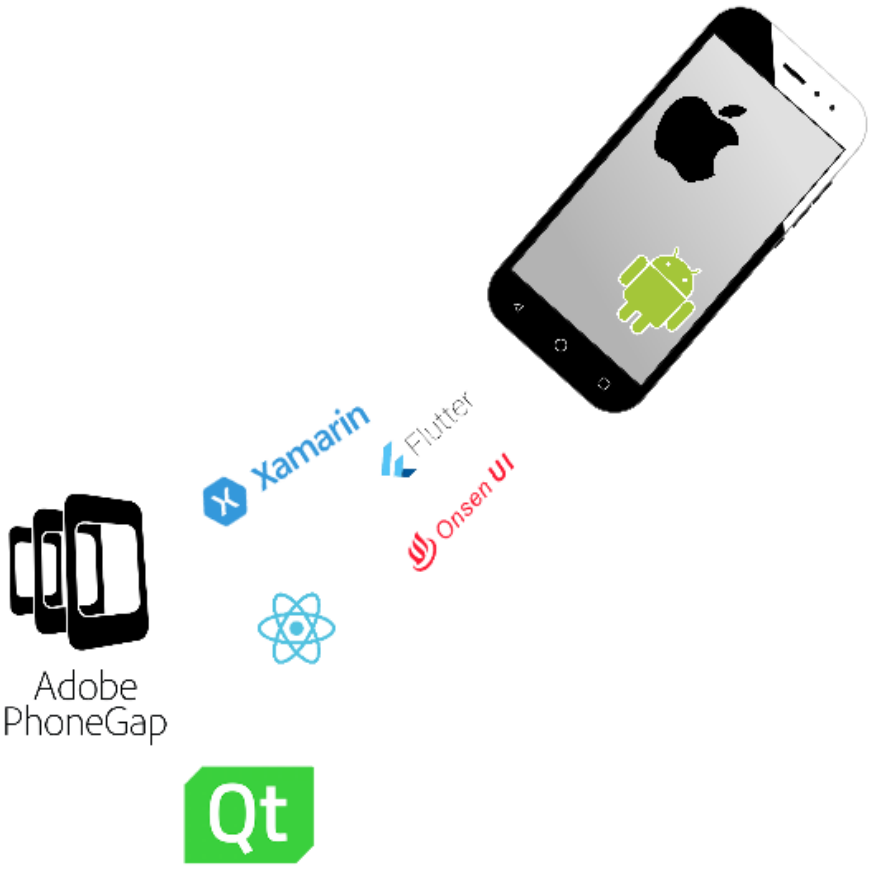
Niko Tauriainen

Alustariippumattomien sovelluskehysten sopivuus mobiilisovellusten luontiin



Insinööri (AMK)

Tieto- ja viestintätekniikka

Syksy 2019

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Tauriainen Niko

Työn nimi: Alustariippumattomien sovelluskehysten sopivuus mobiilisovellusten luontiin

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: alustariippumattomuus, sovelluskehys, mobiilisovellus, web-sovellus, natiivisovellus, hybridisovellus

Tämä opinnäytetyö tehtiin toimeksiantona kajaanilaiselle ohjelmistoalan yritykselle KajaPro Oy:lle. KajaPro Oy tuottaa tuotekehityspalveluita ja ICT-konsultointipalveluita.

Työn tavoitteena oli tutkia eri sovelluskehysten mahdollisuuksia tuottaa alustariippumattomia mobiilisovelluksia. Työ toteutettiin pääosaisesti tutkien sovelluskehysten ominaisuuksia ja niiden toimintaa osana kehitystä. Työssä tutkittiin ja vertailtiin kuutta eri sovelluskehystä. Niistä valittiin kolme, joilla tuotettiin alustariippumaton mobiilisovellus.

Tutkimus toteutettiin vertailemalla eri alustariippumattomuustapoja ja eri sovelluskehyksiä. Sovelluskehyksien vertailu jaettiin niiden tarjoamiin ominaisuuksiin, käyttöliittymätyökaluihin ja tietojärjestelmätyökaluihin. Näistä kolmesta osa-alueesta haettiin tietoja ja tuotettiin taulukoita, joiden perusteella luotiin johtopäätelmät kolmesta parhaasta sovelluskehyksestä.

Työn kehitysvaiheen tarkoituksena oli ottaa selvää sovelluskehyksien toiminnasta käytännössä. Sovelluskehyksiä käyttäen tuotettiin eri kehitysympäristöjen avulla alustariippumaton mobiilisovellus. Mobiilisovelluksen tuli lähtökohtaisesti toimia Android ja iOS-alustoilla ja pystyä käyttämään alustan kameraa viivakoodien skannaamiseen. Skannauksen jälkeen tuli viivakoodit asettaa tietokantaan.

Sovellukset toteutettiin luomalla kullakin sovelluskehyksellä uusi projekti. Projektissa tuotettiin vaadittavat tietokanta, skanneri ja käyttöliittymäominaisuudet. Kehityksen ohessa vertailtiin kehyksien käyttöasteelle saamista, kehityksen helppoutta kehittäjälle ja vaadittujen lisäosien käyttöä. Kukin sovelluskehys pystyi tuottamaan vaatimuksien mukaisen mobiilisovelluksen. Mobiilisovellus lopuksi testattiin erinäisillä Android ja iOS-alustoilla.

Tulosten perusteella suositeltiin yritykselle yhtä tutkituista sovelluskehyksistä. Kyseinen sovelluskehys sopi parhaiten pienten mobiilisovellusten tekoon, kuitenkin omaamalla laajan kokoelman tarpeellisia ja helpottavia työkaluja. Sovelluskehyksellä oli myös erittäin helppoa kehittää alustariippumattomasti. Yritys jatkoi tämän jälkeen sovelluskehyksen tutkimista ja sillä kehittämistä.

Abstract

Author(s): Tauriainen Niko

Title of the Publication: Suitability of Cross-Platform Frameworks for Creating Mobile Applications

Degree Title: Bachelor of Engineering, Information and communication technologies

Keywords: cross-platform, framework, mobile software, web application, native app, hybrid app

This Bachelor's thesis was made as a commission for software company KajaPro Oy. KajaPro Oy provides product development and ICT consulting services.

The main objective of this thesis was to study different frameworks and their potential to produce cross-platform mobile apps. The thesis was mainly executed as a study of framework features and their usability as part of development. This thesis will study and compare 6 different frameworks. From these 6 frameworks, 3 were picked for developing a cross-platform mobile app.

The study was made by comparing different cross-platform development methods and different frameworks. Comparison of frameworks was split in three; given properties, user interface tools and file system tools. On these three fields information was analyzed, and EXEL-tables were created, that led to conclusions of the best three frameworks.

The intent of thesis's development phase was to study frameworks in action. By using these frameworks within different IDE: s a cross-platform mobile app was developed. In principle the produced mobile app should be usable with Android an iOS platforms and use the phone's camera to scan barcodes. After scanning, the barcodes would be placed to the database.

Apps were programmed by creating new code projects with all the frameworks. In the project, the required features were developed. During development, their quality in making the framework usable, ease of development for the developer and using the required tools were compared. In the end, all the frameworks could produce the required software. After development, the mobile app was tested with different Android and iOS platforms.

According to the study and development test, one framework was recommended for the company. The recommended framework was the best in its ability to create small mobile software, while having a large collection of necessary tools to ease the development. The framework also performed well on its cross-platform abilities. The company then continued researching and developing the framework.

Sisällys

Symboliluettelo

API (Application programming interface) Ohjelmointirajapinta, on ennalta määriteltyjä tekniikoita, joiden tarkoitus on ohjelmiston välisten komponenttien kommunikoinnin mahdollistaminen.

CSS (Cascadin Style Sheets) Porrastetut tyyliarkit, jotka kuvailevat HTML dokumentin tyylin ja sen, miten ne tulisi näyttää.

HTML (Hypertext Markup language) Hypertekstin merkintäkieli, jolla kuvataan hyperlinkkejä sisältävää tekstiä.

IDE (Integrated development environment) Ohjelmointiympäristö, joka tarjoaa ohjelmointia helpottavia työkaluja, kuten tekstieditorin ja lähdekoodin kääntäjän.

JSX (JavaScript XML) Syntaksilisäosa JavaScript kieleen, kertoo alustalle, miltä käyttöliittymän tulisi näyttää.

npm (node.js package manager) Paketointityökalu node.js pakettien lataamiseen ja asentamiseen.

SDK (Software development kit) Kokoelma sovelluskehitystyökaluja, jotka helpottavat kehitystä tietyllä alustalla.

SQL (Structured Query Language) Kyselykieli, jolla voi keskustella sitä tukevien relaatiotietokantojen kanssa.

UI (User Interface) Käyttöliittymä, on sovelluksen graafinen osa ja se, mitä käyttäjä näkee käyttäessään sovellusta.

XML (Extensible Markup Language) Merkintäkieli, joka kuvaa rakenteellisia merkkauskieliä.

XAML (eXtensible Application Markup Language) XML-pohjainen merkintäkieli, joka kuvaa UI:ta Xamarin.Forms-sovelluksissa.

1 Johdanto

1.1 Työn tausta

KajaPro on kajaanilainen ICT-alan yritys, joka tekee tuotekehitystä ja tarjoaa konsultointipalveluita. Heillä oli tarjota minulle sopiva opinnäytetyön aihe. He halusivat kartoittaa markkinoilla olevia alustariippumattomia sovelluskehyksiä mobiilisovellusten tekoon ja selvittää mikä sovelluskehys olisi käyttökelpoisin heidän tarpeisiinsa. Heiltä löytyy myös aiemmin kehitetty sovellus, joka tarjoaa alustariippumattoman sovelluskehityksen. Kuitenkin kyseistä sovellusta ei ole käytetty projektikehitykseen viime aikoina ja sen käyttäminen vaatisi erittäin spesifisen projektin. Sovelluksen päivittäminen toimivaksi nykyisille alustoille tulisi olemaan merkityksetöntä, jos markkinoilta löytyy jo parempia alustariippumattomia sovelluskehyksiä mobiilialustoille, joista löytyy tuki ja pidetään ajankohtaisesti päivitettynä.

1.2 Työn tavoite ja rajaus

Työn tavoitteena on selvittää eroja nykyaikaisista alustariippumattomista sovelluskehyksistä, löytää yrityksen tarpeisiin sopiva mahdollisimman käyttökelpoinen tapa luoda mobiilisovelluksia ja tutkia sekä demonstroida, kuinka lähestyä alustariippumatonta sovelluskehitystä mobiilialustoilla.

Työ tehtiin lähtökohtaisesti tutkimukselliselta pohjalta, mutta se myös omaa kehitteellisiä ominaisuuksia. Sovelluskehyksien tutkimukseen ja vertailuun liittyen kehitin työssä läpikäytyjä alustariippumattomia sovelluskehyksiä käyttäen yksinkertaisia mobiilisovelluksia. Mobiilisovellusten päätavoitteena oli esitellä sovelluskehyksillä kehittämistä käytännössä ja millä tavoin eri sovelluskehykset mahdollistavat eri mobiilialustoilla löytyvien toiminnollisuuksien kuten kameran ja tiedostojen tallentamisen käytön. Samalla pyrittiin myös tutkimaan, miten suuria eroja löytyy alustariippumattomien sovelluskehysten komponenttien käytössä, kun sitä verrataan natiiviin mobiilisovelluskehitykseen. Tutkimuksen ohella kehitettyjen mobiilisovelluksien aihe valikoitui yrityksen tulevaisuuden tarpeen mukaan.

Kehitettävän mobiilisovelluksen lisäksi työn ohessa tuli kirjoittaa dokumentit mobiilisovellusvaiheeseen valituista sovelluskehyksistä. Dokumenttien avulla oli tarkoitus antaa yritykselle neuvoja sovelluskehyksien käytöstä ja helpottaa tutkimuksen tai kehittämisen jatkamista tulevaisuudessa. Kuitenkin työn loppuvaiheessa todettiin opinnäytetyön ja tehtyjen esimerkkien olevan riittäviä tähän tarkoitukseen, eikä erillistä dokumentaatioita kirjoitettu.

Internetistä löytyvä tarjonta alustariippumattomille sovelluskehyksille on valtaisa, eikä olisi työn mielekkyyden ja selkeyden puolesta käytännöllistä käydä läpi liian monta eri sovelluskehystä. Edellä mainituitten syitten takia rajaan läpi käytäviä sovelluskehyksiä yhteisön yleisesti käyttämiin ja kehityskelpoisiksi todettuihin.

2 Alustariippumattomuus

Alustariippumaton ohjelmointi on ohjelmointia, missä sovelluksia tai palveluita kirjoitetaan monelle tietokonealustalle yhteensopivaksi [1]. Kyseinen tietokonealusta voi olla sulautettu järjestelmä, työpöytä tai mobiilialusta. Alustariippumattoman ohjelmoinnin tavoitteena on mahdollistaa kirjoitetun sovelluksen käyttö eri alustoilla ja pyrkiä pienentämään tarvittavan lähdekoodin määrää. Eli ohjelmaa ei tarvitse kirjoittaa jokaiselle alustalle erikseen vaan voidaan periaatteessa käyttää yhtä lähdekoodia sovelluksien tuottamiseen.

Alustariippumattomuudella saavutetaan nopeampia kehityssyklejä verrattuna natiivin sovelluksen kehittämiseen monelle alustalle. Tämä vapauttaa aikaa sovelluksen muiden osien ja komponenttien paranteluun. Kuitenkin natiivin kehittämisen tarjoamien ominaisuuksien puute saattaa tuottaa vaikeuksia, varsinkin jos etsitään natiivisovellukselle tyypillisiä toimintoja, kuten sensoreihin ja tiedostorakenteeseen liittyvää toiminnallisuutta. Alustariippumattomia mahdollisuuksia näiden toimintojen tuottamiseen on olemassa, mutta ne yleensä vaativat erillisten pakettien ja lisäosien asentamisen ja eivät toimi natiivin omaisilla nopeuksilla.

Monet IDE:t eli ohjelmointiympäristöt tarjoavat mahdollisuuden ladata ja asentaa valmiita paketteja, kirjastoja tai sovelluskehyksiä, jotka tarjoavat alustariippumattoman lähdekoodin omaavien sovelluksien testaamisen sekä tuottamisen [2]. Näiden avulla voi alustariippumattoman sovelluksen luominen olla helppoa ja nopeaa.

2.1 Natiivisovellus

Natiivisovellus suunnitellaan ja kehitetään toimivaksi vain tietyllä alustalla tai käyttöjärjestelmällä. Natiivisovellus on kehitykseltään alustariippumattoman sovelluksen vastakohta. Natiivisovellukset kirjoitetaan useimmiten alustakohtaisella ohjelmointikielellä. Mobiilialustoilla kuten Androidilla kirjoitetaan Javaa ja iOS:llä Objective-C:tä tai Swiftiä. Jos natiivisovellus halutaan kirjoittaa jollain muulla kielellä kuin on tarkoitettu, tarvitaan väliin kääntäjiä tai kehityspakkauksia.

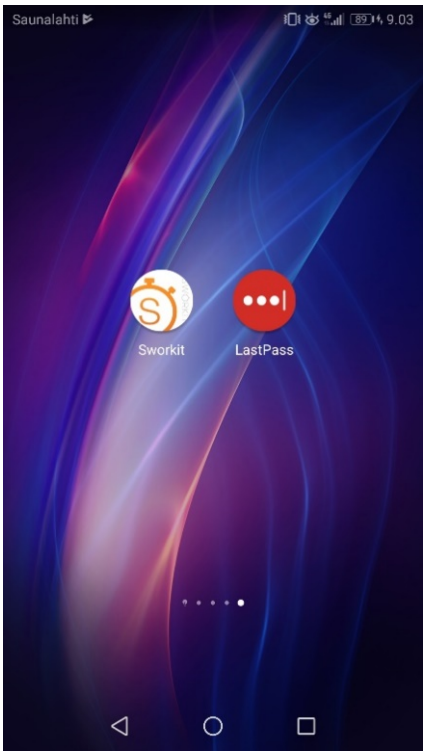
Natiivimobiilisovellusta kehittäessä kehittäjä pystyy varmistamaan, että tuotettava mobiilisovellus suoriutuu täydellisesti halutulla alustalla. Käyttäjien ja kehittäjien ei tarvitse huolehtia vioista

tai viiveistä, joita saattaa ilmaantua odottamattomasta yhteensopimattomuudesta ja erinäisten API:en kommunikoinneista Web- ja hybridisovelluksissa. Tällä tavalla voidaan myös taata johdonmukaisuus, koska sovelluksen toiminta on synkronoitu käyttöjärjestelmää ylläpitävien ensisijaisten ohjelmistojen kanssa. [3.] Käyttämällä järjestelmän tukemaa omaa lähdekoodia päästään suoraan käsiksi alustakohtaisiin komponentteihin, eikä tarvitse käydä läpi ylimääräisiä muunnoksia tai lisätä erillisiä lisäosia, jotka mahdollistavat samat ominaisuudet muilla sovellustyypeillä.

