OULUN AMMATTIKORKEAKOULU

Oskari Pulkkinen

ANDROID-SOVELLUS PYSÄKKIKATOSTEN LAADUNVARMISTUKSEN JOUKKOISTAMISEEN

ANDROID-SOVELLUS PYSÄKKIKATOSTEN LAADUNVARMISTUKSEN JOUKKOISTAMISEEN

Oskari Pulkkinen
Opinnäytetyö
Kevät 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä(t): Oskari Pulkkinen
Opinnäytetyön nimi: Android-sovellus pysäkkikatosten laadunvarmistuksen joukkoistamiseen
Työn ohjaaja: Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: Kevät 2018
Sivumäärä: 41

Opinnäytetyön aiheena oli luoda palvelin- ja mobiilisovellus, jonka avulla käyttäjät voivat luoda havaintoja pysäkkikatosten kunnossapitopuutteista. Lopputuotteena olisi Android-sovellus, joka julkaistaisiin Googlen Play-kaupassa. Sovelluksen avulla voitaisiin kirjata paikkaan sidottuja havaintoja, joihin käyttäjä voisi myös lisätä kuvatiedostoja. Rajatulle joukolle tarjottaisiin myös karttapohjainen näkymä tehtyjen kirjauksien tarkastelua varten. Työ liitettäisiin mahdollisesti osaksi Rovaniemen lupauspohjaista alueurakkaa.

Työn tavoitteena oli suunnitella, toteuttaa ja testata toimiva järjestelmä tilaajan asettamaan määräaikaan mennessä sekä vastata heidän asettamiinsa vaatimuksiin tuotteen osalta. Laajemmin tavoitteena olisi myös selvittää joukkoistetun tiedon keräämisen sekä jakamisen toteutustapoja ja toimivuutta.

Mobiilisovelluksen kehitystyö suoritettiin käyttämällä alustariippumatonta Microsoftin omistamaa Xamarin.Forms-kehitysympäristöä, koska sovellus halutaan mahdollisesti julkaista tulevaisuudessa myös iOS-käyttöjärjestelmälle.

Palvelintoteutuksen alustaratkaisuksi valikoitui Microsoftin tarjoama Azure-pilvipalvelu. Microsoftin ASP.NET Core-frameworkia käyttämällä luotiin Azure Web App Service, joka huolehtii sovelluskokonaisuuden tietoliikenteestä ja autentikoinnista. Tietokantapalveluna toteutuksessa käytettiin Azure CosmosDB:tä. Käyttäjänhallinta ja autentikointi toteutettiin käyttämällä Azure Active Directory B2C:tä sekä Microsoft Authentication Libraryä.

Projektin lopputuotteena saatiin toimiva ja helppokäyttöinen mobiilisovellus sekä palvelintoteutus, jotka täyttivät tilaajan työlle asettamat vaatimukset.

Asiasanat: Xamarin, Azure, alustariippumattomuus, joukkoistaminen, mobiilisovellukset, Android

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software Development

Author(s): Oskari Pulkkinen
Title of thesis: Android application for crowdsourcing the quality assurance of bus stop shelters
Supervisor(s): Pertti Heikkilä
Term and year when the thesis was submitted: Spring 2018
Number of pages: 41

The subject of the thesis was to create a mobile application for crowdsourcing the quality assurance of bus stop shelters. The end project would be an Android application, which would be published to Google Play Store. Application would allow users to make coordinate-bound observations with images. A limited group of users could access a view, which could be used to examine the crowdsourced observations on a map.

The goal was to plan, build and test a working system and to answer to requirements set by the client before deadline. The broader goal was to research how gathering and sharing crowdsourced data could be executed and how it would work.

The development of the mobile application was performed by using Microsoft's cross-platform mobile framework Xamarin.Forms, which would make the possible future porting of the software to iOS-platform easier.

The server implementation was built on Microsoft Azure Cloud services. The Azure Web App Service, which handles the system's data traffic and authentication, was developed by using ASP.NET Core-framework. The database service chosen for the project was Azure CosmosDB. User management and authentication were implemented by using Azure Active Directory B2C and Microsoft Authentication Library.

The end product of this project was a working and easy-to-use mobile application and server implementation, which fulfilled the requirements set by the client.

Keywords: Xamarin, Azure, cross-platform, crowdsourcing, mobile application, Android

SISÄLLYS

1 JOHDANTO 6
2 TAUSTATIETOA 7
2.1 Työn kuvaus 7
2.2 Tavoitteet ja vaatimukset 7
3 SUUNNITTELU 9
3.1 Käytettävien menetelmien ja teknologioiden arviointi 9
3.1.1 Palvelintoteutus 9
3.1.2 Mobiilisovellus 10
3.2 Tietorakenteiden suunnittelu 12
3.3 Mobiilisovelluksen käyttöliittymän suunnittelu 12
4 KÄYTETTÄVÄT TEKNISET MENETELMÄT JA OHJELMISTOT 14
4.1 Xamarin Forms -kehitysalusta 14
4.2 ASP.NET Core 14
4.3 Microsoft Azure-ympäristö 15
4.3.1 Azure CosmosDB 15
4.3.2 Azure Web App Service 15
4.3.3 Azure Active Directory B2C 16
5 TOTEUTUS 17
5.1 Palvelimen toteutus 17
5.2 Mobiilisovelluksen toteutus 21
5.2.1 Navigoinnin implementointi sovellukseen 22
5.2.2 Sovelluksen käyttöliittymän luominen 23
5.2.3 Toiminnollisuuksien lisääminen ja laajennuksien käyttöönotto 28
5.2.4 Karttakirjaston valitseminen ja käyttöönotto 29
5.2.5 Viranomaisnäkymän toteuttaminen 30
5.2.6 Kehitystyössä vastaan tulleet ongelmat ja testaus 35
6 YHTEENVETO 38
LÄHTEET 40

1 JOHDANTO

Opinnäytetyön teko aloitettiin joulukuussa 2017. Opinnäytetyö tehtiin Tietomekka Oy:lle, jolta opinnäytetyön aihetta vastaavan toteutuksen tilasi Lapin ELY-keskus.

Ennen opinnäytetyön tekemistä suoritin kaikki aikaisemmat yrityslähtöiset projektit Tietomekalla. Projektien välissä työskentelin yrityksessä ohjelmistosuunnitteluharjoittelijana. Tietomekka on vuonna 1988 perustettu oululainen ohjelmistoja ja tietopalveluja sähköiseen toiminnanohjaukseen tarjoava yritys. Yrityksen yhteistyökumppaneita ovat muun muassa ovat infran kunnossapitoon osallistuvat viranomaiset, urakoitsijat ja konsultit.

Opinnäytetyön aiheena oli luoda palvelin- ja mobiilisovellus, jonka avulla käyttäjät voivat luoda havaintoja pysäkkikatosten kunnossapitopuutteista. Lopputuotteena olisi Android-sovellus, joka julkaistaisiin Googlen Play-kaupassa. Sovelluksen avulla voitaisiin kirjata paikkaan sidottuja havaintoja, joihin käyttäjä voisi myös lisätä kuvatiedostoja. Rajatulle joukolle tarjottaisiin myös karttapohjainen näkymä tehtyjen kirjauksien tarkastelua varten. Työ liitettäisiin mahdollisesti osaksi Rovaniemen lupauspohjaista alueurakkaa.

Työn tavoitteena oli suunnitella, toteuttaa ja testata toimiva järjestelmä tilaajan asettamaan määräaikaan mennessä sekä vastata heidän asettamiinsa vaatimuksiin tuotteen osalta. Laajemmin tavoitteena olisi myös selvittää joukkoistetun tiedon keräämisen sekä jakamisen toteutustapoja ja toimivuutta.

2 TAUSTATIETOA

2.1 Työn kuvaus

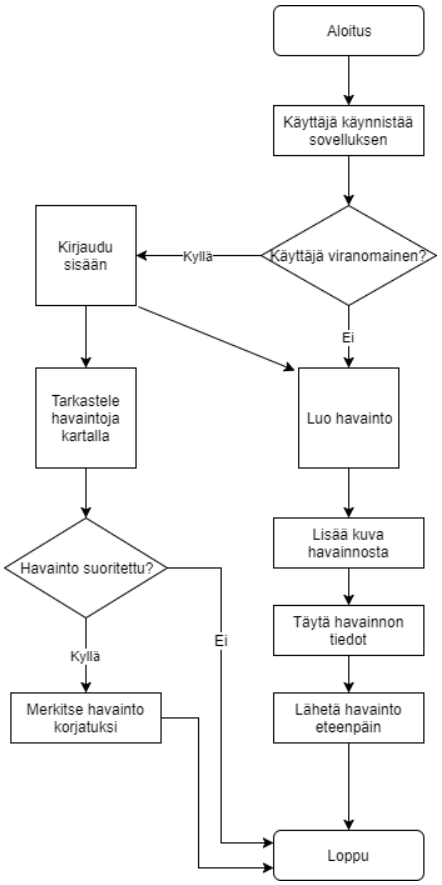
ELY-keskukselle tulee pysäkkikatosten osalta palautetta niiden rikkomisesta ja talvikunnossapidon puutteista. Pysäkkikatosten kunnossapito ja laadunvarmistus haluttaisiin saada hallintaan nykyistä paremmin. Tieto katosten puutteista tulee yleensä myöhään, mikä pitkittää niiden korjaamista.

Pysäkkikatosten laadunvarmistus halutaan joukkoistaa asiakastyytyväisyyden parantamiseksi. Samalla koestetaan, miten joukkoistaminen soveltuu tienhoidon tarpeisiin. Työn avulla voitaisiin tarjota parempaa palvelua tienkäyttäjille sekä lisätä asiakastyytyväisyyttä ja ELY-keskuksen positiivista viestinnällistä imagoa.

Opinnäytetyö liitetään mahdollisesti osaksi Rovaniemen lupauspohjaista alueurakkaa. Toteutus liittyisi kolmeen lupaukseen: asiakastyytyväisyys, digitaalisen toimintamallin kehittäminen sekä tilaaja-tuottaja-toimintamallin kehittäminen.

2.2 Tavoitteet ja vaatimukset

Tavoitteena on luoda sovellus, jolla kuka tahansa sovelluksen ladannut käyttäjä voi tehdä havaintoja pysäkkikatosten puutteellisesta kunnossapidon tasosta. Laajempana tavoitteena on selvittää joukkoistetun tiedon keräämisen ja jakamisen toteutustapoja sekä toimivuutta. Vaatimusten perusteella loin sovelluksen toimintakaavion (kuva 1), joka toimii apuna sovellusta kehittäessä.



KUVA 1. Suunnitelma sovelluksen toiminnasta

Sovelluksella tulee pystyä kirjaamaan paikkaan sidottuja havaintoja. Havaintoihin tulee pystyä liittämään kuvatiedostoja. Rajatulle joukolle (tässä tapauksessa viranomaisille) tulee tarjota karttapohjainen näkymä tehtyjen kirjauksien tarkastelua varten. Lopullinen Android-laitteille luotu mobiilisovellus julkaistaan mahdollisesti Googlen Play-kaupassa.

