SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

# FORMS-MOBIILISOVELLUS

## Taustatutkimus, suunnittelu ja toteutus

TEKIJÄ: Jesse Jaakola

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

| | |
|---|---|
| Koulutusala<br>Tekniikan ja liikenteen ala | |
| Koulutusohjelma/Tutkinto-ohjelma<br>Tietotekniikan koulutusohjelma | |
| Työn tekijä(t)<br>Jesse Jaakola | |
| Työn nimi<br>Forms-mobiilisovellus – Taustatutkimus, suunnittelu ja toteutus | |
| Päiväys 1.6.2019 | Sivumäärä/Liitteet 38/6 |
| Ohjaaja(t)<br>Jussi Koistinen, lehtori, Keijo Kuosmanen, lehtori | |
| Toimeksiantaja/Yhteistyökumppani(t)<br>TCD Consulting and Research Oy | |


Tiivistelmä

Opinnäytetyön tarkoituksena oli toteuttaa alustariippumaton mobiilisovellus, jonka avulla voidaan täyttää ja tarkastella lomakkeita. Forms-mobiilisovelluksen toiminta perustuu aiemmin .NET-ohjelmistokehyksellä luotuun lomakeohjelmistoon, TCD Forms-ohjelmistoon. Opinnäytetyön toimeksiantaja on TCD Consulting and Research osakeyhtiö, joka tarjoaa Forms-ohjelmistoa.

Opinnäytetyön aikana selvitettiin ja tutkittiin eri ohjelmistokehyksiä, joilla projekti on käytännöllistä toteuttaa. Tutkimustyön tuloksen pohjalta valittiin projektin ohjelmistokehykseksi Xamarin.Forms, jonka avulla voidaan luoda sovellus monelle alustalle lähes yhdellä koodipohjalla.

Opinnäytetyön lopputuloksena saatiin toimiva mobiilisovellus Android- ja iOS -järjestelmäalustoille. Työn lopullinen toteutus täytti kaikki ennalta määritellyt kriteerit niin, että sovellus voitiin julkaista beetatestaukseen. Valikoitu tekniikka, jolla sovellus luotiin vaikutti hyvin lupaavalta projektia aloittaessa, mutta osoittautui käytännössä vielä ongelmalliseksi, sillä kehitystyöhön käytettävää aikaa kului kehyksen tarjoamien työkalujen ja kirjastojen säätämiseen.



Avainsanat
Mobiilisovelluskehitys, Alustariippumattomuus, Xamarin.Forms, Android, iOS

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

| | | | |
|---|---|---|---|
| Field of Study<br>Technology, Communication and Transport | | | |
| Degree Programme<br>Degree Programme in Information Technology | | | |
| Author(s)<br>Jesse Jaakola | | | |
| Title of Thesis<br>Forms Mobile Application – Background Research, Planning and Implementation | | | |
| Date | 1 June 2019 | Pages/Appendices | 38/6 |
| Supervisor(s)<br>Mr Jussi Koistinen, Senior Lecturer, Mr Keijo Kuosmanen, Senior Lecturer | | | |
| Client Organisation /Partners<br>TCD Consulting and Research Ltd | | | |


Abstract

The goal of this thesis was to implement a cross-platform mobile application that can be used to view and fill forms. The Forms mobile application is based on the TCD Forms software which is created with the .NET Framework. The thesis was commissioned by TCD Consulting and Research Ltd, the company behind Forms-software.

During the thesis, various mobile application platforms and software frameworks that could be used in the project were studied. Based on the research results, Xamarin.Forms was selected as the project's software framework. With Xamarin.Forms you can create an application for many platforms using mostly a single code base.

The final result of the thesis was a functional mobile application for Android and iOS system platforms. The Forms mobile application fulfilled all the pre-defined criteria for the project, so the application was published for beta testing. The selected technology, Xamarin.Forms, seemed very promising in the beginning of the development. However, it turned out to be problematic because some of the time reserved on the development work was spent adjusting Xamarin.Forms's tools and libraries.



Keywords
Mobile app development, Cross-platform, Xamarin.Forms, Android, iOS

SISÄLTÖ

# 1 JOHDANTO

Mobiililaitteet kuuluvat nykypäivän työ-, opiskelu- ja vapaa-aikaan hyvin läheisesti. Ne ovat matkassa mukana lähes aina, ja niitä käytetään missä ja milloin tahansa. Ne ovat suurelle osalle ihmisiä pakollisia työvälineitä, jotka usein helpottavat ja parantavat päivittäistä työelämää. Erään tutkimuksen mukaan huomattiin että noin 60% yrityksen työntekijöistä on sitä mieltä, että mobiililaitteiden tarjoamat hyödyt ovat lisänneet heidän produktiivisuuttaan työelämässä. Osa tutkimukseen osallistuneista työntekijöistä myös ilmoittivat että mobiililaitteiden käyttö työajalla on parantanut heidän luovuutta huomattavasti. (The Economist Intelligence Unit, 2016 s. 15) Toisin kuin ajatellaan, puhelimen käyttö työajalla ei toimikaan häiriötekijänä vaan saattaa parhaassa tapauksessa parantaa yksilön työnteon tehokkuutta sekä laatua.

Samalla kun mobiililaitteiden käyttö on kasvanut hurjaa vauhtia, erilaiset mobiililaitteiden alustat ja tyypit ovat myös lisääntyneet. Mobiilialustojen jakautuminen on aiheuttanut sen, että sovelluksen kehittäjän on yhä vaikeampaa tavoittaa valtaosaa mobiililaitteiden käyttäjäkuntaa sovelluksellaan. Mobiilisovelluksen kehittäminen useammalle alustalle vaatii usein eri laitetyyppien ja ohjelmistokehyksien osaamisen, joten näin työn määrä kasvaa suhteessa alustojen lukumäärään. Alustariippumattoman mobiilisovelluksen tuottaminen sitä tukevalla ohjelmistokehyksellä nopeuttaa sovelluksen kehittämistä useammalle alustalle, sillä sovellus voidaan ohjelmoida vain yhdelle koodipohjalle.

## 1.1 Tausta

Tämän työn toimeksiantajana on TCD Consulting and Research osakeyhtiö. TCD Consulting and Research Oy:n toimialaa on liikkeejohtamisen konsultointi-, koulutus- ja tieteelliset tutkimuspalvelut, sekä ohjelmistojen suunnittelu-, tuotanto- ja markkinointi. Työn toimeksiantaja haluaa heidän tarjoamastaan Forms-ohjelmistosta sopivan mobiilisovellustoteutuksen. Sovelluksen tarkoituksena on parantaa palvelun kysyntää ja markkinoita.

Forms-ohjelmisto on yhtiön asiakkaille tarkoitettu verkossa toimiva lomaketyökalu. Työkalun avulla palvelun käyttäjä pystyy luomaan, tarkastelemaan, muokkaamaan ja täyttämään verkossa käytettäviä lomakkeita.

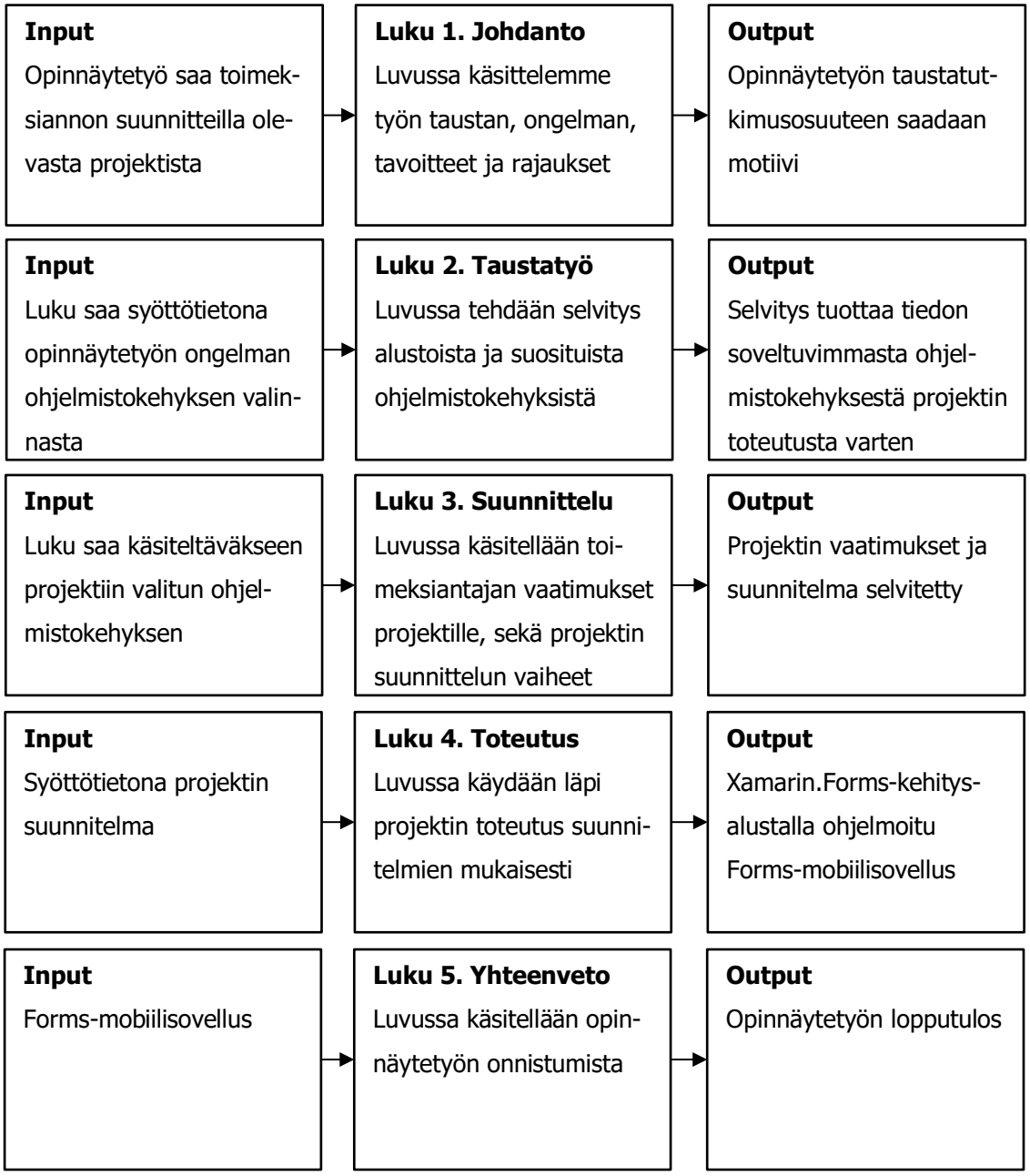
## 1.2 Tavoitteet ja rajaukset

Opinnäytetyön aihe, Forms-mobiilisovellus, on katsaus siitä mitä on suunnitella ja kehittää alustariippumaton mobiilisovellus nykyaikaisilla tekniikoilla jo olemassa olevasta verkkosovelluksesta. Työhön kuuluvat seuraavat osa-alueet: taustatutkimus, suunnittelu ja toteutus. Taustatutkimuksen tavoitteena on tehdä selvitys saatavilla olevista mobiiliohjelmoinnissa käytettävistä kehitystekniikoista. Suunnitteluosiossa tutustutaan projektin toimeksiantoon, jonka jälkeen taustatutkimuksen perusteella valitaan sopivin tekniikka sovelluksen tuottamista varten. Toteutusvaiheessa käydään läpi projektin toteutusta ja lopputulosta.

Työssä keskitytään mobiilisovelluksen kehittämiseen iOS- ja Android -alustoille, sillä nämä järjestelmät ovat vallanneet kaikkien mobiilijärjestelmien maailmanlaajuisesta markkinasuudesta yli 97% (StatCounter, 2019). Opinnäytetyössä etsitään Forms-projektiin sopiva kehitystekniikka, mikä vastaa toimeksiantajan ja työntekijän kriteerejä projektille. Toimeksiantajan näkökulmasta on tärkeää pitää projektin jatkuvuus mahdollisena niin, että sovelluksen ylläpitäminen, jatkokehitys ja päivittäminen ei aiheita tulevaisuudessa ongelmia.

## 1.3 Työn rakenne

Tämän opinnäytetyön rakenne koostuu taustatutkimuksesta ja ohjelmistotuotannon eri vaiheista. Työn vaiheita kuvaa seuraava Kaavio 1, jossa kuvataan opinnäytetyön rakenne yksityiskohtaisesti input-output-kaaviona.




Kaavio 1. Opinnäytetyön rakenne input-output-kaaviossa

## 1.4 Lyhenteet ja määritelmät

Ohjelmistokehys = alusta, jonka pohjalle ohjelmoija voi toteuttaa oman koodin
Käyttöliittymäkirjasto = käyttöliittymän ohjelmoimisessa hyödynnettäviä valmiita graafisia komponentteja sisältävä kokoelma
Alustariippumattomuus = cross-platform, ohjelmisto, joka ei ole sidoksissa tiettyyn käyttöjärjestelmään
HTML5 = HTML-kuvauskielen versio, jolla voidaan kuvata käyttöliittymässä esiintyviä elementtejä
CSS = tekniikka, jolla määritellään HTML-kuvauskielellä luodulle näkymälle ulkoasu
JavaScript = ohjelmointikieli, jolla voidaan luoda toimintoja HTML-kuvauskielen elementeille
MIT-lisenssi = vapaan ohjelmiston lisenssi – sallii lähdekoodin käytön kaupallisissa suljetun lähdekoodin ohjelmistoissa (Open Source Initiative)
Apache 2.0 lisenssi = vapaan ohjelmiston lisenssi – sallii lähdekoodin käytön vapaan, avoimen, omistetun ja suljetun koodin kehittämiseen (Apache, 2004)
Ohjelmointirajapinta = ohjelmiston tarjoama rajapinta, jolla voidaan tarjota tietoa toisiin järjestelmiin
Abstrakti kerros = ohjelmointirajapinnan sisältävä kerros, jonka avulla ohjelmoijan ei tarvitse olla tietoinen rajapinnan alaisista metodeista
CLR = Common language runtime, ajonaikainen käyttöympäristö, jossa voidaan suorittaa .NET ohjelmia
CLI = Command-line interface, komentorivi, johon voidaan suorittaa järjestelmän toimintoja
Node.js = avoimen lähdekoodin palvelin ympäristö (W3Schools, 2019)
npm = node package manager, kirjastojenhallintatyökalu
BCL = Base Class Library, perusluokkakirjasto
JSON = JavaScript Object Notation, kevyehkö tiedonvälitys formaatti (Ecma International, 2017)
APK = Android application package, Android-sovelluksen tiedostotyyppi

# 2 TAUSTATUTKIMUS

Opinnäytetyön alussa tehtiin selvitys saatavilla olevista kehitystekniikoista. Taustatutkimuksen tavoitteena oli perehtyä eri sovellustyyppeihin ja ohjelmistokehyksiin. Tekniikan etsiminen oli haasteliasta, sillä erilaisia tekniikoita on suuri määrä saatavilla. Projektin toteutusta mahdollistavia kehitysalustoja pystyttiin kuitenkin rajaamaan hyvin, sillä sovellus pitää tuottaa alustariippumattomaksi. Tarkoittaen sitä, että lopputuote on oltava saatavilla sekä Android- ja iOS -alustoille. Ongelmaksi ilmenee natiivitoteutusten vaatimat resurssit, sillä molemmat alustat omaavat erilaiset kehitysympäristöt ja ohjelmointikielet.