Natiivisovellusten kehittäminen voi kuitenkin olla enemmän aikaa vievää, jos kehitettävänä oleva sovellus tullaan lopulta julkaisemaan muillakin alustoilla. Kehittäessä yleensä käytetään alustakohtaista lähdekoodikieltä, niin eri alustojen kielten ja syntaksien opetteluun tulee käyttää aikaa. Eri alustat usein myös käyttävät eri ohjelmointiympäristöjä, kuten Android Studio Androidilla ja Xcode iOS:lla, ja niidenkin opetteluun voi tuhlautua aikaa. Niinpä kehittäjien on suunniteltava tarkkaan, millä tavoin he lähestyvät mahdollisimman hyödyllisesti sovellusten kehittämistä. Onko tuotettavana oleva sovellus kehitettävissä vähemmän aikaa vievillä alustariippumattomilla työkaluilla, vai tarvitseeko sovelluksen käyttää laajempia natiiviohjelmoinnin mahdollistamia järjestelmätoimintoja?

2.2 Web-sovellus

Web-sovellusten sovelluskehykset antavat lähtökohtaisesti mahdollisuuden kirjoittaa yhtä lähdekoodia, joka toimii useammalla alustalla. Web-sovellukset ovat web-rajapintaa hyödyntäviä sovelluksia, jotka näyttävät puhelimessa ihan normaaleilta sovelluksilta (kuva 1), mutta käytännössä käyttäjä näkee sovelluksen avatessaan muokatun web-sivun. Web-sovellukset myös vaativat jatkuvan internetyhteyden toimiakseen.



Kuva 1. Web-sovellus vasemmalla ja natiivisovellus oikealla Android-alustalla.

Web-sovellusten kehittämiseen käytetään useimmiten web-sovelluksiin erikoistuneita kieliä, kuten HTML, CSS, AngularJS ja JavaSript. Yksinkertaisten web-sovellusten kehittäminen voi olla yksinkertaista myös ohjelmoinnin vasta aloittaneille. Standardoidut web-teknologiat tarjoavat web-sovelluksien kehittäjille lähtökohtaisen alustariippumattomuuden, joka myös vähentää yrityksien tarvetta investoida eri ohjelmoijatiimeihin [4]. Eli yhden koodikielen opettelulla web-sovelluskehyksiä käyttämällä voidaan tuottaa laajasti alustariippumattomia sovelluksia sekä tuoda näitä sovelluksia kaikkien alustoiden käyttäjille.

Web-sovellusten suosio on kasvanut viimeisen kymmenen vuoden aikana, niin kehittäjien kuin käyttäjien keskuudessa, kun 3G ja 4G ovat mahdollistaneet nopean tiedonsiirron myös mobiilialustoilla. Nopeampi mobiiliverkko on antanut kehittäjille mahdollisuuden luoda tehokkaampia web-sovelluksia. Kaupallistumassa olevan 5G-tiedonsiirtomenetelmän luulisi lisäävän vielä entisestään web-sovelluksen valitsevien kehittäjien määrää.

Web-sovellukset pystyvät kopioimaan natiivisovelluksille omaisia järjestelmätoimintoja luomalla sovellukseen WebView (Android) ja UIWebView (iOS) näkymiä, kuten geo-merkkaus, geo-synkro-

nointi, kamera ja osoiteluettelo, jotka ottavat yhteyttä alustan komponentteihin. Käyttökokemuksesta kuitenkin tulee kankeampi kuin vastaavan natiivisovelluksen [5]. Natiivisovelluksilla on mahdollista kutsua suoraan järjestelmän API:en kautta näitä ominaisuuksia ja siksi tuottaa niitä nopeammin kuin web-sovelluksilla. Kuitenkin web-sovellusten kehittämiseen erikoistuneet sovelluskehykset pyrkivät optimoimaan tätä sovelluksen ja järjestelmän välistä kommunikaatiota omilla rajapinnoillaan. Yleensä web-sovelluksen käyttäjäkokemus voi lopulta hyvinkin vastata natiivisovellusta.

2.3 Hybridisovellus

Hybridisovellus on web-sovelluksen tavoin lähtökohtaisesti alustariippumaton, vaikka se ei käytä web-pohjaisia ratkaisuja sovelluksen grafiikan piirtoon. Hybridisovellus käyttää piirtoon joko sovelluskehyksen omaa grafiikkamoottoria tai alustan omia UI-komponentteja ja toimii piirrollisesti kuin natiivisovellus. Alustan UI-komponenttien tai erillisen moottorin käyttö lisää sovelluksen tehokkuutta ja web-sovelluksille tyypillinen käyttöliittymän kankeus vähenee. Alustasta erillisten UI-toimintojen avulla nämä sovelluskehykset pystyvät tuottamaan sovelluksiin rikkaampia UI-elementtejä ja piirtämään niitä natiivisovelluksille omaisilla nopeuksilla.

Hybridisovellukset tuotetaan web-sovellusten tavoin yhden lähdekoodikielen avulla. Kuitenkin web-sovelluksista poiketen hybridisovellusten kehittämisessä usein käytetään pienissä määrin myös alustakohtaista lähdekoodia. Tällaista lähdekoodia käytetään sellaisissa kohdissa, missä alustasta riippuen järjestelmäkohtaisia tietoja haetaan eri tavoin. Näitä kohtia voivat olla tiedostorakenteeseen liittyvät toiminnot, kuten kuvien tai tiedostojen haku ja esille tuonti.

Hybridisovelluskehyksetkään eivät kuitenkaan pysty käyttämään kaikkia järjestelmän komponentteja ilman erillistä rajapintakeskustelua. Tämän takia hybridisovelluksia ei kehitetä kovin laskelmallisiksi, vaan yleensä hybridisovellukset ovat UI-painotteisia ja toiminnoiltaan yksinkertaisia.

2.4 Alustariippumattomuustapojen vertailu

Web-sovellusten kehittäminen voidaan aloittaa web-sovelluskehysten avulla varsin vauhdikkaasti. Tuki tiettyjen alustan komponenttien ja ominaisuuksien kanssa on rajallista tai lähes olematonta web-sovelluskehyksille. Web-sovellukset myös vaativat jatkuvan internetyhteyden toimiakseen, ja lisäksi web-sovellukset joutuvat käymään läpi erinäisiä rajapintoja ja lisätyökaluja tuottaakseen natiivin omaisia toimintoja, mikä yleensä ilmenee kankeampana sovelluksen toimintana verrattuna vastaavaan natiivisovellukseen.

Hybridisovellukset ovat lähempänä natiivia kokemusta, mutta vaativat ainakin osittain myös alustapohjaisia komponentteja tai lähdekoodia. Hybridisovelluksissa pystytään kuitenkin pääosin kirjoittamaan käyttämällä yhtä kieltä ja ne keskittyvät enemmän sovelluksien graafisiin piirteisiin.

3 Sovelluskehykset

Sovelluskehyksiä, puhekielisesti Frameworkeja, käytetään tyypillisesti laajemman skaalan komponenttien tuottamiseen pienemmällä vaivalla. Sovelluskehykset tarjoavat kehittäjälle suuren määrän eri komponentteja ja ominaisuuksia sovelluksen kehittämiseen ilman, että ne täytyy kirjoittaa pohjalta lähtien.

Mobiilisovelluskehyksille yleistä on alustan tai kehyksen omien UI-ominaisuuksien käyttämisen helppous. Näin kehittäjät saavat helposti aikaan hienoilta näyttäviä mobiilisovelluksia pienellä työllä. Osa mobiilisovelluskehyksistä pystyy lisäksi tarjoamaan komentoja alustan järjestelmän toimintoihin, kuten tiedoston hallintaan, ja sensoreihin, kuten kamera.

Kehittäjille sovelluskehykset tarjoavat hyvän määrän valmista pohjaa kehittämisen aloittamiseen. Kuitenkin tulee sovelluskehyksen valintaa pohtia tarkemmin ennen kehittämisen aloittamista, koska sovelluskehykset vaihtelevat laajasti tarjoamissaan ominaisuuksissa ja tarkoitukseen sopimattoman sovelluskehyksen valinta vain lisäisi työn määrää [5].

Internetistä löytyy valtava määrä eri sovelluskehyksiä. Tässä työssä käytettävät sovelluskehykset valittiin yleisyyden ja teknisen yhteisön kokemuksien kautta. Alla on jaettu luettelo nopeasti löydetyistä mobiilisovelluskehyksistä (taulukko 1). Muutama sovelluskehys omasi niin hybridi- kuin web-sovelluskehyspiirteitä.

Web-sovelluskehykset:	Hybridisovelluskehykset:
Apache Cordova	React native
Crosswalk Project	Flutter
iPFaces	Qt
iUI	jQuery Mobile
NEXT	Ionic
NSB/appStudio	Framework 7
Sencha Touch	Onsen UI
Ratchet	Quasar Framework
Lungo	MoSync
Junior	Codename One
jQT	
Famo.us	
Jo	
PhoneGap	
Onsen UI	
jQuery Mobile	
Framework7	
Mobile Angular UI	
Onsen UI	
Quasar Framework	
RhoMobile suite	
The-M-Project	

Taulukko 1. Löydetyt sovelluskehykset.

3.1 Flutter

Flutter on Googlen kehittämä mobiiliapplikaatio-SDK, jolla pystyy rakentamaan korkealaatuisia mobiiliapplikaatioita natiivikokemuksella iOS:lle ja Androidille. Flutter kategorioituu alustariippumattomuudessa hybridisovelluskehyksen ja natiivin kehityksen väliin, ja sitä kehitetään Googlen kehittäjien ja yhteisön yhteisen panoksen kautta. [6.] Flutterin ensimmäinen julkinen testiversio julkaistiin toukokuussa 2017. Kyseessä on siis suhteellisen uusi sovelluskehys, mutta on silti kasvattanut kehittäjäkuntaansa. Flutter julkaisi ensimmäisen kokoversionsa, version 1.0, joulukuussa 2018, minkä kautta sen oma dokumentaatio täydennettiin ajan tasalle helpottaen kehittämistä.

Kehittäminen Flutterilla perustuu Dart-ohjelmointikielen ja widget-nimisten pienten ohjelmistojen yhteiskäyttöön. Widgetit käytännössä toimivat kehyksenä kaikelle sovelluksessa piirrettävälle toiminnalle. Widgetit asettavat UI-komponenteille koot ja paikat näytössä. Nämä paikat sitten täytetään tarpeen mukaan eri UI-komponenteilla, kuten lista, painike tai tekstikenttä.

Flutter ei käytä ollenkaan alustan omia UI-elementtejä vaan käyttää taustalla Googlen ylläpitämää Skia-grafiikkakirjastoa [6]. Samaa kirjastoa käytetään esimerkiksi Google Chrome -selaimessa grafiikan piirtoon. Muiden ohjelmistokomponenttien ja alustakohtaisten toimintojen käyttöön voidaan käyttää Flutter-kehittäjien mallintamia ja tuottamia paketteja.

Paketit kokonaisuudessaan ovat toinen iso osa Flutterilla kehittämistä. Paketeilla voidaan tuoda sovelluksiin toiminnallisuutta, mitä ei löydy Flutterista suoraan itsessään. Pakettien käyttäminen ohjelmoinnin osana on helppoa. Haluttujen pakettien nimet kirjoitetaan Flutterin luomaan erilliseen tiedostoon, josta ne voidaan ladata osaksi lähdekoodia. Saatavilla olevista paketeista on kerätty erillinen listaus internetiin. Paketteja voi etsiä hakusanan mukaan, ja kunkin paketin sivulta löytyy yleensä käyttöesimerkki ja versiohistoria, mistä voi tarkastella paketin toimintaa ja muutoksia [7].

Flutter sopii hyvin yksinkertaisten ja pienten sovellusten luontiin. Laajempien sovellusten laskennallinen tehokkuus on heikohkoa, mutta isotkin UI-painotteiset sovellukset toimivat kelvollisesti Skia-kirjaston ansiosta. Yhteisön kehittäjien tekemät paketit mahdollistavat laajasti eri alustakohtaisten toimintojen, kuten kameratuen lisäämisen sovellukseen. Kokeneille Javascript- tai C-pohjaisten kielten ohjelmoijille on Flutteriin totuttautuminen helppoa Dart-kielen samantyylisen syntaksin avulla [8].

3.2 Qt

Qt tarjoaa laajan ohjelmointiympäristön kehittäjille. Qt:ta voidaan käyttää erinäisten tietokone- tai mobiilisovelluksien luontiin [9]. Qt on kuitenkin opinnäytetyössä listatuista sovelluskehyksistä heikoin alustariippumattomuuden tarjonnassaan. Qt vaatii erilliset koodipohjat Android- ja iOS-alustoille, ja vaikka Qt:n kääntäjä pystyy kääntämään yhtenäisen lähdekooditiedoston samaan aikaan molemmille alustoille, pitää lähdekoodi kirjoittaa erikseen molemmille alustoille.

3.3 React Native

React Native on JavaScript-kieltä käyttävä sovelluskehys, joka pohjautuu Facebookin kehittämään avoimen lähdekoodin JavaScript-kirjastoon nimeltä React. React Native on suunniteltu kirjoittamaan natiivisti piirtyviä mobiiliapplikaatioita iOS:lle ja Androidille [10]. React Nativen ensimmäinen julkinen versio tuli jakoon GitHub-versionhallintasovelluksessa vuoden 2015 toukokuussa. Flutteriin verrattuna React Native on ehtinyt jo kasvattaa käyttäjäkuntaa ja yhteisöä useamman vuoden ajan. React Nativen kehittäjät myös ylläpitävät nopeaa parin viikon välistä päivityssykliä [11]. Päivitysten tahti varmistaa uusien ominaisuuksien lisäämisen nopeasti ja nopeat korjaukset yhteisön tai kehittäjien löytämiin ”bugeihin”.

React Native -sovellukset kirjoitetaan käyttämällä JavaScriptin ja JSX:n syntaksien sekoitusta. Se myös käyttää hyväkseen alustakohtaisia UI-komponentteja ja saa niiden avulla aikaan natiivisovelluksen tapaisen kokemuksen [10]. Alustakohtaiset UI-komponentit erottavat sen web-sovelluskehyksistä, jotka yleensä yrittävät kopioida UI komponenttien ilmettä käyttämällä web-pohjaisia ratkaisuja, kuten HTML ja XML.

React Nativen käyttöönotto voi olla haastavaa aloittelevalle ohjelmoijalle. React Native vaatii tiettyjen CLI-työkalujen eli komentorivityökalujen asentamisen. Asentamiseen voi käyttää apuna komentoriveille luotuja pakettimanagereita. Windowsilla tulee vielä erikseen lisätä muutamia järjestelmämuuttujia mobiilialustan tiedostopolkujen osalta.

React Native sopii hyvin pienten ja yksinkertaisten mobiilisovellusten kehittämiseen. React Nativen etu on sen alustakohtaisten UI-komponenttien käyttö. React Native myös vaatii laajahkoa tuntemusta alustakohtaisista toiminnoista ja komponenteista tuodakseen niitä kehitettävään sovellukseen. Vaikka React Native tarjoaa alustariippumattomia toimintoja laajasti, on esimerkiksi alustan sensoreita koskevat osat yhä kirjoitettava alustakohtaisesti [12].

3.4 Xamarin

Xamarin on hybridisovelluskehys iOS-, Android- ja Windows Phone -alustoille. Xamarin pohjautuu Microsoftin omistamaan Mono-projektiin, joka on hybridisovellustoteutus .NET-ohjelmistokehyksestä. Xamarinilla kehitys on ilmaista yksityisesti sekä pienillä yrityksillä, mutta näitä suuremmilla

yritystasoilla sovelluskehyksen käyttöön tulee ostaa maksullinen lisenssi. Xamarinin ensimmäinen Android-tuki tuli Android 4.4 -versiolle. Ennen tätä Microsoft käytti Mono-nimeä mobiilipohjaisille kehitystyökaluille. Aikaisin Mono-yhteensopivuus Android-version kanssa on Android 1.0:lla. IOS:lla Xamarin-tuki tuli iOS-versio 6:lle, sitä aiemmin myös Mono-versiot olivat käytössä iOS:lla versio 1:sta lähtien. [13.]

Xamarinilla sovellusten kehitykseen käytetään C#-lähdekoodikieltä yhdessä UI:n määrittävän XAML-merkintäkielen kanssa. XAML pohjautuu yleisesti käytössä olevaan XML-merkintäkieleen. Xamarin-lähdekoodi käännetään natiivisti alustakohtaisten kääntäjien avulla, mikä antaa mahdollisuuden tehokkaiden natiivilta näyttävien sovellusten luontiin [13]. Xamarin myös mahdollistaa alustakohtaisen UI-komponenttien luonnin React Nativen tavoin, mikä antaa sovelluksille paremman käyttäjäkokemuksen ja sovellukset näyttävät natiivilta alustasta riippumatta.

Xamarinilla kehittäminen vaatii alustakohtaiset IDE:t Windowsille tai MacOS:lle. Alustan mukaan ladataan internetistä sopiva ohjelmointiympäristö, Visual Studio tai Visual Studio for MacOS [14]. Xamarin-projekteja ei voida kääntää muilla IDE:llä, koska Xamarin toimii lisäosana Visual Studiolle. Visual Studion asennuksen yhteydessä tulee valita Xamarin-lisäosat asennuksen aikana [14].