3 SUUNNITTELU

Työn toteutus käynnistettiin aloittamalla siinä käytettävien menetelmien sekä teknologioiden arviointi ja soveltuvuus projektin vaatimuksiin.

3.1 Käytettävien menetelmien ja teknologioiden arviointi

3.1.1 Palvelintoteutus

Sovelluksen palvelinpuolen toteuttamiseksi valikoitui jo ennen projektin käynnistämistä Microsoftin ASP.NET Core. Minulla oli ennestään useamman vuoden kokemus ASP.NET Frameworkista. ASP.NET Framework ja ASP.NET Core ovat hyvin samankaltaisia: molemmilla alustoilla käytetään samaa ohjelmointikieltä (C#) ja ne jakavat suuren määrän toiminnollisuuksia keskenään. ASP.NET Core eroaa ASP.NET Frameworkista siinä, että sillä ei voida luoda työpöytäsovelluksia. Core-sovelluksia voidaan ajaa muuallakin kuin pelkästään Windows-ympäristöissä. Se on myös kevyempi ASP.NET Frameworkiin verrattuna, joskin se näkyy joidenkin ominaisuuksien puuttumisena. (1.)

ASP.NET Coreen pohjautuva palvelinratkaisu viedään Microsoftin Azure-pilvipalveluun, jossa sitä ajetaan Web App Servicenä. Web App Service on palvelu web-applikaatioiden eli web-sovellusten ja REST API-rajapintatoteutusten hostaamista varten. Se huolehtii itse tietoturvasta, päivityksistä, skaalautumisesta sekä muista hallinnollisista toimenpiteistä. Vaikka Web Appeja suoritetaan virtuaalikoneilla, itse kehittäjällä ei ole sinne suoraa pääsyä. (2.)

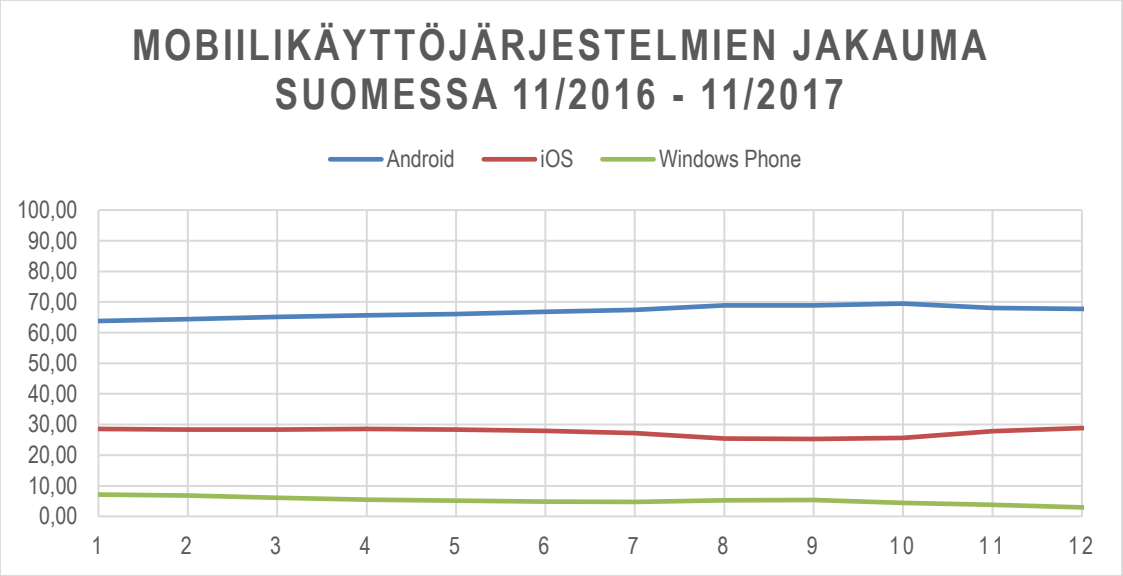
Projektin tietokantaratkaisuksi valittiin Azure CosmosDB, koska se toimii saumattomasti edellä mainittujen teknologiaratkaisuiden kanssa sekä se skaalautuu vaivattomasti. CosmosDB on NoSQL-pohjainen skeematon tietokantatoteutus, jossa säilöttävät tiedot ovat JSON-muotoisia. Täten se tukee myös GeoJSON-tietotyyppiä, mikä on hyvin käytännöllistä projektin luonnetta ajatellen, koska lähes kaikki projektin tietokannassa säilöttävä tieto on paikkatietopohjaista. Sen

pystytys ja käyttäminen on myös hyvin helppoa verrattuna muihin kaupallisiin tietokantapalvelimiin.

Havaintojen mukana tulevat liitteet, eli kuvat, tallennetaan Azure Blob Storageen. CosmosDB:n dokumenttitaltioihin olisi periaatteessa mahdollista tallettaa myös binääridataa, mutta yhden dokumentin maksimikoko on 64 kilobittiä, jolloin suurien kuvien talletus hyvälaatuisena ei olisi mahdollista. Sen sijaan dokumentin JSON-dataan kirjataan Blob Storagessa olevan liitteen tunniste, jonka perusteella haluttu havaintotieto haetaan.

3.1.2 Mobiilisovellus

Aikaisemmin tekemäni Android-sovellukset on tehty Javalla Android Studiota käyttäen. Se on tyypillisin tapa luoda Android-sovelluksia, joskin Kotlin-kielen käyttö sovelluskehitykseen Javan sijaan on yleistynyt viime aikoina. Projektin lopputuotteena oleva sovellus tulee kuitenkin julkiseen käyttöön. Vaikka Android onkin yleisin mobiilikäyttöjärjestelmä Suomessa (kuva 2), muista käyttöjärjestelmistä etenkin Applen iOS tulisi ottaa huomioon.



KUVA 2. Mobiilikäyttöjärjestelmien jakauma Suomessa aikavälillä 11/2016 - 11/2017 (3.)

Kehitystyön helpottamiseksi aloin tekemään tutkimustyötä järjestelmäriippumattomista kehitysalustoista, joilla olisi tuki ainakin iOS:lle ja Androidille. StackOverflow'n julkaiseman tilastotutkimuksen mukaan (4) suosituimpia kehitysalustoja ovat React Native, Xamarin, Cordova ja Ionic. Cordova ja Ionic tukevat laitteita hyvin laajalla skaalalla, mutta alustojen tuottamat sovellukset ovat käytännössä web-sivuja, joita esitetään natiivissa sovelluksessa web-ikkunan avulla (esim. WebView tai Chromium). Sovelluksen tulisi olla mahdollisimman sulava ja nopea, myös karttanäkymässä, jonka vuoksi edellä mainitut alustat hylättiin.

Mielenkiintoisena tulokkaana mobiilikehityksen alustamarkkinoille on saapunut hiljattain Googlen kehittämä ja ylläpitämä Flutter, jolla voidaan luoda sovelluksia samanaikaisesti Android- ja iOS-alustoille. Flutter ei käytä käyttöjärjestelmien tarjoamia natiiveja UI-komponentteja lainkaan, vaan käyttöliittymä piirretään kokonaisuudessaan Skia Canvaksen päälle. Skia Canvas on alustariippumaton kirjasto piirtämistä varten, joka pyrkii hyödyntämään jokaisen alustan rajapintakutsuja mahdollisimman tehokkaasti. Tällä tavoin käyttöliittymä näyttää täysin samalta eri alustoilla. Flutter on kuitenkin vielä kehityksensä alkutaipaleilla (alfa-vaiheessa) ja osittain keskeneräinen, joten se jätettiin pois harkinnoista. (5.)

Jäljelle jäivät React Native ja Xamarin, jotka molemmat kykenevät tuottamaan natiiveja sovelluksia eri alustoille. Näistä voittajaksi valikoitui Xamarin. Ohjelmointityö Xamarinilla toteutetaan C#:lla ja sen kehitystyöhön voidaan käyttää Visual Studiota lisäosineen. Se ei myöskään vaadi erikseen esim. Swift- tai Java-osaamista, toisin kuin React, jossa käyttöjärjestelmäkohtaiset koodit joutuisi kirjoittamaan alustan natiivikielellä. Ohjelmakoodia pystytään myös jakamaan enemmän alustojen välillä Xamarinia käyttämällä. Molempien alustojen huonona puolena on kuitenkin se, että iOS-sovelluksen testaamiseen esimerkiksi emulaattorilla vaaditaan Applen tietokone.

3.2 Tietorakenteiden suunnittelu

Käyttäjien tekemät havainnot tulee säilyttää myöhempää tarkastelua varten. CosmosDB:een tallennettava tieto on JSON-muotoista, joten suurempia rajoituksia ei ole. Ainoastaan yksittäisen dokumentin maksimikoko on rajattu 64 kilobittiin. Ilman maksimikokoa havaintoon liitetty kuva olisi voitu tallentaa Base64-muotoisena CosmosDB:een muiden datatietueiden mukaan.

Ohjelmassa käytetään havainnon hallinnointiin samaa tietorakennetta kuin se joka viedään tietokantaan. Yksittäinen havainto koostuu taulukon 1 tietueista:

TAULUKKO 1. Sovelluskokonaisuudessa käytettävä havainnon tietorakenne

ObservationId	Havainnolle generoitu GUID
Saved	Havainnon luontiaika sovelluksessa
Created	Havainnon tallennusaika tietokantaan
Updated	Havainnon päivitysaika (arkistointi)
Creator	Havainnon luoja
Handler	Havainnon arkistoija
Archive	Tieto siitä, onko havainto arkistoitu
Target	Havainnon kohde (myöhempää jatkokehitystä varten)
Type	Havainnon tyyppi
Description	Havainnon vapaamuotoinen kuvaus
Location	GeoJSON-muotoinen paikkatieto
Attachment	Havaintoon liitetyn kuvan GUID

3.3 Mobiilisovelluksen käyttöliittymän suunnittelu

Parhaan mahdollisen käyttökokemuksen tarjoamiseksi käyttäjällä ohjelman tulee olla helppokäyttöinen, nopea sekä miellyttävän näköinen. Koska toteutus tulee

menemään yleiseen jakeluun Google Play -kauppaan, näihin asioihin haluttiin panostaa projektissa erityisesti.

Ulkoasuluonnosten tekemiseen käytettiin Balsamiq Mockups 3 -ohjelmaa. Sen avulla luonnoksia voidaan tehdä nopeasti. Se sisältää myös laajan valikoiman työkaluja, joiden avulla luonnokset saadaan vastaamaan hyvin pitkälle sitä, miltä lopputuote tulisi näyttämään.

