2.1 Logiikan ulkoistus alustakohtaisesta projektista

Tarkoituksena on luoda Portable Class Library(PCL) -projekti, joka sisältää kaiken sen koodin, mikä ei ole alustasta riippuvainen. Tällöin alustakohtaiset Android- ja Windows Phone -projektit voivat viitata PCL-projektiin, ja käyttää PCL-projektin metodeja ja luokkia.

4.2.1.1 Verkkokutsut

Koska kaikki verkkokutsut palauttavat JSON-merkkijonon, se täytyy muuttaa käyttökelpoiseksi dataksi. Tätä varten luodaan dataparser-luokka, joka sisältää metodit jokaiselle verkkokutsulle. Metodit ottavat parametrikseen merkkijonon, joka deserialisoidaan olioksi tai josta kerätään tietyt tiedot Newtonsoftin Json.NET viitekehyksen avulla.

4.3 Lokalisaatio

Sovelluksen lokalisaatio tehdään ensin Windows-puhelimelle käyttämällä Resources.resx tiedostoa, joka sisältää XML-muodossa lokalisaatiomerkkijonot. Visual Studion resource designerillä on helppo tehdä lokalisointi Windows-puhelimen alustalle. Koska tässä työssä ei käytetä Xamarin.Formsia, ei voida käyttää Resources.resx tiedostoa Android-alustalle. Tällöin Androidille täytyy tehdä oma lokalisointi käyttämällä strings.xml metodia. Merkkijonot kirjoitetaan strings.xml-tiedostoon, jolloin merkkijonoja voi kutsua GetResources() metodilla, joka valitsee merkkijonon laitteen localen mukaan.

5 TOTEUTUS

5.1 Logiikka

Logiikka koostuu metodeista, jotka hakevat, lähettävät tai manipuloivat dataa. Logiikkaan ei kuulu ulkoasun rakentaminen, mutta se tuo ulkoasulle dataa, joka näytetään käyttäjälle graafisessa käyttöliittymässä.

5.1.1 Verkkokutsut

Sovelluksen verkkokutsujen logiikka on yhdessä luokassa nimeltä BackendManager.cs. Tämä luokka sisältää muutaman staattisen metodin peruverkkokutsujen tekemiseen. Verkkokutsut ovat nimetty DoRequest(params) jossa 'params' on kutsun parametrit. Suurin osa verkkokutsuista toimivat samoilla parametrillä, mutta tietyt kutsut vaativat eri parametrejä.

Tässä työssä yleisimmin käytetyn verkkokutsun rakenne on se, että luodaan HttpClient, jolle annetaan URL ja sen parametrit, ja tehdään sen perusteella asynkroninen Http-kutsu. URLin perusparametri on cache-merkkijono, jolle luodaan joka kutsussa uusi uniikki merkkijono välimuistin tyhjentämiseksi. Tällöin kutsun palauttama data on aina täysin tuoretta. Loput URLin parametreistä haetaan käyttäjän syöttämästä KeyValuePair-listasta. KeyValuePair-olio sisältää 'Key'- sekä 'Value' muuttujat. Olion Keytä vastaan oliolla on aina 'Value', eli Key on avainsana jolle annetaan tietty arvo. Tässä työssä sitä käytetään esim. käyttäjätunnuksessa, jossa avainsana on 'username' ja arvo on käyttäjän käyttäjätunnus.

Sisäänkirjautumiselle on oma verkkokutsunsa, joka ottaa parametreiksi URLin ja KeyValuePairin joka sisältää käyttäjätunnuksen ja salasanan, ja suorittaa verkkokutsun HttpClientin GetAsync() metodilla. Myös SendMessagella on oma verkkokutsunsa. Se ottaa parametriksi KeyValuePair-listan ja suorittaa verkkokutsun HttpClientin PostAsync() metodilla. Muut verkkokutsut suoritetaan verkkokutsulla, joka ottaa parametriksi URLin ja KeyValuePair-listan, ja suorittaa verkkokutsun HttpClientin GetAsync() kutsulla.

Näitä perusverkkometodeja kutsutaan kontrollereista sekä BackendCalls-luokasta. BackendCalls-luokka suorittaa kaikki kutsut, jotka eivät palauta dataa. Nämä kutsut lähettävät dataa web-palvelulle, joka päivittää tietokantaa tai suorittaa toimintoja lähetetyn datan perusteella.

```
public static string DoRequest(string _url, string _username, string _password)
{
    using (var handler = new System.Net.Http.HttpClientHandler())
    {
        using (var client = new System.Net.Http.HttpClient(handler, true))
        {
            try
            {
                var pairs = new List<KeyValuePair<string, string>> {
                    new KeyValuePair<string, string>("login", _username),
                    new KeyValuePair<string, string>("password", _password),
                };

                var content = new System.Net.Http.FormUrlEncodedContent(pairs);

                System.Diagnostics.Debug.WriteLine("Logging in...");
                var response = client.PostAsync(_url, content).Result;
                string data = response.Content.ReadAsStringAsync().Result;

                if (response.StatusCode == System.Net.HttpStatusCode.NotFound)
                {
                    System.Diagnostics.Debug.WriteLine("Connection failed");
                    return "404";
                }
```

