Tampereen ammattikorkeakoulu

Monialustaisen mobiilisovelluksen kehitys Xamarin.Forms -alustalla

Antti Laine

OPINNÄYTETYÖ
Toukokuu 2021

Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

LAINE, ANTTI:
Monialustaisen mobiilisovelluksen kehitys Xamarin.Forms-alustalla

Opinnäytetyö 41 sivua, joista liitteitä 4 sivua
Toukokuu 2021

Opinnäytetyön tarkoituksena oli kehittää monialustainen mobiilisovellus Android- ja iOS -käyttöjärjestelmille hyödyntämällä Microsoftin Xamarin.Forms-alustaa täyttämään toimeksiantajan asettamat määrittelyt. Xamarin.Forms:in käyttö mahdollisti sovelluksen kehityksen yhteisestä koodikannasta molemmille käyttöjärjestelmille, säästäen suuresti aikaa ja resursseja verrattuna käyttöjärjestelmien omilla työkaluilla tehtävään kehitystyöhön. Sovelluksen suunnitellussa kiinnitettiin erityistä huomiota käyttöliittymän suunnitteluun ja käytettävyyden takaamiseen.

Kaikkia sovellukselle asetettuja vaatimuksia ei pystytty toteuttamaan valtakunnallisen koronatilanteen vuoksi. Tarkoituksena oli suorittaa sovelluksen laadunvarmistamista valikoiduissa koiranäyttelyissä, mutta näyttelyt jouduttiin perumaan tai siirtämään myöhempään ajankohtaan. Lisäksi sovelluksen julkaisua päätettiin siirtää, kunnes näyttelyitä voidaan järjestää turvallisesti. Tästä huolimatta sovelluksen kehitys vietiin loppuun asti. Lopputuloksena oli julkaisukelpoinen Xamarin.Forms-pohjainen sovellus Android- ja iOS -laitteille.

Asiasanat: mobiilisovelluskehitys, xamarin, xamarin.forms, monialustaisuus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

LAINE, ANTTI:
The Development of a Cross-platform Mobile Application with Xamarin.Forms Framework

Bachelor's thesis 41 pages, appendices 4 pages
May 2021

The purpose of this thesis was to develop a cross-platform mobile application using Microsoft Xamarin.Forms framework in accordance with the requirements specified by the client. With Xamarin.Forms it was possible to develop the application from a single codebase targeting both operating systems, saving a considerable amount of development time and resources compared to native development technologies. Additional attention was paid to user experience and accessibility during the planning and design phases of development.

The final testing and release phases where heavily impacted by the national coronavirus situation. All planned confirmation shows were cancelled during the testing period. Thus, it was decided to postpone the release date until confirmation shows could return to a more stable schedule. Despite the setbacks the development of the application was carried to the end. The result of this work was a release of a ready Xamarin.Forms based application for both Android and iOS devices.

Key words: mobile app development, Xamarin, Xamarin.forms, Cross-platform

SISÄLLYS

ERITYISSANASTO

Leiska	Käyttöliittymän ulkoasua tarkasti esittävä staattinen kuva. Johdettu englannin kielen sanasta layout.
UWP	Universal Windows Platform, Microsoftin luoma alusta Windows pohjaisien sovelluksien kehitykseen.
C#	Microsoftin kehittämä oliopohjainen ohjelmointikieli.
iOS	Applen iPhone-puhelimien käyttöjärjestelmä.
Android	Googlen kehittämä Android-laitteiden käyttöjärjestelmä.
Debug	Virheenjäljitys. Ongelman aiheuttaneen virheen paikantaminen.
.NET	Microsoftin kehittämä universaali ohjelmistokehys.
SSH	Salattuun tietoliikenteeseen tarkoitettu protokolla.
JSON	Yksinkertainen tiedostomuoto tiedonvälitykseen.
Hot Reload	Sovelluksen käyttöliittymän tiedostojen muokkausten visualisointi ajon aikana ilman uudelleenkäynnistystä.
XAML	XML-pohjainen merkintäkieli käyttöliittymien rakenteiden määrittelemiseen.
Natiivisovellus	Käyttöjärjestelmän omilla työkaluilla, kielellä ja kehitystavoilla toteutettu sovellus.
Obfuskaatio	Ohjelmakoodin muuttaminen vaikealukuiseksi koodin toiminnan suojaamista varten.

1 JOHDANTO

Työn toimeksiantaja, Showlink Oy, on suomalainen koiranäyttelyiden palvelujen tarjoaja. Espoolainen Showlink Oy toimii vuosittain yli 160:n koiranäyttelyn palveluntuottajana. Osana sähköistenpalveluiden laajentamista päätettiin kehittää näyttelyiden pääsylippujen lukemiseen tarkoitettu mobiilisovellus. Toistaiseksi liput on luettu tavallisilla viivakoodien lukulaitteilla kannettavientietokoneiden avulla. Mobiilisovellus mahdollistaa lippujen lukemisen näyttelyjärjestäjien omilla mobiililaitteilla ilman erityisiä vaatimuksia.

Sovelluksen kehitystyö annettiin Tamperelaiselle Prettybit Projects Oy:lle, joka on tuottanut aikaisemmin sähköisiä palveluja Showlink Oy:lle. Osana aikaisempia kehitysprojekteja on myös ollut muita mobiilisovelluksia. Kehitystiimi koostui yhdestä mobiilisovelluskehittäjästä, joka on tämän opinnäytetyön tekijä, sekä yhdestä palvelin kehityksen asiantuntijasta.

Tämä opinnäytetyö kattaa sovelluksen kehityksen vaiheet vaatimusten keräämisestä julkaisun suunnitteluun asti. Erityinen painotus on sovelluksen kehityksen aikaisella laadunvarmistamisella, siihen käytetyillä työkaluilla, sekä käyttöliittymän suunnittelun vaiheilla.

Työ on jaettu luvuittain. Luvussa kaksi taustoitetaan sovelluksen tarkoitusta ja vaatimuksia, joiden pohjalta muodostetaan vaatimukset teknologioiden osalta. Lisäksi toinen luku kartoittaa valittujen teknologioiden yhteistoimintaa. Kolmannessa luvussa käydään läpi sovelluksen kehitykseen liittyvät asiat ja luvussa kaksi muodostettujen vaatimuksen toteutus. Kolmas luku sisältää myös käyttöliittymän kehityksen sovelluksen testaamisen periaatteet. Viimeinen, neljäs, luku keskittyy sovelluksen julkaisun ja jatkokehityksen haasteisiin.