## 2.1 Sovellustyypit

Mobiiliapplikaatiot jaetaan pääosin kolmeen eri sovellustyyppin: natiivi-, hybridi- ja web-applikaatioihin. Natiiviapplikaatio on tarkoitettu kohdentamaan tiettyä alustaa kuten Android-, iOS- tai Universal Windows -alustaa. Hybridiapplikaatiot kehitetään kohdentamaan useampaa alustaa yhdellä koodipohjalla ja web-applikaatiot ovat mobiili-optimoituja verkkosivuja, joiden tarkoitus on näyttää samalta kuin laitteelle asennettavat applikaatiot.

### 2.1.1 Natiivisovellus

Natiivisovellus on laitteelle asennettava applikaatio, joka usein pystyy toimimaan ilman internet-yhteyttä, riippuen applikaation luonteesta. Natiivisovellus asennetaa käyttämällä sovelluskauppaa, kuten Google Play tai Applen App Store. Sovellus kirjoitetaan alustalle tarkoitetulla ohjelmointikielellä. Ohjelmointikielet ovat tyypillisesti iOS-järjestelmälle Objective-C tai Swift ja Android-järjestelmälle Java tai Kotlin. (Butusov, 2019) Molempien alustojen ulkoasu suunnitellaan XML-merkintäkielellä. Natiiviapplikaatioiden kehittäminen vaatii oman ohjelmointiympäristönsä, joka on tarkoitettu kyseisille ohjelmointitekniikoille. iOS-sovelluksia kehitetään Xcode-ohjelmointiympäristöllä ja Android-sovelluksia kehitetään Android Studio -ohjelmointiympäristöllä.

Hyödyt

- Pystyvät hyödyntämään kaikkia laiteominaisuuksia
- Ei tarvitse internet-yhteyttä toimiakseen
- Alustalle optimoitu ohjelmistokehys

Heikkoudet

- Jokaiselle alustalle oma koodipohja
- Ylläpito ja päivittäminen useammassa sovelluskaupassa työlästä
- Kehittäminen aikaavievää

### 2.1.2 Hybridisovellus

Hybridisovellus, kuten natiivisovellus, on itse mobiililaitteelle asennettava applikaatio. Hybridisovelluksia on kahdenlaisia: sovellus, joka on toteutettu verkkotekniikoin, sekä sovellus, joka on kehitetty käyttämällä lähes natiivinomaista ohjelmistokehystä.

### 2.1.3 Hybridisovellus, verkkotekniikat

Verkkotekniikoilla (usein HTML5, CSS ja JavaScript) kehitetty hybridisovellus toimii kuten natiivisovellus, mutta hyödyntää laitteen selaimen moottoria renderöidäkseen HTML-dokumentin ja prosessoidakseen JavaScriptin paikallisesti. Tämä verkkosivusta natiivitoteutukseen mahdollistaa sovelluksen pääsyn laitteen natiiviominaisuuksiin, joita ei pelkällä selaimella käytettävällä verkkosivulla ole mahdollista käyttää. Näitä natiiviominaisuuksia ovat usein laitteen GPS-järjestelmä, kamera, kiihtyvyysanturi, kompassi, kontaktiluettelo, notifikaatiot ja paikallinen tallennustila.

Hybridisovellus käyttää hyödykseen ns. verkkosivunäkymä-järjestelmäkomponenttia (Android-järjestelmässä WebView ja iOS-järjestelmässä UIWebView), jonka avulla voidaan näyttää verkkosisältöä kokoruututilassa. Sisällön näyttämisen lisäksi hybriditoteutuksessa voidaan käyttää laitteen natiiviominaisuuksia hyödyntämällä kehykseen luotua ohjelmointirajapintaa. Tällainen alustan natiivi ohjelmointirajapinta on usein toteutettu, ohjelmistokehyksestä riippuen, JavaScript abstraktikerroksella. (Cowart, 2012) Tämä tarkoittaa sitä että kehittäessä mobiiliapplikaatiota ohjelmoija pääsee natiiviominaisuuksiin käsiksi käyttämällä JavaScriptiä.

### 2.1.4 Hybridisovellus, cross-platform

Cross-Platform, eli alustariippumaton applikaatio, kehitetään käyttämällä jotain ohjelmointikehystä, mikä eroaa alustan omasta ohjelmistokehyksestä usein käyttämällä muita kuvaus- ja ohjelmointikieliä. Tämä ohjelmistokehys ei kuitenkaan hyödynnä WebView-komponenttia, vaan ohjelmistokehys tarjoaa käytettävät komponentit, jotka rakentuvat oikeiksi natiiviapplikaation komponenteiksi. (Cowart, 2012)

Hybridiapplikaation ohjelmoimisessa on erityisen hyvää se, ettei sovelluksen kehittäjällä tarvitse olla erityistä osaamista mobiiliohjelmoinnista. Tätä vaihtoehtoa voivat hyödyntää tehokkaasti ohjelmoijat, joilla on osaaminen hallussa verkko-ohjelmoinnin alueelta. (Traversy Media, 2017) Käytettävissä olevien työkalujen avulla sovelluksen kehitysvaiheessa pystytään säästämään aikaa ja resursseja huomattavasti, sillä kehitykseen ei välttämättä tarvita kuin yksi ohjelmoija tai ryhmä toteuttamaan applikaatio kehyksen tukeville alustoille yhdellä koodipohjalla.

Hybridisovellukset eivät kuitenkaan ole vastaus kaikkiin mobiiliohjelmistotuotannossa esiintyviin ongelmiin. Jos suunnitteilla oleva sovellus tarvitsee paljon laitteen resursseja tai vaatii paljon prosessoitavaa logiikkaa, niin silloin verkkotekniikoin kehitetyissä hybridisovelluksissa käytettävä verkkosovelluksiin tarkoitettu selainmoottori voi olla riittämätön. Cross-platform hybridisovelluksen kehittämiseen käytettävät ohjelmistokehykset saattavat ajoittain tuntua myös puutteellisilta, sillä natiiviappli-

kaation kehitykseen tarkoitetut ohjelmistokehykset päivittyvät kovaa vauhtia ja usein kehitysvaiheessa joutuukin tarkistamaan tukeeko ohjelmistokehys tiettyjä laitekohtaista toimintoa. (Butusov, 2019)

Hyödyt:

- Nopea kehittää useammalle alustalle kerralla
- Pystyy käyttämään natiivitoimintoja
- Lähtökohtaisesti yksi koodipohja
- Toimii ilman internet-yhteyttä
- Jakelu toimii kätevästi sovelluskaupoista

Heikkoudet:

- Riippuvainen laitteen suorituskyvystä
- Ei välttämättä tue kaikkia natiivitoimintoja tai -komponentteja
- Osa toiminnoista tulee toteuttaa täysin natiivina

### 2.1.5 Web-sovellus

Mobiililaitteelle tarkoitettu web-sovellus on yksinkertaisuudessaan vain verkkosivusto, minkä tarkoituksena on jäljitellä natiiviapplikaation interaktiivisuutta ja toiminnallisuutta. Web-sovelluksia ei ole tarkoitettu ladattavaksi tai asennettavaksi, vaan niitä käytetään laitteelle asennetulla selainsovelluksella ja verkon välityksellä. (Blair, 2017)

Jokaiselle mobiilikäyttöjärjestelmälle on oma SDK (Software Development Kit), mitä ohjelmoijat käyttävät applikaation kehitykseen. Näin ei kuitenkaan ole web-sovelluksessa, vaan kehitys tapahtuu käyttämällä perinteisiä web-ympäristössä käytettäviä ohjelmointi- ja kuvauskieliä. Mobiililaitteille optimoidut web-sovellukset rakennetaan palvelinympäristöön käyttämällä usein PHP, Node.js tai ASP.NET web-ohjelmistokehyksiä (Cowart, 2012). Web-ohjelmistokehys renderöi näkymän mobiililaitteille sopivaksi.

Hyödyt:

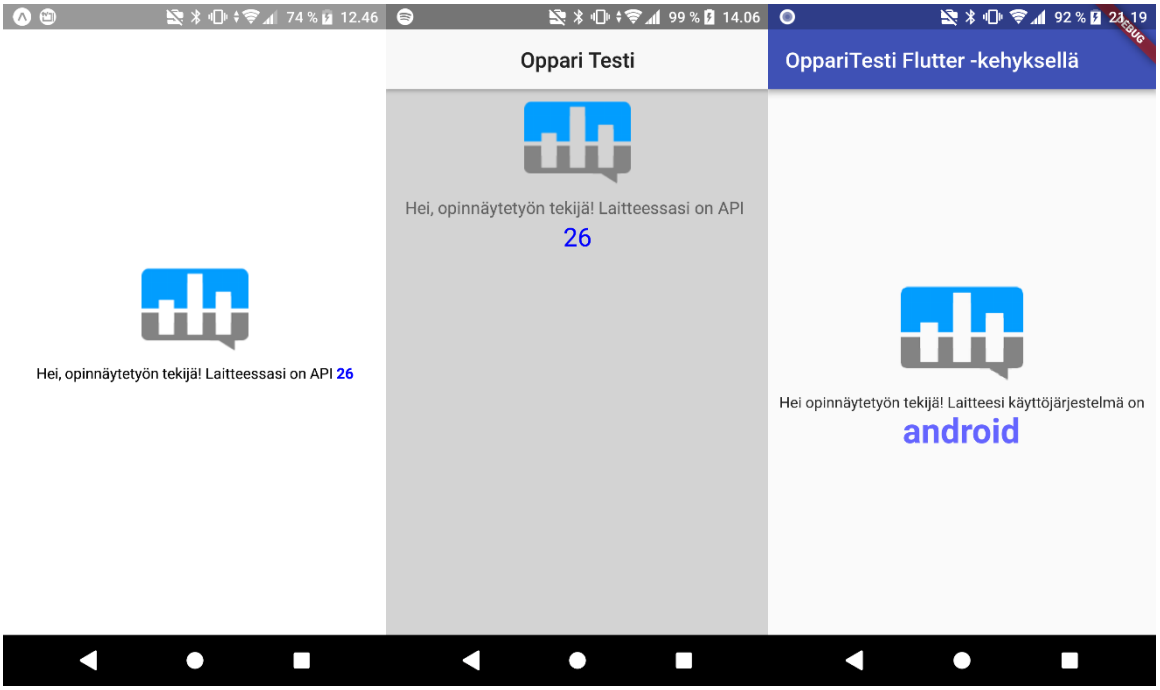
- Kehitykseen käytettävä aika on usein lyhyempi kuin natiivi- tai hybridiapplikaatioiden kohdalla, joten kustannukset ovat pienemmät
- Julkaisu, jakelu ja päivittäminen helpompaa, sillä ei tarvita erillistä alustaa tekemään tätä
- Vain yksi koodipohja kaikille alustoille

Heikkoudet:

- Sovellus ei pysty hyödyntämään alustan natiiviominaisuuksia
- Ei toimi ilman internet-yhteyttä
- Huono suorituskyky vanhemmilla mobiililaitteilla ja verkkoselaimilla

## 2.2 Ohjelmistokehykset

Tässä opinnäytetyössä taustatutkimukseen kuuluu eri tekniikoiden kokeilu Forms-applikaatiota varten, joten osiossa käydään muutamia projektiin sopivia mobiiliohjelmointiin tarkoitettuja ohjelmistokehyksiä lyhyesti läpi. Jotta samaa applikaatiota ei tehtäisi kahteen kertaan, ohjelmistokehykset rajattiin sellaisiin kehyksiin, joilla pystyy tekemään alustariippumattoman sovelluksen. Ohjelmistokehyksen testaus tapahtuu luomalla sen tarjoamalla kehitysympäristöllä testiapplikaatio. Testausta varten luotu applikaatio on laitteelle asennettava sovellus, jossa kokeillaan tulostaa tekstiä ja käyttää jotain ohjelmistokehykselle ominaista järjestelmäkomponenttia. Testauksen jälkeen arvioidaan lyhyesti miten ohjelmistokehys sopii tulevaan projektiin.




Kuva 1. React Nativella (vas.), NativeScriptillä ja Flutterilla (oik.) toteutetut testiapplikaatiot

### 2.2.1 React Native

React Native on Facebookin kehittämä avoimen lähdekoodin ohjelmistokehys, jolla voidaan luoda natiiviapplikaatioita Android- ja iOS -alustoille. Se perustuu Facebookin ylläpitämään React JavaScript -kirjastoon, joka on tarkoitettu mobiiliverkkosivujen kehittämiseen. – Kuten React-kirjasto, React Native hyödyntää JSX JavaScript lisäosaa, jonka avulla voidaan kirjoittaa JavaScript ohjelmia jotka näyttävät HTML-syntaksilta. (Bonney, 2015)

React Native -ohjelmistokehys voidaan asentaa verkosta ladattavalla asennusohjelmalla. Sovelluksen voi kehittää käyttämällä joko Expo- tai React Native CLI -työkaluja. Testiapplikaation rakentamisessa ja virheenjäljityksessä käytettiin Expo-komentoriviä ja -asiakasohjelmaa. React Native ohjelmistokehys ja Expo CLI asennettiin testiympäristöön käyttämällä Node.js Runtime -ympäristöä, projektin ohjelmoimiseen käytettiin Visual Studio Codea.