Loin Balsamiqilla luonnoksen kaikista sovellukseen tulevista näkymistä. Värimaailman suunnittelussa käytin avuksi Googlen kuvahaulla löytyneitä väripaletteja. Päädyin ulkoasuun joka koostuu selkeistä väreistä ja isoista kuvallisista painikkeista, jotta sovellus olisi mahdollisimman helppokäyttöinen, mutta silti miellyttävän ja modernin näköinen (kuva 3).

KUVA 3. Balsamiqilla luotuja luonnoksia käyttöliittymästä

4 KÄYTETTÄVÄT TEKNISET MENETELMÄT JA OHJELMISTOT

4.1 Xamarin Forms -kehitysalusta

Xamarin Forms on Microsoftin omistama (6) Visual Studioon integroitu kehitysympäristö, jolla voidaan luoda natiiveja mobiilisovelluksia useammalle eri alustalle yhtäaikaisesti. Tällä hetkellä tuetut alustat ovat Android, iOS sekä Windows UWP. Sovelluskehitys tapahtuu C#-kielellä. Sovellusten ulkoasu voidaan luoda joko käyttämällä XAML-määrityksiä tai C#-koodia. (7.)

Xamarin Formsin etuna on se, että ohjelmakehitys eri alustoille voidaan tehdä yhdellä ohjelmointikielellä ja suurin osa sovelluksen ohjelmakoodista on jaettua eri alustojen kesken. Joitain toiminnollisuuksia varten joudutaan tekemään alustakohtaista ohjelmointityötä, mutta tällöinkin ohjelmointi tapahtuu C#:llä.

Xamarin Forms tarjoaa myös Xamarin Test Cloudin, jossa sovellusta voidaan testata yhtäaikaisesti eri alustoilla ja lukuisilla eri laitteilla. Laitteille voidaan syöttää komentoja, joita niiden tulee suorittaa sovelluksissa. Sovelluksen toimivuus voidaan todentaa lukemalla komentojen palauttamat arvot ja tarkastaa niiden oikeellisuus tai ottaa kuvankaappauksia käyttöliittymästä. Jälkimmäisessä tapauksessa sovelluksen toimivuuden tarkistaminen täytyy tehdä manuaalisesti tarkastelemalla saatuja kuvia.

4.2 ASP.NET Core

ASP.NET Core on Microsoftin ja yhteisön kehittämä avoimen lähdekoodin web framework eli sovelluskehys, jonka avulla voidaan luoda ja kehittää alustariippumattomia web-sovelluksia. Vaikka ASP.NET Core pohjautuukin ASP.NET Frameworkiin, se on ohjelmoitu kokonaan uudelleen, säilyttäen kuitenkin laajan yhteensopivuuden ASP.NET MVC:n kanssa. Tämä tarkoittaa, että ASP.NET Core -sovelluksia voidaan ajaa joko ASP.NET Coressa tai täydessä .NET Framework -ympäristössä. (7.)

ASP.NET Coren kehitystyössä on pyritty tarjoamaan suorituskykyinen framework pilvessä tai paikallisesti suoritettaville sovelluksille. ASP.NET Core on myös täysin modulaarinen, joten toteutuksen koko voidaan pitää minimaalisena käyttämällä vain sovelluksen toteuttamiseen tarvittavia NuGet-paketteja. (7.)

Visual Studio ja Visual Studio Code tarjoavat ASP.NET Core -integroinnin sovellusten helppoa ja nopeaa kehittämistä varten. Kehitystyötä ei ole kuitenkaan rajoitettu vain edellä mainittuihin editoreihin – kehittäjät voivat käyttää mitä tahansa tekstieditoria ja suorittaa sovellusten kääntämisen terminaalin kautta.

4.3 Microsoft Azure-ympäristö

4.3.1 Azure CosmosDB

Azure CosmosDB on skaalautuva pilvitietokantaratkaisu monenlaisten toteutuksien tarpeisiin. CosmosDB tarjoaa ohjelmointirajapinnat ASP.NET ja ASP.NET Core -alustojen lisäksi myös Javalle, Node.js:lle ja Pythonille. CosmosDB:a voidaan myös käyttää sen tarjoaman Rest-rajapinnan kautta, joka mahdollistaa kyselyjen tekemisen käytännössä millä tahansa ohjelmointikielellä miltä tahansa alustalta. Dokumentit tallennetaan CosmosDB:een JSON-muotoisina. (8.)

Kyselyitä tietokantaan voidaan tehdä SQL-syntaksia käyttäen, LINQ-kyselyillä tai MongoDB-kyselyillä. CosmosDB:ssa on myös GeoJSON-tuki, jonka avulla voidaan indeksoida dokumentit paikkatiedon mukaan, mikä mahdollistaa nopeiden geospatiaalisien kyselyiden suorittamisen. CosmosDB tukee myös triggereitä sekä tallennettuja proseduureja. (8.)

4.3.2 Azure Web App Service

Azure Web App Service on suoritus- ja hostausalusta web-sovelluksille. Se mahdollistaa usealla eri kielellä toteutettujen sovelluksien ajamisten Azuren pilvipal-

veluissa ja skaalaamisen ilman palvelininfrastruktuurin hallinnoimista. Se mahdollistaa myös Continuous Integrationin ja Continuous Development –julkaisuputkien käyttöönoton.

Azure Web App Service tukee .NET, ASP.NET Core, Node.js, PHP, Java, Python, Ruby ja HTML5-toteutuksia. (9.)

4.3.3 Azure Active Directory B2C

Azure Active Directory B2C (Business-to-Customer) on käyttäjän- ja pääsynhallintapalvelu, joka on ensisijaisesti tarkoitettu asiakas- ja kuluttajapalveluiden tarpeisiin. Käyttäjät kirjautuvat palveluisin sovelluskohtaisilla paikallisilla tunnuksilla (esim. oma henkilökohtainen sähköpostiosoite tai käyttäjätunnus), toisin kuin tavallisessa Active Directoryssä, johon täytyy kirjautua AD:n domainiin kuuluvalla sähköpostiosoitteella. B2C mahdollistaa myös rekisteröitymiset ja kirjautumiset palveluihin eri sosiaalisen median tunnusten avulla (Facebook, Twitter jne.). (10.)

B2C:tä voidaan käyttää kokonaan erillisenä palveluna, jonka avulla voidaan hallita kirjautumisia. Sitä käyttävän toteutuksen ei täydy edes sijaita Azuressa. B2C:tä käyttämällä voidaan erotella organisaation työntekijät ja palveluiden käyttäjät kokonaan eri directoryihin. Jokainen käyttäjä on oma itsenäinen kokonaisuutensa B2C:ssa, eli käyttäjät directoryn sisällä eivät näe toisiansa tai toistensa tietoja, toisin kuin tavallisessa Active Directoryssä.

.NET-sovelluksissa AD B2C -autentikointi voidaan implementoida helposti käyttämällä Microsoft Authentication Libraryä (MSAL). Sitä käyttämällä voidaan sovellukseen lisätä OAuth 2.0 –autentikointi, jota voidaan käyttää rekisteröitymiseen, kirjautumiseen ja muihin käyttäjänhallintatehtäviin, esimerkiksi REST-pyyntöjen turvaamiseen sovelluksen ja palvelimen välillä.

5 TOTEUTUS

5.1 Palvelimen toteutus

Palvelintoteutuksen rakentaminen aloitettiin luomalla tyhjä ASP.NET Core Web API-projekti. Koska kaikilla käyttäjillä ei tule olemaan oikeuksia kaikkiin palvelimen tarjoamiin rajapintoihin, projektiin tuli lisätä autentikointitoiminnallisuus. Päätimme yrityksen sisäisessä palaverissa käyttää Azure Active Directory B2C:tä käyttäjän- ja pääsynhallintaa varten. B2C on melko tuore palvelu ja ASP.NET Core on uusi palvelinalusta, jonka vuoksi autentikoinnin käyttöönotossa oli ongelmia.

Testasin autentikoinnin toimivuutta tekemällä Postman-sovelluksella kutsuja luomaani rajapintaan, joka vaati autentikoinnin toimiakseen. Microsoftin tarjoamat koodiesimerkit eivät olleet enää ajan tasalla, joten pyynnöt epäonnistuivat joka kerta ja palvelin palautti statuskoodin 401. Palvelun palauttamien virhekoodien perusteella päädyin siihen lopputulokseen, että esimerkeissä oleva Authority-osoite (jota kautta palvelu tarjoilee autentikointiin tarvittavan datan) on viallinen. Muita keskustelupalstoilta löytyneitä AD-autentikointitoteutuksia tarkastelemalla sain osoitteen muutettua muotoon, jolla autentikointi alkoi toimimaan.

Aloitin rajapintojen määrittelyn kokoamalla vaatimukset mitä tietoliikennettä sovelluksen ja palvelimen välillä tulisi liikkua:

- havainnon ja siihen liitetyn kuvan lähettäminen palvelimelle,
- havainnon arkistointi,
- havaintojen hakeminen kartalle palvelimelta,
- havainnon kuvan hakeminen Blob Storagesta,
- salausavaimen hakeminen,
- sekä sovelluksessa tapahtuneiden virheiden lähetys palvelimelle.

Rajapinnat toteutetaan ASP.net Coressa luomalla Controller-luokka. Controller-luokan metodeille voidaan määrittää HTTP-metodi, esimerkiksi GET tai POST sekä rajapinnan kutsumista varten vaadittava polku. Osaan controllereista lisättiin myös Authorize-attribuutti, joka estää pyynnöt rajapintoihin ilman autentikointia.

Päädyin luomaan neljä controlleria: ErrorController, KeyController, MapController sekä ObservationController.

ErrorControllerin ainoa tarkoitus on ottaa vastaan mobiilisovelluksessa tapahtuneet virheet ja lähettää ne eteenpäin Application Insights -lokianalyysityökaluun, jossa niitä voidaan tarkastella.

KeyControlleriltä mobiilisovellus voi pyytää salausavainta havaintojen paikallista tallennusta varten. Avain haetaan tietokannasta mobiililaitteen uniikin tunnisteen perusteella. Jos tunnistetta vastaavaa avainta ei löydy tietokannasta, luodaan uusi. Koska tämä rajapintatoteutus ei vaadi autentikointia, loin yksinkertaisen RateLimiter-luokan jota käyttäen avain haetaan. Luotu luokka estää rajapinnan jatkuvan kutsumisen vihamielisessä tarkoituksessa. Jos havaitaan, että jostain IP-osoitteesta kutsutaan rajapintaa liian useasti, estetään pyynnöt kyseisestä osoitteesta 24 tunniksi.

MapController palauttaa JSON-muotoisen listauksen klusteroiduista havainnoista. Sille lähetetään parametreinä kartan nykyinen zoomaustaso ja tieto siitä, halutaanko hakuun sisällyttää arkistoidut havainnot. Zoomaustasoa tarvitaan havaintojen klusterointiin. Klusteroinnilla tarkoitetaan havaintojen niputtamista yhden karttamerkin alle, jos karttaa tarkastellaan kaukaa. Tällä pyritään saamaan kartasta selkeämmän näköinen.