2 PROJEKTIN TAUSTA

2.1 Sovelluksen tarkoitus

Sovelluksen on tarkoitus toimia osana Showlink Oy:n sähköisiä näyttelypalveluita. Sovellus mahdollistaa koiranäyttelyiden pääsylippujen lukemisen tuetuilla Android ja iOS laitteilla. Lippujen oikeellisuus todetaan palvelimella, jonka jälkeen lipun tila näytetään sovelluksessa.

2.2 Sovelluksen vaatimukset

Sovellukselle asetetut vaatimukset muodostuvat sen ydinominaisuuksista, jotka tarvitaan toteuttamaan sovelluksen tarkoituksen mukainen toiminnallisuus.

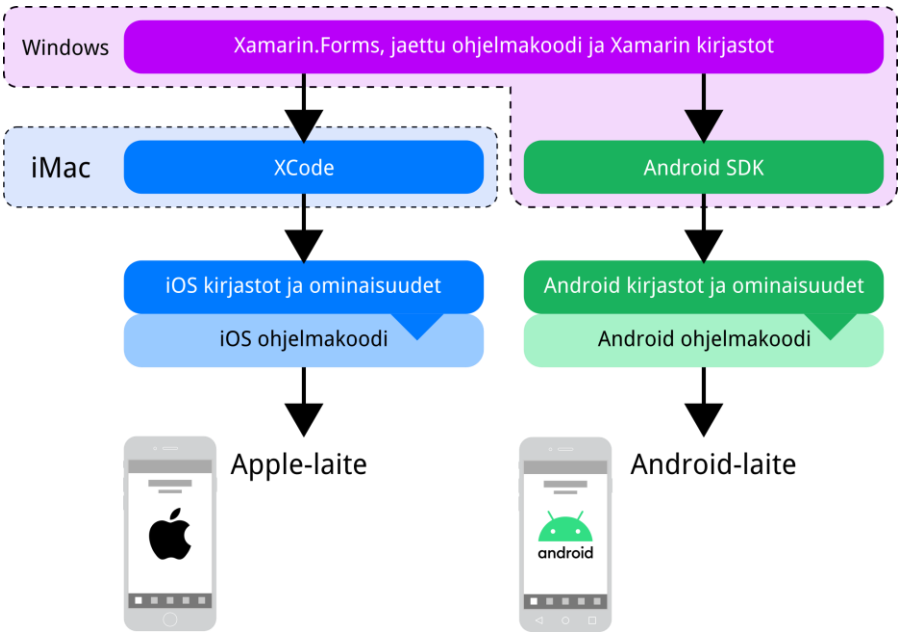
Sovellukselle asetettiin seuraavat tavoitteet:

- Sovellukseen kirjautuminen
- Pääsylipun QR/viivakoodin lukeminen
- Esittää lipun tilanne lukemisen jälkeen
- Esittää näyttelystä tietoja
- Tapa kirjoittaa lipun koodi käsin, jos sen lukeminen ei onnistu
- Virheiden ilmaiseminen
- Automaattinen siirtyminen luku ja tietotilojen välillä
- Tuki käyttöliittymän kielen vaihtamiselle
- Asetuksien tallentaminen
- Tukea mahdollisimman monta laitetta
- Väritys Showlink Oy:n brändin mukaisesti

Valinnaisena lisäominaisuutena käyttöliittymään lisättiin tumman teeman tuki.

2.3 Teknologiapino

Sovelluksen pohjaksi valittiin Xamarin.Forms, sillä projektin tekijällä on aikaisempaa kokemusta Xamarin.Forms-sovelluksien kehityksestä ja kehitysympäristö on jo olemassa nopeuttaen kehitysaikaa. Xamarin.Forms tuo mukanaan omia vaatimuksiaan käytettyjen teknologioiden suhteen (Kuva 1). Androidille kehitys on mahdollista toteuttaa kokonaan Windows-tietokoneella, mutta iOS kehitystä varten ketjussa on oltava jokin Applen tietokone. Tässä projektissa iOS vaatimuksen täyttää vuoden 2016 iMac, jolla on asennettuna XCode 12 ja macOS 11 ”Big Sur”. Android SDK:sta käytettiin uusinta API 30 (v11) versiota.



Kuva 1 Xamarin.Forms sovelluksen teknologiapino

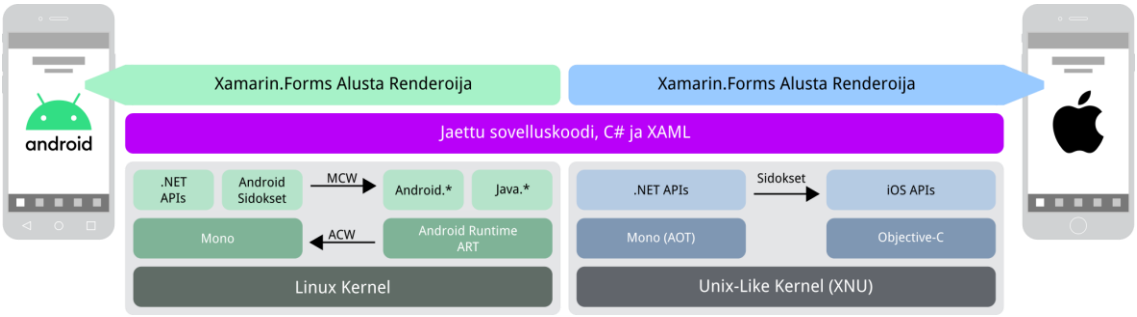
Sovelluksen halutaan toimivan mahdollisimman monella laitteella. Tämä tarkoittaa myös vanhojen käyttöjärjestelmä versioiden tukemista. Androidin minimiversiovaatimukseksi valikoitui Android 5.0 (API 21), joka kattaa 94,1 % kaikista käytetyistä Android-laitteista maailmanlaajuisesti (Android Studio). Minimiversio takaa viivakoodien lukemiseen käytetyn kirjaston toimivuuden. Applen iOS-käyttöjärjestelmän minimiversio on 11, joka kattaa myös Applen tuen ulkopuolella olevia laitteita, kuten iPhone 5S ja iPhone 6.