Hyvät puolet

- Avoin lähdekoodi, MIT-lisenssi
- Renderöi oikeita natiivikomponentteja
- Käyttää React -kirjastoa, joka on entuudestaan tuttu
- Käyttöliittymän kehittäminen todella nopeaa

Huonot puolet

- Ei tue kaikkia natiivikomponentteja
- React Native täysin riippuvainen Facebookista

### 2.2.2 NativeScript

NativeScript on avoimen lähdekoodin ohjelmistokehys, jolla voidaan tuottaa applikaatio iOS- ja Android -alustoille yhdestä JavaScript koodipohjasta. NativeScriptillä voidaan kehittää applikaatio hyödyntämällä joko Angular, Vue.js tai natiivia JavaScript tekniikkaa. (NativeScript, 2018) NativeScript rakentuu useasta osiosta: suorituspalveluista, CLI:stä ja plugineista. NativeScriptin ohjelmointirajapintoja voidaan hyödyntää käyttämällä JavaScript, TypeScript, Angular tai Vue.js -ohjelmistokehyksiä. (NativeScript, 2019)

NativeScript ohjelmistokehys asennettiin, kuten React Native, testiympäristöön käyttämällä Node.js-ympäristön npm-työkalua. Testisovelluksen kehittämiseen käytettiin Visual Studio Code -tekstieditoria.

Hyvät puolet

- Avoin lähdekoodi, Apache 2.0 software lisenssi
- Kehitysnopeus

Huonot puolet

- Käyttöliittymän rakentaminen tuntuu alkeelliselta vain JavaScript- ja XML -kieliä hyödyntäen
- Tekniikan hallitsemisen hankaluus, pitkä oppimiskaari

### 2.2.3 Flutter

Flutter on Googlen tarjoama avoimen lähdekoodin mobiiliohjelmointikehys. Flutter applikaatiot ohjelmoidaan käyttämällä Dart-ohjelmointikieltä, mikä pystytään kääntämään natiivitoteutukseksi Android- ja iOS-alustoille. – Flutter ei käytä minkäänlaista siltaa hallitakseen käytössä olevaa alustaa, toisin kuin React Native- ja NativeScript ohjelmistokehyksissä, vaan ohjelmakoodi käännetään suoraan alustalle sopivaksi. (Stoll, 2018)

Hyvät puolet

- Avoin lähdekoodi
- Käyttöliittymän kehitysnopeus

Huonot puolet

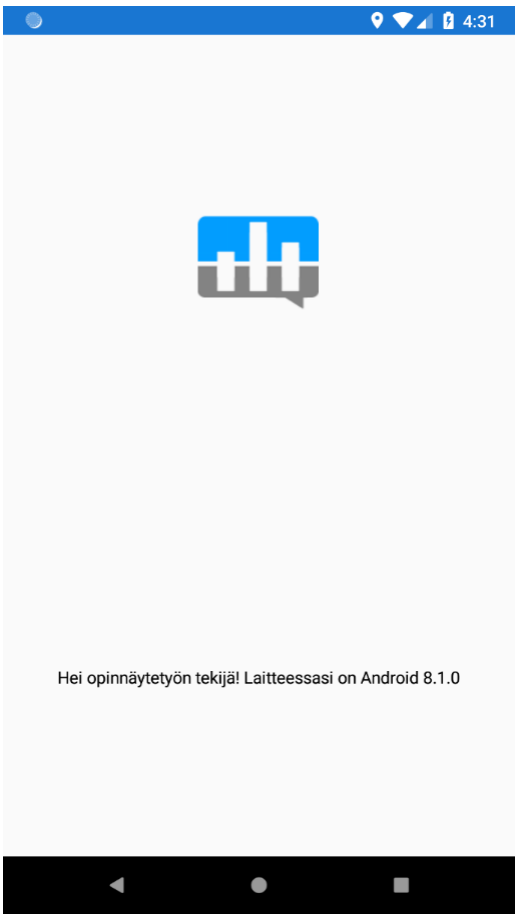
- Osa natiiviominaisuuksista pitää toteuttaa alustakohtaisesti
- Suppeampi kirjastovalikoima

### 2.2.4 Xamarin.Forms

Xamarin on Microsoftin tarjoama avoimen lähdekoodin sovellusalusta natiivimobiiliapplikaatioiden kehittämiseen .NET-ohjelmistokehyksellä. .NET-ohjelmistokehys on Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota voidaan hyödyntää Microsoft Visual Studio -kehitysympäristössä. .NET koostuu työkaluista, ohjelmointikielistä ja -kirjastoista, joilla voidaan kehittää applikaatioita eri järjestelmäalustoille. Xamarin laajentaa .NET-alustaa tarjoamalla mobiiliapplikaation kehitykseen tarkoitettuja työkaluja ja kirjastoja (Microsoft, 2019).

Xamarin ohjelmistokehys perustuu avoimen lähdekoodin Mono-ohjelmistokehyksen konseptiin, joka sisältää perustyökalut sovelluksen kehittämiseen. Näitä työkaluja ovat: C#-kääntäjä, kehitysympäristö, ajonaikainen käyttöympäristö (CLR) ja .NET-kirjastot. Xamarinin kehittäjien ideana oli luoda yksi ohjelmistokehys, jonka avulla voidaan luoda C#-ohjelmointikielellä sovelluksia muihinkin järjestelmiin kuin Windows-käyttöjärjestelmään. Xamarin pitää sisällään erilaisia alustakohtaisia ohjelmointikielen kääntäjiä, joiden avulla Xamarin pystyy hyödyntämään natiiveja ohjelmointirajapintoja. (Applikey Team, 2018)

Xamarin.Forms on käyttöliittymän suunnitelua ja yhden koodipohjan toteuttamista varten oleva ohjelmistokehys, jonka avulla kehittäjä pystyy luomaan yhden käyttöliittymän ja koodipohjan sovellukselle, joka voidaan jakaa iOS-, Android- ja Universal Windows -alustojen välillä (Microsoft, 2019).




Kuva 2. Xamarin.Forms-kehyksellä luotu applikaatio

Hyvät puolet

- Tuloksena lähes täysin natiiviapplikaatio
- Xamarinia tukee suuri määrä plugineita, joilla voidaan toteuttaa vaivattomasti natiiviominaisuuksia hyödyntäviä toimintoja
- Käyttöliittymän suorituskyky

Huonot puolet

- Käyttöliittymän suunnittelu ja kehittäminen työlästä
- Eri alustoille ylimääräisen työn riskit
- Pitkä oppimiskaari jos .NET ohjelmistokehys ei ole entuudesta tuttu

# 3 SUUNNITTELU

## 3.1 Toimeksianto

Opinnäytetyössä toteutetaan natiivi mobiilisovellus, jolla voidaan täyttää Forms-ohjelmiston lomakkeita. Työhön valitaan sopiva tekniikka, jonka avulla voidaan toteuttaa alustariippumaton sovellus, tarkoittaen sitä, että sovelluksen on toimittava sekä iOS- ja Android-alustoilla. Toimeksiantajan ehdotuksia olivat Xamarin, NativeScript ja React Native.

### 3.1.1 Vaatimusten määrittely

Tässä luvussa käydään toimeksiantajan lähtökohtaiset vaatimukset Forms-mobiilisovellukselle. Vaatimukset käytiin läpi toimeksiantajan kanssa palavereissa ja ohjeistustilaisuuksissa. Taulukko 1 on esitelty toiminnot, jotka Forms-mobiilisovelluksen pitää pystyä suorittamaan.

| Vaatimus id | Vaatimuksen selitys |
|---|---|
| R1 | Verkkoyhteyden käyttäminen, http-pyyntöjen suoritus |
| R2 | JSON-muotoisen datan käsittely |
| R3 | Tiedostonhallinta ja kameran käyttö |
| R4 | GPS sijaintitietojen käyttäminen |
| R5 | Tallentaa lomake puhelimen muistiin, jotta sovellus voi toimia offline-tilassa |
| R6 | Lomake tulee pystyä lataamaan joko URL-osoitteella, QR-koodilla tai NFC tagilla |
| R7 | Kirjautuminen sovellukseen TCD tunnuksilla, session tallennus evästeeseen |
| R8 | Muistissa olevien lomakkeiden päivitys |

Taulukko 1. Lomakesovelluksen vaatimukset

### 3.1.2 Yksinkertaiset elementtityypit

Forms-ohjelmisto tarjoaa käyttäjälle 17 erilaista komponenttia, mistä Forms-lomakkeen pystyy rakentamaan. Forms-mobiiliapplikaation tulee tukea ainakin kaikkia yksinkertaisia elementtityyppejä.

| Elementin id | Elementin nimi (komponentin nimi) |
|---|---|
| EL1 | Teksti (Text) |
| EL2 | Otsikko (Header) |
| EL3 | Vaakaviiva (HorizontalLine) |
| EL4 | Tekstikenttä (TextInput) |
| EL5 | Tekstialue (TextArea) |
| EL6 | Numerokenttä (NumberInput) |
| EL7 | Dokumenttilinkki (DocumentLink) |
| EL8 | Monivalinta (CheckBox, CheckBoxList) |
| EL9 | Valinta (Radio button list) |
| EL9 | Päivämäärä (DmyPicker) |

| | |
|---|---|
| EL10 | Sijaintitieto (Gps) |

Taulukko 2. Yksinkertaiset elementtityypit

### 3.1.3 Edistykselliset elementtityypit

Edistykselliset lomakkeen elementit ovat projektin jatkokehitystä varten, mutta lähtökohtaisesti ne kuuluvat projektiin olennaisesti lomakkeiden toimivuuden kannalta, joten ne tulee toteuttaa sovellukseen.

| Elementin id | Elementin nimi |
|---|---|
| AEL1 | Tarkastuslista (CheckList) |
| AEL2 | Yksinkertainen tarkastuslista (SimpleCheckList) |
| AEL3 | Tiedosto (FileInput) |
| AEL4 | Tehtävä (TaskInput) |
| AEL5 | Toimipiste (SiteInput) |
| AEL7 | Iframe-kehys (Iframe) |

Taulukko 3. Edistykselliset lomakkeen komponentit

### 3.1.4 Säännöt

Forms-ohjelmiston lomakkeisiin voi määritellä sääntöjä. Säännöillä voidaan hallita lomakkeen käyttäytymistä esimerkiksi antamalla valintanapille sääntö, jonka mukaisesti tämän pitää esittää tai hävittää joitain elementtejä lomakkeelta.

## 3.2 Tarkoitus

Lomakesovelluksia on ohjelmistomarkkinoilla enemmän kuin riittävästi ja sovellukset kutakuinkin toimivat samalla tavalla kuin toiset kilpailijoiden lomakesovellukset. Lähes kaikki lomakesovellukset tarjoavat saman kattauksen lomakkeilla käytettävistä elementeistä ja toiminnoista, näin markkinoilla menestyäkseen ohjelmalla on oltava jotain muutakin tarjottavaa. Tämän työn, Forms-mobiilisovelluksen, tarkoituksena on tukea ja edistää TCD Consulting and Research osakeyhtiön tarjoamaa Forms-ohjelmiston markkinoita laajentamalle sen käyttö mobiililaitteisiin.

## 3.3 Määrittely

Opinnäytetyöhön käytettävä ohjelmistokehyksen valinta on tärkeä prosessi, sillä oikean tekniikan valitseminen on mobiilisovelluksen toimivuuden kannalta erittäin tärkeää. Ohjelmistokehyksellä ei pelkästään luoda sovellusta, vaan sen avulla on myös tehtävä sovelluksesta ylläpidettävä, skaalautuva ja jatkuva.

Ohjelmistokehyksen valintaa varten tehtiin lista vaatimuksista, jotka kehyksen on pystyttävä täyttämään:

- Kehyksen hallitsemiseen kuluva aika

- Kehyksellä on pystyttävä luomaan alustariippumaton sovellus
- Kehyksellä on pystyttävä kehittämään näkymä molemmille alustoille yhdellä koodipohjalla
- Kehyksen on oltava soveltuva toimeksiantajalle ja kehittäjälle

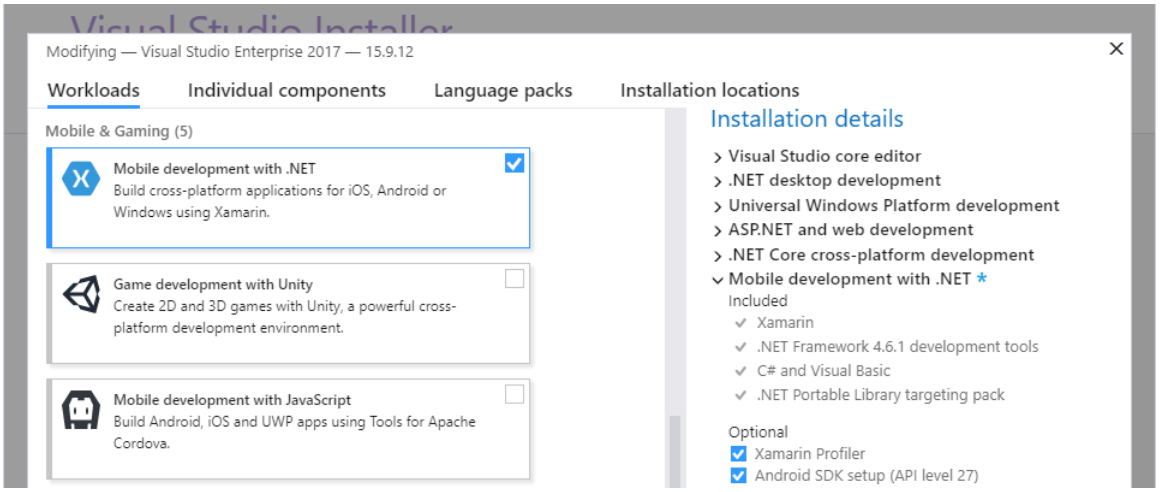
## 3.4 Ohjelmistokehyksen valinta

Forms-mobiilisovelluksen kehittämistä varten valittiin ohjelmistokehykseksi Xamarin Nativea hyödyntävä Xamarin.Forms-ohjelmistokehys. Tekniikka valittiin, sillä se täyttää toimeksiantajan kriteerin applikaation alustariippumattomuudesta, tukee vaadittuja toimintoja, sekä on sopiva kehittäjälle, joka tuntee kehyksessä käytetyt ohjelmointi- ja kuvauskielet. Kehys sopii projektiin hyvin, sillä sen avulla voidaan jakaa sekä loogisen- että käyttöliittymässä käytettävän koodin. Xamarin.Forms-kehys on kelvollinen myös siinä mielessä, koska sen käyttämä .NET-ohjemistokehys on työn toimeksiantajalle tuttu. Xamarin.Forms-kehyksellä luvataan yhtenäistää kaikki koodi jopa 90:een prosenttiyksikköön (Microsoft, 2018).