Itse klusterointitoteutus on melko yksinkertainen (kuva 4). Palvelin hakee tietokannasta kaikki tehdyt havainnot, jonka jälkeen ne käydään yksitellen läpi ja lasketaan, montako havaintoa on kyseisen havainnon säteellä. Säde on laskettu

Bing Maps Tile Systemsin esimerkkikoodista löytyvällä GroundResolution-funktiolla, joka kerrotaan 20:llä. Kaikki säteeltä löytyneet havainnot merkataan käytetyiksi, jottei niitä käydä läpi toiseen kertaan.

```
/// <summary>
/// Palauttaa havainnot paikkatiedon ja määritetyn säteen mukaisesti sekä niille luo klusteroinnin
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
1 reference | Oskari Pulkkinen, 41 days ago | 1 author, 6 changes | 0 exceptions
public List<MapTools.MapMarker> GetClusteredMapMarkers(MapController.MapRequest request)
{
    try
    {
        var list = _documentClient.CreateDocumentQuery<Observation>(UriFactory.CreateDocumentCollectionUri(DatabaseName, "Observation"))
            .Where(x => request.ShowArchived || !x.Archive)
            .ToList();

        var markers = new List<MapTools.MapMarker>();
        var used = new List<Observation>();

        foreach (var obs in list)
        {
            var gr = MapTools.GroundResolution(obs.Location.Position.Latitude, request.ZoomLevel);

            var inRadius = list.Where(x => !used.Contains(x) && MapTools.GetDistance(x.Location, obs.Location) < gr * 20).ToList();

            if (inRadius.Count < 1)
            {
                continue;
            }

            used.AddRange(inRadius);

            markers.Add(
                new MapTools.MapMarker
                {
                    Location = MapTools.GetClusterCenter(inRadius.Select(x => x.Location).ToList()),
                    Observations = inRadius
                });
        }

        return markers;
    }
    catch (DocumentClientException ex)
    {

    }

    return null;
}
```

KUVA 4. Havainnot tietokannasta hakeva ja klusteroiva toteutus

ObservationController pitää sisällään toiminnallisuudet havainnon tallentamiseen, havainnon kuvan hakemiseen sekä havainnon arkistointiin.

Havaintoa tallennettaessa palvelimelle lähetetään JSON-muotoinen luokkarakenne, joka pitää sisällään havainnon tiedot sekä havaintoon liitetyn kuvan Base64-muotoisena. Kuva muutetaan bittijonoksi merkkijonosta, poistetaan havainnosta ja lähetetään Blob Storageen. Kun kuvan lähettäminen on onnistunut, jäljelle jäänyt havainto-objekti viedään tietokantaan. Tähän rajapintametodiin lisättiin AllowAnonymous-attribuutti, jotta havaintoja voidaan lähettää palvelimelle ilman autentikointia.

Havainnon kuva haetaan Blob Storagesta käyttämällä havainto-objektiin tallennettua Attachment-GUIDia. Se palautetaan mobiililaitteelle tiedostona, eli Content-Typeksi asetetaan image/jpeg ja palautettavaksi dataksi kuvan bittijono, jotta CachedImage-laajennus pystyy käyttämään välimuistia kuvien säilömistä ja käsittelyä varten.

Havainnon arkistointia varten palvelimelle lähetetään ArchiveRequest-luokkarakenne, joka pitää sisällään listan havaintojen tunnisteista, jotka tulisi arkistoida sekä arkistoijan käyttäjätunnuksen. Toteutus käy läpi tunnistelistan ja päivittää dokumentteihin yksitellen arkistointistatuksen, arkistointiajan ja arkistoijan.

```
/// <summary>
/// Arkistoi valitut havainnot ja lisää Handler-kenttään arkistoijan käyttäjänimen sekä päivittää Updated-ajan
/// </summary>
/// <param name="observationIds"></param>
/// <param name="username"></param>
/// <returns></returns>
1 reference | Oskari Pulkkinen, 42 days ago | 1 author, 5 changes | 0 exceptions
public async Task<bool> ArchiveObservations(List<string> observationIds, string username)
{
    try
    {
        var documents = _documentClient.CreateDocumentQuery<Observation>(
                UriFactory.CreateDocumentCollectionUri(DatabaseName, "Observation"))
            .Where(x => observationIds.Contains(x.ObservationId))
            .ToList();

        foreach (var doc in documents)
        {
            doc.Handler = new Identification
            {
                Username = username
            };

            doc.Archive = true;

            doc.Updated = DateTime.Now;

            await _documentClient.ReplaceDocumentAsync(
                UriFactory.CreateDocumentUri(DatabaseName, "Observation", doc.Id),
                doc);
        }

        return true;
    }
    catch (DocumentClientException ex)
    {

    }

    return false;
}
```

KUVA 5. Havainnot tietokannasta hakeva ja klusteroiva toteutus

5.2 Mobiilisovelluksen toteutus

Mobiilisovelluksen toteutus aloitettiin luomalla Visual Studiossa Xamarin Forms Shared Project. Se luo projektia varten Visual Studio –solutionin (tiedosto, joka voi pitää sisällään useamman projektin), joka sisältää erillisissä projekteissa alustojen välillä jaettavat sekä alustakohtaiset ohjelmakoodit.

Minulla ei ollut aikaisempaa kokemusta Xamarin Formsin käytöstä kehitystyöhön, joten aloitin perehtymällä kehitysympäristöön. Xamarinin mukana tulee käyttöliittymän esikatselutyökalu, jolla periaatteessa pitäisi pystyä näkemään reaaliajassa miltä UI tulee näyttämään. Kyseinen työkalu osoittautui kuitenkin lähes hyödyttömäksi jatkuvan kaatuilun ja toimimattomuuden vuoksi.

Työn suorittamista hidasti graafisen käyttöliittymän esikatselutyökalun toimimattomuus, joka johti siihen, että jouduin luomaan uuden APK:n joka kerta, kun halusin tarkastella tekemiäni muutoksia. Kun olen aikaisemmin tehnyt Android-kehitystyötä Android Studiolla, olen pystynyt viemään muutokset todella nopeasti puhelimeen käyttämällä sen tukemaa Instant Run -toimintoa. Xamarin-ympäristö ei tue tätä ominaisuutta.

APK:n vieminen Oneplus 5T:hen Xamarinilla on myös erittäin hidasta, jopa 3–4 minuuttia kertaa kohden. Aloin selvittämään syytä sille, mistä tämä johtuu. Asetin MSBuildin kirjoittamaan Visual Studion output-työkaluun lokia kaikesta sen tekemistä vaiheista. Sieltä selvisi, että buildia tehdessä Xamarin vie jostain syystä puhelimeen joka kerta Mono Shared Runtimen sekä Mono Frameworkin, joiden yhteiskoko on yli 100 megatavua. Tämä johtuu ilmeisesti virheestä kehitysympäristössä tai yhteensopimattomuudesta puhelimen Android-version (7.1.1) kanssa. Minulla ei ollut käytössä muita laitteita, joissa olisi ollut yhtä tuore Android, mutta versioilla 5.1.1 ja 6.0.1 ei tapahtunut samaa hidastelua.

Sovelluksen vieminen puhelimeen oli kuitenkin hieman hitaampaa kuin emulaattoriin, joten käytin aluksi sovelluksen testaamiseen suurimmaksi osin emulaattoria. Kameratoiminnallisuuden lisäämisen jälkeen emulaattori kuitenkin kaatui joka

kerta kuvaa ottaessa, jonka vuoksi jouduin luopumaan emulaattorin käyttämisestä.

5.2.1 Navigoinnin implementointi sovellukseen

Alkuperäisessä luonnoksessa ohjelma oli jaettu eri sivuihin: aloitussivu, havaintokohteen valinta, havaintotyypin valinta, kuvauksen kirjoittaminen, kuvan ottaminen sekä yhteenvetosivu. Aloin luomaan ylläolevaa rakennetta käyttämällä Xamarin Formsin tarjoamaa hierarkista navigointia (kuva 6).

KUVA 6. Xamarin.Formsin navigointi visualisoituna (12.)

Hierarkisessa navigoinnissa luodaan ns. sivupakka, johon lisätään aina tarvittaessa uusi sivu päällimmäiseksi. Tällöin navigointi eteen- ja taaksepäin on helppoa ja joustavaa. Toteutuksessani havainnon luomisen päätyttyä pakka tyhjennetään ja palataan aloitussivulle. Yksittäinen sivu luodaan Visual Studiossa lisäämällä projektiin uusi Content Page, joka luo XAML-määrityssivun ja C#-kooditiedoston.

Xamarin Formsissa navigoinnin käyttöönotto oli erittäin helppoa. Seuraavalle sivulle siirrytään kutsumalla Navigation-luokan PushAsync-metodia, jolle annetaan parametrinä haluttu sivuobjekti.

Koska sovelluksen avulla käyttäjän luoma havainto muodostuu sivu kerrallaan niillä syötetyistä datasta, täytyi dataobjektia pitää jotenkin yllä sivujen välillä navigoidessa. Dataa voitaisiin siirtää välittämällä jokaiselle sivulle erikseen tiedot sisällään pitävä objekti. Koin sen olevan kuitenkin jokseenkin kestämätön ratkaisu, jos ohjelma tulisi laajenemaan.

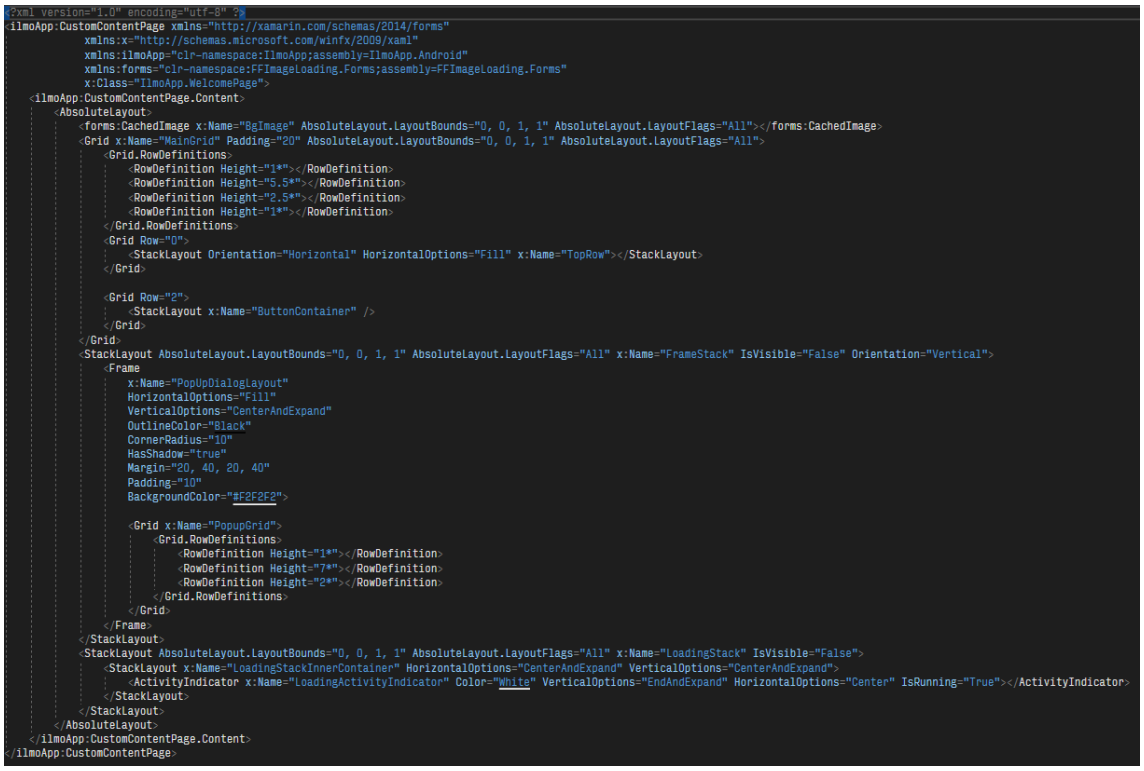
Jokainen Xamarin Forms -sovellus pitää sisällään Application-luokan, joka hallinnoi sovellusta. Se tarjoaa Properties-luokan, johon voi tallentaa haluamiaan tietoja, mutta se ei osaa serialisoida luokkarakenteita (13). Tästä johtuen loin Application-luokkaan kaksi muuttujaa, joista toinen pitää sisällään CurrentObservation-luokan ja toinen PageModel-luokan. Täten kaikki sovelluksen sivut jakavat keskenään yhdenmukaiset tiedot.