2.3.1 Xamarin.Forms

Xamarin.Forms avoimen lähdekoodin käyttöliittymän kehitykseen keskittyvä ohjelmistokehys, joka mahdollistaa monialustaisien sovelluksien kehityksen .NET ohjelmistokomponenttikirjaston avulla (Microsoft 2020a). Xamarin.Forms:in uusin versio tukee sovelluksien kehittämistä monelle käyttöympäristölle:

- Android, Android Wear ja Android Auto
- iOS, watchOS, tvOS ja macOS
- UWP (Windows 10 ja Xbox)

Mobiilisovelluksien kehityksessä Xamarin.Forms integroituu saumattomasti Xamarin.Android ja Xamarin.iOS implementaatioiden kanssa mahdollistaen natiivin koodin kääntämisen yhteisestä C# pohjaisesta koodikannasta. Xamarin.Formsilla kehitetyt mobiilisovellukset ovat käyttöympäristöilleen natiiveja sovelluksia (Kuva 2), jotka oletuksena käyttävät ympäristönsä omia käyttöliittymäratkaisuja. Ulkonäöllisesti Xamarin.Forms sovellukset ovat samanlaisia kuin käyttöympäristön omilla työkaluilla kehitetyt mobiilisovellukset.

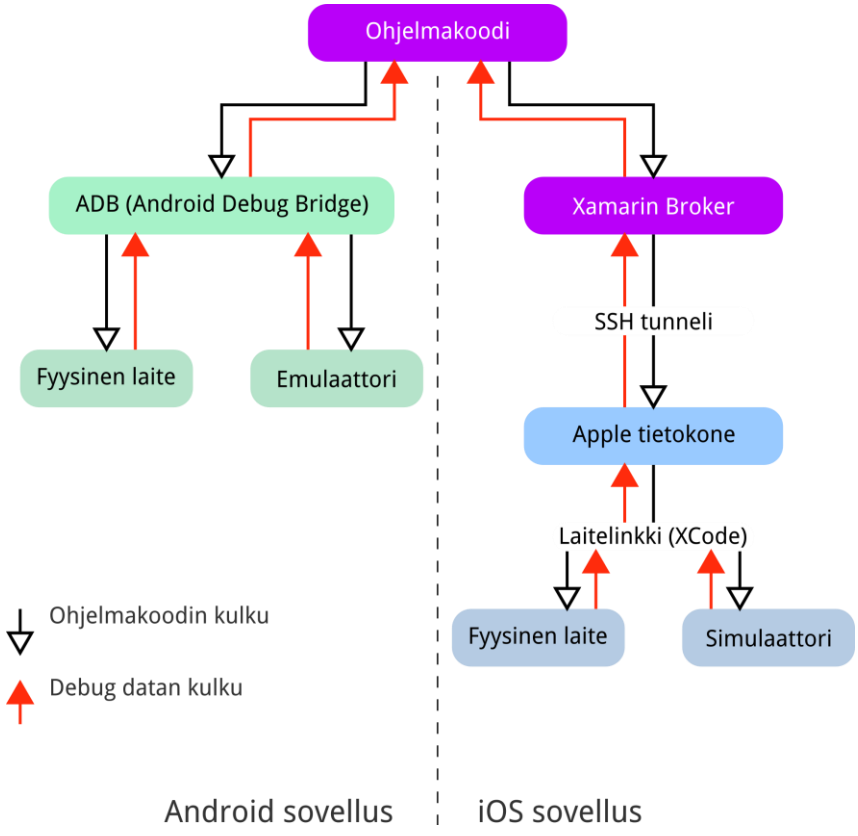


Kuva 2. Xamarin.Formsin sisäinen rakenne

Käyttöliittymän komponentteja voi myös kehittää kohdealustalla. Xamarin.Formsin avulla on mahdollista käyttää alustatasolla kehitettyjä komponentteja saumattomasti jaetussa koodissa. Lipuntarkistussovelluksen kameranäkymä on kehitetty molemmille alustoille erikseen, mutta sen käyttö ei eroa normaalista näkymästä jaetussa koodissa.

Virheidenjäljitys eli debuggaaminen tapahtuu keskeytyspisteiden avulla. Kun ohjelman koodin suoritus saavuttaa keskeytyspisteen, suorittaminen keskeytyy ja Visual Studion työkalujen avulla on mahdollista katsella ja muokata laitteen muistissa olevaa dataa. Keskeytyspisteiden datan toimitustapa riippuu, millä alustalla sovellus on.

Androidilla datan kulku tapahtuu Android Debug Brdige:n (ADB) kautta (Kuva 3), joka hallinnoi Android-laitteita. ADB hallinnoi fyysisiä laitteita sekä emulaattoreita. Android kehityksessä fyysisen laitteen ja emulaattorin ero on hyvin pieni (Android Developer). Android emulaattori on itsessään käyttöjärjestelmän näkökulmasta oikea laite, sillä rautatason tarpeet emuloidaan. Emulaatio sisältää paikantamisen, kamerat sekä kaikki laitteen sensorit (mm. kompassi ja kiihdytysanturit).



Kuva 3 Xamarin.Formsin virheenjäljityksen datan kulku

Applen iOS-sovelluksen debuggaaminen tapahtuu samankaltaisesti kuin Android puolella. ADB:n korvaa Apple-tietokoneella pyörivä XCode (Kuva 3), joka hallinnoi käytettäviä fyysisiä laitteita ja virtuaalisia laitteita. Visual Studio kommunikoi XCoden kanssa Xamarin Brokerin välityksellä SSH-tunnelin läpi.

Xamarin Broker vastaa kaikesta kommunikaatiosta Applen-tietokoneen kanssa kehityksen aikana. XCode on vaadittu osa Xamarin.Forms kehitystä Applen laitteille, joten sen sisällyttämistä projektin kokoonpanoon ei voi välttyä.

Toisin kuin Androidin emulaattori, Apple ei tarjoa laitteiden emulointia, vaan fyysisten laitteiden rinnalla on simulaattori. Laitesimulaattori vastaa oikeaa laitetta näytön resoluution ja kuvasuhteen osalta, mutta simulaattorissa ajettava koodi ei ole yhteensopiva fyysisen laitteen kanssa. Simulaattorilla on myös rajoituksia ominaisuuksien suhteen, kuten kameran puuttuminen.