### 3.4.1 Ohjelmistokehyksen vaatimukset

Xamarin.Forms applikaatioita voidaan kehittää macOS- tai Windows -ympäristöissä. Tässä työssä käytettävien laitteiden käyttöjärjestelmänä on Windows 10, joten Xamarin kehitysympäristö on asennettava Visual Studio -ohjelmankehitysympäristöön. Xamarinin voi asentaa Visual Studio 2017 tai siitä uudempiin versioihin.

Asentaminen tapahtuu käyttämällä Visual Studion asennustyökalua, josta valitaan "Mobile development with .NET" -lisäosa, jonka mukana tulevat Xamarin, .NET-ohjelmistokehys, C# ja Visual Basic, sekä .NET Portable Library. Vaihtoehtoisina lisäosina on valittava niiden alustojen SDK:t joita tullaan kehittämään. (Dunn;ym., 2019)

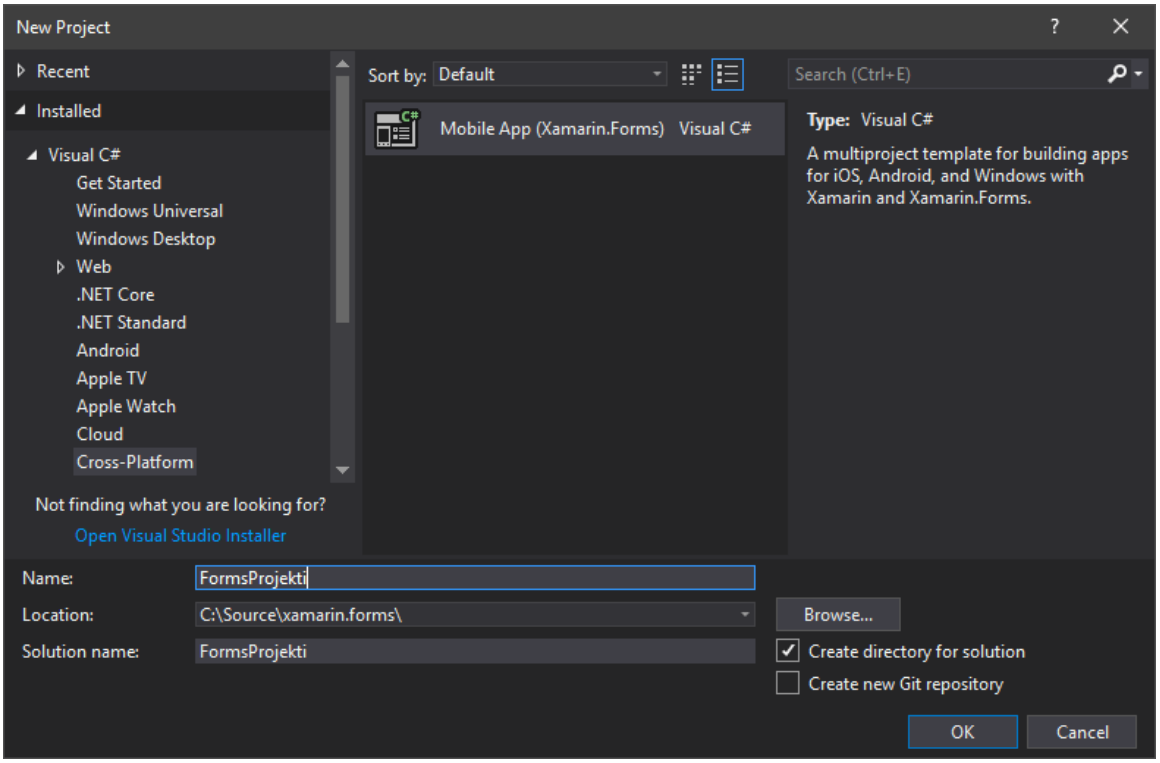



Kuva 3. Alustariippumattoman sovelluksen kehitykseen tarkoitettu lisäosa

# 4 TOTEUTUS

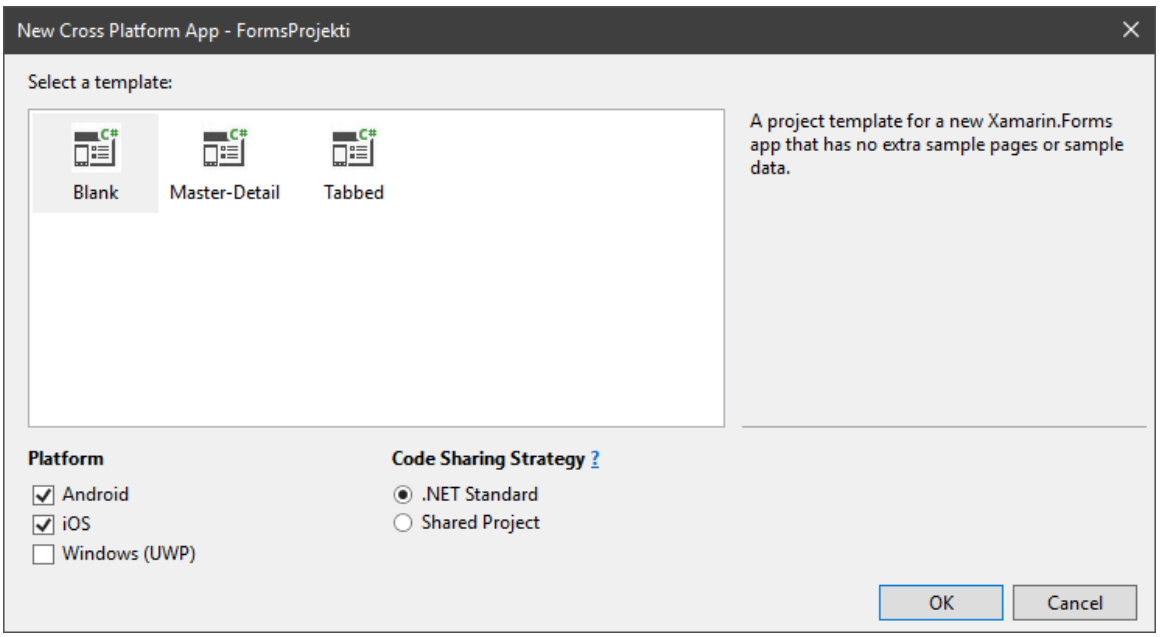
## 4.1 Projektin luominen

Projektin luominen on nopea prosessi, johon Visual Studio -kehitysympäristö tarjoaa yksinkertaisen käyttöliittymän. Xamarin.Forms projekti valitaan käyttämällä Mobile App -multiproject pohjaa (Kuva 4), joka on tarkoitettu alustariippumattoman mobiilisovelluksen kehittämistä varten.




Kuva 4. Uuden projektin luomisnäkymä

Projektin pohjan valinnassa (Kuva 5) on tärkeää huomioida mille alustoille sovellus luodaan ja millä strategialla jaettu koodipohja toimii. Strategioita on kolmea eri tyyppiä: .NET Standard Libraries, Shared Project ja PCL. ”Yhden koodipohjan” -strategian toimintaperiaate perustuu siihen, että useat alustat voivat hyödyntää kertaalleen kirjoitettua koodia rajapintojen välityksellä.




Kuva 5. Alustojen ja koodipohjan jakamisstrategian valinta

### 4.1.1 .NET Standard

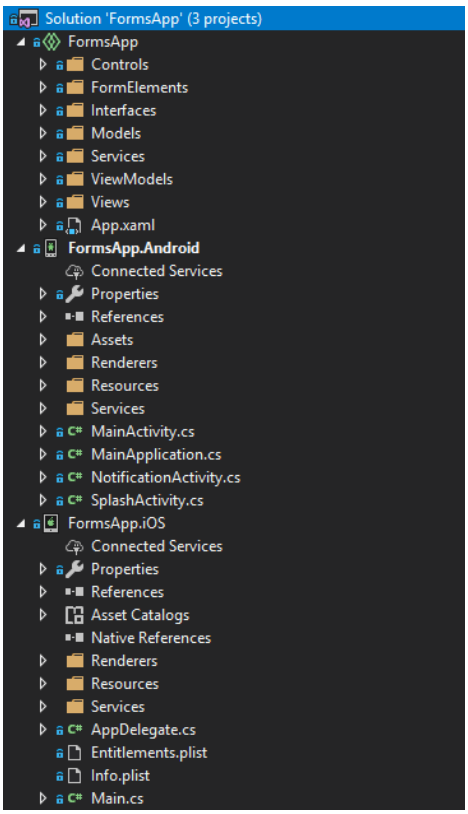
.NET Standard projektissa määritellyt alustat hyödyntävät yhtä koodipohjaa sekä .NET ohjelmointirajapintoja. – NET Standard 2.0 tarjoaa kattavimman listan saatavilla olevista .NET BCL nimiavaruuksista. (Umbaugh;ym., 2018) Xamarin.Forms projekti koostuu .NET Standard Library-, Android- ja iOS -projekteista (Kuva 6). .NET Standard projekti pitää sisällään viittauksia .NET-, Mono- ja .NET Core -ohjelmistokehyksiin, sekä jaetun koodipohjan, minkä kehittäjä haluaa jakaa projektien välillä.

### 4.1.2 Shared Project

Shared Project -strategian avulla alustaprojekteille voidaan jakaa samaa koodia ja resursseja mitkä on määritelty jaetussa projektihakemistossa. – Jaettu projekti hyödyntää kääntäjän direktiivejä (Umbaugh;ym., 2018), esimerkiksi `#if __ANDROID__` -direktiiviä, jonka tarkoituksena on kertoa kääntäjälle, että tämän ehtolauseen sisällä toteutettava koodi suoritetaan vain Android-järjestelmässä.

## 4.2 Rakenne

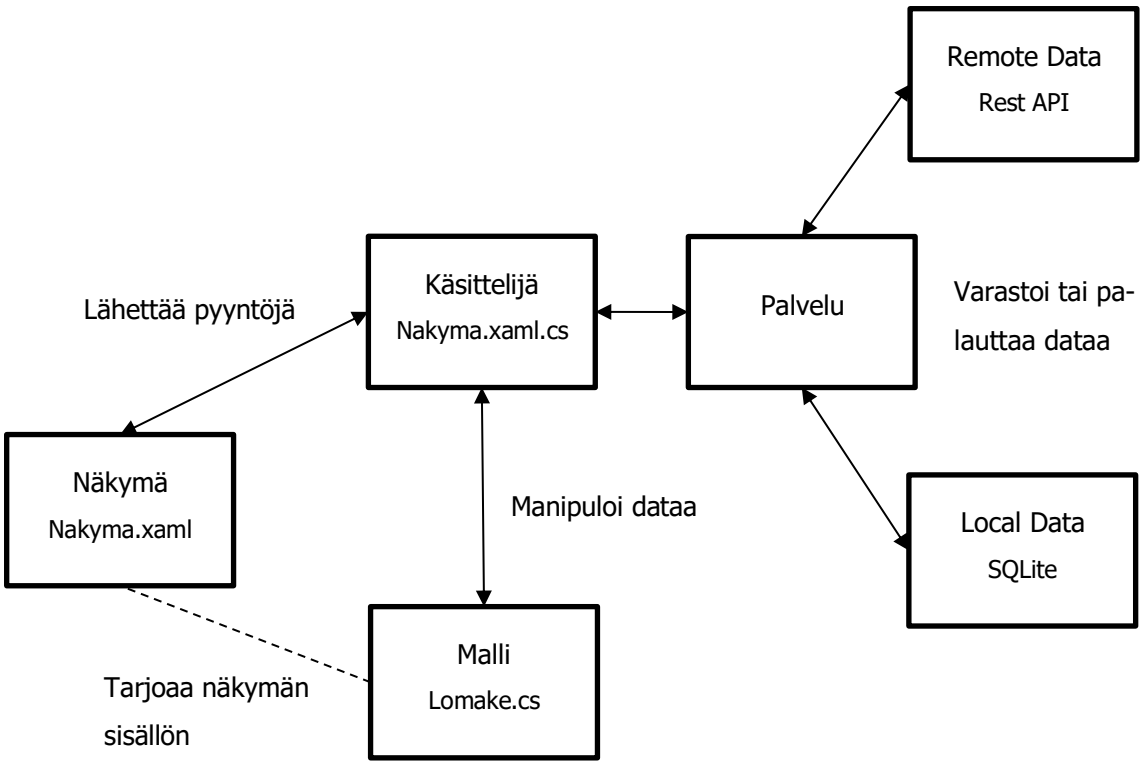
Forms-mobiilisovellus toteutettiin käyttäen .NET Standard -strategiaa, joten projektissa jaettu ohjelmakoodi tuotetaan .NET Standard -projektiin ja natiivit toteutukset tuotetaan Android- ja iOS -projekteihin.




Kuva 6. Visual Studiolla luotu Xamarin.Forms projektin rakenne

### 4.2.1 Toimintaperiaate

Forms-mobiilisovelluksen näkymien toimintaperiaate pohjautuu MVC-arkkitehtuuriin, jonka tarkoituksena on käyttöliittymän erottaminen näkymän toimintojen logiikasta.