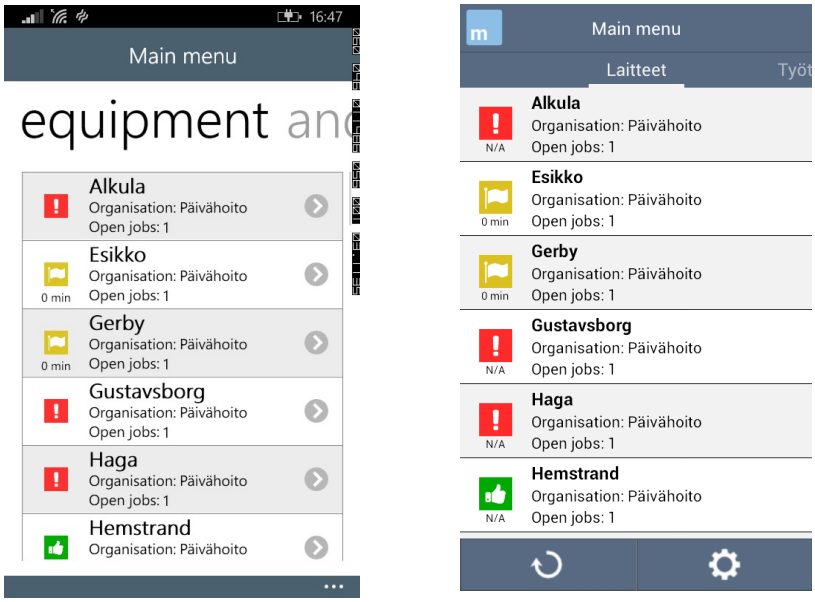
Kuva 6. Osa BackendManager-luokan login-kutsusta

Kuvassa luodaan HttpClientHandler, joka tarvitaan HttpClientin parametriksi. Sen jälkeen luodaan HttpClient, ja muutetaan metodin parametriksi annetut merkkijonot KeyValuePaireiksi, ja muutetaan ne URL-muotoon. Sen jälkeen tehdään PostAsync-kutsu metodin parametriksi annettuun osoitteeseen, jolle annetaan URL-muotoon muutettu data parametriksi. Kutsun vastaus luetaan merkkijonona, joka palautetaan metodin lopussa (ei näy kuvassa).

5.1.2 Equipment

Sovelluksen päävalikossa näytetään käyttäjälle lista laitteista. Päävalikon työsivulla on myös alavalikko, josta voidaan päivittää laitelista tai siirtyä asetukset-näkymään. Laitelista haetaan PCL-projektin ListEquipmentController-luokan metodilla GetEquipmentAsync() joka suorittaa asynkronisen kutsun BackendManagerille. Tälle metodille annetaan parametriksi käyttäjän organisaatio id, ja luodaan KeyValuePair-olio, jolle annetaan avaimeksi 'org_id' ja arvoksi käyttäjän syöttämä parametri. BackendManager tekee tarvittavan verkkokutsun ja

ottaa yhteyden web-palveluun. Web-palvelu voi hakea tietokannasta kaikki käyttäjän parametriksi antaman organisaation laitteet, muuttaa data JSON-merkkijonoksi ja lähettää merkkijonon käyttäjälle vastaukseksi. Tämän jälkeen merkkijono deserialisoidaan olioiksi maintBoxDataParserin metodilla ParseEquipmentData().



Kuva 7. Laitelista Windows-(vas.) ja Android-puhelimilla

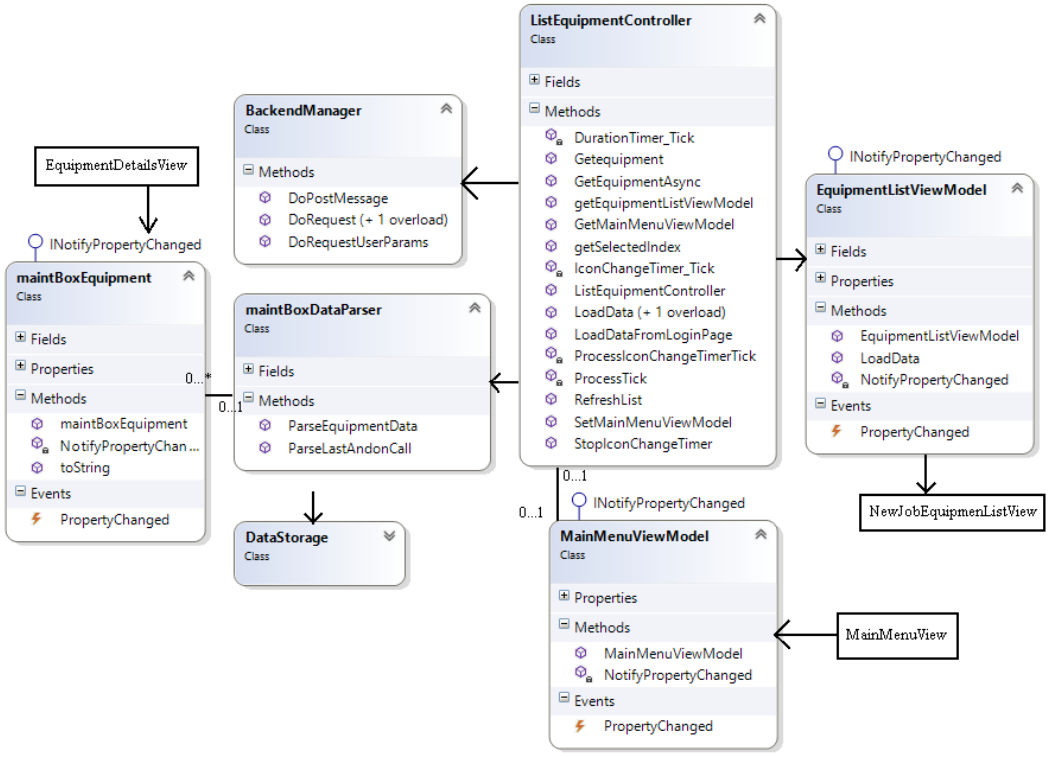
Laitelistassa näytetään käyttäjälle jokaisen laitteen tila, nimi, organisaation nimi sekä laitteen avoimet työt. Kun käyttäjä napauttaa jotain laitetta listasta, suoritetaan navigointikutsu EquipmentDetailsView(Windows) tai ActivityEquipmentDetails(Android) jolloin käyttäjä siirtyy laitteen lisätietonäkymälle. Lisätietonäkymästä hallitaan laitteen tilaa, se sisältää kolme painiketta joilla sen tila voidaan muuttaa Normaliksi, Alarmiksi tai Faultiksi.