2.3.2 Visual Studio ja versionhallinta

Sovelluksen kehitysympäristönä toimii Microsoftin Visual Studio 2019 Enterprise ja sen .NET työkalut. Versionhallintana käytettiin Azure DevOps palvelinta, johon Visual Studio integroituu saumattomasti. Versionhallinnan avulla sovellus on helppo palauttaa toimivaan muotoon virheiden sattuessa, sekä sen avulla projektin tiimin muut jäsenet voivat muokata ja katsella koodia. Tämän sovelluksen tiimi koostui yhdestä henkilöstä, joten koodilla oli vain yksi haara versionhallinnassa. Xamarin sovelluksia on mahdollista kehittää myös macOS käyttöjärjestelmässä Visual Studio for Mac:illä.

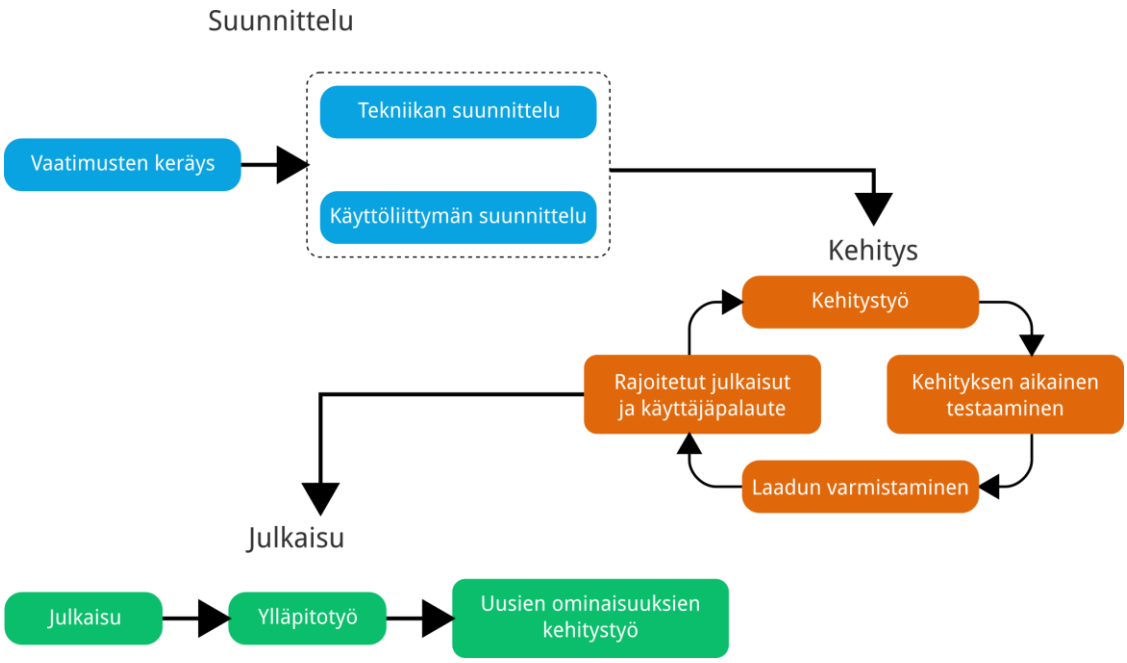
iOS sovelluksien kääntäminen ja salausavaimien varastointi vaatii Applen tietokoneen käyttöä. Windows koneella pyörivä Visual Studio ottaa yhteyttä samassa paikallisverkossa olevaan Applen tietokoneeseen Xamarin Brokerin avulla. Sovelluksen Apple isäntänä käytettiin vuoden 2016 iMac tietokonetta.

3 KEHITYKSEN KULKU

Mobiilisovelluksen kehityskaari mukailee perinteistä vesiputousmallia, joka jakautuu kolmeen osa-alueeseen (Kuva 4). Kehitys alkaa suunnittelemalla projektin kulku alusta loppuun. Suunnitellussa kerätään kasaan sovellukselle asetetut vaatimukset asiakkaan haluamien ominaisuuksien perusteella. Tässä vaiheessa myös luonnostellaan sovellukselle ulkoasu ja sitä koskevat vaatimukset.

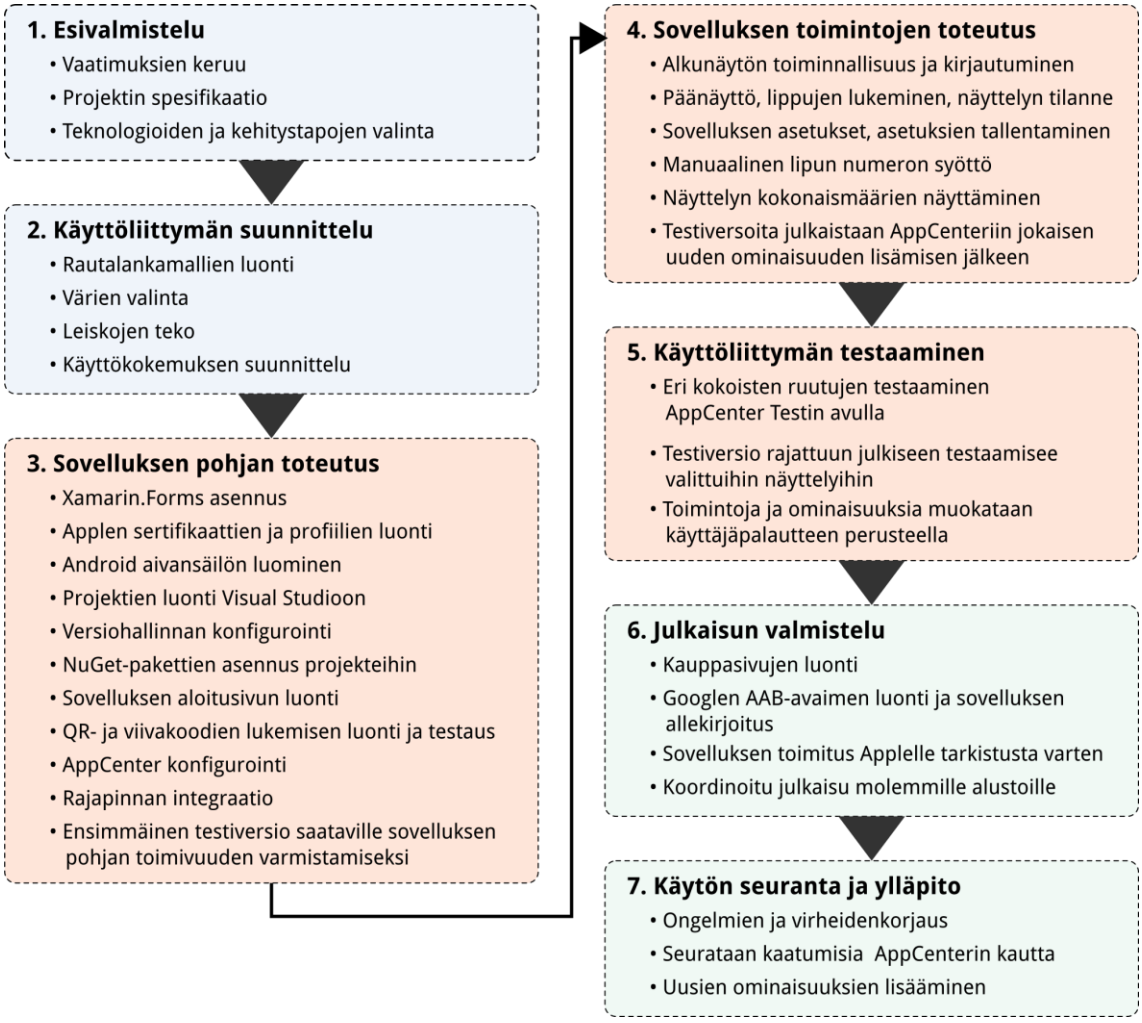
Toinen vaihe kehityksessä on sovelluksen toteuttaminen vaatimuksien ja suunnitelmien perusteella. Kehitys vaiheen tärkeä kulmakivi on jatkuva sovelluksen testaaminen. Ohjelmakoodin ja käyttöliittymän toimintaa testataan jatkuvasti.