Xamarin on erittäin laaja sovelluskehys, johon on saatu upotettua paljon toiminnallisuutta unohtamatta alustariippumatonta kehitystapaa. Kuitenkin kehittäminen Xamarinilla vaatii jossain määrin alustakohtaista tuntemusta. Esimerkiksi tiedostopolkujen hakuun tulee lisätä erillinen käsittely Androidille ja iOS:lle.

3.5 PhoneGap

PhoneGapin on kehittänyt yhtiö nimeltä Nitobi Sofware. ICT-yritys Adobe hankki Nitobi Softwaren vuonna 2011. Hankinnan jälkeen PhoneGapin lähdekoodi lahjoitettiin Apache Software Foundationiin koodinimellä Cordova [15]. Lahjoituksen takia PhoneGap on nyt ilmainen ja avoimen lähdekoodilisenssiä käyttävä muunnos Apache Cordovasta. Apache Cordovaa voidaan pitää PhoneGapin moottorina, mikä tarjoaa käyttöjärjestelmäkomponentteja liittämällä alustan API:t sovelluksen lähdekoodin kanssa. Phonegapin tavoitteena on tuottaa avoimen lähdekoodin ratkaisuja alustariippumattomien mobiilisovellusten luontiin standardisoitujen web-teknologioiden,

kuten HTML, JavaScript ja CSS avulla [15]. PhoneGapilla on kehitetty mobiilisovelluksia jo pidemmän aikaa, ja se on luonut laajan ja kehittyneen yhteisön.

Kehittämiseen PhoneGapilla tarvitaan PhoneGapin työpöytäsovelluksen ja mobiilisovelluksen kombinaatio. Aluksi asennetaan PhoneGap Desktop -työkalu, joko Windowsille tai MacOS:lle. Sen lisäksi asennetaan PhoneGap-mobiilisovellus halutulle mobiilialustalle, joko Android- tai iOS-puhelimelle [15]. Työpöydän ja mobiilialustan tulee olla samassa WiFi-verkossa sovelluksen kehittämisen ja testaamisen mahdollistamiseksi. PhoneGapin työpöytäsovelluksessa luodaan PhoneGap-projekti, joka rakentaa kehittämiseen vaaditut JavaScript- ja HTML-tiedostot. PhoneGap-työpöytä-sovellus luo URL-osoitteen, missä voi tarkastella kehitettävänä olevan sovelluksen ilmettä, joko internet-selaimessa tai PhoneGap-mobiiliapplikaatiossa.

PhoneGap on pitkälle kehitetty web-sovelluskehys, josta löytyy laajasti ominaisuuksia alustariippumattomaan kehittämiseen. Kehittäminen PhoneGapilla on ilmaista ja lähdekoodi on avoin. Aloittelevan mobiilikehittäjän näkökulmasta PhoneGap on erittäin varteenotettava vaihtoehto helppokäyttöisyyden ja ilmaisuuden takia. PhoneGapilla on myös laaja käyttäjäkunta ja internetistä löytyy dokumentaatioita ja opasvideoita kehittämisen eri vaiheisiin.

Tietyt toiminnot, kuten tietokantoihin liittymiset ja tiedostojen tallentamiset, toimivat vielä rajoittavana tekijänä PhoneGapilla kehittäessä, mutta yksinkertaisten mobiilisovellusten luontiin PhoneGap on hyvä vaihtoehto. Suurin osa alustojen järjestelmien API:sta on tullut käyttökelpoisiksi Cordovan 3.0-version mukana.

3.6 Onsen UI 2

Onsen on laaja kokoelma erinäisiä käyttöliittymäkomponentteja erityisesti mobiilisovelluksille [16]. Onsenia voidaan käyttää itsenäisesti kehittämään yksinkertaisia UI-mobiilisovelluksia, tai sen voi ladata lisäosana eri sovelluskehyksille, jos on tarkoitus tehdä vaativampia mobiilisovelluksia [17]. Onsenissa on perehdytty vielä enemmän luomaan natiivin näköisiä UI-komponentteja. Onsenin käyttö on ilmaista ja sen lähdekoodi on avoin.

Onsenilla kehittämiseen tarvitaan osaamista useammasta lähdekoodikielestä. Onseniin kuuluvat CSS-komponentit kirjoitetaan käyttämällä cssnextiä, ja web-komponentit kirjoitetaan käyttämällä

JavaScriptiä. Onsen käyttää valmiita komponentteja erilaisten UI-toimintojen luontiin. [17.] Valmiita komponentteja käyttämällä saadaan aikaan mobiilisovelluksia, jotka näyttävät ja toimivat kuten natiivisovellus. Mutta käyttämällä vain Onsenia ei päästä kiinni alustan omiin sensoreihin tai ohjelmistoihin, koska Onsen periaatteessa toimii vain UI-rajapintana mobiilisovelluksille. Alustakohtaisten toimintojen käyttöön voi Onsenin liittää muihin sovelluskehyksiin tai kieliin, kuten Apache Cordova, Angular.js ja javaScript. Näiden sovelluskehysten yhteiskäyttö Onsenin kanssa mahdollistaa alustatoimintojen käytön ja UI:n muokkaamisen natiivin näköiseksi ja tuntoiseksi.

4 Sovelluskehysten vertailu

Tässä luvussa käydään läpi edellä mainittujen sovelluskehysten sopivuutta mobiilisovelluksien kehittämiseen. Tutkitaan, mitä ominaisuuksia kyseiset sovelluskehykset tarjoavat ja miten sovelluskehysten käyttö eroaa toisistaan. Käydään myös läpi sovelluskehyksien alustasta riippuvia eroavaisuuksia, eli miten Android tai iOS sopii yhteen sovelluskehysten kanssa. Tarkastellaan myös, ovatko alustariippumattomien sovelluskehyksien tarjoamat hyödyt sekä haitat suhteessa natiiviohjelmoinnin tarjoamiin mahdollisuuksiin.

Sovelluskehysten vertailulla on tarkoitus kartoittaa muutama sovelluskehys, jotka sopivat ominaisuuksiltaan ja käytettävyydeltään työhön liittyvän mobiilisovelluksien kehittämiseen.

Työn asiakkaan toiveet huomioiden sovelluskehysten tärkeimpinä ominaisuuksina pidetään tukea kameralle ja erityisesti viivakoodiskannerin mahdollistamisen alustariippumattomasti. Toisena tärkeänä ominaisuutena tullaan pitämään tietokantayhteensopivuuksia ja mahdollisuutta tallentaa tietoja tietokantaan sekä alustan omiin tiedostoihin.

4.1 Grafiikka ja UI

Kaikista työssä läpikäydyistä sovelluskehyksistä löytyy laajasti ominaisuuksia grafiikan piirtoon ja UI:n muokkaamiseen. Tämä johtuu yleisestä mobiilisovellusten ulkönäön ja käytettävyyden tärkeydestä käyttäjille. Tosin grafiikan piirrossa ja UI:n käytössä löytyy eroavaisuuksia mobiilialustasta riippuen, kuten mistä suunnasta ruutua vierittämällä avataan asetukset ja kummalta puolelta puhelimen ruutua näytetään avattu listaus. Löydetyt eroavaisuudet eivät kuitenkaan tuota merkittäviä ongelmia kehittämiseen, sillä työssä käytetyt sovelluskehykset ovat pääsääntöisesti jo huomioineet nämä eroavaisuudet ja osaavat automaattisesti muuttaa käsittelyä alustan perusteella. Näin ollen yhdellä lähdekoodilla voidaan tuottaa samanaikaisesti Android- ja iOS-yhteensopivia UI-elementtejä. Taulukossa 2 on kirjattu sovelluskehysten tukemia UI-elementtejä.

Merkintä ”+” tarkoittaa, että tuki ominaisuudelle löytyy suoraan sovelluskehyksestä. Merkintä ”O” tarkoittaa, että tuki ominaisuudelle on mahdollista hankkia kolmannen osapuolen kirjastojen tai pakettien kautta. Merkintä ”-” tarkoittaa, että ominaisuutta ei tueta työn kirjoitushetkellä.

	Sovelluskehys:					
Ominaisuusluettelo:	Flutter	React Native	PhoneGap	Onsen UI 2	Xamarin	Qt
UI:						
Animaatiot	+	+	+	O	+	+
Kuvat, Ikonit	+	+	+	+	+	+
Fonttien muutos/teksti tuki	+	+	O	+	+	+
Kosketustuki	+	+	+	+	+	+
Kosketuseletuki	+	+	+	+	+	+
Kosketuksella ruudunvieritystuki	+	+	O	+	+	+
Ruudun skaalaus	+	+	+	+	+	+
WebView	O	+	+	+	+	+

Taulukko 2. Sovelluskehysten tarjoamat UI-ominaisuudet.

Taulukosta voidaan päätellä, että kaikki läpikäydyt sovelluskehykset pystyvät tuottamaan tarpeellisia UI-elementtejä riittävästi ainakin lisäosien tukemana. Onsen UI ei sen yksinkertaisuuden puolesta pysty suoraan tukemaan animaatioita, mutta liittämällä sen toiseen sovelluskehykseen, kuten React Native, voidaan animaatioita tuottaa sen kautta. PhoneGapista ei suoraan löydy tukea erillisten fonttiperheiden ja ruudunvierityksen tukemiseen, mutta tässäkin tapauksessa voidaan tarpeen mukaan käyttää ladattavia Apache Cordova -kirjaston tarjoamia paketteja. Flutter ei sen oman Skia-grafiikkakirjaston takia tarvitse tukea webviewille, mutta sen voi halutessaan ladata kolmannen osapuolen pakettina.

4.2 Kehitystyökalut, tietokannat ja tiedostorakenne

Tällä osa-alueella sovelluskehyksistä löytyy suuriakin eroavaisuuksia ominaisuuksien tarjontaan. Web-sovelluskehykset, kuten PhoneGap ja Onsen UI, eivät lähtökohtaisesti tue suurta osaa alustan tiedostorakenteeseen liittyvistä ominaisuuksista ja tarvitsevat lisäosia tämänlaisen toiminnallisuuden mahdollistamiseksi. Flutter, React Native ja Xamarin ollessaan hybridisovelluskehyksiä voivat jo lähtökohtaisesti keskustella osan alustan järjestelmä-API:n kanssa ja siksi pystyvät tuottamaan helpommin tämänkaltaista toiminnallisuutta. Qt:n ollessa laajempi ohjelmointiympäristö se tukee näitä ominaisuuksia, mutta tarvitsee siihen natiiviluokkia ja lähdekoodia. Taulukosta 3 selviää eri sovelluskehysten tarjoamia mahdollisuuksia kehitystyökaluihin, tietokantoihin ja tiedostorakenteeseen liittyvissä ominaisuuksissa.

	Sovelluskehys:					
Ominaisuusluettelo:	**Flutter**	**React Native**	**PhoneGap**	**Onsen UI 2**	**Xamarin**	**Qt**
Kehittäminen ja Tiedostot:						
kolmannen osapuolen lisätyökalu/ kirjastotuki	+	+	+	+	+	+
Alustariippuvat luokat	+	+	O	-	+	+
Tietokantatuki	+	+	+	-	+	+
Tiedostoon pääsy	+	+	O	-	+	+
Ohjelman allekirjoitus ja sovelluksen julkaisu	+	+	+	+	+	+
Automaattinen version päivitys	+	O	+	-	+	-
Ajonaikainen lähdekoodin muunnos	+	+	+	+	+	-
Emulaattorituki	+	+	+	O	+	+

Taulukko 3. Sovelluskehysten tarjoamat ominaisuudet kehittämiseen ja tietojenhallintaan.

Taulukosta huomataan, että kaikki sovelluskehykset tukevat kolmannen osapuolen lisätyökaluja ja kirjastoja. Tuki lisätyökaluille ja kirjastoille tuo myös Web-sovelluskehykset Onsen UI:n ja PhoneGapin näiltä osin samalle tasolle hybridisovelluskehysten kanssa.

Alustariippuvat luokat, jotka ovat tarpeellisia esimerkiksi mobiilialustalta löytyvien tiedostojen hakuun, löytyvät luonnostaan kaikista hybridisovelluskehyksistä. Molemmat web-sovelluskehykset ollessaan Apache Cordovasta pohjautuvia sovelluskehyksiä voivat käyttää Cordovan Plugin-järjestelmiä lisäämään Android- ja iOS-toimintoja.

Tietokantojen tukeminen on vähän monimutkaisempaa Web-sovelluskehyksissä kuin hybridisovelluskehyksissä. Niistä ei löydy tukea lainkaan yhteydettömille tietokannoille, mutta PhoneGapilla tehty sovellus on mahdollista liittää verkossa oleviin tietokantoihin. Onsen sen sijaan ei suoraan tue lainkaan tietokantoja, jos sitä ei käytetä lisäosana toista sovelluskehystä tai ohjelmointiympäristöä. Kaikki hybridisovelluskehykset tarjoavat SQL-pohjaisia tietokantamahdollisuuksia niin yhteydettömille kuin verkossa oleviin tietokantoihin. React Nativeen ja flutteriin voidaan liittää SQL-pohjaisia tietokantoja erillisten pakettien ja lisäosien kanssa. Xamarinkin tukee yhteydettömiä tietokantoja omilla työkaluilla, mutta verkkotietokantoihin tulee väliin rakentaa verkkopalvelin, joka yhdistää sovelluksen ja tietokannan, esim. Azure-pilvipalvelimet. Qt taas tarjoaa lähtökohtaisesti omat syntaksit SQL-komentojen käyttöön ja tukee yleisiä SQL-pohjaisia tietokantoja.

Tiedostoihin ja tiedostorakenteeseen käsiksi pääsy on myös vaikeaa web-sovelluskehyksille. PhoneGapissa on mahdollista käyttää Cordovan tarjoamia lisäosia tuottamaan tiedoston hakutoimintoja, kun taas Onsen UI vaatii sen sisällyttämisen erilliseen sovelluskehykseen tai ohjelmointiympäristöön. Hybridisovelluskehykset pystyvät hakemaan tiedostoja ilman erillisiä lisäosia, kuitenkin

tässä tarvitaan hiukan alustakohtaista lähdekoodia. Flutter ja React Native tarvitsee alustatarkistuksen alustan omalla lähdekoodilla ennen tietokantakäsittelyä tietääkseen, minkä alustan tiedostorakennetta läpikäydään haun yhteydessä. Xamarinilla kirjoittaessa Xamarin.Android- ja Xamarin.iOS-projekteja voidaan tiedostoja käyttää suoraan. Xamarin.Forms-projekteissa tulee lähdekoodissa erotella tiedostojen käsittely alustan mukaan Flutterin ja React Nativen tavoin. Qt:lla taas joudutaan tiedoston haut kirjoittamaan täysin alustakohtaisella lähdekoodilla.

Kaikilla sovelluskehyksillä on mahdollista tuottaa sertifioituja sovelluksia niin Android- kuin Apple Store -kauppapaikoille. Osalla sovelluskehyksistä on myös mahdollista toteuttaa automaattinen version päivitys, eli tunnistaako sovellus uuden version lisäyksen ja osaako se automaattisesti päivittää nykyisen version. Tämä ominaisuus on enemmänkin kehittäjien kehityskokemusta parantava, eikä kerro merkittävästi, onko toinen sovelluskehys toista parempi.

Ajonaikainen lähdekoodin muunnos taas helpottaa kehittäjiä vähentämällä vaadittua aikaa pienten lisäysten, kuten tekstikenttien muutosten testaamisessa. Tämäkin ominaisuus on tuotu esiin suurimmassa osaa sovelluskehyksistä ja oli yllättävän suuressa painoarvossa sovelluskehyksien markkinoinnissa [18]. Qt ei tällä hetkellä tue kyseistä ominaisuutta.

Huomattiin myös, että sovelluskehykset mahdollistavat emuloidun mobiilialustan käyttämisen kehittämisen aikana, mikä on kehittäjien näkökulmasta erittäin tarpeellinen ominaisuus. Onsen UI:sta puuttuu suora tuki emulaattoreille, mutta jälleen voidaan Onsen UI liittää osaksi toista sovelluskehystä emuloitujen alustojen käyttämiseksi.

4.3 Mobiilialustan sensorit

Merkittävällä osalla työn sovelluskehyksistä ei löydy suoraa tukea mobiilialustan sensoreiden käyttöön, mutta lisäosien avulla pystyvät kaikki sovelluskehykset tuottamaan tarpeelliset toiminnallisuudet. Tällä osa-alueella sovelluskehysten käyttö eroaa natiiviohjelmoinnista eniten. Mobiilialustan natiivikielellä kirjoittaessa on alustan sensoreiden käyttö suoraan kytköksissä lähdekoodikielen syntakseihin ja sensoreita käyttävät sovellukset ovat helppo tuottaa. Taulukossa 4 on merkitty sovelluskehysten tarjoamat tuet sensorien käyttöön.