Painikkeen painallus suorittaa navigoinnin joko CreateFaultReport-sivulle ('Normal') tai pyytää käyttäjää lähettämään viestin tiimille/henkilölle ('Alarm', 'Fault'). CreateFaultReport-sivulla pyydetään käyttäjää syöttämään vikatietoraportin tiedot, jotka kerätään ja lähetetään web-palveluun asynkronisella verkkokutsulla. Kun taas käyttäjä lähettää viestin, lähetetään web-palveluun

asynkroninen verkkokutsu, joka muuttaa laitteen tilan ja lähettää andon-viestin käyttäjän syöttämälle vastaanottajalle. Kun laitteen tila muutetaan Alarmiksi, laitelistassa näytetään laitteen tilakuvan alapuolella ajastin joka laskee alaspäin 10 minuutista. Aika päivitetään 6 sekunnin välein. Kun 10 minuuttia on kulunut, web-palvelu muuttaa tilan Faultiksi.

Tällä toiminnolla siis tarkkaillaan laitteiden tilaa, ja toimitaan yhteistyössä andon-toiminnon kanssa.

5.1.2.1 Equipment class-diagram



Kuva 8. Equipment-luokan class-diagrammi

Kuvan 8 class-diagrammista nähdään, että ListEquipmentController-luokka hakee datan web-palvelusta BackendManagerin avulla. Web-palvelu palauttaa BackendManagerille JSON-merkkijonon, jonka se palauttaa ListEquipmentControllerille. Merkkijonosta luodaan lista Equipment-olioita

DataParserin avulla. Lista lisätään globaaliin DataStorage-olioon. Tämän jälkeen ListEquipmentController lataa datan MainMenuViewModeliin sekä EquipmentListViewModeliin, joita käytetään näkymien datakonteksteina. Equipment-oliota käytetään EquipmentDetails näkymän datakontekstina.