Ohjelman käynnistyessä palvelimelta haetaan JSON-muotoinen PageModel-luokkaa vastaava dataobjekti, joka pitää sisällään määritykset siitä, mitä painikkeita eri sivuilla tulee näyttää. Havainnon kohde- ja tyyppisivuilla näytettävät painikkeet luodaan dynaamisesti kyseisten määritysten mukaisesti.

CurrentObservation-luokka vastaa tietokantaan tallennettavaa tietorakennetta. Sitä täytetään sitä mukaa, kun havainto valmistuu. Kun havainto on valmis, sen sisältämät tiedot validoidaan, muunnetaan JSON-objektiksi ja lähetetään palvelimelle, jossa se talletetaan tietokantaan.

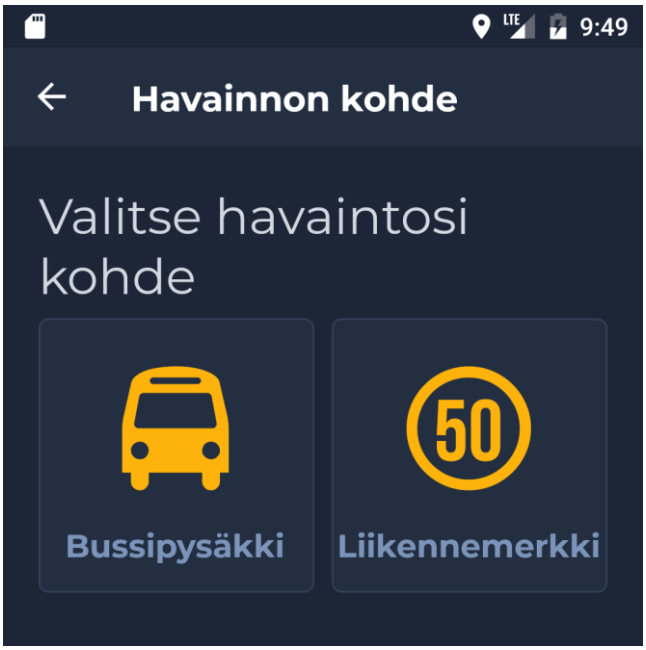
5.2.2 Sovelluksen käyttöliittymän luominen

Aloin luomaan sovelluksen käyttöliittymää tekemieni luonnosten pohjalta. Ensimmäiseksi vaiheeksi valikoitui sovelluksen painikkeiden teko. Oletuksena Xamarin Forms -ohjelmissa painikkeet ovat eri alustojen natiiveja painikkeita, joten niitä ei voitu käyttää tähän tarkoitukseen. Käytin ulkoasujen määritykseen yhtäaikaisesti XAML- ja C#-määrityksiä, koska koin layoutien muodostamisen XAML-määrityksillä selkeämmäksi (kuva 7). Dynaaminen ja event-pohjainen sisältö luodaan C#-koodilla.



KUVA 7. Sovelluksen etusivun XAML-määritystiedosto

Painikkeiden täytyi sisältää kuvake ja teksti, sekä taustan tuli olla reunoista pyöristetty neliö (kuva 8). Forms ei tarjoa käyttöliittymäpalikkaa eri muotojen tekemistä varten, joten tähän tehtävään valikoitui lisäosa XFShapeView. Loin apuluokan painikkeiden muodostusta varten, koska samankaltaisia painikkeita tarvitaan suuri määrä ohjelman sisällä. Apuluokalla on staattinen Create-metodi, jolle syötetään luokan sisällä määritelty Parameters-luokka, jonka avulla voidaan määrittää painikkeen sisältö ja ulkoasu (kuva 9).



KUVA 8. Valmiit sovelluksessa käytettävät painikkeet

Painike koostuu ShapeViewistä, joka on painikkeen tausta sekä pitää sisällään kuuntelijan kosketusta varten. Sen sisältönä ovat StackLayoutiin asetetut Labelit painikkeen ikonia sekä tekstiä varten. Ikonit olisi voitu toteuttaa myös kuvina, mutta päätin käyttää tässä toteutuksessa fonttia, joka pitää sisällään halutut ikonit. Fontiksi valikoitui Font Awesome, jota olen käyttänyt aikaisemminkin vastaavan kaltaisiin tarpeisiin. Se sisältää suuren määrän erilaisia vektorikuvia, jotka ovat hyvin käytännöllisiä sovelluksen ulkoasun piristämistä ja selkeyttämistä varten. Projektin luonteesta johtuen tarvitaan myös ikoneita, joita mikään fonttipaketti ei pidä sisällään. Se ei ole kuitenkaan ongelma, koska Font Awesomea voidaan muokata lisäämällä omia vektorimuotoisia ikoneita. Käytin fontin kustomointiin Fontelloa. Sen avulla fonttiin voidaan lisätä SVG-muotoisia vektorigrafiikoita. Loin vektorimuotoiset (bussi, liikennemerkki jne.) ikonit käyttämällä ilmaista Inkscapea.

```
27 references | Oskari Pulkkinen, 34 days ago | 1 author, 8 changes
public static CustomShapeView Create(Parameters parameters)
{
    View buttonContent;

    var icon = Icon.Create(parameters.IconTag, parameters.IconSize, popUp: parameters.PopUpButton);

    var legend = new Label
    {
        Text = parameters.Legend,
        FontSize = parameters.LegendSize,
        TextColor = !parameters.PopUpButton ? Colors.BaseFg : Colors.PopUpButtonFg,
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.End,
        VerticalTextAlignment = TextAlignment.End,
        HorizontalTextAlignment = TextAlignment.Center,
        Margin = new Thickness(0, -15, 0, 10),
        FontAttributes = FontAttributes.Bold,
        FontFamily = "Montserrat-Regular.otf#Montserrat",
        LineBreakMode = LineBreakMode.NoWrap
    };

    if (parameters.IconOnSide || parameters.ListButton)
    {
        buttonContent = new Grid
        {
            ColumnDefinitions = new ColumnDefinitionCollection
            {
                new ColumnDefinition
                {
                    Width = new GridLength(2, GridUnitType.Star)
                },
                new ColumnDefinition
                {
                    Width = new GridLength(8, GridUnitType.Star)
                }
            }
        };

        icon.HorizontalOptions = LayoutOptions.Center;
        icon.Margin = new Thickness(5, 5, 0, 0);

        legend.Margin = new Thickness(-icon.FontSize, 0, 0, 0);
        legend.VerticalTextAlignment = TextAlignment.Center;
        legend.HorizontalOptions = LayoutOptions.Center;
        legend.VerticalOptions = LayoutOptions.CenterAndExpand;

        ((Grid)buttonContent).Children.Add(icon, 0, 0);

        if (!string.IsNullOrEmpty(parameters.Legend))
        {
            ((Grid)buttonContent).Children.Add(legend, 1, 0);
        }
    }
```

```
    else
    {
        buttonContent = new StackLayout();

        ((StackLayout)buttonContent).Children.Add(icon);

        if (!string.IsNullOrEmpty(parameters.Legend))
        {
            ((StackLayout)buttonContent).Children.Add(legend);
        }
    }

    var button = new CustomShapeView
    {
        ShapeType = parameters.ShapeType,
        HeightRequest = parameters.Height,
        WidthRequest = parameters.Width,
        Color = !parameters.PopUpButton ? Colors.ButtonBg : Colors.PopUpButtonBg,
        HorizontalOptions = parameters.HzOptions,
        VerticalOptions = parameters.VertOptions,
        CornerRadius = 10,
        BorderColor = !parameters.PopUpButton ? Colors.ButtonBorder : Colors.PopUpButtonBorder,
        BorderWidth = 1f,
        Content = buttonContent,
        Margin = parameters.Margin,
        IconLabel = icon,
        TextLabel = legend,
        MaxHeight = parameters.MaxHeight
    };

    if (parameters.ListButton)
    {
        button.HeightRequest = (int)Text.GetFontSize(dsize: 60);
    }

    if (parameters.OnTap != null)
    {
        button.GestureRecognizers.Add(new TapGestureRecognizer
        {
            Command = parameters.OnTap
        });
    }

    if (parameters.SquareAspectRatio)
    {
        button.SizeChanged += Button_SizeChanged;
    }

    return button;
```

KUVA 9. Apufunktio sovelluksessa käytettävien painikkeiden luomista varten

Halusimme koestaa myös käyttöliittymäluonnoksesta poikkeavaa tapaa käyttää sovellusta, jossa etusivulta siirrytään suoraan yhteenveto-sivulle, jossa navigoinnin sijaan kaikki toiminnot voidaan suorittaa yhdellä sivulla. Jotta tämä olisi mahdollista ja sovellus olisi helppokäyttöinen sekä miellyttävän näköinen, tarvittiin sovelluksen päälle avautuva dialogi.

Xamarin Forms ei tarjoa valmista käyttöliittymäpalikkaa tätä varten. Testasin myös Popup-lisäosaa, mutta sitä käyttäneet kehittäjät olivat raportoineet kaatumisista, joten päätin luoda oman toteutuksen (kuva 10).

KUVA 10. Valmis dialogitoteutus

Jotta Xamarin Formsissa voidaan luoda kerroksittaisia käyttöliittymiä, täytyy kaikki sivulla olevat elementit kietoa RelativeLayoutin tai AbsoluteLayoutin sisään. Jokaiselle sivulle, jossa oli tarve dialogille, luotiin Frame-elementti, jonka sisään pystytään luomaan halutun kaltainen ulkoasu.