Kaava 1. Sovelluksen näkymien toimintaperiaate

### 4.2.2 MVC

MVC, eli Model View Controller, on ohjelmistoarkkitehtuurityyli, minkä avulla ohjelmiston kehitys voidaan jakaa kolmeen toisistaan eriteltyyn osaan. MVC-arkkitehtuuri on suosittu verkko- ja mobiilisovelluskehityksen parissa.

### 4.2.3 Malli

Model, eli malli, on tietokerros, jonka tehtävänä tässä projektissa on sisällön tarjoaminen näkymille ja elementeille. Esimerkiksi projektiin tehty luokka Form, vastaa lomakkeen tietosisällöstä, joka tarjoaa tiedon lomakkeella esiintyvistä elementeistä ja säännöistä, joista lomakenäkymä voidaan generoida.

### 4.2.4 Näkymä

View, eli näkymä, on käyttäjälle suunniteltu graafinen käyttöliittymä. Näkymä määrittää esimerkiksi Lomake-mallin ulkoasun ja tiedot käyttäjälle, sekä sallii näiden tietojen muokkauksen.

### 4.2.5 Ohjain

Controller, eli kontrolleri, käsittelijä tai ohjain, toimii MVC-arkkitehtuurissa logiikan käsittelyn roolissa, eli tässä työssä se toimii näkymän ja tietolähteiden välissä (kts. Kaava 1). Kontrolleri hoitaa lomakkeiden latauspyynnöt, sekä ilmoitusten tallennuspyynnöt http-palvelun kanssa. Jotta applikaatiota voidaan käyttää offline-tilassa, sen tulee tallentaa lomakkeet online-tilassa laitteen omaan kestomuistiin. Tätä varten luotiin tietokantapalvelu, joka hyödyntää SQLite-tietokantaa. Kontrollerin tehtävänä on tarjota näkymän toiminnoille logiikka ja päivittää mallin, esimerkiksi lomakeobjektin, sisältö.

### 4.2.6 Tietopalvelu

Jotta kontrolleri saatiin eriteltyä datan hakemis- ja käsittelyprosesseista, sitä varten luotiin tietopalvelumoduli. Palvelumodulin tarkoitus on käsitellä data operaatiot, jotta lomaketietojen käsitteleminen olisi helpompaa näkymien kontrollereissa. Servicellä, eli palvelulla, on kolme päätehtävää: käyttäjän autentikointi, lomake- ja ilmoitusdatan käsittely.

### 4.2.7 Autentikointi

Forms-mobiilisovelluksessa kirjaudutaan sisään käyttämällä palveluun luotua kirjautumismetodia, jonka tehtävänä on lähettää kirjautumispyyntö toimeksiantajan järjestelmään. Onnistuneen pyynnön jälkeen tallennetaan pyynnön mukana kulkeutuva eväste, joka takaa myöhemmin sovellukseen autentikoimisen ilman uudelleen kirjautumista.

#### 4.2.8 Lomakedatan käsittely

Lomakedatan käsitteleminen koostuu useammasta osasta. Ensiksi käsittelijä lähettää GET-pyynnön TCD Rest -rajapintaan, jonka jälkeen se varastoi saapuvan datan laitteeseen JSON-muodossa. Tämä toimenpide usein tapahtuu kirjautumisen jälkeen, sovelluksen aukaisemisen yhteydessä tai lomakkeen avaamisen aikana (QR-koodilla avaaminen). Lomakedata saapuu JSON-kokoelmassa, joka pitää sisällään listan lomakkeista JSON-objekti -formaatissa. Jokainen JSON-objektimuodossa oleva lomake tallennetaan laitteen SQLite-tietokantaan. Kun lomakenäkymä pyytää käsittelijältä lomaketta, se pyytää palvelun ensiksi hakemaan kyseisen lomakkeen JSON-objektin laitteen SQLite-tietokannasta, jonka jälkeen se muutetaan lomakeobjektiksi ja lopuksi palautetaan näkymän käsittelijälle.

#### 4.2.9 Ilmoitusten käsittely

Ilmoituksen käsittely lähtee liikkeelle, kun täytetty lomake tallennetaan. Tallentamisen aikana täytetty lomake tallennetaan laitteelle, jonka jälkeen se lähetetään suoraan Rest rajapintaan POST-pyyntönä, jonka tarkoituksena on tallentaa täytetty lomake Forms -ohjelmiston tietokantaan. Jos ilmoituksen lähetys epäonnistuu tai laitteella ei ole sinä hetkenä verkkoyhteyttä, ilmoitus tallennetaan odottaviin lähetyksiin. Tätä varten luotiin taustaprosesseja natiivitoteutuksille, jotka huolehtivat ilmoitusten lähettämisestä, kunnes verkkoyhteys löytyy taas. Toiminto tuli tehdä natiivina toteutuksena, sillä Xamarin.Forms ei tue taustapalveluita, koska ne toteutetaan alustoilla eri tavoin.

### 4.3 Näkymät

Tässä toteutuksen osiossa käydään läpi projektiin luotujen näkymien toteutustekniikat, ulkoiset piirteet ja tarkoitukset. Sovelluksen näkymät suunniteltiin ja toteutettiin käyttämällä XAML- ja C# -tiedostojen yhdistelmää.

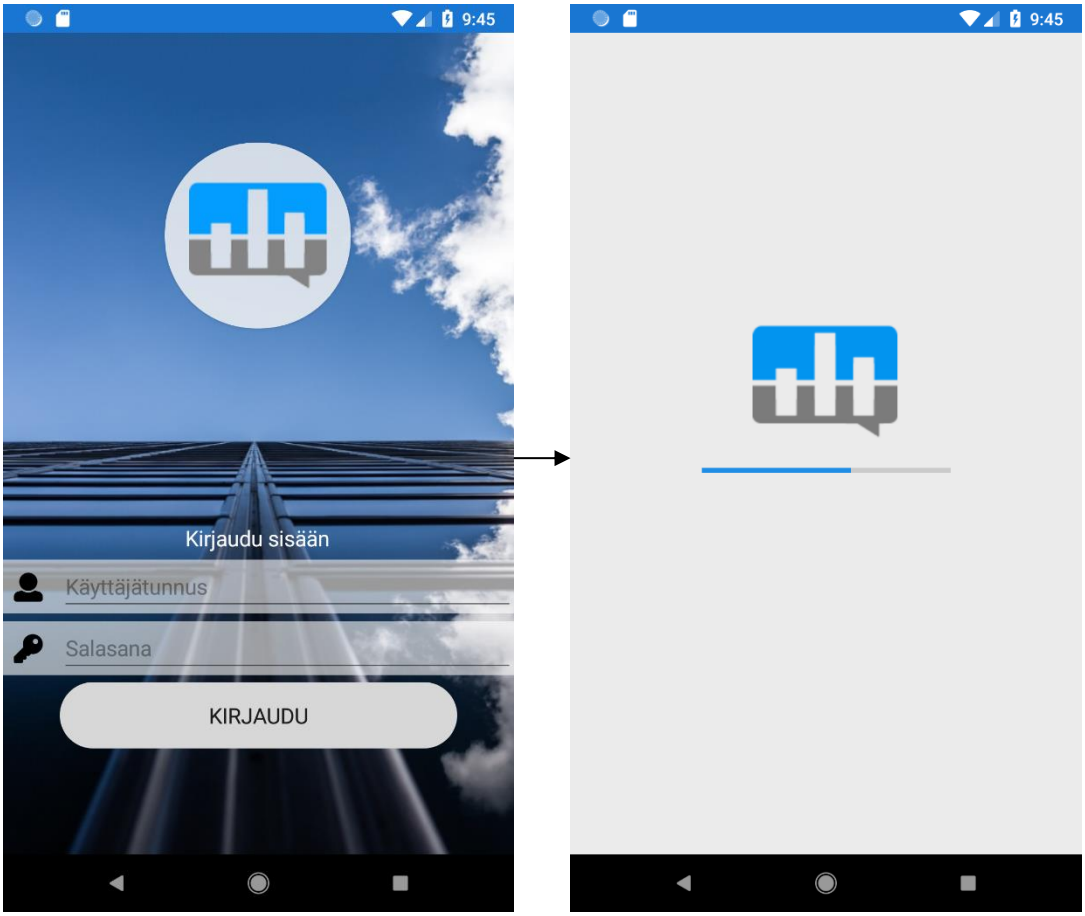
#### 4.3.1 XAML

Tässä opinnäytetyössä sovelluksen näkymät luodaan käyttämällä XAML-kuvauskieltä. XAML on Microsoftin kehittämä versio XML-merkintäkielestä. Xamarin.Forms kehyksessä XAML toimii yhdessä sille määrätyn luokan kanssa, yleensä Application-, Page- tai ContentView -luokkien yhteydessä. Tämän luokan avulla voidaan kontrolloida XAML-näkymän komponentteja ja niiden ominaisuuksia. Yksi XAML-merkintäkielen tärkeimmistä ominaisuuksista on datakytkentä (Data Binding), jonka avulla voidaan kontrolloida näkymän sisältöä mallista tai näkymämallista käsin.

#### 4.3.2 Kirjautumisnäkymä

Käyttäjän autentikointi suoritetaan käyttämällä .NET-ohjelmistokehyksen tarjoamaa Http Client -kirjastoa tietopalvelussa. Kirjaston avulla luodaan HTTP POST-pyyntö, jonka parametreiksi upotetaan enkoodattuna käyttäjätunnukset ja formaatin missä data lähetetään. Jos palvelin vastaa pyyntöön myöntävästi, käyttäjän autentikoiva eväste tallennetaan laitteelle, jotta sovellukseen ei tarvitse kir-

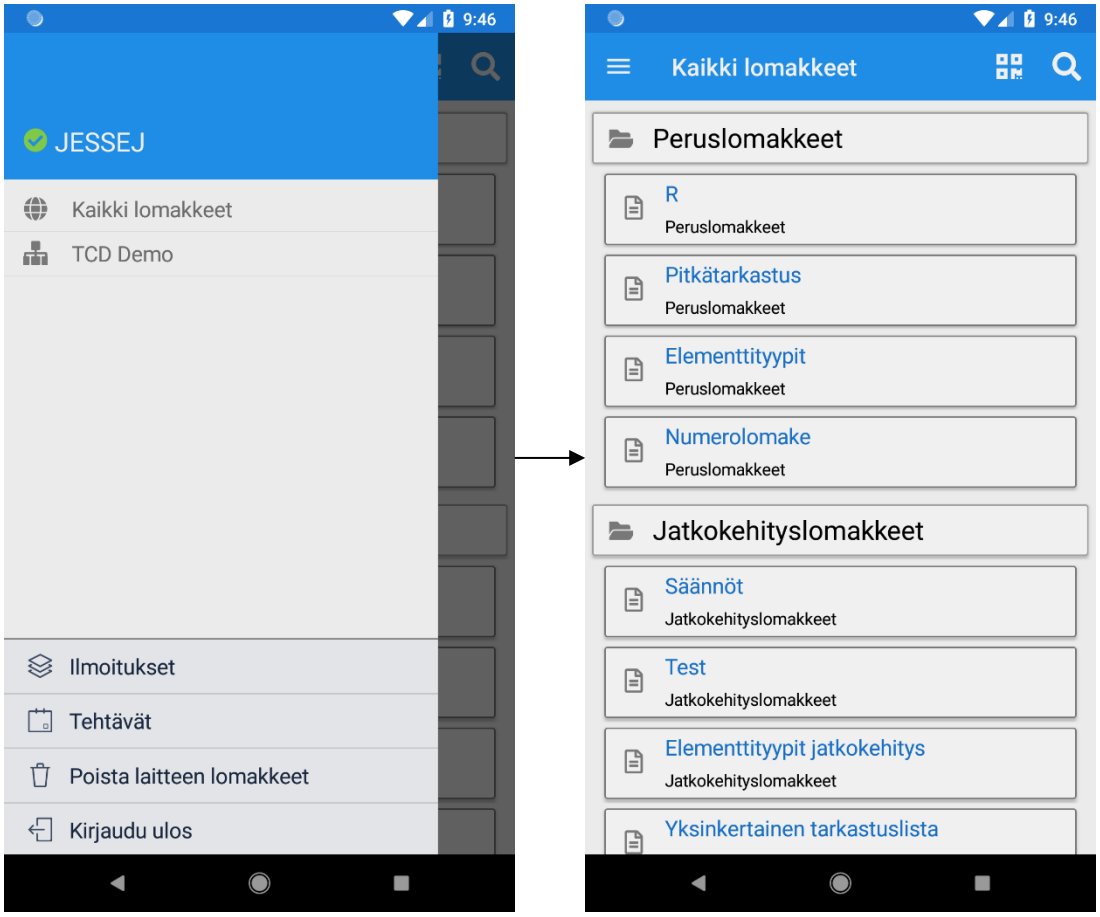
jautua uudestaan. Näkymän tarkoitus on luoda käyttöliittymä sovellukseen kirjautumista varten. Kirjautumisen pystyy myös ohittamaan, jos sovellusta halutaan käyttää offline-tilassa. Sovellus ohittaa tämän näkymän, jos laitteeseen on jo kirjauduttu aikaisemmin sisään.