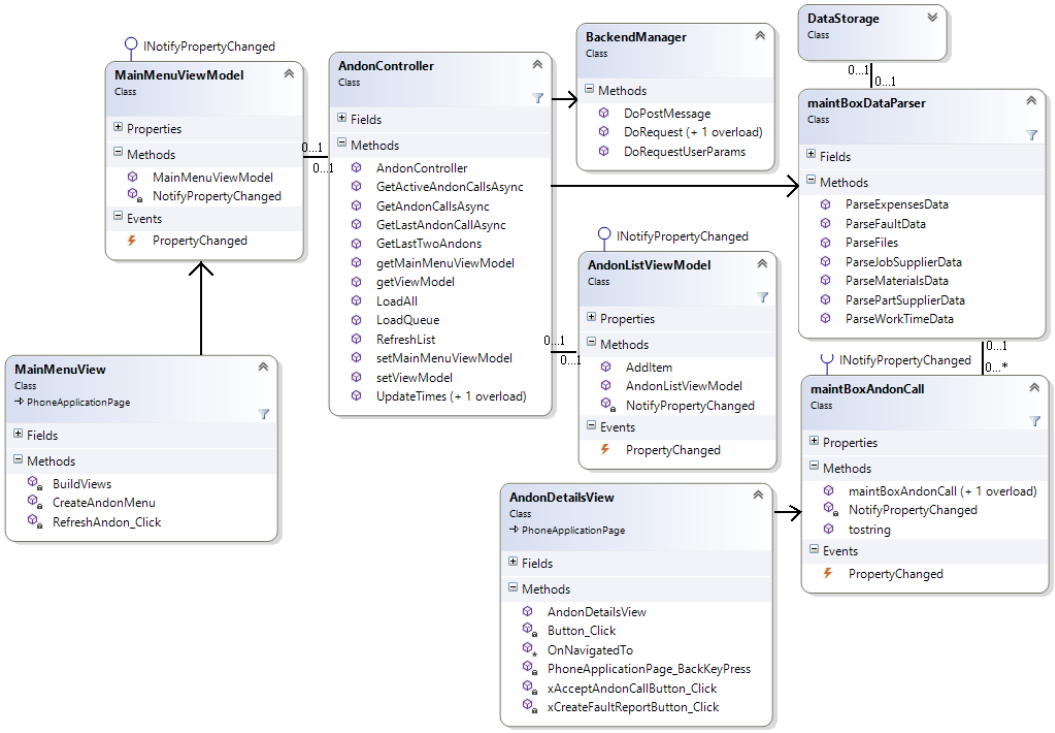
5.1.3 Andon

Sovelluksen päävalikossa näytetään käyttäjälle kaksi listaa andoneista. Toinen lista on 'avoimille' andoneille, eli andoneille joille ei ole määritelty suorittajaa.

Kuva 9. Andonlistan ulkoasu Windows alustalla

Päävalikon työsivulla on myös alavalikko, josta voidaan päivittää andonlistat tai siirtyä asetukset-näkymään. Listaus tapahtuu samalla tavalla kuin laitteille, paitsi andonit jaetaan kahteen ryhmään, avoimet ja ei-avoimet eli jonossa olevat. Käyttäjä voi ottaa andonin suoritettavakseen lähettämällä web-palveluun vastausviestin andonista, tällöin andon ei ole enää avoin ja siirtyy jono-listalle. Tämä tapahtuu painamalla 'Complete'-painiketta AndonDetails-sivulla, jonne käyttäjä siirtyy kun napauttaa listasta andonia.

5.1.3.1 Andon class-diagram



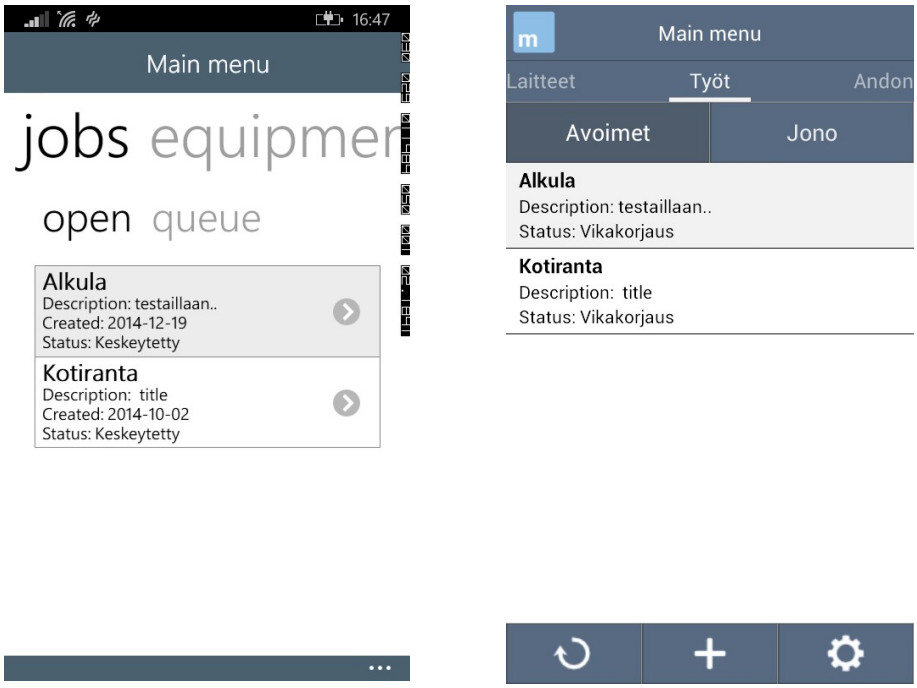
Kuva 10. Andon-luokan class-diagrammi

Kuvasta 10 nähdään, että AndonController hakee andon datan web-palvelusta BackendManagerin avulla, jolloin web-palvelu palauttaa kutsulle vastauksena JSON-merkkijonon. Merkkijono parsetetaan DataParserin avulla, joka muuttaa merkkijonon listaksi AndonCall-olioita. Lista lisätään globaaliin DataStorage-datavarastoon josta andonit voidaan hakea jatkossa. Sen jälkeen AndonController asettaa AndonListViewModel ja MainMenuViewModel datan, jolloin MainMenuView pystyy näyttämään listan andoneista asettamalla luodun viewmodelin datakontekstiksi. Andon-oliota käytetään AndonDetails näkymän datakontekstina.

5.1.4 Job

Sovelluksen päävalikossa näytetään käyttäjälle kaksi listaa töistä. Toisen lista on 'avoimille' töille, eli töille joille ei ole määritelty suorittajaa. Listaus tapahtuu samalla tavalla kuin laitteille, paitsi työt jaetaan andoneiden tapaan kahteen ryhmään, avoimet ja ei-avoimet eli jonossa olevat.

Päävalikon työsivulla on myös alavalikko, josta voidaan päivittää työlistat, tehdä uusi työ tai siirtyä asetukset-näkymään. 'New job'-painiketta painaessa käyttäjä navigoidaan laitelistasivulle, josta käyttäjä valitsee laitteen jolle uusi työ kohdistuu. Laitteen valinnan jälkeen käyttäjä navigoidaan sivulle, jossa käyttäjä asettaa tarkemmat lisätiedot sekä kuvan työlle. Tämän jälkeen 'Save'-painiketta painamalla käyttäjä siirtyy takaisin päävalikkoon, ja uusi työ lähetetään verkkokutsun avulla web-palvelulle, joka lisää sen tietokantaan.