Viimein vaihe on sovelluksen julkaisu tuotantoon ja asiakkaan käyttäjien käytettäväksi. Julkaisun jälkeen sovelluksen käyttöä ja suoriutumista seurataan tarkasti. Esiintyvät virheet ja ongelmat ratkaistaan julkaisun jälkeisissä korjauspäivityksissä. Päivitykset sisältävät myös mahdollisesti uusia ominaisuuksia. Käyttäjäpalautteen seuraaminen ja siihen reagointi on tärkeä osa julkaisun jälkeistä työtä.



Kuva 4 Kehityksen kolme päävaihetta ovat suunnittelu, kehitys ja julkaisu

Kehityksen vaiheista johdettiin ensimmäisenä suunnitelma kehitystyön vaiheiden toteuttamisesta (Kuva 5). Suunnitelma kartoittaa projektin vaiheet tarkimmin ja asettaa ne ajalliseen järjestykseen. Tarkkoja päivämääriä työn vaiheille ei asetettu, sillä lopullinen julkaisupäivämäärä ei ollut tiedossa suunnitelman laadinnan aikana.

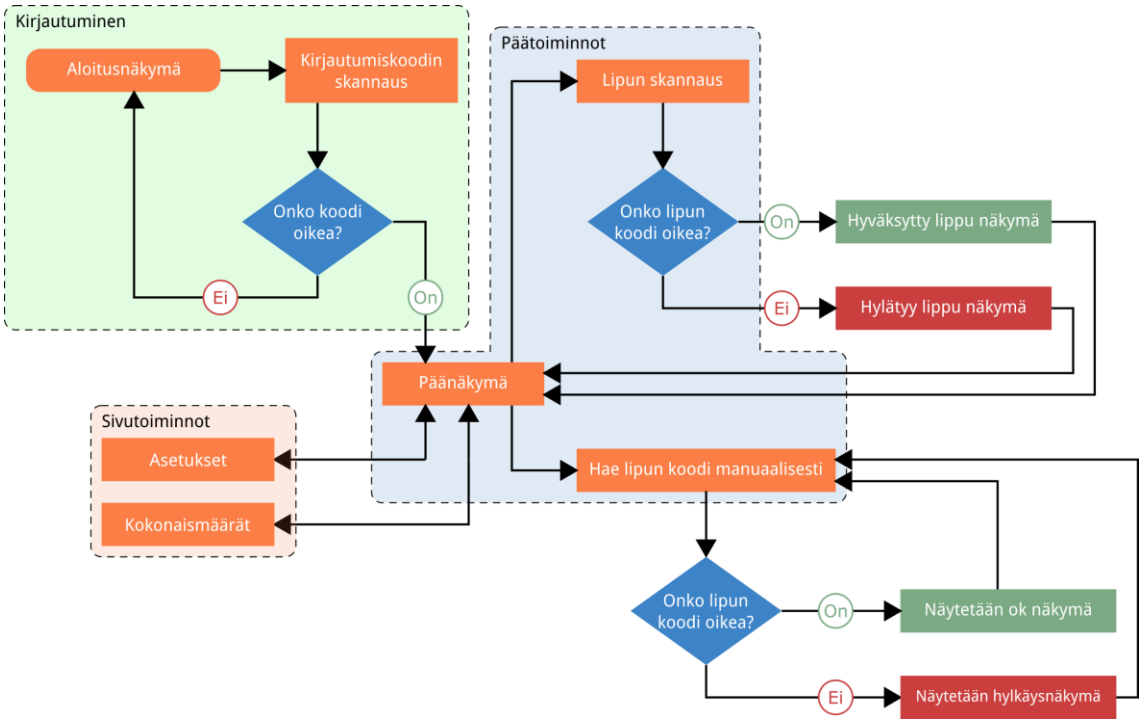


Kuva 5 Sovelluksen kehityksen suunnitellut vaiheet

3.1 Käyttöliittymän suunnittelu

3.1.1 Käyttöliittymän kulkukaaviot

Sovelluksen käyttöliittymän kehitys aloitettiin kulkukaavion laatimisella (Kuva 6). Kulkukaavio esittää eri toimintojen ja näkymien suhdetta toisiinsa ja kartoittaa, miten eri toiminnot toimivat keskenään. Kulkukaaviolla kartoitetaan, miten käyttäjän on mahdollista liikkua sovelluksen toimintojen välillä ja, mitä askelia toimintojen käyttäminen vaatii. Kulkukavion avulla on solmukohtien ja liian pitkien toimintoketjujen hahmotus helpottuu ja ne voidaan ratkaista jo suunnittelun aikana.