Kuva 7. Kirjautumisnäkymä (vas.) ja sitä seuraava latausnäkymä kirjautumisen jälkeen

### 4.3.3 Lomakelistausnäkymä

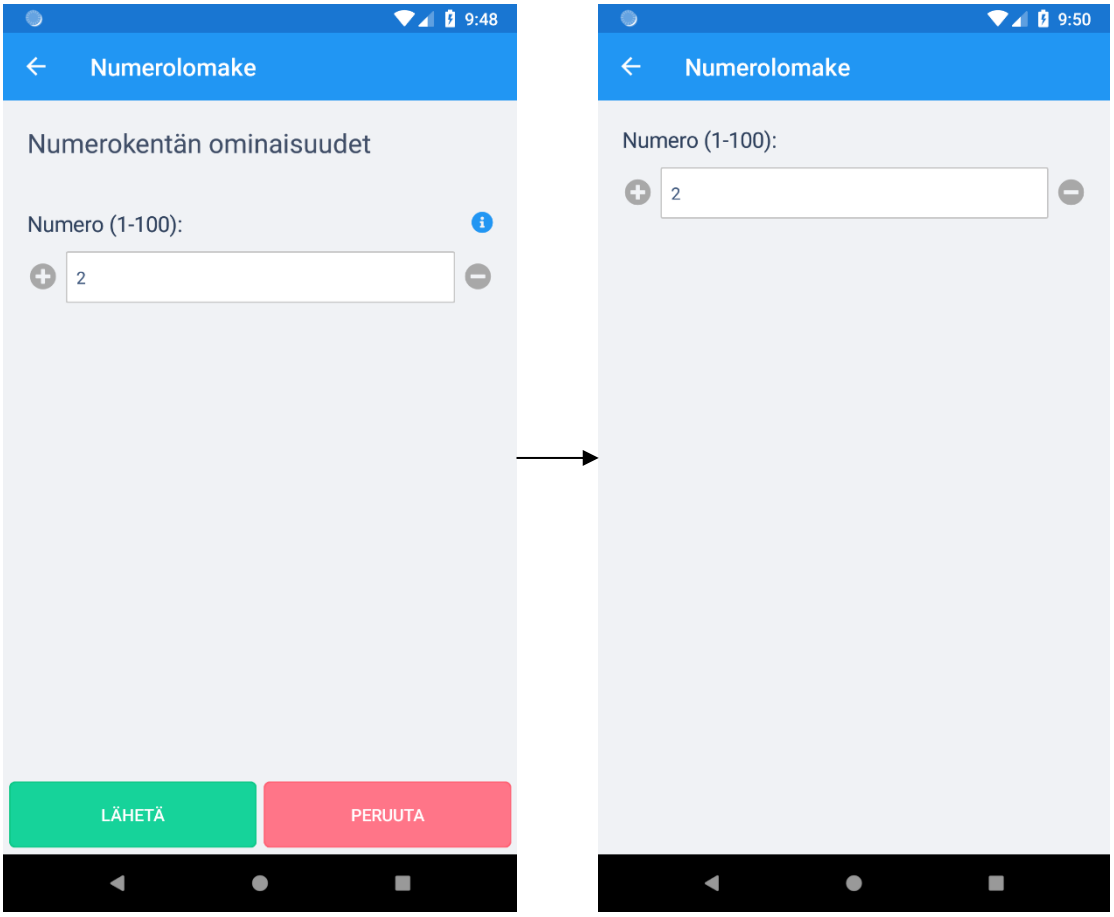
Lomakkeiden listausnäkymässä listataan käyttäjälle tarkoitetut lomakkeet ryhmitellysti. Käyttäjä voi Forms-ohjelmistossa määritellä erilaisia lomakeryhmiä, sekä kuulua eri organisaatioihin joiden perusteella lomakkeita voidaan lajitella.




Kuva 8. Vasemmalla valikkonäkymä ja oikealla lomakelistausnäkymä

### 4.3.4 Lomakenäkymä

Lomakenäkymän käyttöliittymässä voidaan täyttää ja lähettää täytetty lomake käsittelijälle. Käsittelijän tehtävänä on sitten pyytää tietopalvelun tallentamaan ilmoitus laitteen kestomuistiin ja lähettämään se edelleen palvelimelle POST-pyynnöllä, jos tämä on mahdollista. Jos laite on offline-tilassa, täytetty lomake tallennetaan odottaviin ilmoituksiin ja käynnistetään taustasovellus, joka lähettää ilmoituksen palvelimelle, kun käyttäjä on kirjautunut sisään ja verkkoyhteys on käytettävissä. Lomakenäkymää voidaan myös hyödyntää lomakkeen ilmoituksen tarkastelussa, jossa ilmoitus rakennetaan samalla tavalla kuin täytettävä lomakenäkymä, mutta tallennettujen arvojen kanssa.




Kuva 9. Lomakenäkymä (vas.) eräästä lomakkeesta, sekä täytetty lomake

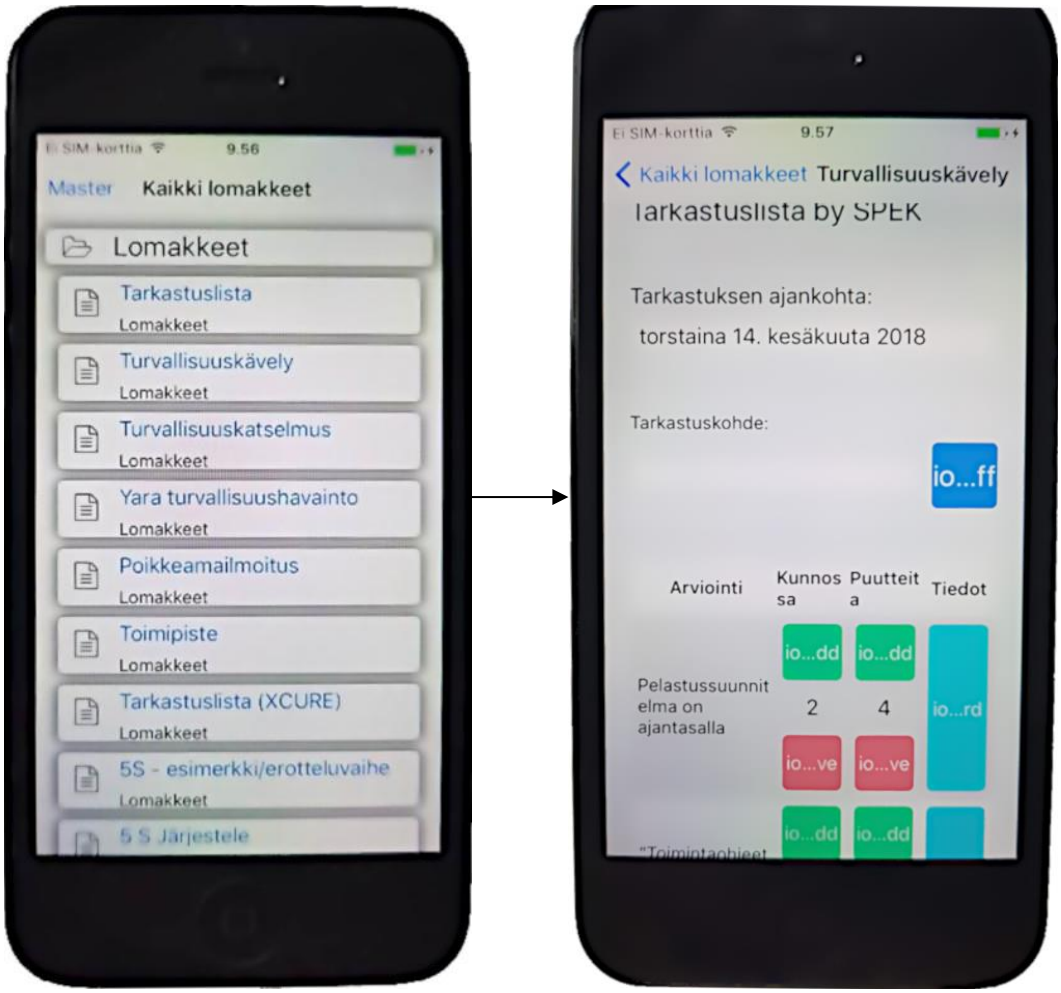
Lomakenäkymän luonti lähtee POST-pyynnöstä REST API -rajapinnalle, joka palauttaa datan käyttäjän kaikista lomakkeista JSON-tiedostomuodossa. Palvelumodulin tehtävänä oli muuttaa JSON-data kontrollerin käyttöä varten sopivaksi tietomuodoksi. Palvelumoduli palauttaa näkymälle Lomake-objektin, joka pitää tiedon sille määrätyistä elementeistä. Nämä elementit generoidaan lomakkeen XAML-tiedostoon C#-tiedostossa, jossa määritelty metodi lisää näkymät järjestyksessä lomakkeen elementeille annettuun layout-komponenttiin.

```
foreach (Models.Element formElement in form.Elements)
{
    var layoutElement = new StackLayout
    {
        Margin = new Thickness(20),
    };

    switch (formElement.ElementType)
    {
        ...

        case "text":
            Text text = new Text(formElement);
            _views.Add(text, typeof(Text));
            _elements.Add(text.ElementId, text);
            layoutElement.Children.Add(text);
            break;

        ...
    }
}
```

Lähdekoodi 1. Dynaamisten elementtien generointi lomakenäkymän C#-tiedostossa

### 4.3.5 Näkymät iOS-laitteella

Xamarin.Forms UI API jakaa käyttöliittymässä käytettävän koodin, jolloin sovelluksen ulkoasusta tulee lähes saman näköisiä molemmille alustoille. Näkymät luodaan kuitenkin alustojen natiivikomponenteista, jolloin niiden käyttäytyminen ja ulkoasu eroaa hieman alustojen välillä (vrt. Kuva 8 ja Kuva 10).




Kuva 10. Listausnäkymä (vas.) ja lomakenäkymä (oik.) kehitysvaiheessa iOS-alustalla

## 4.4 Elementit

Tässä osiossa käsitellään lomakkeen elementtien toimintaperiaate ja ulkoiset kehitysoperaatiot. Koska elementtejä luotiin projektiin seitsemäntoista erilaista variaatiota, on niitä turha käydä kaikkia läpi, sen sijaan käymme yhden elementin toimintaperiaatteen läpi yksinkertaisena esimerkkinä. Tarkastelun kohteeksi otetaan tekstintulostuskomponentti, eli tekstikenttä (EL1), josta luotiin yksinkertaistettu versio tarkastelua varten. Elementti luodaan Lomake-mallin Elements-listasta, joka koostuu Element-luokan objekteista. Element-objektista voidaan luoda tekstielementti, sillä objektin määrittelemä luokka pitää sisällään tiedon elementin tyypistä. Tekstikomponentti voidaan lisätä näkymään määrittelemällä se näkymän XAML-tiedostossa (Lähdekoodi 2) tai luomalla se dynaamisesti näkymän C#-tiedostossa (Lähdekoodi 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="FormsProjekti.Nakyma">
  <ContentView.Content>
      <StackLayout>
          <Label x:Name="tervehdys"
                 FontSize="Medium"
                 FontAttributes="Bold" />
      </StackLayout>
  </ContentView.Content>
</ContentView>
```

Lähdekoodi 2. XAML-tiedosto yksinkertaisesta tekstikentästä, Nakyma-esimerkkikomponentista

XAML-tiedostossa määritellään komponentin ulkoasu käyttämällä XML-tyyppistä syntaksia. Lähdekoodi 2 Label-näkymä on Xamarin.Forms kehyksen tarjoama yksinkertainen tekstikenttä, jolle voidaan antaa erilaisia attribuutteja. Projektissa ei käytetä tekstikenttänä pelkkää Label-luokkaa, sillä elementin näkymä voi omistaa muitakin komponentteja, kuten tooltip-komponentti, jonka tarkoituksena on tulostaa selvennys siitä mitä kenttään tulisi syöttää.

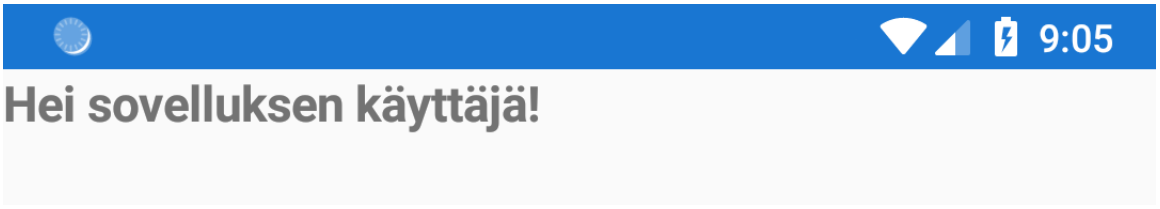
```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace FormsProjekti
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Nakyma : ContentView
    {
        public string Tervehdys
        {
            get { return (string)GetValue(TervehdysProperty); }
            set { if (Tervehdys != value) SetValue(TervehdysProperty, value); }
        }

        private static BindableProperty TervehdysProperty = BindableProperty.Create(
            propertyName: "Tervehdys",
            returnType: typeof(string),
            declaringType: typeof(Nakyma),
            defaultValue: "Terve",
            defaultBindingMode: BindingMode.OneWay,
            propertyChanged: TervehdysPropertyChanged
        );

        private static void TervehdysPropertyChanged(
            BindableObject bindable,
            object oldValue,
            object newValue
        ) {
            var control = (Nakyma) bindable;
            control.tervehdys.Text = newValue.ToString();
        }

        public Nakyma ()
        {
            InitializeComponent();
        }

        public Nakyma (Element element)
        {
            InitializeComponent();
            tervehdys.Text = element.Text;
        }
    }
}
```

Lähdekoodi 3. Nakyma-esimerkkikomponentin C#-tiedosto

Elementit luodaan perimällä ContentView-luokka, jonka tarkoituksena on toimia elementin kehyksenä, johon sisältö rakennetaan. Nakyma-komponentille voidaan määritellä BindableProperty ominaisuus, jonka tarkoitus on päivittää elementissä näkyvä tieto.