Kuva 11. Työlista Windows(vas.) ja Android puhelimilla

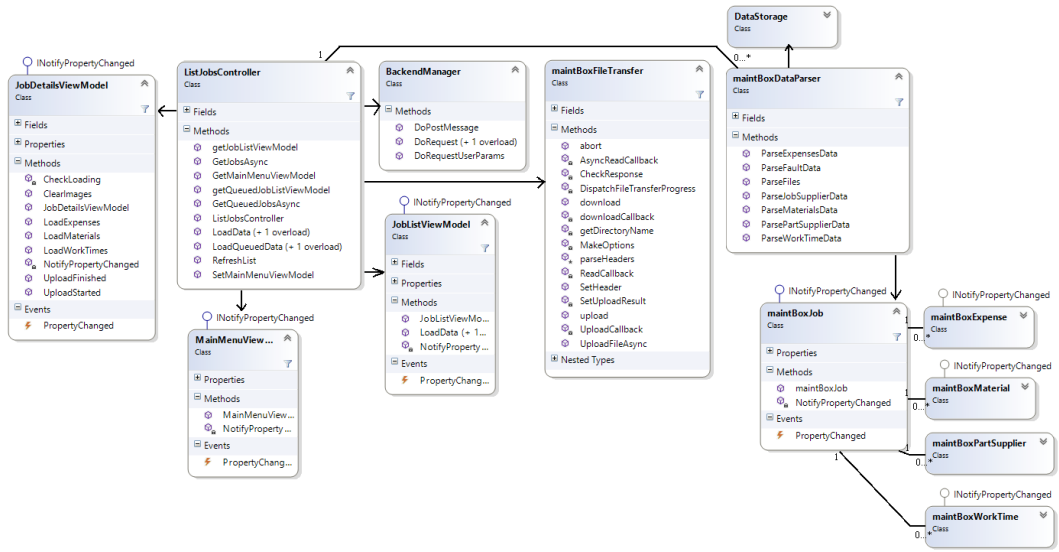
Kun käyttäjä valitsee pääsivun työjonosta työn, sovellus siirtyy työn lisätietosivulle. Lisätietosivu koostuu kolmesta alisivusta joiden välillä käyttäjä voi navigoida pyyhkäisemällä näyttöä. Ensimmäisellä sivulla 'general' on yleiset tiedot työstä sekä lista työajoista ja painike uuden työajan luomiseen. Toisella

sivulla 'additional' on listat työn materiaaleille ja kustannuksille sekä painikkeet niiden lisäämiseen. Kolmannella sivulla on työn valokuvat ja painike uuden kuvan ottamiseen ja kuvan hakemiseen puhelimen kuva-albumista.

Uuden työajan lisäämisnäkymä on kaavake, josta käyttäjä asettaa tiedot työajalle kuten työn tyyppi (ylityö, normaali, viikonloppu), työajan pituus sekä merkinnät. 'Save'-painiketta painettaessa ohjelma suorittaa verkkokutsun, joka lähettää uuden työajan tiedot web-palvelulle joka tallentaa ne tietokantaan, jonka jälkeen käyttäjä navigoidaan takaisin JobDetails sivulle. Uuden materiaalin ja kustannuksen lisääminen toimii samalla tavalla. Valokuvan lisääminen toimii suoraan JobDetails-näkymän Photo-sivulta. Käyttäjän otettua/valittua valokuvan, ohjelma kerää sen tiedot, muuttaa sen biteiksi ja lähettää web-palveluun, joka lisää sen tietokantaan.

Käyttäjän painaessa JobDetails-sivulla olevaa 'Complete'-painiketta, sovellus lähettää web-palvelulle kutsun muuttaa työn tila tehdyksi. Tällöin työ poistuu web-palvelun 'Open'- tai 'Queue'-listalta, eikä työ ole enää työdatassa, joka web-palvelusta haetaan.

5.1.4.1 Job class-diagram



Kuva 12. Job-luokan class-diagrammi

Kuvasta 12 nähdään, että ListJobsController-luokka hakee datan web-palvelusta BackendManagerin avulla. Web-palvelu palauttaa BackendManagerille JSON-merkkijonon, jonka se palauttaa ListJobsControllerille. Merkkijonosta luodaan lista Job-olioita DataParserin avulla. Lista lisätään globaaliin DataStorage-olioon. Tämän jälkeen ListJobsController lataa datan MainMenuViewModeliin sekä JobDetailsViewModeliin, joita käytetään näkymien datakonteksteina.

5.2 Lokalisaatio

Lokalisaatio on valmis Windows-alustalle. Se toimii käyttämällä Resources.resx tiedostoa, joka sisältää merkkijonot, joita halutaan muuttaa käyttökielen mukaan. Resurssitiedostot nimetään kielen mukaan; resurssitiedosto, joka sisältää suomenkieliset merkkijonot on nimeltään Resources.fi-FI.resx, ja englanninkielinen taas Resources.en-US.resx. Tällöin ohjelma osaa automaattisesti hakea oikeat merkkijonot laitteen tai sovelluksen localen perusteella.