Yhteenveto-sivulle siirryttäessä annetaan luokalle boolean-parametri, joka määrittää, onko kyseessä havainnon pikakirjaus. Jos se on tosi, asetetaan sivun eri elementeille kosketuksen kuuntelijat, jotta havainto pystytään täyttämään. Kosketuksen tapahtuessa dialogi täytetään dynaamisesti tarvittavilla elementeillä.

Kun havainto on täytetty ja valmis lähetettäväksi eteenpäin, havainnon sisältö validoidaan. Jos validointi onnistuu, tallennetaan havainto ensin levylle. Havainto muutetaan JSON-objektiksi, joka Base64-enkoodataan ja salataan, jottei havaintoa voida muuttaa vahingossa tai tahallisesti sinä aikana, kun sitä säilytetään levyllä. Sen jälkeen havainto lähetetään palvelimelle. Jos lähetys onnistuu, levylle tallennettu havainto poistetaan. Muussa tapauksessa se pidetään tallessa niin kauan, kunnes se on voitu lähettää palvelimelle onnistuneesti.

5.2.3 Toiminnallisuuksien lisääminen ja laajennuksien käyttöönotto

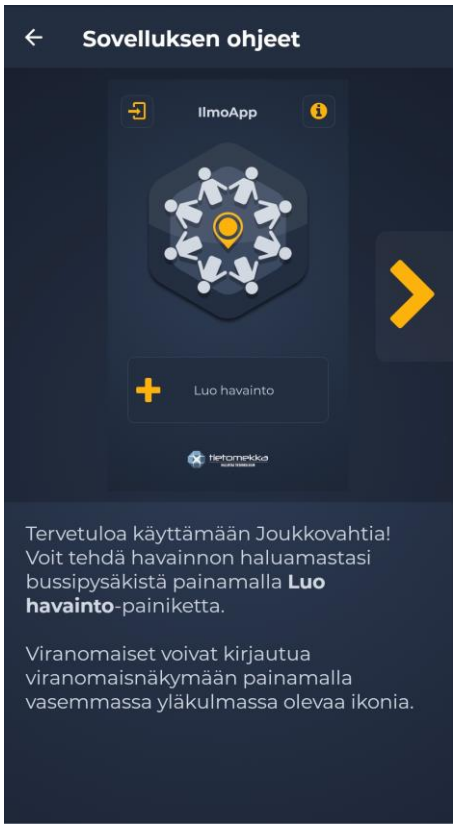
Lisäsin kameratoiminnallisuuden sovellukseen käyttämällä MediaPlugin-lisäosaa. Lisäosa tarjoaa toiminnot kuvien ja videoiden ottamiseen tai valitsemiseen galleriasta alustasta riippumatta. Lisäosan käyttäminen itsessään oli hyvin helppoa: kamera avautuu kutsumalla TakePhotoAsync-funktiota, joka palauttaa kuvan, jos kuvan ottaminen on onnistunut.

Jotta MediaPluginin avulla otettu kuva voidaan näyttää myös toisella sivulla kuin vain sillä, jossa se on napattu, täytyy kuvan tuottama bittivirta kopioida talteen. Toteutin sen kopioimalla GetStream-funktion bittivirran bittitaulukkoon, joka tallennettiin ohjelman CurrentObservation-luokkaan. Jotta kuva voidaan näyttää toisella sivulla Image-elementissä, bittitaulukko täytyy kopioida bittivirtaan (esim. MemoryStreamiin).

Alun perin käytin FFImageLoading-lisäosan CachedImage-elementtiä kameralla otetun kuvan näyttämiseen, koska sen avulla kuvaa ei täydy muodostaa joka kerta uudestaan, kun siirrytään sivulta toiselle, vaan se haetaan välimuistista. Oneplus 5T:llä tämä johti jostain syystä joka kerralla virheeseen, vaikka muilla laitteilla toteutus toimi oikein. Tämän vuoksi jouduin muuttamaan CachedImaget takaisin tavallisiksi Xamarin Formsin Imageiksi, joka hidastaa kuvien näyttämistä jonkin verran.

Paikannuspalveluiden hyödyntämistä varten käytettiin alustariippumatonta Geolocator-lisäosaa. Se ilmoittaa sovellukselle aina, kun tarjolla on uusi sijaintitieto,

jossa sitä voidaan hyödyntää. Paikannuspalveluiden käyttämistä varten tarvitaan lupa puhelimelta, joiden pyytämiseen käytettiin Permissions-lisäosaa.



KUVA 11. Ensimmäisellä käynnistyskerralla näytettävä käyttöehtodialogi

Lisäsin sovellukseen myös ensimmäisellä käynnistyskerralla näytettävän käyttöehtodialogin (kuva 11), joka vaatii hyväksynnän sovelluksen käyttämistä varten. Tieto dialogin hyväksynnästä tallennetaan SharedPreferencesiin. Hyväksynnän jälkeen näytetään opastus sovelluksen käyttöön. Käyttöehtoja ja opastusta voidaan tarkastella myös myöhemmin painamalla etusivulla olevaa info-painiketta.

5.2.4 Karttakirjaston valitseminen ja käyttöönotto

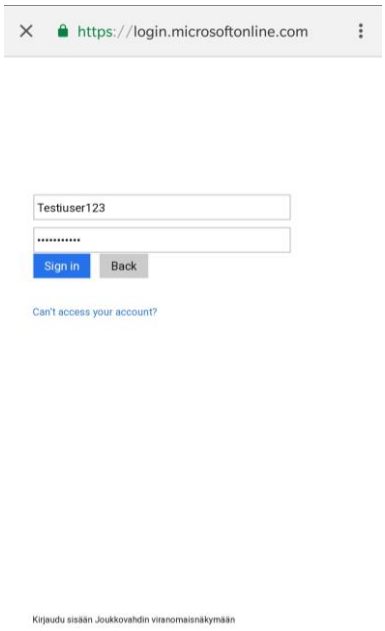
Xamarin Forms pitää sisällään alustariippumattoman karttakirjaston, jota voidaan käyttää karttapohjien näyttämiseen mobiilisovelluksissa. Se käyttää mobiililaitteiden natiiveja karttakirjastoja ja rajapintoja, joten lopullinen toteutus on erilainen joka laitteella. Kuitenkin muun muassa tästä johtuen kirjasto on toiminnoiltaan hyvin rajallinen, joten päätin selvittää mitä erilaisia mahdollisuuksia karttanäkymän tuottamiseen olisi.

Vaihtoehtoja oli käytännössä kaksi: TKCustomMap-kirjasto tai Xamarin.Google.Maps -kirjasto. Koestin molempia kirjastoja lisäämällä niiden vaatimat elementin sovelluksen käyttöliittymään ja tarjoamalla niille kartalla näytettävää dataa. Näistä kahdesta kirjastosta Xamarin.Google.Maps avautui nopeammin ja oli sulavampi käyttää (kartalla liikkuminen, zoomaus yms.) sekä sen päivitystahti oli nopeampi, joten se valikoitui käytettäväksi.

Kartan lisääminen sovellukseen on hyvin yksinkertaista: sovelluksen layout-määrityksiin lisätään karttaelementti ja määritetään sille koko, jonka jälkeen karttaa voidaan käyttää. Kartalle voidaan piirtää tilejä, joilla voidaan esittää haluttua dataa, tai vaihtoehtoisesti näyttää ikoneita, jotka piirretään käyttäjän päässä haluttuihin koordinaattipisteisiin. Koska kartalla yhtäaikaisesti esitettävien objektien määrä tulee jäämään suhteellisen vähäiseksi, päätin piirtää ikonit kartalle käyttäjän laitteella sen sijaan, että ne olisi piirretty palvelimella. Testasin kartan käytettävyyttä sadalla, tuhannella ja kymmenellä tuhannella ikonilla, joista 100 oli vielä käytettävyydeltään kelvollinen.

5.2.5 Viranomaisnäkymän toteuttaminen

Mobiilisovellus tarvitsi myös viranomaisnäkymän palvelun avulla luotujen havaintojen kartalta tarkastelemista varten. Lisäsin sovelluksen etusivulle painikkeen, jonka avulla voidaan kirjautua palveluun sisään. Painiketta painettaessa luodaan MSAL-kirjaston avulla kirjautumispyyntö Azure AD B2C-palveluun (kuva 12). MSAL-kirjasto pitää muistissa onnistuneet kirjautumiset. Jos muistissa ei ole yhtään validia kirjautumista, sovellus avaa uuteen webselainpohjaiseen ikkunaan kirjautumisnäkymän, johon tulee syöttää käyttäjätunnus sekä salasana. Jos sisäänkirjautuminen onnistuu, käyttäjälle näytetään Luo havainto –painikkeen alla uusi painike, jota painamalla päästään viranomaisten karttanäkymään.



KUVA 12. Käyttäjälle näytettävä kirjautumisnäkymä

Karttaelementtiin haetaan palvelimelta havainnot kartan zoomaustason mukaan karttanäkymän avautuessa (kuva 13). Havainnot haetaan kartalle uudestaan joka kerta, kun kartan liikuttaminen tai zoomaaminen on päättynyt. Oikeassa yläkulmassa on myös painike, jolla voidaan hakea myös arkistoidut havainnot kartalle.

```
/// <summary>
/// Hakee havainnot paikan perusteella
/// </summary>
/// <param name="request"></param>
/// <returns></returns>
1 reference | Oskari Pulkkinen, 37 days ago | 1 author, 2 changes
public static async Task<List<ObservationMapPage.MapMarker>> GetObservations(MapRequest request)
{
    var jsonData = "";

    using (var client = new HttpClient())
    {
        var uri = new Uri(App.BaseUrl + "map");

        var content = new StringContent(JsonConvert.SerializeObject(request), Encoding.UTF8, "application/json");

        client.Timeout = new TimeSpan(0, 0, 0, 30);

        var token = await Task.Run(() => Auth.GetToken());

        try
        {
            client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

            var response = await client.PostAsync(uri, content);

            jsonData = await response.Content.ReadAsStringAsync();
        }
        catch (Exception ex)
        {
            ErrorTools.SendError(ex);
        }
    }

    return JsonConvert.DeserializeObject<List<ObservationMapPage.MapMarker>>(jsonData) ?? new List<ObservationMapPage.MapMarker>();
}
```