```
<StackLayout>
    <local:Nakyma
        Tervehdys="Hei sovelluksen käyttäjä!"/>
</StackLayout>
```

Lähdekoodi 4. Sivunäkymän XAML-tiedostossa käytetty Nakyma-komponentti




Kuva 11. Nakyma-esimerkkikomponentin tulos Android-alustalla

## 4.5 Natiivit toteutukset

Xamarin.Forms-kehyksellä luodaan natiiviominaisuuksia vaativia toimintoja käyttämällä DependencyService-luokkaa. DependencyService on staattinen luokka, joka tarjoaa toiminnallisuuden kutsumalla natiivitoteutusta jaetusta koodista. DependencyService hakee alustan projektista rajapinnan, jonka avulla toivottu toiminto voidaan suorittaa. (Microsoft, 2018) Ensiksi luodaan halutulle toiminnolle rajapinta ja sen metodit .NET Shared -projektiin, jonka jälkeen luodaan toteutus Android- ja iOS-projekteihin, joka implementoi .NET Shared -projektin rajapintaa. Alustan projektille luotuun rajapintaan määritellään alustakohtainen toteutus haluttua toimintoa varten.

### 4.5.1 Speech-to-text

Projektin tekstinsyöttökentille, elementeille EL4 ja EL5, toteutettiin puheesta tekstiksi -lisätoiminto. Tämän toiminnon avulla käyttäjä voi valita tekstin syöttämisen kirjoittamisen sijaan puhumisen laitteen mikrofoniin, jonka alustakohtainen toteutus kääntää tekstiksi.

### 4.5.2 Taustaprosessit

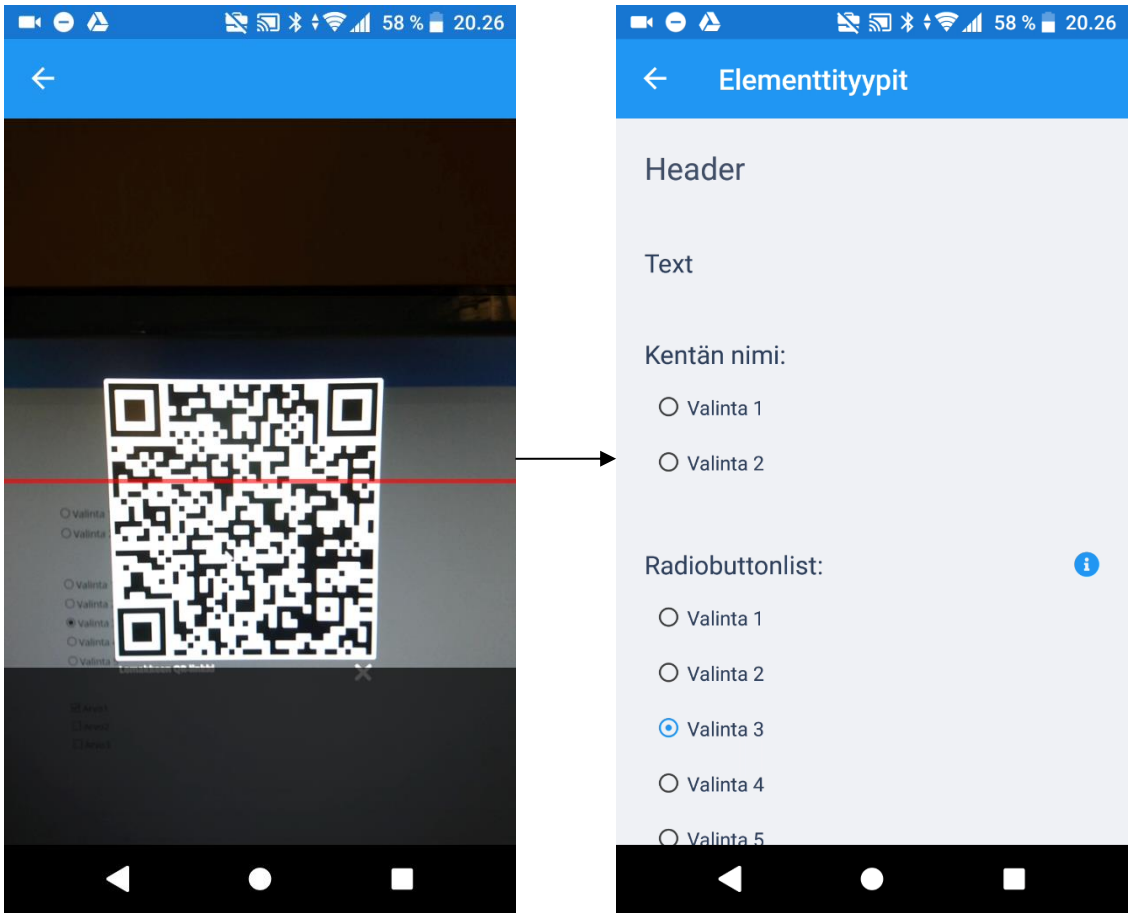
Taustaprosesseja luotiin sovellukselle natiivitoteutuksena, koska Xamarin.Forms ei tarjoa tähän toteutusta eri alustoille. Taustaprosesseja tarvittiin, jotta lomakkeen toimitus onnistuisi palvelimelle asynkronisesti kun laite saa verkkoyhteyden ja käyttäjä on kirjautunut sovellukseen sisään.

### 4.5.3 Tiedostohakemisto

Tiedostohakemistot toteutettiin natiivina, sillä Android- ja iOS -järjestelmien tiedostorakenteet ovat erilaiset. Tiedostohakemistoa käytettiin tiedostojen tallentamiseen ja tarkastelemiseen (AEL3 ja AEL4).

### 4.5.4 QR-koodin lukeminen

QR-koodin lukeminen luotiin projektille, jotta lomakenäkymän voi myös avata kameran avulla, esimerkiksi työmaalle sijoitetusta kuvasta. QR-koodikuvan prosessointi tehtiin ZXing avoimen lähdekoodin kirjastolla, mikä oli mahdollista lisätä NuGet Gallerystä käyttämällä nuget package manageria. NuGet Gallery on keskitetty NuGet pakettien säilytys- ja jakoalusta, josta .NET-kehystä hyödyntäviin projekteihin voidaan lisätä muiden kehittäjien tekemiä paketteja.




Kuva 12. Lomakkeen voi avata myös QR-koodilla

## 4.6 Julkaiseminen beetatestaukseen

Forms-mobiilisovellus julkaistiin beetatestaukseen Play kauppaan. Beetatestauksella tarkoitetaan, että sovellus on päätoiminnoiltaan ja ulkoasultaan kutakuinkin valmis, mutta saattaa sisältää vielä ohjelmointivirheitä. Sovelluksen julkaisuun vaadittiin Google Play Publisher käyttäjätili, jonka rekisteröiminen maksaa 25 dollaria. Toimeksiantajan rekisteröimisen jälkeen Google Play Developer Consolen avulla lisättiin Visual Studiolla luotu beeta testaukseen soveltuva APK-tiedosto.

## 4.7 Ongelmat

Projektin edetessä ilmeni useita ongelmia. Ongelmat olivat pääsääntöisesti kehitysympäristöön liittyviä ongelmia, joihin ei ollut vielä kunnollista ratkaisua kehitetty.

### 4.7.1 Remote iOS

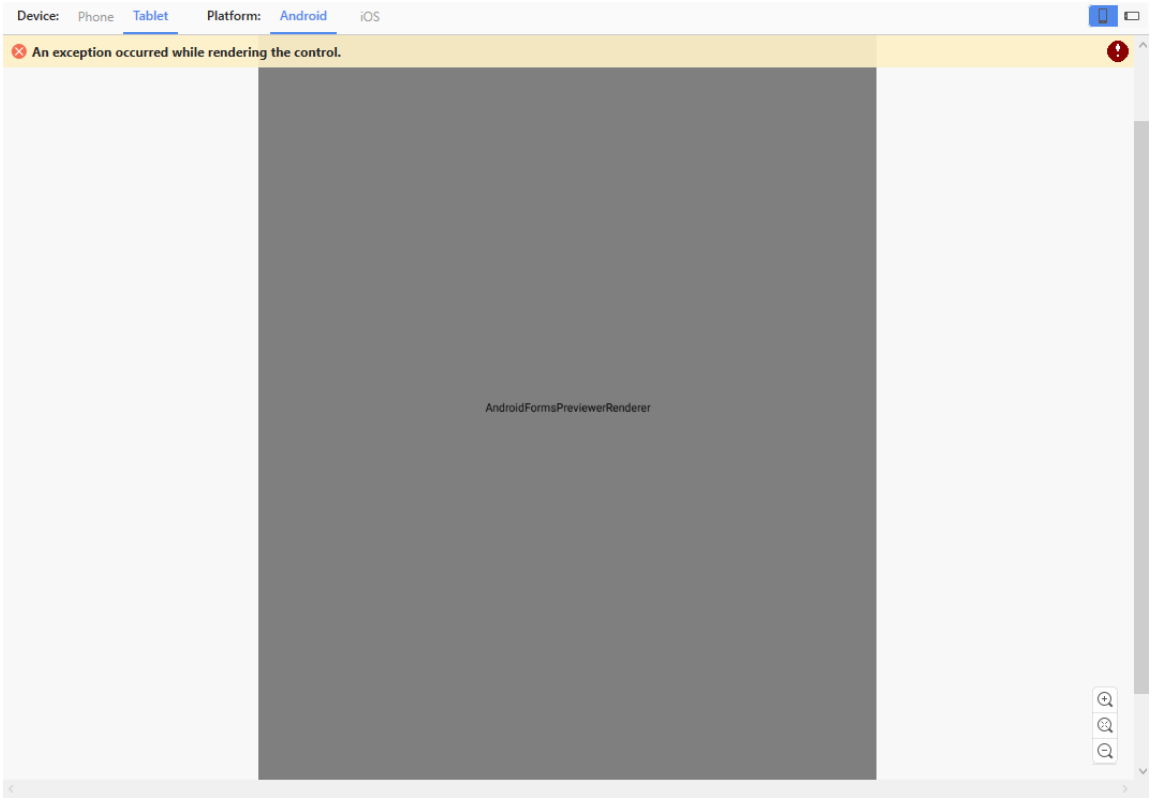
iOS-alustalle voidaan rakentaa applikaatio vain Xcode-ohjelmointiympäristöä käyttämällä. Xcode-ohjelmointiympäristö on tarkoitettu Applen macOS- ja iOS -sovellusten kehittämistä varten. Se sisältää koodieditorin, debuggerin, käyttöliittymäsuunnittelutyökalun ja muun tarvittavan iOS-ohjelmointia varten. (Apple Inc., 2019)

Visual Studion Cross-Platform Mobile Development -lisäosa tarjoaa mahdollisuuden rakentaa ohjelmakoodi ja virheenjäljistystoiminnot iOS-alustan applikaation kehittämistä varten. Näitä toimintoja pystytään hyödyntämään vain, jos Visual Studio -kehitysympäristöön on liitetty macOS-käyttöjärjestelmällä varustettu laite.

Projektin iOS-sovelluksen rakentamista ja virheenjäljitystä testattiin toimeksiantajan macOS-laitteella. Sovellus toimi kutakuinkin odotetusti, mutta bugeja oli nähtävissä sekä käyttöliittymässä että ohjelmakoodissa. Esimerkiksi Kuva 10 oikealla esiintyvästä lomakenäkymästä puuttuvat kuvakkeet kokonaan. Tämän korjaaminen onnistui lisäämällä natiivitoteutukseen puuttuva kuvakekirjasto.

### 4.7.2 XAML Previewer

Käyttöliittymän suunnittelu oli usein virheenjäljityksen yhteydessä tapahtuva toimenpide, sillä Visual Studion tarjoama käyttöliittymä näkymien suunnittelua varten harvoin toimi oikein tai sen käyttö oli hidasta. Tämä osittain johtui siitä, että osa XAML-tiedostoon lisättävistä komponenteista oli dynaamisia.




Kuva 13. XAML Previewer ei toiminut odotetusti

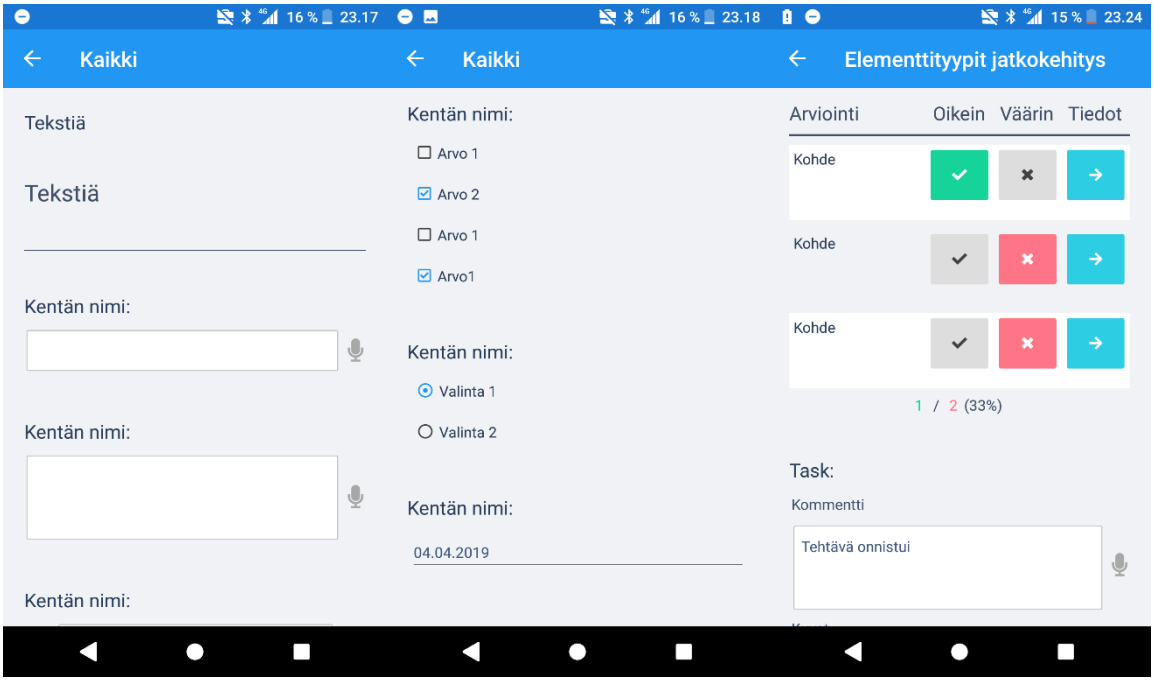
### 4.7.3 NuGet-pakettien päivittäminen

Projekti haluttiin pitää ajan tasalla Android SDK:n kanssa, joten oli päivitettävä Xamarin.Android.Support-kirjastoja useampaan otteeseen. Päivitysten yhteydessä ilmeni ongelmia muiden NuGet-pakettien kanssa, sillä kaikkien näiden pakettien oli oltava yhteensopivia tietyn version omaa-

vien Xamarin.Android.Support-kirjastojen kanssa. Projektin toteutuksen aikana lisättiin useita kymmeniä NuGet-paketteja, joiden ylläpitäminen osoittautui turhan monimutkaiseksi. Toteutuksen aikana Visual Studio ehdotti pakettien päivittämistä, joten ne päivitettiin kaikki kerralla, jonka jälkeen kehitysympäristö meni sekaisin, sillä riippuvuudet olivat konfliktissa keskenään. Paketit jouduttiin toteutuksen aikana useampaan otteeseen palauttamaan aiempiin versioihin.

## 4.8 Tulos

Lopputuloksena saatiin lomakkeita hyödyntävä mobiilisovellus, joka toimii vaihtoehtona verkossa käytettävälle Forms-ohjelmistolle. Lopputulos täytti kaikki ennalta määritellyt kriteerit, mutta sovellukselle kuitenkin jäi vielä mahdollisuuksia jatkokehitykselle. Toimeksiantajan ja kehittäjän vaatimuksiin päästiin sekä teknisten että käyttöliittymien toteutusten puolesta.




Kuva 14. Lomakkeen eräitä elementtejä

## 4.9 Jatkokehitys

Forms-mobiilisovellukselle ideoitiin kehityksen aikana useampia ominaisuuksia, joita voitaisiin toteuttaa jatkokehityksenä.

### 4.9.1 Lomakkeiden hallinta

Lomakkeiden luominen ja editointi tapahtuu verkkosovelluksessa drag-and-drop-tyyppisellä järjestelmällä, jossa haluttu elementti raahataan lomakkeen sisään. Mobiilisovellukseen ideoitiin samanlaista toimintoa, mutta mobiilisovelluksessa lomakkeen elementtejä voitaisiin lisätä valintaruudun ja painikkeen avulla. Tästä näkymästä luotaisiin JSON-objekti, joka lähetettäisiin uutena lomakkeena rajapinnan yli Forms-ohjelmistoon.

### 4.9.2 Asetukset ja teema

Forms-mobiilisovellukseen ideoitiin asetukset-näkymää, jossa käyttäjä pystyisi vaihtamaan sovelluksen näkymien ulkoisia tekijöitä, kuten värimaailmaa tai fonttikokoa, sekä rajoittamaan sovelluksen käyttämiä resursseja, esimerkiksi taustadatan käyttämistä.

# 5 YHTEENVETO

Opinnäytetyön tavoitteena oli saada laajempi ymmärrys alustariippumattoman sovelluksen kehittämisestä eri tyyppisillä ohjelmistokehyksillä, sekä ohjelmistosuunnittelun ja -toteutuksen eri vaiheista. Näihin tavoitteisiin päästiin, joten opinnäytetyö onnistui suunnitelmien mukaisesti. Vaikka kehitystyö suoritettiin minulle uudella kehitystekniikalla, Xamarin.Forms:lla, onnistui työn toteuttaminen sujuvasti. Uuden tekniikan opiskelu suunnittelun ohessa takasi työlle sopivasti haasteita. Projektia aloittaessani minulla oli hyvä käsitys Android-sovelluskehityksestä, sekä kokemusta C#-ohjelmoinnista, joita hyödynsin opinnäytetyössä.

Tutkimusvaiheessa tekemäni kartoitus erilaisista kehitysvaihtoehdoista sovellustuotannossa osoittautui erittäin hyödylliseksi suunnitteluvaiheessa. Tästä opin, että jokaiseen tulevaan projektiin tulee vaatia suhteellisen kattava taustatutkimus, jotta työn suunnittelu ja toteuttaminen sujuisi helpommin.

Projektin lopputuotteena saatiin toimiva ja helppokäyttöinen lomakesovellus. Sovellukset eivät koskaan ole valmiita, sillä ohjelmistotekniikat ja ratkaisut kehittyvät ja muuttuvat jatkuvasti. Sovellusta olisi voinut jatkaa moneen suuntaan, mutta isoimmaksi jatkokehityksen mahdollisuudeksi asettaisin lomakkeiden luonnin ja muokkaamisen mobiilissa. Olen kuitenkin hyvin tyytyväinen työn tulokseen, sillä sovellus toimii kuten pitää ja se täyttää kaikki kriteerit, mitä sovellukselle annettiin.

Opinnäytetyön aikana sain myös hyvää kokemusta .NET Frameworkilla toteutetusta järjestelmästä, Forms-ohjelmistosta, jonka toimintaperiaatetta kävimme toimeksiantajan kanssa yhdessä läpi. Tästä sain hyvän käsityksen, miten ja millä tekniikoilla monimutkaisempia järjestelmiä voidaan toteuttaa.

Työn edetessä opin hallitsemaan Xamarin.Forms-ohjelmistokehyksen, mutta projektin toteuttamisen jälkeen osasin kuitenkin olla kriittinen kyseisestä tekniikkaa kohtaan. Toteuttamisen aikana heräsi useampaan otteeseen kysymys toteuttamisen nopeudesta natiivina, sillä jotkut projektin kriittiset osat oli toteutettava kuitenkin natiivina. Työn lopputuloksesta voin sanoa, että yhden koodipohjan osuus koko projektista oli n. 80%. Tämä luku kertoo, että työ kannatti tehdä Xamarin.Forms-kehyksellä, sillä käyttöliittymän tekemiseen kului huomattavasti vähemmän aikaa mitä olisi kulunut molempien alustojen natiivitoteutuksessa yhteensä.

# 6 LÄHDELUETTELO

**Apache. 2004.** Apache License, Version 2.0. *Apache.* [Online] Tammikuu 2004. [Viitattu: 4. Huhtikuu 2019.] http://www.apache.org/licenses/LICENSE-2.0.

**Apple Inc. 2019.** Xcode - Apple Developer. *Apple Developer.* [Online] 2019. [Viitattu: 10. Toukokuu 2019.] https://developer.apple.com/xcode/.

**Applikey Team. 2018.** Xamarin Forms vs Xamarin Native: What Fits You Best? *Applikey Solutions sivusto.* [Online] 29. Toukokuu 2018. [Viitattu: 4. Helmikuu 2019.] http://applikeysolutions.com/blog/xamarin-forms-vs-xamarin-native-what-fits-you-best.

**Blair, Ian. 2017.** Buildfire.com. *Buildfire.com.* [Online] BuildFire, 17. Heinäkuu 2017. [Viitattu: 12. Huhtikuu 2018.] https://buildfire.com/choose-native-hybrid-web-mobile-app/.

**Bonney, Eisenman. 2015.** *Learning React Native by Bonnie Eisenman.* s.l. : O'Reilly Media, 2015. 978-1491989142.

**Butusov, Mykhailo. 2019.** TechMagic. *TechMagic.* [Online] 6. Helmikuu 2019. [Viitattu: 14. Huhtikuu 2019.] https://blog.techmagic.co/native-vs-hybrid-apps/.

**Cowart, Jim. 2012.** Telerik.com. *Telerik.com.* [Online] Telerik AD, 14. Kesäkuu 2012. [Viitattu: 16. Toukokuu 2019.] https://www.telerik.com/blogs/what-is-a-hybrid-mobile-app-.

**Dunn, Graig ja Britch, David. 2019.** Installing Xamarin - Xamarin. *Microsoft.* [Online] 2. Huhtikuu 2019. [Viitattu: 10. Huhtikuu 2019.] https://docs.microsoft.com/en-us/xamarin/get-started/installation/index?pivots=win-vs2017.

**Ecma International. 2017.** ECMA-404. *Standardi.* 2017.

**Microsoft. 2019.** Get Started with Xamarin. *Microsoft.* [Online] 2019. [Viitattu: 23. Maaliskuu 2019.] https://docs.microsoft.com/en-us/xamarin/get-started/.

**—. 2018.** Introduction to DependencyService - Xamarin. *Microsoft.* [Online] Microsoft, 15. Syyskuu 2018. [Viitattu: 14. Toukokuu 2019.] https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction.

**—. 2019.** Xamarin Documentation. *Microsoft.* [Online] 2019. [Viitattu: 10. Huhtikuu 2019.] https://docs.microsoft.com/en-us/xamarin/.

**—. 2018.** Xamarin.Forms Quickstart Deep Dive. *Microsoft.* [Online] 27. Marraskuu 2018. [Viitattu: 23. Maaliskuu 2019.] https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/deepdive?pivots=windows.

**NativeScript. 2018.** Getting Started with NativeScript - YouTube. *YouTube.com.* [Online] NativeScript, 24. Huhtikuu 2018. [Viitattu: 12. Joulukuu 2018.] https://www.youtube.com/watch?v=KNgx3JAyCtk.

**—. 2019.** NativeScript. *NativeScript.* [Online] NativeScript, 2019. [Viitattu: 12. Joulukuu 2018.] https://www.nativescript.org/.

**—. 2019.** NativeScript Docs. *NativeScript.* [Online] NativeScript, 2019. [Viitattu: 12. Joulukuu 2018.] https://docs.nativescript.org/core-concepts/technical-overview.

**Open Source Initiative.** opensource.org. *opensource.org.* [Online] [Viitattu: 5. Huhtikuu 2019.] https://opensource.org/licenses/MIT.

**StatCounter. 2019.** Mobile Operating System Market Share Worldwide | StatCounter Global Stats. *StatCounter Global Stats.* [Online] 18. Toukokuu 2019. [Viitattu: 18. Toukokuu 2019.] http://gs.statcounter.com/os-market-share/mobile/worldwide/2018.

**Stoll, Scott. 2018.** In plain English: So what the heck is Flutter and why is it a big deal? *Medium.* [Online] Medium, 3. Toukokuu 2018. [Viitattu: 23. Tammikuu 2019.] https://medium.com/flutter-community/in-plain-english-so-what-the-heck-is-flutter-and-why-is-it-a-big-deal-7a6dc926b34a.

**The Economist Intelligence Unit. 2016.** *Mobility, perfomance and engagement.* s.l. : The Economist Intelligence Unit, 2016.

**Traversy Media. 2017.** *YouTube.* [Online] Traversy Media, 4. Elokuu 2017. [Viitattu: 15. Huhtikuu 2019.] https://www.youtube.com/watch?v=ZikVtdopsfY.

**Umbaugh, Brad ja Dunn, Graig. 2018.** Sharing Code Overview - Xamarin. *Microsoft.* [Online] 6. Elokuu 2018. [Viitattu: 22. Tammikuu 2019.] https://docs.microsoft.com/fi-fi/xamarin/cross-platform/app-fundamentals/code-sharing.

**W3Schools. 2019.** W3Schools. *W3Schools.* [Online] Refsnes Data, 2019. [Viitattu: 25. Tammikuu 2019.] https://www.w3schools.com/nodejs/nodejs_intro.asp.

# 7 LIITTEET

Lähdekoodi 5. React Native -kehyksellä toteutetun päänäkymän lähdekoodi