Name	Value
AcceptButton	Accept
AddButton	Add
AddExpensesTitle	Add expenses
additionalButton	additional
additionalButtonAndroid	Additional
AddMaterialsTitle	Add materials
Address	Address:
AddWorktimeTitle	Add worktime
AlarmButton	Alarm
AlbumButton	Album

Kuva 13. Resource Designer en-US

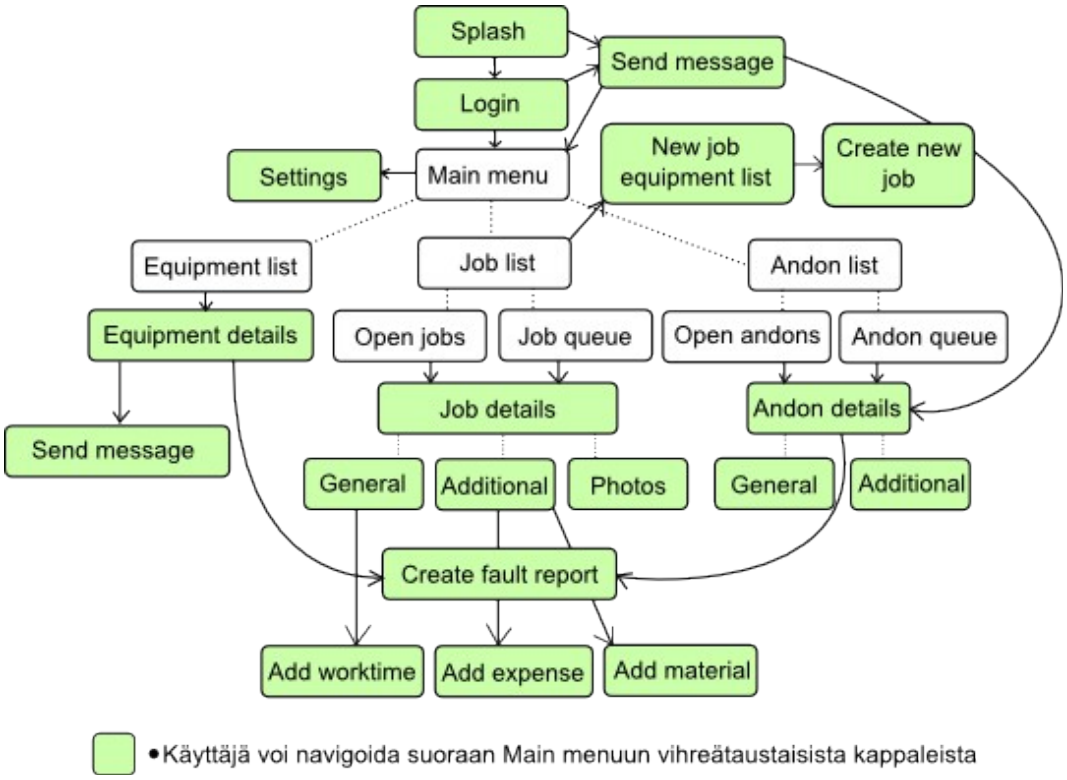
Name	Value
AcceptButton1	Hyväksy
AddButton1	Lisää
AddExpensesTitle1	Lisää kulu
additionalButton1	lisätietoja
additionalButtonAndroid	Lisätietoja
AddMaterialsTitle1	Lisää materiaali
Address1	Osoite:
AddWorktimeTitle1	Lisää työaika
AlarmButton1	Hälytys
AlbumButton1	Albumi

Kuva 14. Resource Designer fi-FI

Kuvissa 13 ja 14 nähdään Visual Studio Resource Designerin helppokäyttöisyys. Designerin vasemmalla puolella asetetaan tunnusmerkkijono, joka saa vasemmallapuolella olevan arvon. Resurssien valinta toimii tiedostonimen perusteella. Sovelluksessa määritellään käytetty lokaali, ja sovellus käyttää automaattisesti oikean tiedoston merkkijonoja.