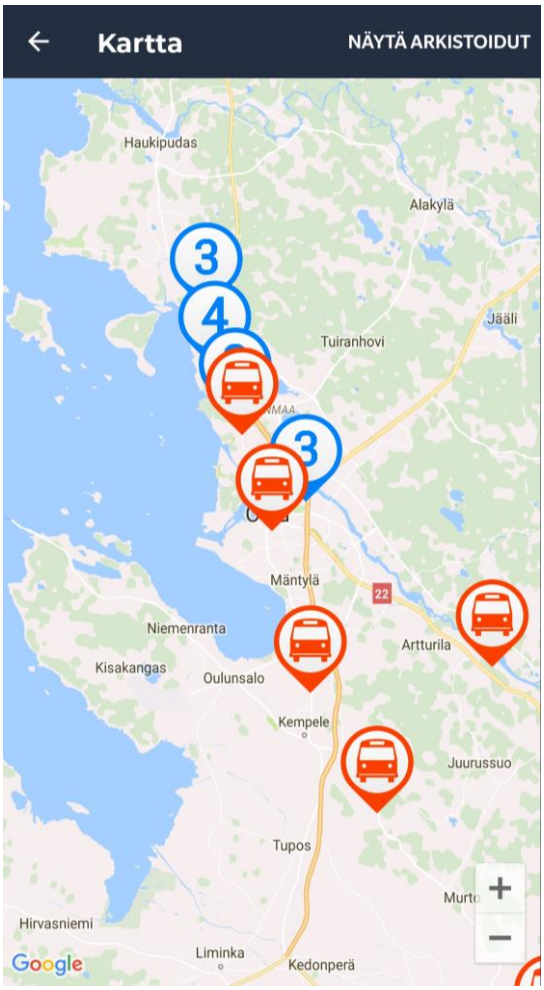
KUVA 13. Havaintojen hakeminen palvelimelta käyttäen MSAL-autentikointia

Havainnot klusteroidaan palvelimen päässä, jotta kartan suorituskyky ja selkeys pysyisivät kohtuullisina. Tämä tarkoittaa sitä, että lähekkäin olevat havainnot niputetaan yhden karttamerkin alle. Tälläisessä ikonissa on havaintojen lukumäärä, joka piirretään tyhjään ikoniin dynaamisesti (kuva 14) käyttäen Xamarin.Formsin SKPaint-kirjastoa, joka muistuttaa pitkälti .NET Frameworkin Graphics-kirjastoa.

```
/// <summary>
/// Piirtää tyhjään markkeriin sen alla olevien havaintojen lukumäärän
/// </summary>
/// <param name="count"></param>
/// <returns></returns>
1 reference | Oskari Pulkkinen, 34 days ago | 1 author, 1 change
private BitmapDescriptor CreateMultipleMarker(int count)
{
    var assembly = this.GetType().GetTypeInfo().Assembly;

    using (var s = assembly.GetManifestResourceStream("IlmoApp.Droid.multiplemarker.png"))
    {
        var marker = SKBitmap.Decode(s);
        var canvas = new SKCanvas(marker);

        var textPaint = new SKPaint
        {
            Color = new SKColor(0, 129, 250),
            FakeBoldText = true,
            TextSize = .65f * marker.Width
        };

        var textBounds = new SKRect();
        textPaint.MeasureText(count.ToString(), ref textBounds);

        var x = (marker.Width / 2.0f) - textBounds.MidX;
        var y = (marker.Height / 2.0f) + 15;

        canvas.DrawText(count.ToString(), x, y, textPaint);

        var img = SKImage.FromBitmap(marker);
        var encodedImg = img.Encode(SKEncodedImageFormat.Png, 85);
        var stream = encodedImg.AsStream();

        return BitmapDescriptorFactory.FromStream(stream);
    }
}
```

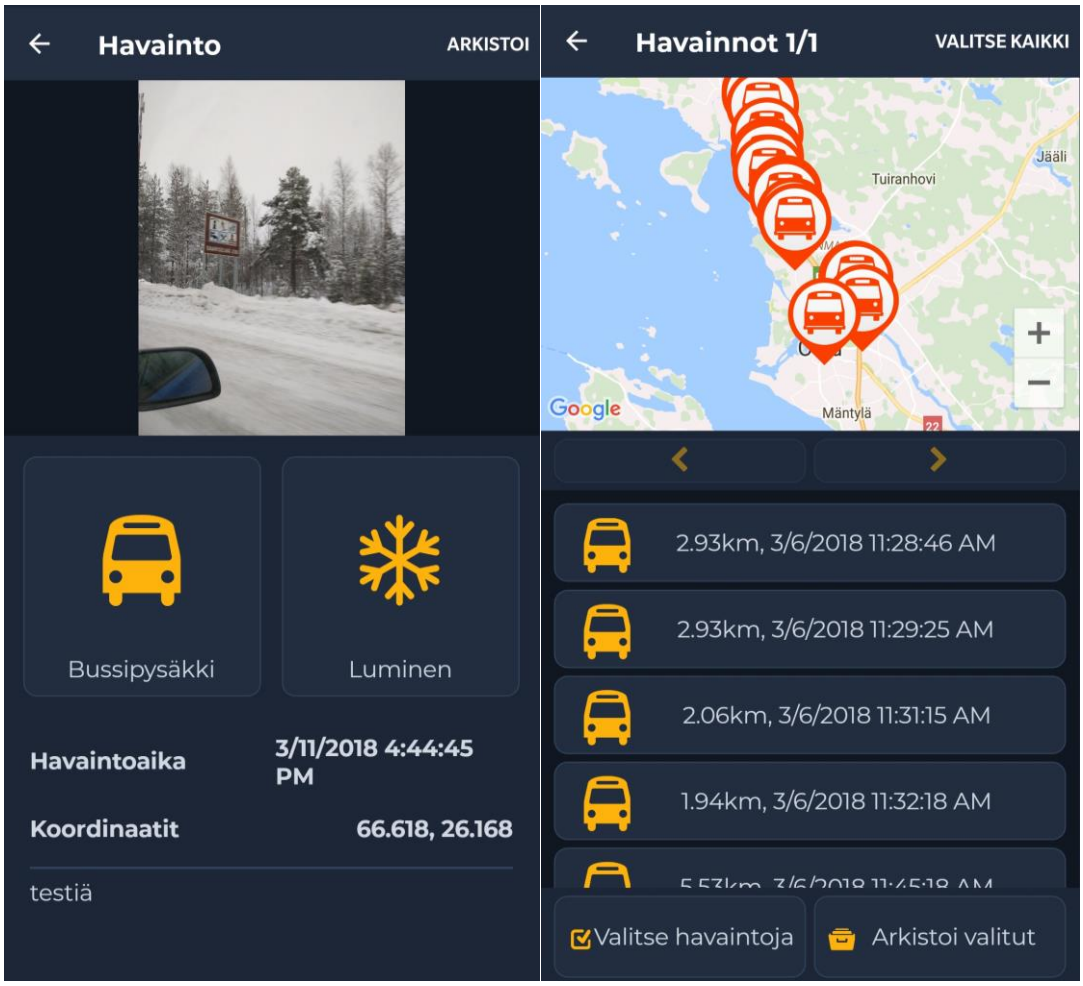
KUVA 14. Numeron ikoniin piirtäminen SKPaint-kirjastoa käyttäen

Kartalle piirrettäviin karttamerkkeihin sisällytetään sen alla olevien havainnot (kuva 15). Karttamerkkiä painettaessa, riippuen siitä onko sen alla yksi vai useampi havainto, avataan joko havaintonäkymä tai listanäkymä havainnoista.



KUVA 15. Sovelluksen viranomaisnäkymän karttanäkymä

Jos kyseessä on yksittäinen havainto, avataan karttamerkkiin tallennetusta data-objektista samankaltainen näkymä kuin havaintoa luodessa (kuva 16). Erona tässä näkymässä edellä mainittuun on se, ettei havaintoa voida muokata jälkikäteen. Viranomainen pystyy tarkastelemaan havaintoon liitettyä kuvaa isompana tai vaihtoehtoisesti arkistoimaan sen.



KUVA 16. Vasemmalla yksittäisen havainnon näkymä, oikealla useamman havainnon listanäkymä.

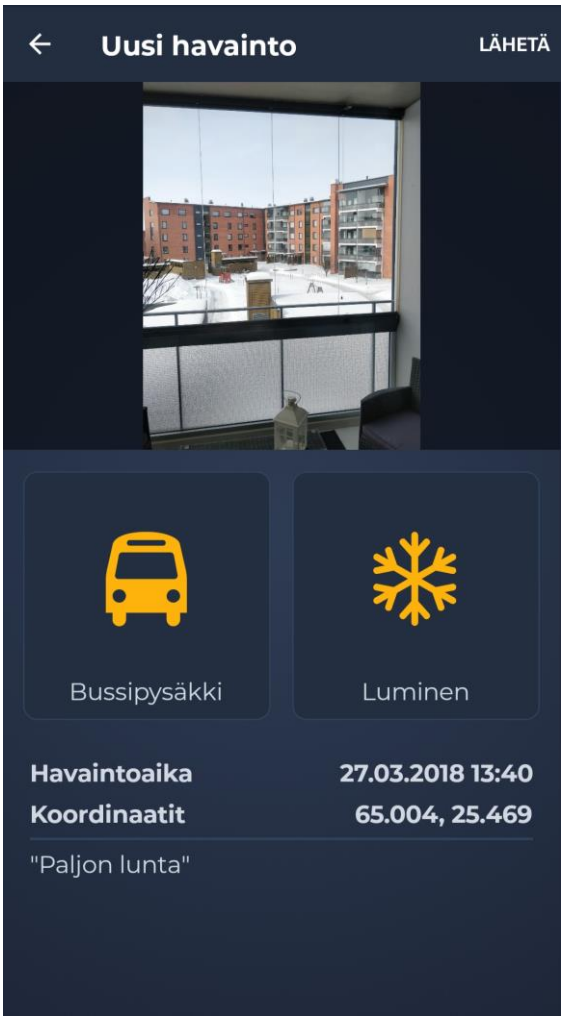
Jos karttamerkki pitää sisällään useamman havainnon, avataan uusi näkymä, jossa on pieni kartta joka näyttää avatut havainnot, sekä sivutettu listanäkymä havainnoista (kuva 16). Listalla olevista havainnoista näytetään etäisyys sen hetkiseen kartan keskikohtaan sekä havainnon tallennusaika. Listanäkymässä voidaan myös valita havaintoja sekä arkistoida niitä. Yksittäiseen havaintoon voidaan siirtyä painamalla kartalla näkyviä karttamerkkejä tai valitsemalla havainto listasta.

5.2.6 Kehitystyössä vastaan tulleet ongelmat ja testaus

Yrityksellä oli ylimääräisenä Samsung Galaxy Note 4 -puhelimia, joten päätin testata lähes valmista sovellusta käyttämällä kyseisiä laitteita. Yllätyksekseni sovellus reagoi kosketuksiin erittäin huonosti. Ainoastaan hyvin pehmeä, hipaisun kaltainen kosketus rekisteröitiin. Löysin Xamarinin virhekirjausjärjestelmästä viittaukseen samankaltaiseen ongelmaan. Siellä kerrottiin myös ongelman olevan korjattu Xamarin Formsin versioon 2.4.0.38779. Päivitin XF:n sen uusimpaan julkaisuversioon (2.5.0.1), joka ei korjannut ongelmaa. Ongelma katosi kuitenkin päivittämällä aiemmin mainittuun versioon, joka oli ensimmäinen julkaisu johon korjaus oli tehty.