	Sovelluskehys:					
Ominaisuusluettelo:	Flutter	React Native	PhoneGap	Onsen UI 2	Xamarin	Qt
Sensorit:						
Kamera	O	O	O	O	O	+
Sijaintitiedot/GPS	O	+	O	O	+	+
Ruudun kääntö/vatupassi	+	+	O	+	+	+
Bluetooth	O	+	O	O	O	+
WiFi	O	+	O	O	O	+
NFC	O	O	O	O	O	+

Taulukko 4. Sovelluskehysten tuet järjestelmän sensoreille.

Taulukosta nähdään, että kaikki sovelluskehykset pystyvät tuottamaan toimintoja alustan sensoreiden käyttöön ainakin lisäosien tukemana. Enemmän alustariippuvaisesta kehityksestä tunnettu Qt tukee suoraan kaikkia alustan sensoreita omien syntaksien kautta, mutta taas tarvitaan erilliset alustakohtaiset luokat Androidille ja iOS:lle.

Flutterin toiminta perustuu erinäisten pakettien lisäämiseen kehitettävän sovelluksen tarpeen mukaan. Kaikkiin tarvittaviin sensoreihin löytyy valmiit paketit joko yhteisön tai Googlen tekeminä Flutterin omilta pakettisivuilta.

React Nativesta löytyy suoraan jo suurin osa sensoreiden tuista, ja puuttuvat kameran ja NFC-tuet voidaan hankkia kolmannen osapuolen kehittämistä paketeista. Xamarinkin tarvitsee lisäosia sensoreille. Kaikille sensoreille löytyy kolmannen osapuolen tekemiä paketteja, jotka voidaan lisätä Xamarin projekteihin kehityksen alussa.

Web-sovelluskehykset, PhoneGap ja Onsen UI, eivät pääse käsiksi alustan sensoreihin ilman lisäosia. Apache Cordovan paketit sensoreille toimivat hyvin yhteen PhoneGapin kanssa ja niiden yhteenliittäminen kehittämisen aikana on helppoa kutsumalla vain Cordova-kirjastoa. Onsen UI voidaan taas liittää yhteen toisen sovelluskehyksen kautta sensoritoimintojen tuottamiseksi.

Hybridisovelluskehyksillä ja Web-sovelluskehyksillä ei ole suurempia eroavaisuuksia sensoreita käyttävien sovellusten kehittämisessä. Yleisesti ei-natiivit sovelluskehykset tarvitsevat lisäosia tukeakseen laajasti alustan sensoreiden käyttämistä.

4.4 Sovelluskehysten käyttöönotto

4.4.1 Sovelluskehyksien lataaminen ja asentaminen

Työssä käytetyt sovelluskehykset tulee ladata erikseen internetistä. Kunkin sovelluskehyksen omilta internetsivuilta löytyy yleispätevät ohjeet kehyksien lataamiseen. Tulee kuitenkin huomioida latauksen yhteydessä, millä työpöydällä kehitetään ja mille alustalle kehitetään. Osa sovelluskehyksistä vaatii eri ohjelmointiympäristöjä tai paketteja Windowsille, Linuxille ja MacOS:lle. Alle on koottu selventävä taulukko 5 sovelluskehysten tukemista alustoista ja millä työpöydillä voi kehittää halutulle alustalle.

	Kehykset					
Työpöytä tyypit:	Flutter	React Native	PhoneGap	Onsen UI	Xamarin	Qt
Linux	Android	Android	Android	Android	ei tueta	Android
MacOS	iOS/Android	iOS/Android	iOS/Android	iOS/Android	iOS/Android	iOS/Android
Windows	Android	Android	iOS/Android	Android	Android	Android

Taulukko 5. Työpöydät ja sovelluskehysten tukemat mobiilialustat

Flutterilla, React Nativella ja Xamarinilla tehtävän sovelluksen lähdekoodin Android- ja iOS-osiot pystytään kirjoittamaan Windows-alustalla, mutta testaaminen ja tarpeen mukaan Apple-sertifiointi tulee vielä hoitaa MacOS-työpöydällä, jolla on Xcode-kehitysympäristö ja Apple-kehittäjälisenssi.

PhoneGapilla on mahdollista Windows-puolella luoda tarvitut iOS-avaimet kehittämiseen, mutta sertifikaatin saaminen sovelluksen lataamiseen Apple Storeen tarvitsee vielä MacOS:n ja xCoden [19].

Xamarin ei tue kehittämistä Linux-käyttöjärjestelmäisillä työpöydillä. Xamarin-kehittäjät päättivät olla tukematta suurta määrää eri Linux-versioita, koska kullekin versiolle tulisi tehdä oma painos Xamarinista. Linuxilla voidaan kuitenkin käyttää MonoDevelop-järjestelmiä tuottamaan alustariippumattomia sovelluksia, mille myös Xamarin pohjautuu [20].

Sovelluskehyksien asentamisessa Qt on ainoa työkalu, joka tarjoaa kaiken tarvittavan kehittämisen aloittamiseen. Ollessaan täysi ohjelmointiympäristö, joka tarjoaa kääntäjät ja tarvittavat mobiilikirjastot latauksen yhteydessä, Qt:lla on nopeaa aloittaa mobiilisovellusten kehitys.

Hybridisovelluskehykset, Flutter ja React native, vaativat erilliset tekstieditorit ja kääntäjät, tai kokonaisen IDE:n. Flutterin voi suoraan liittää moneen ohjelmointiympäristöön erillisenä työkaluna, esim. intelliJ, Android Studio, Xcode ja VsCode. Ohjelmointiympäristön valinta on kiinni omasta mieltymyksestä ja mahdollisista alustan pakottamista rajoituksista, kuten iOS:n vaatima Xcode. React Nativen voi myös liittää moneen ohjelmointiympäristöön, ja jos työpöydältä löytyy tarvittavat työkalut, voi React Nativea käyttää vain tekstieditorin ja komentokehotekomentojen avulla.

Xamarin on erilainen verrattuna muihin hybridisovelluskehyksiin. Xamarin vaatii toimiakseen Visual Studio -ohjelmointiympäristön, sillä se toimii Visual Studion alustariippumattomuuden mahdollistavana lisäosana, minkä takia Xamarin projekteja ei voida kääntää erillisissä ohjelmointiympäristöissä tai tekstieditoreissa.

PhoneGap vaatii kehitykseen oman työpöytä- ja mobiilisovelluksen. Työpöytäsovelluksessa luodaan PhoneGap-projektit. Luotu projekti sen jälkeen liitetään yhteen mobiilisovelluksen kanssa. Mobiilisovellus toimii mallina kehitettävänä olevalle projektille. Liitoksen jälkeen voidaan projektia kehittää ja muokata halutussa tekstieditorissa tai ohjelmointiympäristössä. Liitoksen avulla muutokset projektissa tulee suoraan näkyviin mobiilisovelluksessa.

Onsen UI:lla itsessään pystytään tuottamaan vain web-sovelluksia. Eli kehittääkseen laajempia mobiilisovelluksia Onsen UI tulee liittää toiseen sovelluskehykseen. Helpoin ja Onsen UI:n kehittäjien ehdottama tapa on liittää Onsen UI yhteen Cordovan kanssa. Cordova asennetaan ja projektit luodaan käyttämällä komentokehotetyökaluja. Projektin luotua voidaan Onsen UI paketti liittää projektiin, ja Onsenin käyttö voidaan aloittaa haluamalla tekstieditorilla tai ohjelmointiympäristöllä.

4.4.2 Eroavaisuudet Android ja iOS alustoilla

Alustoista riippuvat eroavaisuudet sovelluskehysten käyttöönotosta ovat erittäin vähäisiä. Kuitenkin edellä mainittuun iOS-puolen testaamiseen ja sovelluksen Apple Storeen lisäämiseen vaaditaan Apple-kehittäjälisenssi ja Xcode sertifikaattien tuottamiseen.

4.5 Valinta skannerisovelluksen kehittämiseen

Tässä luvussa on tarkoitus valita sopivimmat sovelluskehykset skannerisovellusten kehittämiseen.

Ensimmäisenä Onsen UI jää pois skannerisovelluksen kehittämisestä. Onsen UI ei tuo tarpeeksi toiminnallisuutta alustakohtaisten järjestelmien ja sensoreiden lisäämiseksi. Onsen UI on mahdollista liittää osaksi toista sovelluskehystä tai ohjelmointiympäristöä, mutta tämä aiheuttaa tarpeetonta monimutkaisuutta kehitykseen. Lopulta ottaen huomioon muiden sovelluskehysten laajempi ominaisuustarjonta jää Onsen UI:lta lisää toivottavaa. Onsen UI kuitenkin tarjoaa paljon alustariippumattomia UI-komponentteja ja tältä puolelta se on tarjonnaltaan parhaiden sovelluskehysten joukossa.

Toisena jää pois Qt. Qt on ominaisuuksien ja järjestelmäkomponenttien puolesta erittäin hyvätasoinen ohjelmointiympäristö. Sitä voidaan käyttää tuottamaan vaativiakin mobiilisovelluksia, mutta erittäin vähäinen tuki alustariippumattomaan kehitykseen aiheuttaa liikaa vaatimuksia mobiilisovellusten kehittämiseen. Qt olisi loistava valinta, jos tulisi tuottaa natiiveja mobiilisovelluksia Androidille ja iOS:lle.

Kolmantena jää pois PhoneGap. PhoneGap on Onsenin tavoin Web-sovelluksien kehittämiseen tarkoitettu, ja sen takia ovat alustan järjestelmiä koskevat komennot erittäin rajalliset. PhoneGap kuitenkin pystyy tarjoamaan näitä komentoja Cordova-kirjaston välityksellä, mutta tämä kasvattaa kehityksen monimutkaisuutta. Lisäksi PhoneGapin heikkoutena voidaan pitää sen kehityksen

aikana vaatimaa työpöytäsovelluksen ja mobiilisovelluksen välistä yhteyttä, mitä ilman kehittäminen ei onnistu. PhoneGap-projektien kääntäminen alustoille ole yhtä yksinkertaista kuin Flutterilla, React Nativella tai Xamarinilla, koska ne ovat liitettyinä täysiin ohjelmointiympäristöihin.

Jäljellä olevat kolme sovelluskehystä React Native, Flutter ja Xamarin ovat erittäin sopivia alustariippumattomaan mobiilisovelluskehitykseen. Kaikki ovat hybridisovelluskehyksiä ja näin ollen tarjoavat helpommin alustan sensoreihin ja tiedostojärjestelmiin liittyviä ominaisuuksia. React Native eroaa muista, sillä sen sovellukset käännetään komentorivin kautta, eikä sen lähdekoodin käännöstä voida suorittaa ohjelmointiympäristöstä tai tekstieditorista. React Native myös tukee vähiten tunnettuja ohjelmistoympäristöjä, mikä voi tuottaa kehittäjille haasteita. Flutter taas tukee laajaa määrää eri ohjelmointiympäristöjen syntakseja, kunhan työkalut ovat lisättyinä. Xamarin tukee vain eri Visual Studio -versioita, mikä saattaa olla mobiilikehittäjillä vähän tunnettu IDE. Xamarinin hyvä puoli on se, että se on Microsoftin luoma ja tukee laajasti Visual Studion toimintoja, joten kunhan Visual Studioon tottuu, on Xamarinilla kehittäminen luontevaa.

Valitut sovelluskehykset eroavat toisistaan niiden tavasta käsitellä lisäosia ja kirjastoja. Flutter ja Xamarin molemmat käyttävät yksiselitteisiä pakettijärjestelmiä. Flutterilla paketit lisätään pubspec.yaml-tiedostoon, mistä ne voi ladata suoraan ja ottaa käyttöön lähdekoodissa. Xamarinin paketit ladataan suoraan projekti-kansioon Add Packages -toiminolla. Toiminto avaa erillisen ikkunan mistä hakusanan kautta voi etsiä ja ladata haluamansa paketin. Ladattujen pakettien syntaksit ja komennot tulevat suoraan käyttöön lähdekoodissa. React Nativessa paketit pitää manuaalisesti ladata komentorivin kautta ja asettaa projektikansioon. Siinä missä Flutter ja Xamarin paketit toimivat ilman erillisiä liitoksia kaikilla mobiilialustoilla, tulee React Nativen paketit liittää projekteihin alustakohtaisesti.

Näillä perusteilla on erittäin vaikea valita Flutterin, Xamarinin ja React Nativen välillä. Niinpä kaikki kolme valitaan mobiilisovelluksen kehittämiseen. Kehityksen aikana käydään laajemmin läpi näiden sovelluskehysten toimintaa käytännössä ja saadaan sitä kautta selville, mikä sovelluskehys lopulta sopii parhaiten yrityksen tarkoituksiin.

5 Sovelluskehyksillä kehittäminen

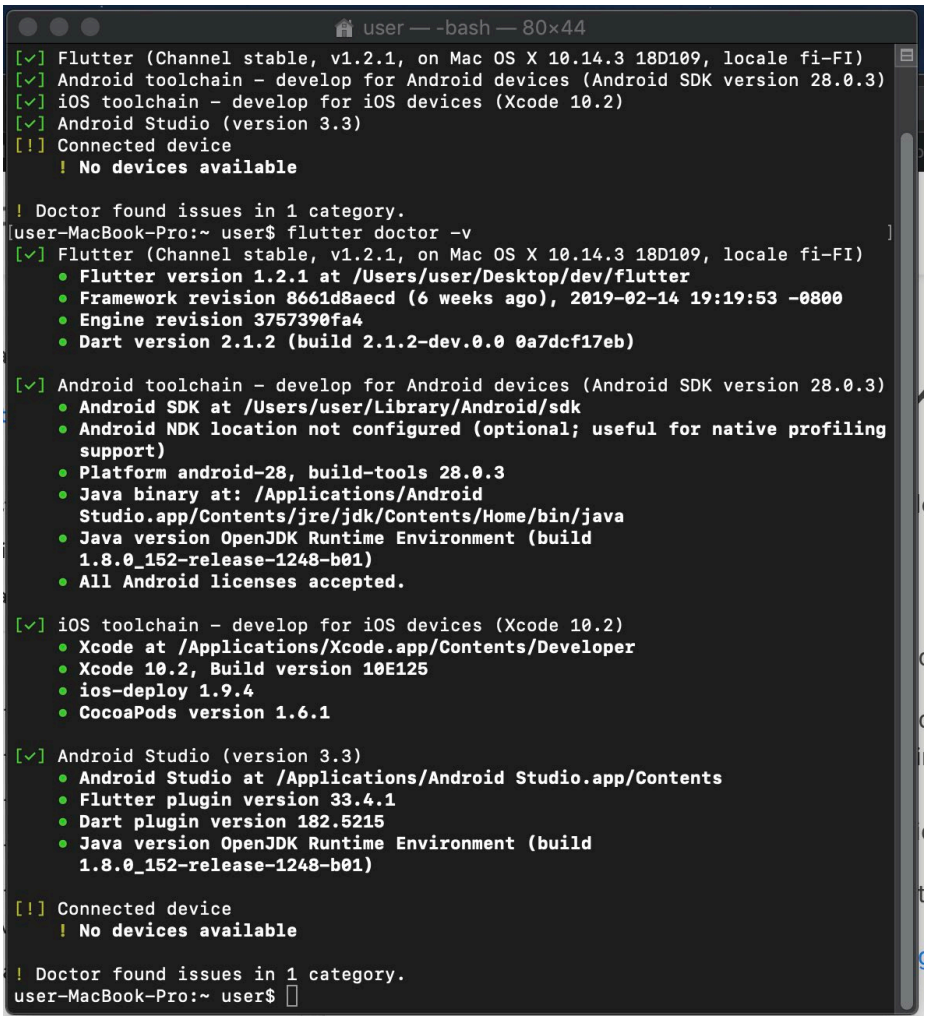
Tässä luvussa käydään läpi mobiilisovelluksen kehittämistä valituilla sovelluskehyksillä. Testattaviksi sovelluskehyksiksi valikoitui aiemmin tehtyjen päätelmien mukaan Flutter, React Native ja Xamarin.

Sovelluskehyksillä kehitettävän mobiilisovelluksen tulee pystyä skannaamaan 1D-viivakoodi ja asettaa viivakoodista saatu merkkijono sen skannausajankohdan kanssa tietokantaan. Kaikki valitut sovelluskehykset pystyvät ainakin ominaisuustarjontansa puolesta toteuttamaan vaaditun tapaisen mobiilisovelluksen. Kehittämistehtävällä onkin tarkoitus vertailla sovelluskehysten toimintaa enemmän käytännön puolelta. Tarkastellaan myös sovelluskehysten asennusta ja järjestelmävaatimuksia. Vertaillaan myös sovelluskehysten liittämistä ohjelmointiympäristöihin ja miten ne toimivat niiden kanssa.

Mahdollisimman tarkan vertailun mahdollistamiseksi mobiilisovelluksen spesifikaatiot pidetään yksinkertaisina. Mobiilisovelluksen on pystyttävä tarjoamaan skanneriominaisuus tietokannalla ja toimia Android- ja iOS-alustoilla. Tavoitteena on käyttää mahdollisimman vähän vaadituista ominaisuuksista poikkeavaa toimintaa.