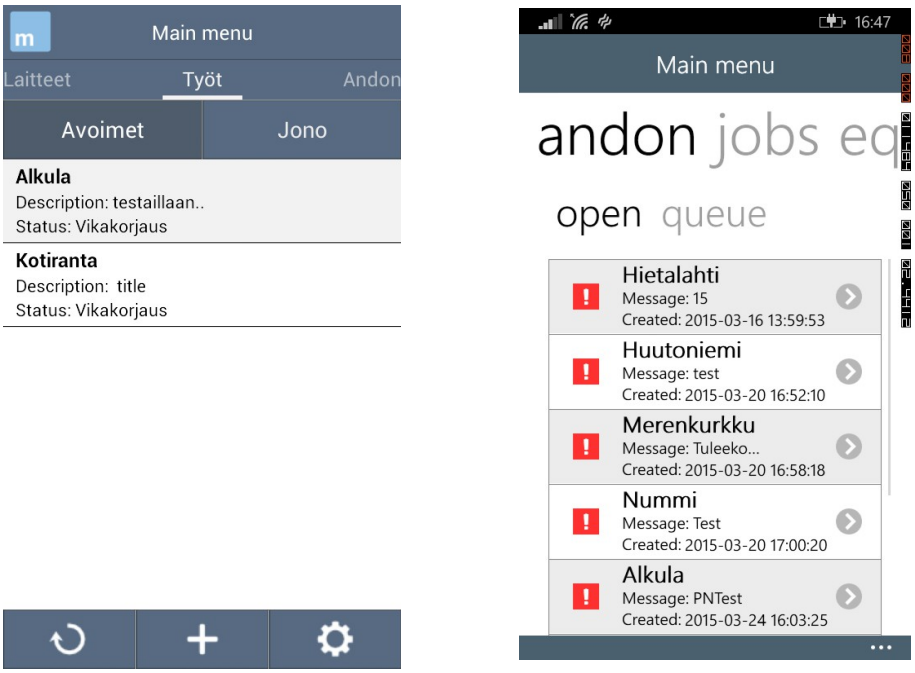
5.3 Ulkoasu

Ulkoasu on rakenteeltaan sellainen, että sovelluksen päävalikko toimii hubina josta pääsee sovelluksen muihiin osiin vaivattomasti. Kaikille sivuille (paitsi Login ja MainMenu) on tehty painike, joka navigoi käyttäjän Main Menu sivulle.



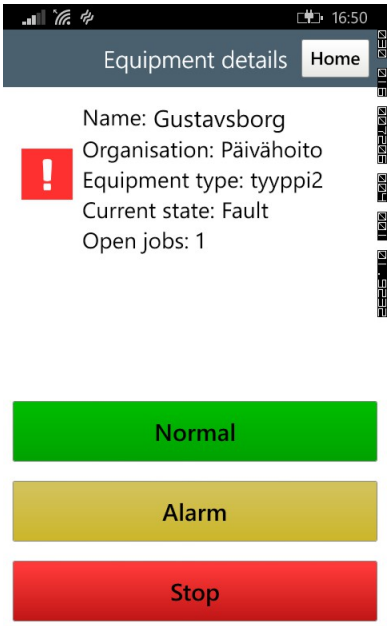
Kuva 15. Näkymät

Ulkoasun tärkeimpiä tekijöitä ovat sen helppokäyttöisyys, sulavuus ja selkeys. Ulkoasu on kokenut muutoksia projektin aikana, ja uusimman version päävalikossa käyttäjä pystyy siirtymään kolmen sivun välillä sormen pyyhkäisyllä ilman latausaikoja.



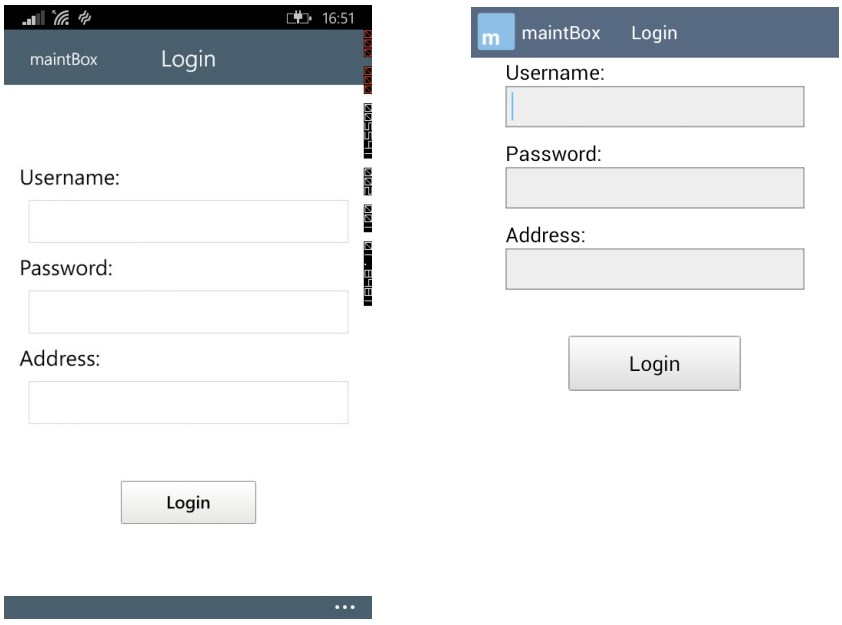
Kuva 16. Päävalikko Windows(vas.) ja Android alustoilla

Ulkoasun värimaailma on tärkeää ottaa huomioon graafista käyttöliittymää luodessa. Esimerkiksi laitteiden tila-ikonin väri ja laitteen lisätietosivulla olevien tilanmuutospainikkeiden värit ovat samaa sävyä.



Kuva 17. Buttonien ja ikonin värisävyn yhtenäisyys

Ulkoasu pyrittiin pitämään suunnilleen samanlaisena eri alustoilla. Android-version(vas) värit ovat täsmälleen samat, mutta alla olevassa kuvassa 18 puhelimen kirkkaus- ja väriasetukset saavat sen näyttämään erilaiselta.



Kuva 18. Login näkymä Windows(vas.) ja Android alustoilla

6 TESTAUS

Sovelluksen testaus suoritettiin pienimuotoisena yrityksen sisäisenä testailuna sovelluksen kehityksen yhteydessä, ja sovellusta testattiin myös hieman käytännössä. Web-palvelun puolella luotiin eri tietokantoja testi- ja tuotantoversioita varten. Sovelluksen kehityksessä käytettiin eri testitietokantoja, jotta nähtiin, miten sovellus toteutti eri yritysten vaatimukset. Eri tietokantoihin otettiin yhteys asettamalla sovelluksen kirjautumisen yhteydessä osoite halutulle tietokannalle.