Alun perin sovelluksen lokalisointiin oli tarkoitus käyttää resx-tiedostoja. Visual Studion rajoitusten vuoksi jaetussa projektissa (Shared Project, .shproj) ei ole mahdollista käyttää resursseja. Keskustelupalstoilla kerrottiin, että resx-tiedostoja voitaisiin mahdollisesti käyttää luomalla solutioniin ns. PCL-projekti, joka pitäisi sisällään ainoastaan resurssitiedostot. En saanut lukuisista yrityksistä huolimatta tätä toimimaan, joten päätin toteuttaa oman lokalisaatioluokan. Se on yksinkertaisuudessaan lista StringResource-objekteista, joka puolestaan pitää sisällään merkkijonon tunnisteen sekä merkkijonon suomeksi ja englanniksi.

Käyttöliittymän käytettävyystestaus suoritettiin yrityksen toimistolla. Valikoin testaajiksi neljä henkilöä. Annoin heille laitteen, johon sovellus oli asennettu, ja pyysin heitä tekemään havainnon. Sovellusversiossa oli kaksi käyttötapaa: navigaatiopohjainen toteutus ja pikakirjaus. Pikakirjaus (kuva 17) valikoitui kaikkien testihenkilöiden mielestä paremmaksi vaihtoehdoksi kahdesta, koska havainnon tekeminen oli sillä nopeampaa ja yksiselitteisempää. Täten päätin poistaa navigaatiopohjaisen toteutuksen käytöstä, mutta pidin sen projektissa mahdollisen ohjelman laajentumisen vuoksi.



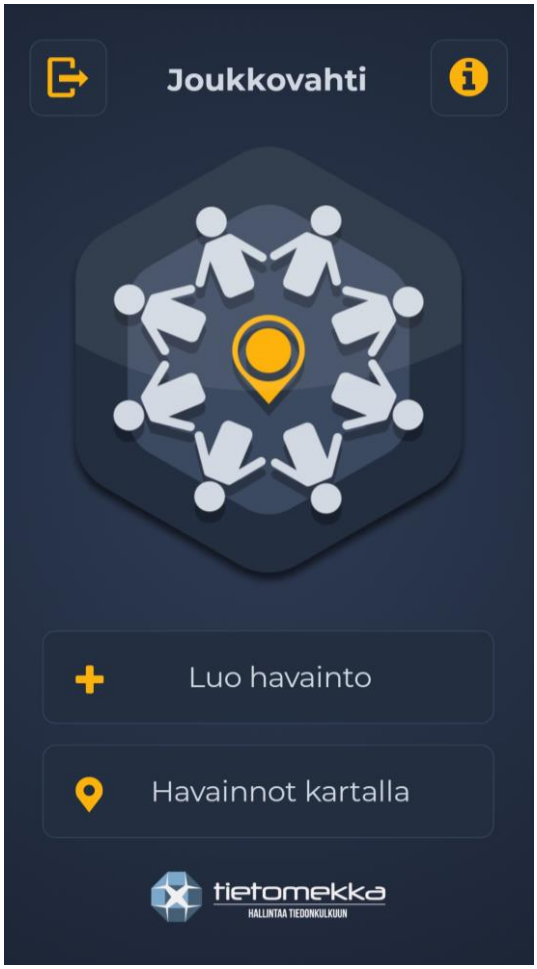
KUVA 17. Havainnon luomisnäkymä

Sovelluksen etusivulla (kuva 18) näytetään oletuksena vain yksi iso painike, jota painamalla voidaan luoda havainto. Viranomaisnäkymän lisäämisen jälkeen näkymässä näytetään myös toinen painike, jos käyttäjä on kirjautunut sisään.

Tämä toi kuitenkin mukanaan ongelman: painikkeet sisällään pitävä StackLayout ei päivittänyt korkeuttaan, kun painike asetettiin näkyville, jonka vuoksi painikkeet olivat liian pieniä. Jos kävin toisella sivulla ja tulin takaisin etusivulle, painikkeet olivat oikean kokoisia. Kokeilin kaikkia keskustelupalstoilta löytämiäni keinoja päivittääkseni layoutin ja sen sisältämien elementtien koot, mutta mikään ei korjannut tilannetta. Ongelma tuli kuitenkin korjata, jotta käyttöliittymä pysyisi halutun näköisenä.

Päätin kokeilla mitä tapahtuisi, jos piilotan painikkeet sisällään pitävän layoutin ja asetan sen uudestaan näkyville manuaalisesti. Tämä korjasi layoutin korkeuden

ja painikkeet näyttivät taas siltä miltä pitikin. Lisäsin karttanäkymän painikkeelle PropertyChanged-kuuntelijan, joka tarkkailee, milloin elementin IsVisible-arvo muuttuu. Kuuntelija asettaa painikkeet sisällään pitävän layoutin piilotetuksi ja näyttää sen uudelleen. Tällä tavoin sain ongelman korjattua, joka oletettavasti johtuu virheestä Xamarin.Formsissa.



KUVA 18. Sovelluksen etusivu

Kun sovellus toimitettiin testikäyttöön tilaajalle, ilmeni ongelma, jolloin karttapainike ei aina näkynyt etusivulla sen jälkeen, kun käyttäjä oli käynyt jollain toisella sivulla. Xamarin.Formsin navigointijärjestelmän pitäisi pitää tallessa sivujen tila, jos niitä ei ole poistettu navigointipinosta, enkä saanut toistettua kyseistä ongelmaa millään laitteella. Tämä ongelma saattaa johtua virheestä Xamarin.Formsissa, käyttäjän laitteesta tai niiden yhdistelmästä. Sain ongelman korjattua kuitenkin helposti päivittämällä painikkeiden näkyvyyden aina, kun sivu näytettiin uudelleen.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda mobiilisovellus sekä palvelintoteutus, joka mahdollistaisi havaintojen tekemisen helposti ulkona liikuttaessa. Työn aihe vaikutti mielenkiintoiselta ennen projektin aloittamista ja mielenkiinto pysyi yllä koko työn tekemisen ajan. Projektin kehitystyö suoritettiin minulle uusilla kehitysalustoilla, Xamarin.Formsilla ja ASP.NET Corella. Tämä lisäsi toiminnallisuuksiltaan suoraviivaisen sovelluksen kehitystyöhön sopivasti haasteita.

Projektin lopputuotteena saatiin toimiva ja helppokäyttöinen mobiilisovellus sekä palvelintoteutus, jotka täyttivät työlle asetetut vaatimukset. Sovellusta käyttäneiden, vaihtelevasti teknologiaa hallitsevien henkilöiden yleinen mielipide oli se, että sovellus on helppo ja suoraviivainen käyttää sekä sen ulkoasu miellytti silmää. Myös tilaajan testihenkilöiltä, joilla oli pääsy myös karttanäkymään, tuli positiivista palautetta.

Projekti opetti minulle paljon Xamarin.Formsin käytöstä sekä alustariippumattomien mobiilisovellusten kehitystyöstä. Vaikka Xamarinia voidaan käyttää myös pelkästään Android-sovellusten kehitystä varten, koen Android Studion olevan parempi sekä siihen valmiimpi kehitysympäristö. Alustariippumattomien sovellusten kehitykseen oletusarvoisesti käyttäisin jotain olemassa olevista yleisesti hyväksi todetuista web-pohjaisista frameworkeista, jos se projektin luonteesta perusteella olisi mahdollista.

Sain myös kokemuksia ASP.NET Coren käyttämisestä palvelintoteutuksen luomista varten sekä Azuren CosmosDB-tietokannasta. ASP.NET Core vastaa suurimmilta osin .NET Frameworkia joitain poikkeuksia lukuun ottamatta. Alusta vaikutti nopealta ja kehitystyö sekä testaaminen vaikuttivat suoraviivaiselta natiivin Visual Studio –tuen johdosta. CosmosDB osoittautui toimivaksi tietokantaratkaisuksi alussa ilmenneiden ongelmien jälkeen, joihin oli jokseenkin vaikea päästä kiinni puutteellisten virheviestien johdosta. Suhtautuisin siihen kuitenkin varauksella skaalaltaan suurien, monimutkaisten projektien tietokantaratkaisuna, koska

se ei nykyisellään tarjoa samoja ominaisuuksia kuin perinteisemmät SQL-ratkaisut.

Sovellus on myös tarkoitus laittaa yleiseen jakeluun Google Play -kauppaan, mutta opinnäytetyön kirjoitushetkellä sitä ei ole vielä tehty. Sovelluksen toimintaa iOS-laitteilla ei päästy kokeilemaan ennen opinnäytetyön julkistamista.

LÄHTEET

1. .NET Core vs .NET Framework: How to Pick a .NET Runtime for an Application. 2017. Stackify. Saatavissa: https://stackify.com/net-core-vs-net-framework/. Hakupäivä 21.12.2017.

2. Web Apps overview. 2017. Microsoft Azure, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-overview. Hakupäivä 21.12.2017.

3. Mobile Operating System Market Share Finland. 2017. StatCounter, GlobalStats. Saatavissa: http://gs.statcounter.com/os-market-share/mobile/finland. Hakupäivä 20.12.2017.

4. Robinson, David 2017. Exploring the State of Mobile Development with Stack Overflow Trends. StackOverflow. Saatavissa: https://stackoverflow.blog/2017/05/16/exploring-state-mobile-development-stack-overflow-trends/. Hakupäivä 20.12.2017.

5. Pedley, Adam 2017. Flutter Could Be Xamarin's Next Big Competitor. Xamarin Help. Saatavissa: https://xamarinhelp.com/flutter-xamarins-next-big-competitor/. Hakupäivä 3.1.2018.

6. Guthrie, Scott 2016. Microsoft to acquire Xamarin and empower more developers to build apps on any device. Microsoft. Saatavissa: https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/. Hakupäivä 11.1.2018.

7. Part 1. Getting Started with XAML. 2018. Microsoft. Saatavissa: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/get-started-with-xaml?tabs=vswin. Hakupäivä 17.05.2018.

8. Introduction to ASP.NET Core. 2018. Microsoft. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/. Hakupäivä 8.3.2018.

9. Welcome to Azure Cosmos DB. 2018. Microsoft Azure, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction. Hakupäivä 8.3.2018.

10. Web Apps overview. 2017. Microsoft Azure, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-overview. Hakupäivä 8.3.2018.

11. Azure AD B2C: Focus on your app, let us worry about sign-up and sign-in. 2016. Microsoft Azure, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-overview. Hakupäivä 25.2.2018.

12. Hierarchical Navigation. 2017. Microsoft Xamarin, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical. Hakupäivä 22.5.2018.

13. App Class. 2016. Microsoft Xamarin, Microsoft. Saatavissa: https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/application-class/#Properties_Dictionary. Hakupäivä 22.5.2018.