5.1 Kehittäminen Flutterilla

Flutterilla kehittäminen alkaa asentamalla työpöydälle Flutter-SDK sen internetsivuilta löytyvästä latauslinkistä. Ensin ladataan oikea Flutter-versio työpöydän järjestelmän mukaan. MacOS:lla ja Linuxilla tulee asentaa muutama järjestelmätyökalu ennen Flutterin asentamista, kun Windowsilla riittää PowerShell- ja Git-työkalujen löytyminen tietokoneesta. Listaus tarvittavista työkaluista ja tarkemmat asennusohjeet löytyvät Flutterin internetsivuilta. Flutterin lataustiedosto on paketoitu, joten latauksen jälkeen paketoitu tiedosto avataan kansioon. Seuraavaksi lisätään mahdollisesti tarvittavat järjestelmämuuttujat, kuten polut, valittuun Flutter-kansioon. Järjestelmämuuttujien lisäämisen jälkeen voidaan Flutterin mukana tulleita komentorivikomentoja käyttää komentorivissä. Komentoriville voidaan kirjoittaa asennusta helpottava komento ”flutter doctor -v”, jota käyttämällä näkee tarvittavat ja puuttuvat työkalut (kuva 2).



Kuva 2. Komennon tarjoama listaus Flutter-v1.2.1:n tarvitsemista työkaluista.

Yllä oleva kuvakaappaus on otettu työpöydältä, jossa on jo asennettuna tarvittavat työkalut Flutterilla kehittämiseen. Kuitenkin kuvasta näkee, ettei työpöytään ole kytkettynä sopivaa mobiilialustaa. Flutter tunnistaa mobiilialustaksi minkä tahansa Android-SDK 16- ja sitä uudemmat Android-alustat, ja iOS 8- ja sitä uudemmat iOS-alustat. Mobiilialusta voi olla myös emuloitu, eli Android Studion ja Xcoden tarjoamat emulaattorit toimivat alustana. iOS:lla puhelin on vaadittu kameraominaisuuden testaamiseen, kun sen emulaattorissa ei ole kameraa. Osasta Android-emulaattoreita löytyy tuki virtuaalitodellisuudella toteutetulle kameralle, mikä helpotti viivakoodiskannerin testauksessa.

Kuten aiemmin mainittu, Flutterin voi liittää moneen eri IDE:hen. Kuitenkin Flutterin tarjoama suora yhteensopivuus tietyille IDE:ille kannattaa ottaa huomioon IDE:tä valittaessa. Flutterin

omista dokumentaatioista löytyy suoraan ohjeet liitokseen Android Studioon, Xcodeen, IntelliJ:hin ja Visual Studio Codeen. Kehittämisen aloitus voi siis olla helpompaa joillakin näistä IDE:stä.

IDE:stä riippuen uuden projektin luominen vaihtelee vähän. Android Studiolla, Xcodella ja IntelliJ:llä voidaan suoraan valikosta valita uusi Flutter-projekti. Visual Studio Codella avataan uusi Flutter-projekti ”view > Command Palette” -taulukon alta. Projektin luomisen jälkeen IDE:t rakentavat vaaditut esitiedostot, ja sen jälkeen projekti on valmis lähdekoodin kirjoittamiselle. Projektiin on liitetty pieni testisovellus, jolla voi varmistua asennuksien ja alustojen toiminnasta.

Projekti käännetään liitetylle tai emuloidulle mobiilialustalle ja näyttää kehitettävänä olevan sovelluksen. Käännetty projekti on Flutterilla automaattisesti ”hot reload” -tilassa, eli projektin lähdekoodiin tehdyt muunnokset tuodaan ajonaikaisesti mobiilialustalla näkyvään sovellukseen. Tällöin voidaan nopeasti tarkastella pienten lähdekoodimuunnosten tekemiä eroja sovelluksen puolella. Suurempien muunnosten tekemisen jälkeen kannattaa lähdekoodi kääntää uudestaan, koska on mahdollista, että virhekonsoli ilmoittaa virheestä, vaikka lähdekoodi voikin olla oikein kirjoitettu.

5.1.1 Projektin luonti

Opinnäytetyössä luotu Flutter-mobiilisovellus on kehitetty käyttämällä MacOS-työpöytää ja Android Studio IDE:tä. MacOS:n valittiin työpöydäksi sen tarjoamien iOS-työkalujen ja iOS-tuen takia. Android Studio valittiin IDE:ksi sen käytettävyyden ja edellisen tuntemuksen takia.

Sovelluksen kehitys aloitettiin luomalla tyhjä Flutter-projekti Android Studiolla, mikä onnistuu suoraan Flutterin liitoksen jälkeen Android Studion valikosta. Vastaluodun projektin mukana tulleet lähdekoodit testiapplikaatioon poistettiin omien lähdekoodien alta. Tämän jälkeen lisättiin tarvittavat paketit skanneriominaisuuden luomiseen pubspec.yaml-tiedostoon. Pubspec.yaml -

tiedosto luodaan automaattisesti uuden Flutter-projektin luonnin yhteydessä. (Liite 1 1/4, kuva 10).

Tämä projekti vaati kolme pakettia. SQFLite-paketti tarjoaa SQL-tietokantarajapinnan. Sen avulla voidaan luoda myös yhteydettömiä tietokantoja, mikä ei olisi mahdollista web-sovelluksissa. Paketin mukana tulee myös tarvittavat syntaksit, kuten SQL-kyselykieli, tietokantojen rakentamiseen ja niiden käyttöön.

Toinen paketti Path_provider tarjoaa toiminnot alustan tiedostojärjestelmien ja niiden tiedostopolkujen käyttöön. Tämän avulla voidaan esim. hakea, tallentaa tai poistaa kuvia ja tiedostoja alustalla. Tässä projektissa sitä käytettiin kuvien ja tietokannan tallentamiseen.

Kolmantena pakettina vaaditaan barcode_scan-paketti. Barcode_scan tarjoaa tärkeimmän osan sovelluksen toiminnallisuudesta. Sen avulla voidaan käyttää mobiilialustan kameraa viivakoodien lukemiseen. Kuvassa 11 (Liite 1 1/4, kuva 10), näkyvä cupertino_icons-paketti ei ole välttämätön, mutta se tarjoaa kokoelman natiivialustan kuvakkeiden näköisiä kuvakkeita, jotka saattavat selkeyttää sovelluksen UI:ta.

5.1.2 Tietokannan luonti

Pakettien lisäämisen jälkeen aloitetaan lähdekoodin kirjoittaminen tietokantaluokasta. Projektiin Lisätään database.dart-niminen dart-tiedosto. Avataan database.dart-tiedosto ja luodaan Singleton-luokka, joka hoitaa tietokannan käsittelyn SQL-kyselyiden avulla. Singleton-luokka varmistaa, että on olemassa vain yksi instanssi tietokannasta.

Tietokantatiedoston luonnin jälkeen lisätään projektin pääluokkaan metodit tietokannan luontiin, uuden viivakoodin lisäämiseen, kaikkien tietokannassa olevien viivakoodien hakemiseen ja viivakoodien poistoon. Tietokannan toiminnallisuutta on esitelty liitteistä löytyvässä kuvassa (Liite 1 1/4, Kuva 11).

SQFLite-tietokantapaketin avulla luotuun tietokantaan voidaan ottaa vastaan vain Map-säiliöitä, joten projektiin luodaan lisäksi erillinen tiedosto, johon lisätään Model-luokka, joka hallinnoi tiedon Map-säiliöiksi muuntamisen. Luokan sisälle lisätään funktiot toMap ja fromMap tuottamaan tarvittavat käännökset tietokantaan (Liite 1 2/4 kuva 12). Model-luokka on suora käännös

JSON:sta dart:convert-kirjaston avulla. Pooja Bhaumik on kirjoittanut erinomaisen blogin tästä käännöksestä Flutter-yhteisön sivulla, mistä voi lukea tarkemmin sen toiminnasta [21].

Model-luokan lisäksi projektiin lisättiin vielä tietokannan avustajaluokka. Avustajaluokan tarkoituksena on käsitellä tietokantaa ja mahdollistaa sen käytön muualla projektissa yksinkertaisten funktiokutsujen avulla. Avustajaluokkaan lisätään kuuntelija, joka pitää reaaliaikaista yhteyttä tietokantaan. Kuuntelijan lisäksi lisätään avustajaluokkaan metodit viivakoodien hakuun, lisäämiseen ja poistoon (Liite 1 3/4, kuva 13).

5.1.3 Skanneritoiminnallisuuden luonti

Mobiilisovelluksen skanneritoiminnallisuus luodaan projektin pääluokkaan, missä käsitellään UI:n signaaleja. Luokkaan lisätään uusi painike, jota painamalla ajetaan initPlatformState-funktio. Tämän funktion avulla avataan barcode_scan-paketin tarjoama skannausominaisuus. Kameran avauduttua ja viivakoodin skannauduttua funktio palauttaa string-tekstijonona viivakoodin numerosarjan pääluokkaan luodulle muuttujalle.

Viivakoodin palauduttua pääluokan initPlatformState-funktiossa luodaan vielä uusi viivakoodiobjekti. Tämä viivakoodiobjekti täytetään tietokantaan menevällä tiedoilla, kuten ID:n, skannatun viivakoodin numerosarjan ja skannaushetken kanssa. Tämän objektin tiedot sitten lisätään tietokantaan Barcode-funktion kautta (Liite 1 3/4, kuva 14).

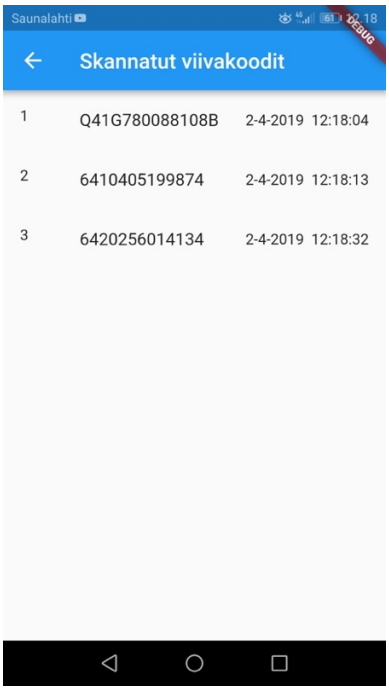
5.1.4 Käyttöliittymän rakentaminen

Mobiilisovelluksen UI on Flutterissa rakennettu käyttämällä widgetejä. Tässä mobiilisovelluksessa on kaksi pääwidgetiä, jotka näyttävät kaksi erillistä näkymää. Ensimmäisessä luotiin Scaffold-widget, joka toimii muiden widgetien pohjana. Scaffoldille asetettiin lapseksi Container, joka pitää sisällään ”aloita skannaus” -painikkeen ja ”skannauksen tulos” -tekstikentän. Näiden lisäksi, scaffoldille lisättiin upotettu painike, jota painamalla avataan listanäkymä, johon on listattu kaikki tietokannasta löytyvät viivakoodit ja ajat.

Sovelluksen listanäkymä on rakennettu eri tyyliin kuin ensimmäinen näkymä. Tässä tapauksessa rakennetaan näkymän käyttöliittymä ”StreamBuilder” -widgetin avulla, johon asetetaan tietokannasta saadut viivakoodit. Tietokannasta otetaan sen hetkinen tilannekatsaus, jonka tietojen kautta se lisää rakennettavaan listaan uusia lista-alkioita tietokannasta löytyvien viivakoodien verran. Eli lista-alkiot näyttävät tietokannasta löytyneiden viivakoodien numerosarjat ja kellonajat (Liite 1 4/4, kuva 15).

Lopuksi sovellusta testattiin mobiilialustalla. Androidilla testaukseen käytettiin Huawei Honor 8 -puhelinta. iOS-testauksessa käytettiin Xcoden tarjoamaa simulaattoria. Alla on kuvankaappaukset sovelluksen ilmeestä Android-mobiilialustalla (kuva 3, kuva 4).

Kuva 3. Flutteria käyttämällä luodun sovelluksen pääsivun UI.



Kuva 4. Sovelluksen listanäkymä.

5.2 Kehittäminen Xamarinilla

Xamarinilla kehittäminen alkoi asentamalla Xamarinin vaatimat työkalut. Ensin asennetaan Android SDK- ja Android API-työkalut. Nämä asentuvat joko erillisesti lataamalla tai Android Studion asennuksen yhteydessä. Xamarin-sovelluksia voi kirjoittaa niin macOS:lla kuin Windowsilla, kunhan käyttökärjestelmää vastaava Visual Studio on asennettuna. MacOS taas vaaditaan, jos halutaan testata sovelluksen toimintaa myös iOS-laitteilla. Vaikkakin Visual Studio versioissa 2017 ja 2019 tuli etänä toimiva iOS-etäsimulaattori Windowsille, etäsimulaattorin käyttöönotto vaatii macOS-koneen samassa WiFi-verkossa Windows-koneen kanssa.

Xamarin toimii Visual Studion lisäosana, joten Visual Studion asentaminen on vaadittua. Visual Studio asennetaan sen omilta internetsivuilta. Ennen asentamista tulee valita omiin tarkoituksiin sopiva Visual Studio -versio. Versioita on kolmea tyyppiä: VS Community, VS Professional ja VS Enterprise. VS community on ilmainen versio, ja muista versioista tulee maksaa lisensointimaksut. Asentajan lataamisen jälkeen avataan asennusohjelma. Asennusohjelman kautta valitaan kehi-

tyksessä tarvitut työkalut Visual Studiolle. Xamarin-projektien luontiin valitaan ”Mobile & Gaming” -välilehdeltä ”Mobile development with .NET” -paketti, joka tarjoaa Xamarin-lisäosat. Haluttujen työkalujen lisäämisen jälkeen painetaan asenna-nappia.

Asennuksen valmistuttua avataan Visual Studio. Visual Studio tarjoaa suuren määrän eri projektimahdollisuuksia, joista Xamarin-projekteja ovat Xamarin.Forms, Xamarin.Android ja Xamarin.iOS. Xamarin.Forms on ainoa projektityyppi, joka tuottaa suoraan alustariippumattomia sovelluksia. Xamarin.Forms-projekti rakentuu kolmen eri kansion kanssa: yhteisen lähdekoodin omaava kansio, Android-kansio ja iOS-kansio. Android- ja iOS-kansioihin kirjoitetaan tarpeen mukaan alustariippuvat lähdekoodit. Yhteinen kansio taas pitää sisällään projektin cs-luokat ja XAML-sivut. Cs-luokkiin kirjoitetaan sovelluksen toiminnallisuudet, ja XAML-tiedostoihin kirjoitetaan UI:n ilme.

Projekti ajetaan asettamalla projektin Android- tai iOS-osio Visual studion Startup-valinnaksi. Tämän jälkeen käyttämällä ”run”-komentoa projekti asentuu valitulle mobiilialustalle. Xamarinista ei vielä löydy tukea ajonaikaisille lähdekoodin muunnoksille, joten muutoksien jälkeen tulee projekti rakentaa uudestaan. Kuitenkin Flutterin tavoin voi mobiilialusta olla simuloitu tai aito. Xamarin tukee Android SDK 19- ja uudempia versioita, kuten myös iOS 8- ja uudempia iOS-versioita. Visual Studioon on lisätty Android-emulaattoreille oma laitehallinta, jossa voi luoda virtuaalisia Android-alustoja. MacOS:ää käytettäessä Xcoden kautta saadaan iOS-simulaattorit.

5.2.1 Skannerisovelluksen kehittäminen

Opinnäytetyöhön liittyen Xamarinilla kehitetty mobiilisovellus on tehty käyttämällä MacOS-työpöytää ja Visual Studio Community -versiota. MacOS valittiin sen suoraan tarjoaman iOS-tuen takia, ja Visual Studio valikoitui IDE:ksi, koska se vaaditaan Xamarin-projektien kehitykseen.

Aluksi luodaan Visual Studiolla uusi tyhjä Xamarin.Forms-projekti. Forms-projektit olivat alustariippumattomia. Projektin mukana tulleet testisovelluksen luovat XAML- ja cs-tiedostot poistettiin. Tämän jälkeen lisättiin tarvittavat paketit projektiin. Pakettien lisäys käy pakettimanagerin kautta helposti (Liite 2 1/3, kuva 16).

Projektiin lisättiin kolme pakettia: ”sqlite-net-pcl” -paketti lisättiin tuottamaan tietokanta, mihin tallennettiin skannatut viivakoodit, ”ZXing.Net.Mobile” ja ”ZXing.Net.Mobile.Forms” -paketit

tuottivat vaaditun skanneriominaisuuden. Paketit asentuvat automaattisesti pakettimanagerista lisäämisen jälkeen. Kuitenkin paketit tulee asentaa erikseen molempien alustoiden omiin kansioihin.

5.2.2 Tietokannan luonti

Xamarin-mobiilisovelluksen kehittäminen alkoi tietokannasta. Tämä lähestymistapa mahdollisti hyvän pohjan luonnin muiden projektin ominaisuuksien lisäämiselle. Tietokannan lisäys aloitettiin luomalla Barcode.cs-luokka, joka on template-luokka viivakoodiobjekteista, kuten barcodeModel-luokka Flutterissa. Viivakoodiobjekteihin on upotettu kaikki tietokannan tarvitsema tieto: viivakoodin ID, viivakoodin numerosarja ja viivakoodin ottoaika.