Sovelluksen testaus yrityksen sisällä koostui omakohtaisesta testailusta, ja satunnaisesta sovelluksen kehityksen valvojan testailusta. Aina, kun sain osan sovellusta valmiiksi, yritin rikkoa sen kaikilla mahdollisilla keinoilla, kuten internet-yhteyden katkaisemisella, painikkeiden nopealla jatkuvalla painamisella ja kenttien tyhjiksi jättämisellä. Tämän avulla löysin suurimmat bugit ja osasin korjata ne. Myös työn valvoja kokeneena ohjelmistokehittäjänä osasi paikantaa ja aiheuttaa paljon bugeja satunnaisilla testailuillaan. Bugin esiintymisen ja korjaamisen jälkeen yritin aiheuttaa bugin uudelleen, ja yritin varmistaa, ettei korjauksesta syntynyt uusia bugeja. Näin saimme yrityksen sisäisen testailun avulla korjattua mahdollisimman paljon bugeja.

Kun sovelluksen koettiin olevan valmis tuotantokäyttöön, se otettiin testauskäyttöön asiakasyritysten testipalvelimille. Näiden testailujen seurauksena löydettiin uusia bugeja. Jotkin näistä bugeista oli vaikea paikantaa, sillä sovellus saattoi toimia hyvin itsellä, mutta asiakasyrityksen käyttäjällä saattoi esiintyä sovelluksen kanssa jokin ongelma. Ongelmista piti ensin selvittää, olivatko ne web-palvelussa vai mobiilisovelluksessa. Tutkin itse mobiilisovelluksesta ongelmaan liittyvät metodit ja luokat, kun taas web-palvelun kehittäjä tutki web-palvelun osat, tällöin saimme kaikki mahdolliset tekijät tarkastettua.

Testausta jatketaan niin pitkään, kunnes sovellus on täysin stabiili, ja aina uusien päivitysten yhteydessä sovellusta läpikotaisin, jottei jokin huomaamaton uusi bugi aiheuta ongelmia sovelluksen käytössä käytännössä.

7 YHTEENVETO

Tämän opinnäytetyön vaatimukset onnistuttiin täyttämään hyvin. Sovellus toimii hyvin Windows puhelimella, ja Android puhelimen versio on suurimmalta osin tehty. Lukuunottamatta pieniä bugeja, jotka esiintyivät opinnäytetyön lopulla, onnistuin työssä hyvin. Suurimmat ajankuluttajat tässä työssä olivat HTTP-kutsujen toimivuus asynkronisina, Windows- ja Android ulkoasujen luomiseen. Sovellus oli niin suuri, että paljon ohjelmakoodista on hieman sekavaa ja moni luokka sisältää enemmän metodeja ja attribuutteja kuin haluaisin. Tulevaisuudessa mietitään enemmän ohjelmakoodin rakennetta ja luettavuutta.

Työn alussa mobiilisovelluskehityksestä ei ollut aiempaa kokemusta, eli jouduttiin perehtymään siihen ja etsimään tutoriaaleja ja kysymään apua aina, kun törmättiin uuteen asiaan. Työtä tehdessä opittiin kuitenkin nopeasti, ja onnistuttiin hyvin asiallisen ja toimivan sovelluksen luomisessa.

Koska opinnäytetyö on tärkeähkö projekti toimeksiantajayritykselle, ohjelmakoodia pyritään kommentoimaan ja selkeyttämään mahdollisimman hyvin. Opinnäytetyön aikana opittiin todella paljon monialustaisesta kehityksestä, Xamarinista sekä mobiilisovelluskehityksestä yleisesti.

Mobiilisovelluksen kehitys ei pääty tähän, sillä sovelluksen web-versio kehittyy koko ajan, ja mobiilisovellusta päivitetään aina tarvittaessa. Sovellukselle otetaan myös palautetta sen käyttäjiltä, jolloin saatetaan kehittää muutoksia sovellukseen.

LÄHTEET

Burnette, E. 2010. Hello, Android : introducing Google's mobile development platform. Raleigh. Pragmatic Bookshelf.

Iversen, J. 2014. Learning mobile app development : a hands-on guide to building apps with iOS and Android. Harlow. Addison-Wesley.

Whitechapel A. & McKenna S. 2013. Windows Phone 8 Development Internals. Microsoft.

Xamarin 2015. Cross-Platform guides. Viitattu 1.4.2015 http://developer.xamarin.com/guides/

Google 2015. Introduction to Android. Viitattu 5.5.2015 http://developer.android.com/guide/index.html

Tutorialspoint 2014. UML Use Case Diagram Viitattu 28.5.2015 http://www.tutorialspoint.com/uml/uml_use_case_diagram.htm

Microsoft 2015. MSDN class library. Viitattu 19.1.2015 https://msdn.microsoft.com/en-us/library/d11h6832%28v=vs.71%29.aspx

IBM 2015. UML basics: The class diagram Viitattu 31.5.2015 http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/