Kuva 6 Sovelluksen käyttöliittymän kulkukaavio

Lipuntarkistussovelluksen kulkukaaviossa on yksi selkeä solmu, päänäkymä, josta käyttäjä pääsee kaikkiin muihin toimintoihin. Tämä johtuu sovelluksen litteästä käyttöliittymästä, jossa toimintoihin päästään vaihtamalla näkymää välilehtimäisesti päänäkymässä. Tarkoituksena on nopea siirtyminen toimintojen välillä, joten päänäkymän solmu on odotettu sivuvaikutus.

3.1.2 Rautalankamallit

Ensimmäinen askel sovelluksen graafisen ulkoasun luomisessa on rautalankamallin tekeminen. Malli esittää sovellusta hyvin korkealla tasolla. Käyttöliittymän eri komponentit esitetään pelkistettyinä laatikoilla, tekstiä tai fontteja ei juurikaan esiinny ja värit eivät ole vielä mukana. Rautalankamalli tarkastelee myös komponenttien välisiä kontrastisuhteita. Käytettävyyden yksi tärkeä kulmakivi on riittävän suuri kontrasti käyttöliittymän osien välillä. Kehitys tapahtuu yleisesti joko suoraan paperille tai vektorigrafiikkaohjelmalla.

Käyttöliittymän rautalankamalli (Kuva 7) tehtiin kulkukaavion perusteella täyttämään sille asetetut vaatimukset perusominaisuuksien osalta.

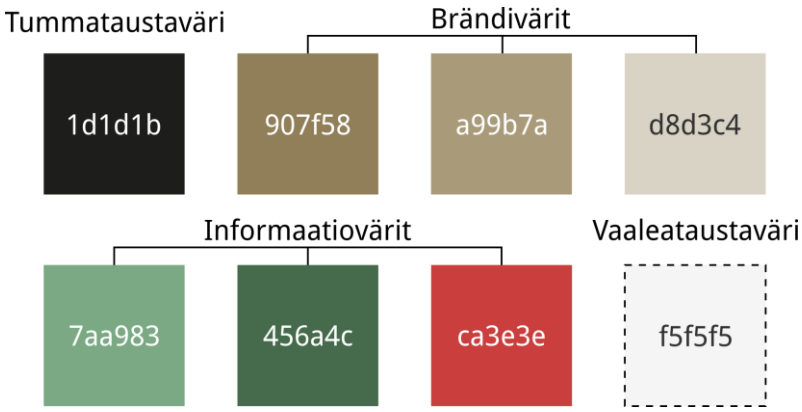
Kuva 7 Mustavalkoinen rautalankamalli sovelluksen ruuduista.

Vaalean teeman rautalankamallista tehtiin tummalle teemalle oma rautalankamalli (Kuva 8) tummanteeman yleisilmeen arvioimiseksi. Käytännössä tummateema invertoi tausta ja etualavärit, mutta tekstien kohdalla käytetään harmaata valkoista.

Kuva 8. Tummateemoitettu rautalankamalli.

3.1.3 Sovelluksen värit

Yhtenä vaatimuksena oli brändätä sovellus Showlink Oy:n omalla brändivärillä. Sovelluksen muut värit valittiin brändivärin ympärille (Kuva 9). Muut sovelluksen värit muodostettiin vaihtelemalla brändivärin valoisuutta ja värisävyä. Näin värit sopivat hyvin yhteen keskenään. Sovelluksen tummateema käyttää samoja värejä, mutta tekstit käyttävät harmaata valkoisen sävyä puhtaan valkoisen sijasta. Puhdas valkoinen mustalla taustalla on vaikeasti luettava liian suuren kontrastisuhteen vuoksi.



Kuva 9. Sovelluksen värit koostuvat brändiväristä, sen sävyistä, informaatioväreistä ja taustan kahdesta väristä.

Informaatiovärit seuraavat tavallista punaisen ja vihreän suhdetta. Vihreää väriä käytetään ilmaisemaan odotettua toiminnallisuutta, kuten onnistunutta lipun lukemista. Kun taas punaista käytetään virheiden ilmaisuun. Värisokeuden huomioimiseksi vihreälle otettiin käyttöön rinnakkainen vaaleampi vaihtoehto, joka takaa riittävän suuren kontrastieron punaiseen väriin. Kontrastieron avulla värit on mahdollista erottaa toisistaan myös täysin värittömässä kontekstissa, kuten kuvan 9 viidennessä ja kuudennessa näkymässä.

3.1.4 Käytettävyys ja värisokeus

Värisokeus vaikuttaa suuresti, kuinka käyttäjä kokee käyttöliittymän käytön. Noin 8 %:lla miehistä ja 0,5 %:lla naisista maailmanlaajuisesti on joko puna-viher- tai viherpunaheikkous. Puna-viherheikoista miehistä jopa 40 % on tietämätön omasta väriheikkoudestaan (Saarelma O.). Sovelluksien kehityksessä on otettava myös värisokeat henkilöt huomioon.

Käyttöliittymän kehitykseen sovellettiin seuraavia ohjeita koskien värisokeutta:

- Värit eivät ilmaise tärkeää tietoa
- Samanlaisten värien välillä tulee olla kontrastiero
- Vaaleiden ja tummien värien välillä tulee olla suuri kontrastiero
- Värien määrää tulee rajata
- Informaatio välitetään symbolien ja värien avulla

Ohjeet johdettiin ICALEPCS2017 konferenssissa olleen puhujan, Sakire Aytac:in kirjoittamasta artikkelista (S. Aytac), joka käsitteli hiukkaskiihdyttimien ohjauslaitteiden käyttöliittymien suunnittelua. Vaikka mobiilisovelluksen käyttöliittymä on täysin virtuaalinen, sen suunnittelu pohjautuu samoihin tekniikoihin värisokeuden osalta kuin tavallisten fyysisten käyttöliittymien suunnittelu.

Värien toimivuuden testaamisessa hyödynnettiin aiemmin tehtyjä rautalankamalleja (Kuva 10 ja 11). Värejä testattiin muokkaamalla väritetyt rautalankamallit vastaamaan eri värisokeuden tyyppejä (Liite 1), jolloin normaalisti näkevä kehittäjä pystyy arvioimaan värien kontrastin riittävyyden. Kaikista haastavin värisuhde on punaisen ja vihreän välinen toimivuus. Normaalisti näkevälle kontrastiero on suuri, mutta puna-vihersokea henkilö ei erota värejä toisistaan tai erottaa ne heikosti. Sovelluksen väreissä kontrastieron takaamiseksi punainen väri on valoisuudeltaan tummempi kuin päävihreä.