Barcode.cs-luokan luonnin jälkeen luodaan tietokannalle oma cs-luokka. Tämä luokka luo sqlite-tietokannan mobiilialustan tiedostorakenteeseen ja käsittelee tiedon siirtoa tietokantaan ja sieltä pois. Tähän luokkaan luodaan metodit uuden tietokannan luontiin, viivakoodien hakemiseen, lisäämiseen ja poistamiseen tietokannasta. Aiemmin ladattu sqlite-paketti tarjoaa suoraan syntaksit näille ominaisuuksille, eikä ole tarvetta kirjoittaa SQL-kyselykieltä (liite 2 1/3, kuva 17).

Tietokantaa varten tulee kirjoittaa mobiilialustan tiedostorakenteen poluille pääsevät metodit. Alustojen tiedostorakenteiden erojen takia ei tätä voida kirjoittaa täysin alustariippumattomasti Xamarinissa, joten lisätään alustojen omiin kansioihin luokka polkujen hakuun ja projektin yhteiseen kansioon lisätään vielä erillinen luokka alustalta tulevan polun käsittelyyn. Alustojen ”LocalFileHelper” -luokat pohjautuvat pääkansion luokasta ILocalFIleHelper:stä. Alustojen luokkiin kirjoitetaan vain yksi metodi, joka tarkistaa, onko tiedostopolulla jo tietokantakansio, ja jos kansiota ei löydy, sellainen luodaan. Esimerkki luokasta löytyy liitteistä (Liite 2 2/3 kuva 18).

Tietokannan käyttöön vaaditaan vielä lisäys sovelluksen pääluokkaan. Pääluokkaan lisätään metodi tietokannan lukemiseen. Lisäys pääluokkaan on tarpeellinen, että tietokanta haetaan suoraan sovelluksen käynnistämisen yhteydessä. Esimerkki pääluokkaan lisättävästä käsittelijästä löytyy liitteistä (Liite 2 2/3 kuva 19).

5.2.3 Skanneritoiminnallisuuden luonti

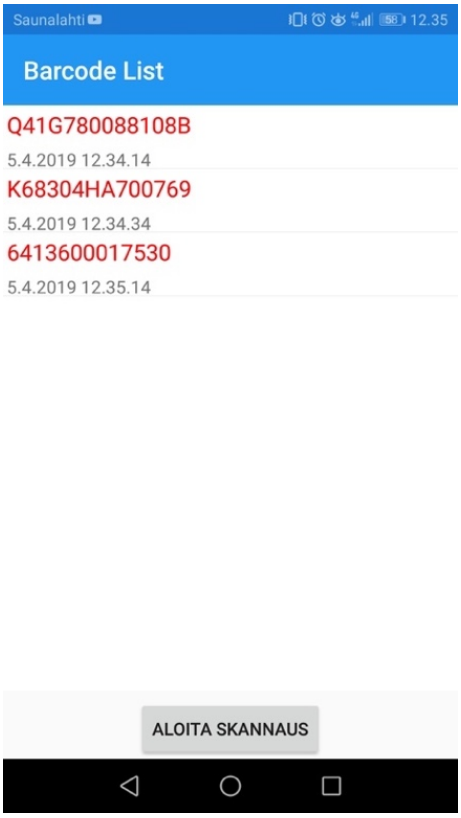
Skanneritoiminnallisuuden tässä projektissa mahdollistaa ”ZXing”-paketti. Paketti toimii luomalla automaattisesti uuden XAML-sivun, joka avaa kameran skannerina ja käyttäjän skannattua viivakoodin se palauttaa viivakoodin. Tässä projektissa tämä toiminnallisuus on asetettu toteutumaan, kun skannausnappia on painettu. Esimerkki skanneripaketin käytöstä löytyy liitteistä (Liite 2 2/3 kuva 20).

5.2.4 Käyttöliittymän rakentaminen

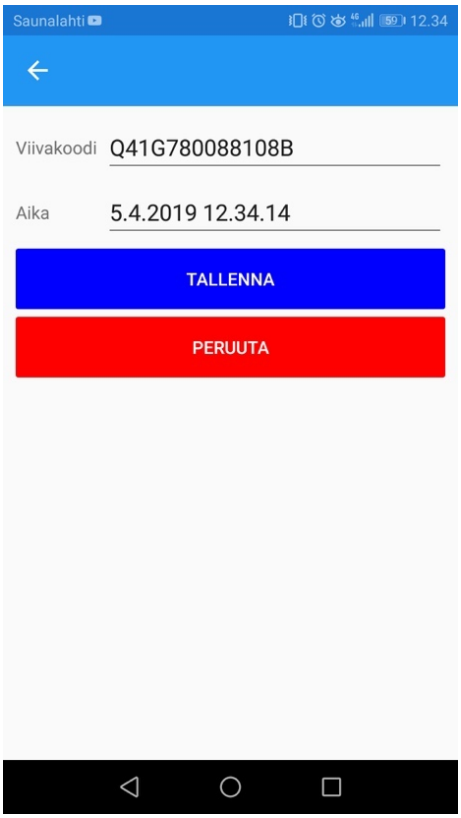
Xamarin-projektit voivat tuottaa UI:n käyttämällä XAML-sivuja tai vain cs-luokkia. XAML-sivut XML-sivujen tavoin kertovat, miltä sovelluksen UI:n tulee näyttää. Tässä projektissa on käytetty kolmea eri XAML-sivua, ensimmäinen sivu mikä avautuu ohjelman käynnistäessä näyttää listan kaikista tietokannasta olevista viivakoodeista. Toinen sivu avautuu, kun skannerilla on skannattu viivakoodi. Tämä sivu näyttää skannatun koodin numeron ja ajanottohetken. Sivulla on myös painikkeet viivakoodin tallentamiseen tai toiminnon peruuttamiseen. Kolmas sivu avautuu, kun ensimmäisen sivun listassa olevaa viivakoodia on painettu. Kolmannella sivulla voi viivakoodia tarkastella ja poistaa sen tietokannasta. XAML-sivut kirjoitetaan XML-sivujen tapaan lokeroimalla tieto tunnisteiden väliin.

XAML rakentuu ContentPage-nimisen tunnisteen sisään. Sen sisälle lisätään tarvittavat layoutit ja UI-esineet. Tässä sovelluksessa on käytetty muutamia eri layouteja ja yksinkertaisia UI-esineitä. Sivujen pohjalla on StackLayout, joka rakentaa UI-esineet lineaariseen järjestykseen. Ensimmäisen sivun StackLayoutin sisälle on lisätty listaelementti, joka täyttää koko StackLayoutin lukuuottamatta sivun alareunassa olevaa painiketta. Listaelementin sisäisille alkioille on lisätty oma GridLayout, joka jakaa listan esineen kahteen osaan: yläosaan, mihin tulostetaan viivakoodi, ja alaosaan, mihin tulostetaan skannausaika. Lisäksi alkioille voi lisätä myös tapahtumia. Tässä sovelluksessa alkiota painamalla avataan uusi sivu, mistä voi tarkistaa tiedot ja poistaa alkion tiedot tietokannasta. Listasivun XAML-lähdekoodi löytyy liitteistä (Liite 2 3/3, kuva 21)

Toisen ja kolmannen sivun UI on käytännössä sama, vain tekstikenttien tekstit ovat erilaiset ja tallenna-nappi muutettu poista napiksi. Näillä sivuilla Stacklayoutiin on suoraan lisätty Grid-Layout. GridLayout pitää sisällään neljä tekstikenttää, jotka on jaoteltu kahteen riviin ja sarakkeeseen. Gridlayoutin lisäksi Staclayoutiin lisätään 2 painiketta. Painikkeet tulevat GridLayoutin alle järjestyksessä. Alla olevista kuvista näkee mobiilisovelluksen ilmeen Android-puhelimella (kuva 5, kuva 6).



Kuva 5. Xamarinilla tehdyn listasivun UI.



Kuva 6. Xamarinilla tehdyn toisen sivun UI. Kolmannen sivun UI on sama lukuun ottamatta TALLENNA-nappia, joka on muutettu POISTA-napiksi.

5.3 Kehittäminen React Nativella

Kehittäminen React Nativella vaati react-native-cli-paketin. Paketti ladataan komentorivin kautta käyttämällä Node.js-pakettimanageria eli npm:ää. Lataamisen jälkeen asetetaan kuntoon IDE. Androidia varten tuli ladata uusin versio JDK-paketista ja testausalustaa vastaavan Android SDK:n. Taas on mahdollista ladata työpöydälle Android Studio ja sitä kautta hallita ja asentaa tarvittavat SDK:t ja emulaattorit. IOS-puolta kehittäessä vaaditaan työpöytä, mistä löytyy Xcode- ja Xcode CLI -työkalut.

React Native -projekteja voi muokata haluamassaan IDE:ssä avaamalla vain projektikansiot ja JavaScript-tiedostot. IDE:ksi voi valita React Nativen syntaksia tukevia IDE:tä, kuten Nuclide, Atom tai Visual Studio Code, kuitenkaan se ei ole pakollista. IDE:n valinnan jälkeen Androidilla tulee vielä asettaa ympäristömuuttujat poluille. IOS:lla ei ole tarvetta muuttaa järjestelmämuuttujia

Seuraavaksi luodaan uusi React Native -projekti tietokoneen komentorivin kautta käyttämällä komentoa ”react-native init <projektin nimi>”. Luomisen jälkeen projekti rakentuu automaattisesti projektikansioon. Rakentumisen jälkeen siirrytään komentorivillä projektikansioon ja ajetaan projekti halutulla alustalla käyttämällä komentoja react-native run-android/run-ios. Ajokomennon jälkeen projekti asentuu mobiilialustalle ja avaa pienen testiapplikaation, mistä voi tarkistaa, että React Native toimii oikein. React Nativella on Flutterin tavoin ”hot reload” -ominaisuus eli pienet muutokset lähdekoodiin asentuvat ajonaikaisesti.

Pakettien lisääminen on hiukan haastavampaa React Nativella. React Nativella tulee paketit erikseen ladata ja linkata projektin kanssa käyttämällä komentorivikomentoja. Suurin osa paketeista tukee linkkausta suoraan, mutta forumeilta löytyy myös paketteja, jotka eivät tue linkkausta ja pitää erikseen liittää projektiin kirjoittamalla riippuvaisuuksia tiedostoihin.

5.3.1 Skannerisovelluksen kehittäminen

Mobiilisovellus kehitettiin käyttämällä macOS-työpöytää, mihin oli asennettuna Xcode ja Android Studio tukemaan emulaattoreita ja simulaattoreita. Androidin testaamiseen käytettiin virtuaalista SDK 26 -versiota. iOS-testaukseen käytettiin virtuaalista 12.2-versiota.

Kehittäminen aloitettiin luomalla uusi React Native -projekti komentorivillä. Seuraavaksi avattiin projektikansio Android Studio IDE:ssä ja poistettiin projektin asennuksen yhteydessä luodut testilähdekoodit.

Seuraavaksi lisättiin paketit käyttämällä npm:ää komentorivin kautta. Projektiin lisättiin kaksi pakettia: react-native-sqlite-storage ja react-native-camera. Erilliselle skanneripaketille ei ollut tarvetta, kun react-native-camera -paketti tuki suoraan skannausta monilla viivakoodityypeillä. Pakettien latauduttua liitetään ne projektirakenteeseen käyttämällä ”react-native link <paketin nimi>” -komentoa komentorivillä. Liittäminen lisää tarvittavat riippuvuudet android.manifest- ja build.gradle-tiedostoihin, kuten myös info.plist-tiedostoon iOS-puolella.

Pakettien lisäämisen jälkeen lisätään vaadittavat luokat projektiin. Android Studiolla tulee JavaScript-luokat lisätä omana tiedostona ja asettaa ”.js”-tiedostopääte. Projektiin kuuluu viisi luokkaa: App, barcodeDelete, barcodeList, mainScreen ja camera. Luokkien lisäämisen jälkeen voidaan projektin kehittämistä jatkaa

5.3.2 Tietokannan kehittäminen

React Native -projektin tietokantaa käsitellään eri tavoin lähdekoodissa kuin Flutter- tai Xamarin-projekteissa. React Nativen sqlite-tietokannan tarjoava paketti oli erittäin yksinkertainen, eikä tarjonnut helpottavia komentoja tai ominaisuuksia. Paketti tarjosi vain käsittelijän tietokantakyselyiden lähettämiseen ja yhteyden luontiin, joten tietokanta tuli kehittää suoraan upottamalla SQL-kyselykieltä raakatekstinä.

Toinen eroavaisuus aiempiin projekteihin oli se, että tässä projektissa ei luotu erillistä luokkaa tietokannan käyttöön. Yhteys tietokantaan luodaan erikseen jokaisessa luokassa, joka tarvitsee sitä. Jakamalla tietokanta käsittelyt sitä tarvitsevien luokkien kesken sekoittaa hiukan lähdekoodia, mutta erillisiä avustajia ja malliluokkia ei tarvita.

Itse tietokanta luotiin mainScreen-luokassa sen ajonaikaisen järjestyksen takia. Näin saadaan tietokanta käyntiin suoraan sovelluksen käynnistyttyä. Tietokantaan luotiin yksi taulu nimeltä table_barcode, johon kuului kolme alkiota: bar_id, bar_code ja bar_time. ID asetettiin pääavaimeksi ja automaattisesti kasvavaksi, kun tietokantaan lisätään alkioita. ID:n avulla voidaan sitten käsitellä alkioita helpommin.

Tietokantaa käsiteltiin vielä kolmessa luokassa. Ensin camera-luokkaan lisättiin SQL-komennot viivakoodin lisäämiseen tietokantaan. Viivakoodi lisätään skannauksen jälkeen. Alkioon asetettiin tiedot skannauksen tuloksen ja skannausajan mukaan (liite 3 1/2 kuva 22).

Seuraavaksi lisättiin BarcodeList-luokkaan komennot kaikkien alkioiden hakuun. Luokassa on lista säiliö, mikä täyttyy uusilla viivakoodi esineillä tietokannassa olevien alkioiden määrän mukaan. Esimerkki lähdekoodia kaikkien alkioiden hakuun (Liite 3 1/2 kuva 23).

Lopuksi lisättiin barcodeDelete-luokkaan funktiot poistoon. Viivakoodin poisto hoidetaan poistamalla viivakoodi tietokannasta annetun ID:n mukaan (liite 3 2/2 kuva 24).

5.3.3 Skanneritoiminnallisuuden luonti

Kaikki sovelluksen skanneritoiminnallisuus hoidetaan camera-luokassa. Luokkaan pitää muistaa lisätä riippuvuutena kamera-paketti, koska React Nativessa lisäosat pitää erikseen lisätä niitä käyttäviin tiedostoihin. Camera-luokan piirtäjä palauttaa kameraobjektin, minkä näkymä täyttää sovelluksen ruudun kameranäkymällä. Kameralle asetetaan tarpeelliset ominaisuudet, kuten käytetään vain alusta takakameraa ja asetetaan salamavalo pois päältä.

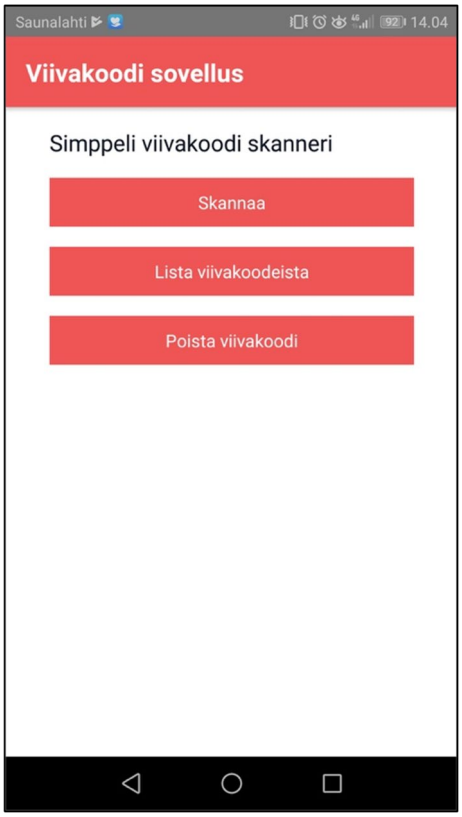
Camera-paketin mukana tulee myös automaattinen käsittelyfunktio sille, kun kamera havaitsee viivakoodin ikkunassa. Yliajetaan käsittelyfunktiolle oma käsittely luokan sisään ja palautetaan viivakoodi string-muodossa viivakoodin havaitsemisen jälkeen. Palautuksen jälkeen asetetaan viivakoodi sen ottoajan kanssa tietokantaan. Tallentamisen jälkeen siirretään sovellusnäkymä takaisin pääsivulle.