```
import React from 'react';
import { Text, View, Image, Platform } from 'react-native';

export default class App extends React.Component {
  render() {

    let tcd_logo = {
      uri: 'https://www.tcdcon.com/img/logo.png'
    };

    let platform_version = Platform.Version;

    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <Image source={tcd_logo} style={{width: 100, height: 100}} />
        <Text>
          Hei, opinnäytetyön tekijä! Laitteessasi on API
          <Text style={{fontWeight: 'bold', color: '#00F'}}> {platform_version}</Text>
        </Text>
      </View>
    );

  }
}
```

Lähdekoodi 6. NativeScript -kehyksellä toteutetun päänäkymän lähdekoodi

```
import { Component, OnInit } from "@angular/core";
import { device } from "tns-core-modules/platform";


@Component({
    selector: "Home",
    moduleId: module.id,
    templateUrl: "./home.component.html"
})

export class HomeComponent implements OnInit {

    api_version = "0";

    constructor() {
        this.api_version = device.sdkVersion;
    }

    ngOnInit(): void {
        // Init your component properties here.
    }
}
```

Lähdekoodi 7. NativeScript -kehyksellä toteutetun päänäkymän XML-kuvaus

```
<ActionBar class="action-bar">
    <Label class="action-bar-title" text="Oppari Testi"></Label>
</ActionBar>

<StackLayout
    class="page"
    orientation="vertical"
    width="100%"
    height="100%"
    backgroundColor="lightgray">

    <Image
        src="https://www.tcdcon.com/img/logo.png"
        width="100"
        height="100"
        horizontalAlignment="center"
        verticalAlignment="center">
    </Image>

    <Label
        fontSize="16"
        text="Hei, opinnäytetyön tekijä! Laitteessasi on API"
        width="90%">
    </Label>

    <Label
        fontSize="24"
        text="{{ api_version }}"
        horizontalAlignment="center"
        color="blue">
    </Label>

</StackLayout>
```

Lähdekoodi 8. Flutter-kehyksellä toteutetun testiapplikaation päänäkymän lähdekoodi

```
class _MyHomePageState extends State<MyHomePage> {

  String urlLogo = 'https://www.tcdcon.com/img/logo.png';
  final String _os = Platform.operatingSystem;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Image.network(
              urlLogo
            ),
            Text(
              'Hei opinnäytetyön tekijä! Laitteesi käyttöjärjestelmä on',
            ),
            Text(
              '$_os',
              style: TextStyle(
                fontSize: 32,
                fontWeight: FontWeight.bold,
                color: Color.fromARGB(255, 100, 100, 255)
              ),
            ),
          ],
        ),
      )
    );
  }
}
```

Lähdekoodi 9. Xamarin.Forms testiapplikaation etusivunäkymän logiikka

```
using Plugin.DeviceInfo;
using System;
using Xamarin.Forms;

namespace OppariTesti
{
    public partial class MainPage : ContentPage
    {
        public string DeviceOS
        {
            get
            {
                return Device.OS.ToString();
            }
        }

        public string DeviceVer
        {
            get
            {
                return CrossDeviceInfo.Current.Version.ToString();
            }
        }

        public string HelloText
        {
            get
            {
                return
                    $"Hei opinnäytetyön tekijä! Laitteessasi on " +
                    $"{ DeviceOS } " +
                    $"{ DeviceVer }";
            }
        }

        public MainPage()
        {
            InitializeComponent();
            BindingContext = this;

            image.Source = "https://www.tcdcon.com/img/logo.png";
        }
    }
}
```

Lähdekoodi 10. Xamarin.Forms testiapplikaation kuvauskielenä toimii XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:OppariTesti"
             x:Class="OppariTesti.MainPage">

    <StackLayout Orientation="Vertical">

        <Image
            x:Name="image"
            HeightRequest="100"
            WidthRequest="100"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />

        <Label
            x:Name="label"
            Text="{Binding HelloText}"
            TextColor="Black"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />

    </StackLayout>

</ContentPage>
```