Kuva 10. Teemaväreillä väritetty rautalankamalli.

Kuva 11. Tummanteeman ulkoasu sovelluksen teemaväreillä.

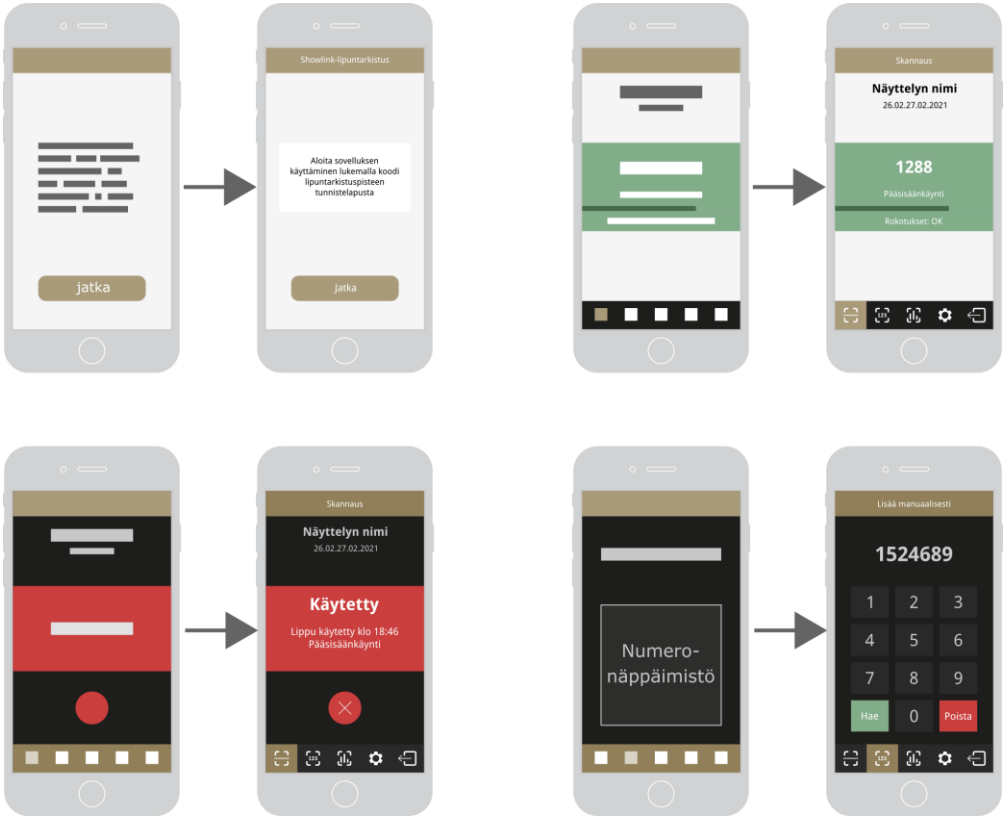
Kun punaista tai vihreää väriä käytetään ilmaisemaan informaatiota, kuten hylättyä tai hyväksyttyä lippua, tiedon välittämiseen ei käytetä pelkkää väriä. Apuna käytetään selkeää tekstiä, symboleita ja animaatioita. Esimerkiksi hylätyn lipun näkymä eroaa hyväksytystä lipusta animoidulla virheen kuittausnapilla, jossa on sulkemista indikoiva symboli.

3.1.5 Visuaalisuuden vaikutus käytettävyyteen

Käyttäjän kokema käyttöliittymän laatu on käänteisesti verrannollinen käyttöliittymän visuaalisien elementtien määrään nähden (Taba et al.). Suurimäärä erilaisia käyttöliittymän elementtejä tuntuu vaikeasti lähestyttävältä ja ahtaalta. Käyttöliittymän lähestyttävyyden takaamiseksi visuaalisien elementtien välillä käytetään riittävästi tyhjää tilaa ja vain välttämätön informaatio näytetään.

3.2 Käyttöliittymän asettelu

Käyttöliittymän suunnittelun viimein vaihe on käyttöliittymän asettelun tekeminen. Asettelussa rakennetaan rautalankamallien pohjalta käyttöliittymän elementtien sommittelu eli leiska, joka esittämään kehitettävän sovelluksen ulkoasun tarkempaa ja mahdollisesti lopullista muotoa. Sovelluksen ulkomuoto ei missään vaiheessa ole kiveen kirjattu, mutta se pyritään suunnittelemaan kattavasti kehitystyön helpottamiseksi. Liite 3 sisältää kaikki sovellusta varten tehdyt leiskat.



Kuva 12. Rautalankamallin kehittyminen leiskaksi lisäämällä yksityiskohtia, kuten tekstit ja symbolit

3.2.1 Interaktiivinen testaaminen

Leiskojen avulla on mahdollista kerätä palautetta sovelluksen käytettävyydestä ennen varsinaisen kehitystyön aloittamamaista hyödyntämällä interaktiivisia prototyyppejä. Leiskoista kootaan sovellusta muistuttava kokonaisuus, jossa liikutaan näyttöjen välillä määritettyjen kosketuspisteiden avulla. Interaktiivisen prototyypin luomiseen hyödynnettiin InVision-palvelua.

Interaktiivisen prototyypin luominen valmiista leiskoita kestää hyvin vähän aikaa, ja se mahdollistaa käyttöliittymän testaamisen jo ennen ensimmäistä koodiriviä. Prototyyppiä käyttämällä käyttöliittymän ongelmat ja tunnelma saadaan tuotua esiin tavalla, joka ei ole mahdollista pelkästään kuvia tuijottamalla. Saatuun hyötyyn nähden prototyypin luominen on hyvin kannattavaa.

3.3 Kehitystyö

Sovelluksen ohjelmakoodin kehitys suoritettiin suunnitelman mukaisesti kolmessa vaiheessa. Ensimmäiseksi toteutettiin sovelluksen pohja, tyhjä Xamarin.Forms-sovellus, joka kääntyi molemmille kohdealustoille ongelmitta käyttäen uusinta versiota kaikista vaadituista kirjastoista. Toisessa vaiheessa tyhjän pohjan päälle kehitettiin vaaditut sovelluksen ominaisuudet. Viimeisessä kehityksen vaiheessa sovelluksen käyttöliittymää testattiin syvällisemmin hyödyntämällä App Center:in tarjoamaan App Center Test -palvelua, sekä julkaisemalla rajattuja testiversioita asiakkaan kommentoitavaksi. Kerätty palaute ja automattisten testien tulokset huomioitiin muokkaamalla sovelluksen toimintoja ja ulkoasua.