5.3.4 Käyttöliittymän rakentaminen

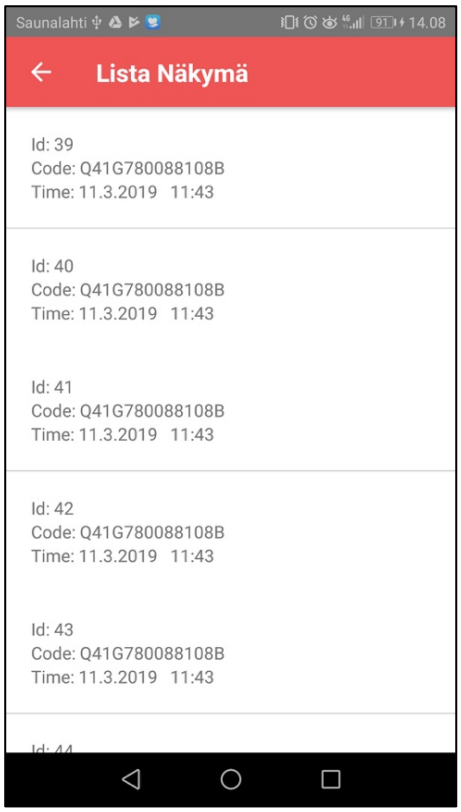
React Nativen UI rakentuu piirtäjän ja määriteltyjen tyylien mukaan. Piirtäjät palauttavat aina view-objektin, joka pitää sisällään kaikki UI:lle sillä hetkellä piirtyvät esineet. Eri esineille voidaan antaa eri tyylit, joko erikseen esineen luonnin yhteydessä, tai kirjoittamalla tyylilistan, mikä on kokoelma tyyleistä. Tässä projektissa loin tyylilistan avulla suurimman osan objekteista, vain pienille tekstikentille kirjoitin tyylit erikseen alustuksen yhteydessä. Esimerkki lähdekoodia sovelluksessa käytettyjen painikkeiden tyylilistasta (liite 3 2/2 kuva 25).

Projektin pääluokkaa lukuun ottamatta jokainen luokka luo oman sivunäkymän. Projektissa on siis neljä sivua. Sivujen takia tulee sovellukseen lisätä navigaattori, joka pitää hallussaan eri sivujen välisiä yhteyksiä ja siirtymisiä. Käytin React Nativen suoraan tarjoamaa stackNavigator-nimistä objektia. StackNavigator-objektiin lisäsin yhden sivun jokaista luokkaa kohti, jonka avulla kutsumalla toista sivua, navigaattorin kautta, voidaan siirtyä suoraan toiselle sivulle.

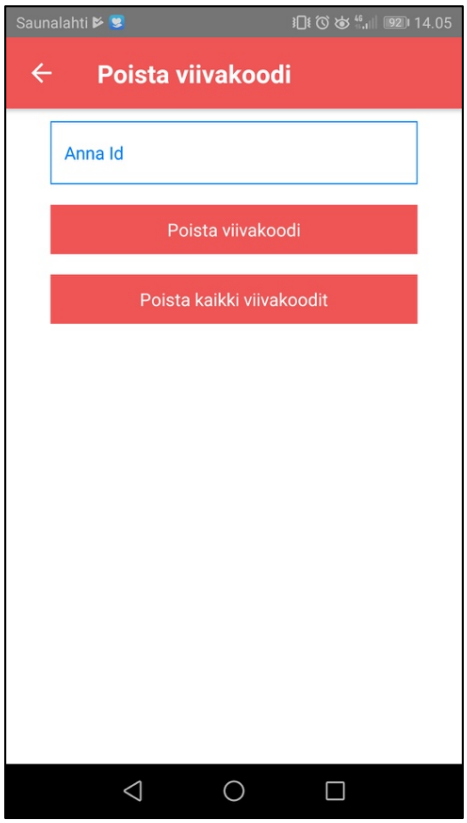
Käyttöliittymän lisäämisen jälkeen React Nativella tehtävä alustariippumaton mobiilisovellus oli valmis. Sovelluksen ulkonäköä Android-mobiilialustalla (kuva 7, kuva 8, kuva 9).



Kuva 7. Sovelluksen pääsivun ilme.



Kuva 8. Listaus tietokannassa olevista viivakoodeista.



Kuva 9. Viivakoodin poistaminen annetun ID:n mukaan.

6 Pohdintaa

Kuten aiemmin todettiin, jokaisella sovelluskehyksellä pystyi tuottamaan vaatimuksien mukaisen alustariippumattoman mobiilisovelluksen. Kehittämisprosessi kuitenkin vaihteli sovelluskehyksestä riippuen. Lopullisen vertailun tarkoituksena olikin vertailla kehittämisprosessien vaiheita ja missä toiset sovelluskehykset onnistuivat toisia paremmin.

Ajallisesti sovelluskehyksillä sovelluksien kehittämiseen meni suunnilleen yhtä kauan. Kunkin sovelluksen kehittämiseen käytettiin noin 16 tuntia. Kehittämiseen kuului sovelluskehyksien ja tarvittavien työkalujen asentaminen, lähdekoodin kirjoittaminen, sovellusten testaus useammalla alustalla ja virheiden korjaaminen.

Kehittäjälle kaikki kehitystehtävässä olevien sovelluskehysten käyttämät lähdekoodikielet olivat tuntemattomia, niin siltä osin vertailua voidaan pitää tasa-arvoisena. Valitut sovelluskehykset myös liitettiin jo tutuiksi tulleisiin kehitysympäristöihin, eikä aikaa pitänyt käyttää ympäristöjen oppimiseen.

Sovelluskehyksen asentaminen ja pääseminen kehitysvaiheeseen oli huomattavasti helpointa Xamarinilla, vaikka kaikki sovelluskehykset tarjosivat erinomaiset oppaat kehityksen alkuun pääsyssä. Xamarin tuli vain ladata normaalin Visual Studio -asennusohjelman mukana, kun muissa sovelluskehyksissä vaadittiin IDE:n asentaminen erikseen.

Android-kehitykseen kaikki sovelluskehykset vaativat erinäiset android työkalut, joten tällä osa-alueella ei noussut yksikään sovelluskehys toisen ohi. Työkalut asentuivat suoraan internetistä ladattavan asennusohjelman kautta, eikä siinä ollut suurempia ongelmia millään sovelluskehyksellä. iOS-sovelluksien kirjoittamiseen ei vaadittu erikseen mitään uutta. Testaamiseen ja sovellusten ajamiseen tulee olla macOS, missä on uusin versio Xcodesta.

Pakettien ja lisäosien lisääminen projekteihin oli helppoa Flutterissa ja Xamarinissa niiden omien pakettimanagereiden tai järjestelmien avulla. React Nativella tuli paketit ladata ja asentaa komentorivin kautta. Paketit tuli myös linkata erikseen projektiin. Automaattinen linkkaaja ei toiminut SQL-paketin kanssa. Samaa ongelmaa oli raportoitu internetfoorumeilla muidenkin pakettien yhteydessä. SQL-paketti piti siis linkata manuaalisesti lisäämällä paketti erilliseen kansioon ja

asettamalla riippuvuudet android.manifest- ja info.plist-tiedostoihin. Flutterin ja Xamarinin tarjoamat paketit antoivat yksinkertaiset syntaksit tietokannan tekoon ja käsittelyyn. React Nativella paketti ei tarjonnut suurempia hyötyjä vaan vain rajapinnan käsittelyä varten, joten sillä piti kirjoittaa tietokantamuutokset käyttämällä SQL-kyselyitä.

Kaikilla sovelluskehyksillä rakennetut projektit pystyivät suoraan asentumaan mobiilialustalle, kunhan mobiilialusta oli liitettynä tai emuloituna. Flutter ja React Native tarjosivat myös mahdollisuuden suorittaa ajonaikaisia muutoksia lähdekoodissa. Ennen kehitysvaihetta tämä ominaisuus tuntui turhalta hienostelulta, mutta kehityksen aikana nopeutti esimerkiksi pienten UI-muutoksien tekemistä.

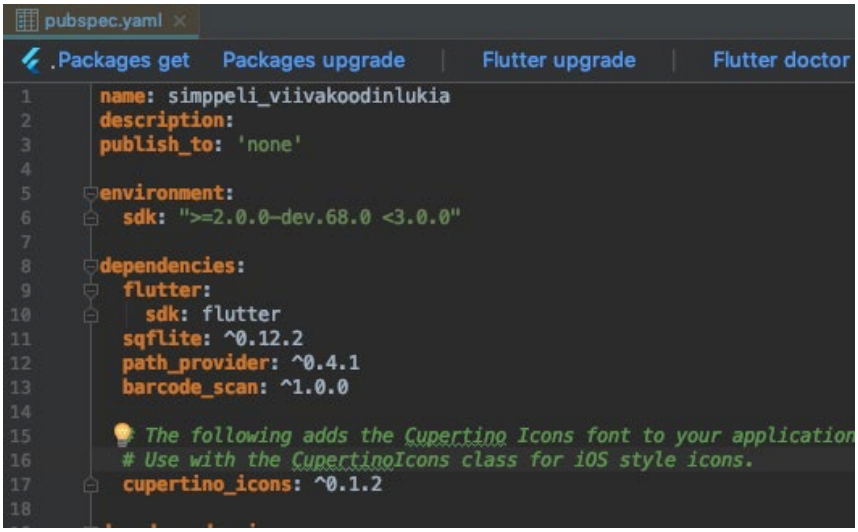
7 Yhteenveto

Työn tavoitteena oli tutkia ja vertailla alustariippumattomaan kehitykseen kelpaavien sovelluskehysten tarjontaa ja toimintaa. Työn alussa huomattiin, että puhdas tutkimusmainen malli ei olisi sopiva tässä aiheessa. Niinpä työ jakautui kahteen pääosaan. Ensimmäisessä enemmän tutkielmallisessa osassa vertaillaan ja tutkitaan sovelluskehysten toimintaa ja ominaisuustarjontaa. Toiseen enemmän kehitykselliseen osaan, sitten valittiin ensimmäisen osan tulosten perusteella kolme sopivinta sovelluskehystä. Valituilla sovelluskehyksillä kehitettiin yksinkertainen alustariippumaton sovellus. Lopulta näistä kolmesta valittiin yrityksen tarpeisiin sopivin sovelluskehys.

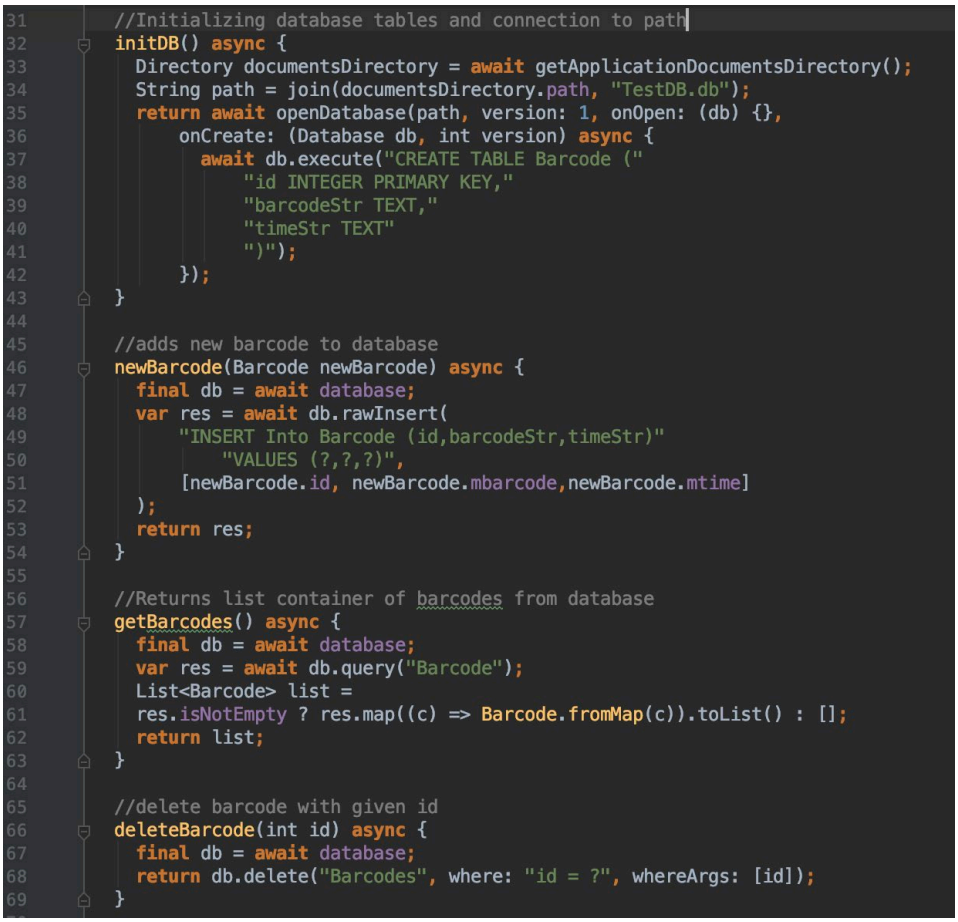
Lähteet

(1) Techopedia. Cross-platform Development. https://www.techopedia.com/definition/30026/cross-platform-development (haettu 30. 1 2019)
(2) https://en.wikipedia.org/wiki/Cross-platform_software (haettu 19.2.2019)
(3) Raasch Jon. JavaScript Programming: Pushing the Limits, John Wiley & Sons, Incorporated, 2013. ProQuest Ebook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=1315441. (haettu 19.2.2019)
(4) Abrosimova Kate. Native VS Cross-Platform App Development. Why You Shouldn't Work With Cross-Platform, 15. 9. 2014. https://medium.com/yalantis-mobile/native-vs-cross-platform-app-development-why-you-shouldnt-work-with-cross-platform-2e85e0619226 (haettu 3.4.2019)
(5) Manchanda A. Native Vs Hybrid Apps Development – Finding Clarity in Confusion, 20. 11 2018 https://www.netsolutions.com/insights/native-vs-hybrid-apps-from-confusion-to-clarity (haettu 1.2.2019)
(6) Flutter FAQ. https://flutter.io/docs/resources/faq (haettu 30. 1 2019)
(7) Dart Packages. https://pub.dartlang.org/ (haettu 30.1.2019)
(8) A Tour of the Dart Language. https://www.dartlang.org/guides/language/language-tour (haettu 27.2.2019)
(9) The Qt Company Ltd. Qt Creator Manual. http://doc.qt.io/qtcreator/ (haettu 30. 1 2019)
(10) Eisenmann Bonnie. Learning React Native. https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html (haettu 30. 1 2019)
(11) Facebook. A frameword for building native apps with React. https://github.com/facebook/react-native (haettu 14.2.2019)
(12) Chrzanowska Natalia. React Native Pros and Cons – Facebook's Framework in 2019 (update). https://www.netguru.com/blog/react-native-pros-and-cons (haettu 14.2.2019)
(13) Wikipedia. Xamarin. https://en.wikipedia.org/wiki/Xamarin (haettu 30.1.2019)
(14) Burns A, Britch D, Boggan P, Dunn C, Umbaugh B. Get Started With Xamarin. https://docs.microsoft.com/fi-fi/xamarin/cross-platform/get-started/ (haettu 14.2.2019)
(15) Adobe Systems Inc. Frequently Asked Questions. https://phonegap.com/about/faq/ (haettu 30. 1 2019)
(16) Onsen UI. Getting Started. https://onsen.io/v2/guide/ (haettu 30. 1 2019)
(17) Onsen UI. Introduction to Bindings. https://onsen.io/v2/guide/frameworks.html (haettu 15.2.2019)
(18) Flutter. Fast Development. https://flutter.dev (haettu 3.2.2019)
(19) Devlin Ian, Building an iOS signing key for PhoneGap in Windows. http://www.ian-devlin.com/blog/2012/11/phonegap/building-an-ios-signing-key-for-phonegap-in-windows/ (haettu 15.3.2019)
(20) Developing with Xamarin for Linux. https://stackoverflow.com/questions/36492583/developing-with-xamarin-for-linux (haettu 15.3.2019)
(21) Bhaumik Pooja. Parsing complex JSON in Flutter, 8. 6. 2019 https://medium.com/flutter-community/parsing-complex-json-in-flutter-747c46655f51 (haettu 1.4.2019)

Kuvia Flutter-lähdekoodeista.



Kuva 10. Otos pubspec.yaml-tiedostosta mihin lisätään projektin tarvitsemat paketit.



Kuva 11. Projektin tietokantaluokan toimintaa.

databaseBloc.dart barcodeModel.dart

```dart
import 'dart:convert';

Barcode barcodeFromJson(String str) {
  final jsonData = json.decode(str);
  return Barcode.fromMap(jsonData);
}

String barcodeToJson(Barcode data) {
  final dyn = data.toMap();
  return json.encode(dyn);
}

class Barcode {
  int id;
  String mbarcode;
  String mtime;

  Barcode({
    this.id,
    this.mbarcode,
    this.mtime
  });

  factory Barcode.fromMap(Map<String, dynamic> json) => new Barcode(
    id: json["id"],
    mbarcode: json["barcodeStr"],
    mtime: json["timeStr"],
  );

  Map<String, dynamic> toMap() => {
    "id": id,
    "barcodeStr": mbarcode,
    "timeStr": mtime,
  };
}
```

Kuva 12. Otos Map-tiedostotyypin muuttavasta luokasta.

databaseBloc.dart barcodeModel.dart

```dart
import 'dart:async';
import 'package:flutter_app_213/barcodeModel.dart';
import 'package:flutter_app_213/database.dart';

//creating barcode helper class
class BarcodesBloc {
  final _barcodeController = StreamController<List<Barcode>>.broadcast();

  // returns stream to be used as widgets streambuilder body
  get barcodes => _barcodeController.stream;

  //closes controller stream
  dispose() {
    _barcodeController.close();
  }

  //implementing controller stream with database
  getBarcodes() async {
    _barcodeController.sink.add(await DBProvider.db.getBarcodes());
  }

  //Helper constructor gets barcodes from database
  BarcodesBloc() {
    getBarcodes();
  }

  //delete barcode with given id from database
  delete(int id) {
    DBProvider.db.deleteBarcode(id);
    getBarcodes();
  }

  //Addign new barcode to database
  add(Barcode barcode) {
    DBProvider.db.newBarcode(barcode);
    getBarcodes();
  }
}
```

Kuva 13. Tietokanta-avustajaluokka.

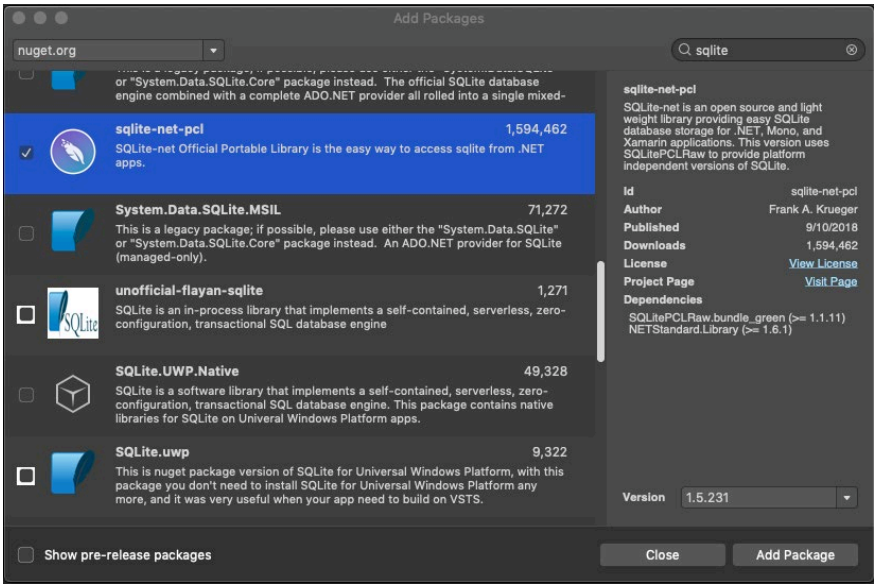
```dart
// Scanner method async
Future<void> initPlatformState() async{
  String barcodeScanRes;
  //Calling scanner.dart files scan method. #ff6666 is only color code for scan line
  try {
    barcodeScanRes = await FlutterBarcodeScanner.scanBarcode("#ff6666");
  } on PlatformException {
    barcodeScanRes = 'Failed to get platform version.';
  }

  //Check to see if device has camera or is allowed
  if (!mounted) return;

  //setting the got barcode into string
  setState(() {
    _scanBarcode = barcodeScanRes;
  });
  //Creating new barcodeModel and sending it to helper
  Barcode add = new Barcode(mbarcode: _scanBarcode, mtime: _currentTime);
  bloc.add(add);
}
```

Kuva 14. Skannausnappia painamalla avataan kamera ja odotetaan viivakoodin löytymistä.

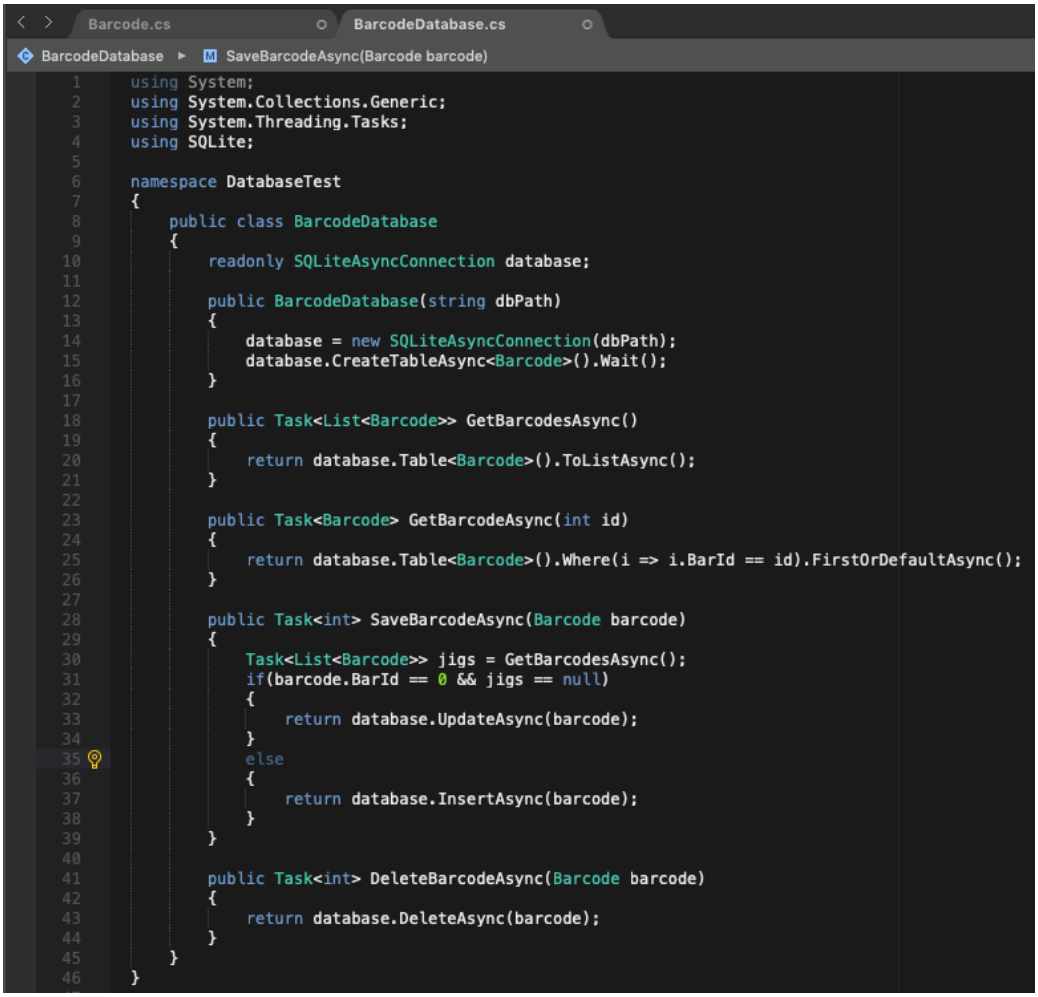
```
Widget build(BuildContext context) {
  // Use the Todo to create our UI
  return Scaffold(
    appBar: AppBar(title: Text("Skannatut viivakoodit")),
    body: StreamBuilder<List<Barcode>>(
      stream: bloc.barcodes,
      builder: (BuildContext context, AsyncSnapshot<List<Barcode>> snapshot){
        if(snapshot.hasData){
          return ListView.builder(
            itemCount: snapshot.data.length,
            itemBuilder: (BuildContext context, int index){
              Barcode item = snapshot.data[index];
              return new ListTile(
                title: Text(item.mbarcode),
                leading: Text(item.id.toString()),
                trailing: Text(item.mtime),
                onLongPress: (){
                  bloc.delete(item.id);
                },
              ); // ListTile
            },
          ); // ListView.builder
        }else{
          return Center(child: CircularProgressIndicator());
        }
      },
    ), // StreamBuilder
  ); // Scaffold
}
```

Kuva 15. Esimerkki-widgetti. Tämä widgetti rakentaa sovelluksen lista sivun.

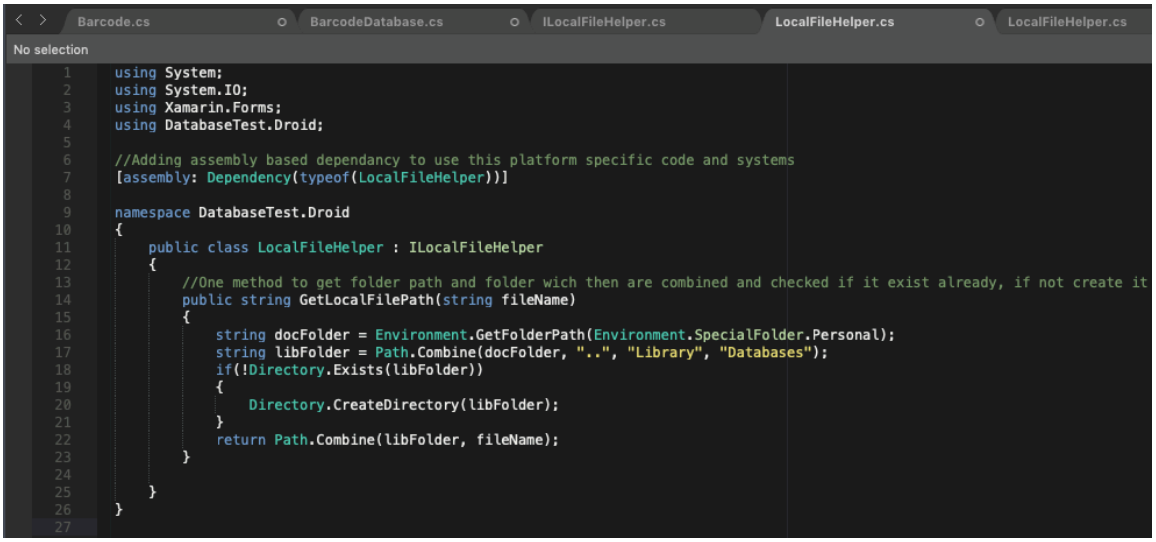
Kuvia Xamarin-lähdekoodeista.



Kuva 16. Pakettilistanäkymä, missä ollaan lataamassa sqlite-pakettia.



Kuva 17. Xamarin-sovelluksen tietokantaluokka.




```
using System;
using System.IO;
using Xamarin.Forms;
using DatabaseTest.Droid;

//Adding assembly based dependancy to use this platform specific code and systems
[assembly: Dependency(typeof(LocalFileHelper))]

namespace DatabaseTest.Droid
{
    public class LocalFileHelper : ILocalFileHelper
    {
        //One method to get folder path and folder wich then are combined and checked if it exist already, if not create it
        public string GetLocalFilePath(string fileName)
        {
            string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");
            if(!Directory.Exists(libFolder))
            {
                Directory.CreateDirectory(libFolder);
            }
            return Path.Combine(libFolder, fileName);
        }

    }
}
```

Kuva 18. Alustakansioon kirjoitettava polun hakija. IOS-puolella muutetaan ”DatabaseTest.droid” -riippuvuus ”DatabaseTest.iOS” -riippuvuudeksi.

```
    public static BarcodeDatabase Database
    {
        get
        {
            if(database == null)
            {
                database = new BarcodeDatabase(DependencyService.Get <ILocalFileHelper>().GetLocalFilePath("test.db3"));
            }
            return database;
        }
```

Kuva 19. Pääluokkaan lisätty polun hakija, joka saa oikean polun mobiilialustan luokasta.

```
        async void Handle_Clicked(object sender, System.EventArgs e)
        {
            //Additional options for scanner page, setting autorotate true to allow landscape mode, forcing scanner to use back camera,
            //Also one can limit what barcode formats scanner can scan, this might help detection speed of the scanner
            var options = new MobileBarcodeScanningOptions
            {
                AutoRotate = true,
                UseFrontCameraIfAvailable = false,
                TryHarder = true,
            };

            //Creating new XAML page with options
            var scan = new ZXingScannerPage(options) { DefaultOverlayTopText = "Tasaa viivakoodi punaisen viivan mukaan"};

            //Creating new Barcode item that will be filled with scanned barcode and time of scan
            var barcodeItem = new Barcode();

            //Open the scanner page
            await Navigation.PushAsync(scan);

            //Strings to hold result and current time
            string scanResult = "";
            string dateTime = DateTime.Now.ToString();

            //Checker that triggers when scanner finds barcode
            scan.OnScanResult += (result) =>
            {
                scanResult = result.Text;
                barcodeItem.BarCode = scanResult;
                barcodeItem.BarTime = dateTime;

                //Set app to stop scanning
                scan.IsScanning = false;
                Device.BeginInvokeOnMainThread(async () =>
                {
                    //Close scanner page
                    await Navigation.PopToRootAsync();
                    //Open page where user can inspect the scanned barcode and decide wether to save it or not
                    await Navigation.PushAsync(new BarcodePage() { BindingContext = barcodeItem as Barcode });
                });
            };
        }
    }
}
```

Kuva 20. Skannausnappia painamalla avataan skanneri ja tallennetaan viivakoodi tietokantaan.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="DatabaseTest.BarcodeListPage">
    <ContentPage.Content>
        <StackLayout>
            <ListView x:Name="BarcodeListView" HorizontalOptions="FillAndExpand" BackgroundColor="White" VerticalOptions="FillAndExpand" RowHeight="50" ItemSelected="Barcode_ItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <Grid RowSpacing="+5" Padding="+3">
                                <Grid.RowDefinitions>
                                   <RowDefinition Height="*" />
                                </Grid.RowDefinitions>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="*" />
                                    <ColumnDefinition Width="*" />
                                </Grid.ColumnDefinitions>
                                <StackLayout Grid.Column="0">
                                    <Label Text="{Binding BarCode}" TextColor="Red" FontSize="Medium"/>
                                    <Label Text="{Binding BarTime}" />
                                </StackLayout>
                            </Grid>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
            <Button Text="Aloita Skannaus" VerticalOptions="End" HorizontalOptions="Center" Clicked="Handle_Clicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Kuva 21. Esimerkkiviivakoodi listan XAML-tiedostosta.

Kuvia React Native-lähdekoodeista

```
onBarCodeRead(scanResult) {
      //forcing camera to shut down after scanning
      this.componentWillUnmount();

      var that = this;
      //parsing current date/time from system time
      var days = new Date().getDate();
      var month = new Date().getMonth();
      var years = new Date().getFullYear();
      var hours = new Date().getHours();
      var minutes = new Date().getMinutes();
      currentDateTime: '';
      currentDateTime = days+'.'+month+'.'+years+'   '+hours+':'+minutes;

    //Sending query to database to insert scanning result with time of scan
    db.transaction(function(tx) {
                tx.executeSql(
                  'INSERT INTO table_barcode (bar_code, bar_time) VALUES (?,?)',
                  [scanResult.data, currentDateTime],
                  (tx, results) => {
                    console.log('Results', results.rowsAffected);
                    if (results.rowsAffected > 0) {
                      Alert.alert(
                        'Huomio!',
                        'Viivakoodi tallennettu',
                        [
                          {
                            text: 'Ok',
                            onPress: () =>
                              //sending app back to first screen
                              that.props.navigation.navigate('Home'),
                          },
                        ],
                        { cancelable: false }
                      );
                    } else {
                      alert('Tallennus epäonnistui');
                    }
                  }
                );
              });
  }
```

Kuva 22. Viivakoodin skannauksen yhteydessä tapahtuvat funktiot.

```
db.transaction(tx => {
  tx.executeSql('SELECT * FROM table_barcode', [], (tx, results) => {
    var temp = [];
    for (let i = 0; i < results.rows.length; ++i) {
      temp.push(results.rows.item(i));
    }
    this.setState({
      //Filling flatlistItems container with data from db
      FlatListItems: temp,
    });
  });
});
```

Kuva 23. Haetaan kaikki tietokannasta löytyvät alkiot.

```
db.transaction(tx => {
  tx.executeSql(
    'DELETE FROM  table_barcode where bar_id=?',
    [input_bar_id],
    (tx, results) => {
      console.log('Results', results.rowsAffected);
      if (results.rowsAffected > 0) {
        Alert.alert(
          'Huomio',
          'Viivakoodi poistettu',
          [
            {
              text: 'Ok',
              onPress: () => that.props.navigation.navigate('MainScreen'),
            },
          ],
          { cancelable: false }
        );
      } else {
        alert('ID:tä ei löytynyt, onko ID varmasti oikein?');
      }
    }
  );
});
```

Kuva 24. Viivakoodin poisto tietokannasta annetun ID:n avulla.

```
//Stylesheet that is given to Mybutton prop
const styles = StyleSheet.create({
  button: {
    alignItems: 'center',
    backgroundColor: '#f05555',
    color: '#ffffff',
    padding: 10,
    marginTop: 16,
    marginLeft: 35,
    marginRight: 35,
  },
  text: {
    color: '#ffffff',
  },
});

export default Mybutton;
```

Kuva 25. Esimerkki painikkeelle tehtävästä tyyliluettelosta.