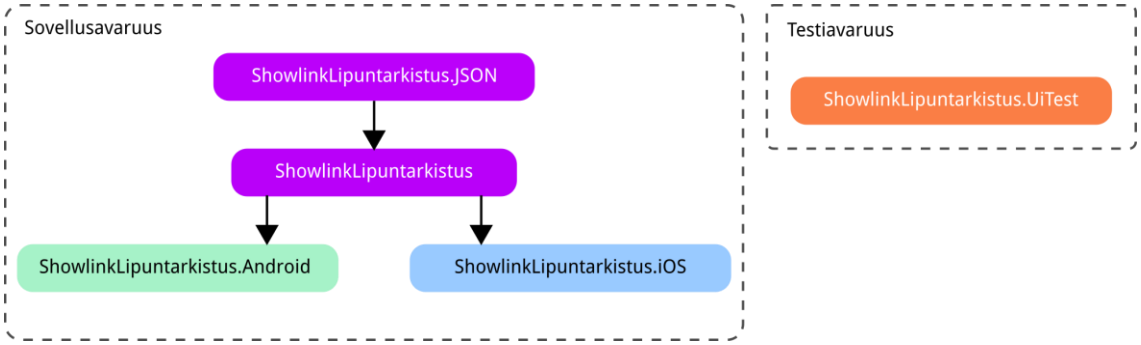
3.3.1 Projektin koodin rakenne

Xamarin.Formsilla toteutettava sovellusprojekti on rakenteellisesti jaettu vähintään kahteen, mutta yleensä useampaan osaan. Yksi osa sisältää jaetun koodin, joka on yhteistä kaikille kohteena oleville alustoille. Toinen osa koostuu alustakohtaisesta koodista, joka muodostaa yhden projektin jokaista alustaa kohden.

Modulaarisuuden ja testaamisen vuoksi on hyvä jakaa tiettyä tehtävää suorittava koodi omaan aliprojektiinsa (Kuva 13). Esimerkiksi tietorajapintaa käyttävä tietoliikennekoodi muodostaa on oma projektinsa. Kun tietoliikennekoodi on omassa projektissaan, se on helppo korvata testausta varten testikoodilla, joka testaa sovelluksen toimintaa kontrolloidussa ympäristössä ilmain yhteyttä verkkoon. Tämä mahdollistaa sovelluksen testaamisen myös ilman rajapintaa, ennen kuin palvelimet ovat käytettävissä.

Lipuntarkistussovelluksessa on mukana viisi projektia:

- ShowlinkLipuntarkistus
- ShowlinkLipuntarkistus.Android
- ShowlinkLipuntarkistus.iOS
- ShowlinkLipuntarkistus.JSON
- ShowlinkLipuntarkistus.UiTest



Kuva 13 Sovelluksen projektit ja niiden riippuvuudet toisistaan

ShowlinkLipuntarkistus. Sisältää ohjelmakoodin, joka on yhteistä kaikille alustoille. Suurin osa sovelluksen logiikasta ja näkymistä on tässä projektissa.

ShowlinkLipuntarkistus.Android. On Androidin aliprojekti. Tämä projekti sisältää kaikki Android-laitteille kehittämistä varten tarvittavan ohjelmakoodin, kirjastot ja resurssit.

ShowlinkLipuntarkistus.iOS. Android-projektin mukaisesti, iOS-projekti koostuu Applen iOS-laitteita varten tarvittavasta ohjelmakoodista.

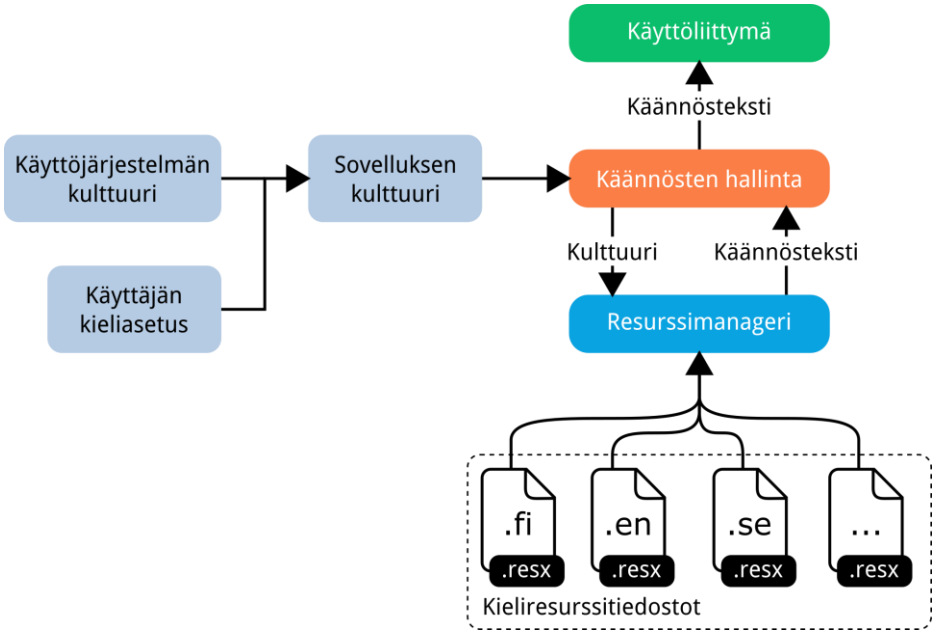
ShowlinkLipuntarkistus.JSON. Yhteyden muodostaminen ja datan siirtäminen on JSON-projektin vastuulla. Projektilla on vähiten riippuvuuksia ja se ei tarvitse Xamarin.Forms referenssejä.

ShowlinkLipuntarkistus.UiTest. Sovellusavaruuden ulkopuolella oleva UiTest-projekti vastaa sovelluksen käyttöliittymän testikoodista. Projekti koostuu automatisoiduista testimäärityksistä, jotka ajetaan sovellusta vasten sen käyttöliittymän testaamista varten. Projektin käännös tuottaa UiTest-kirjaston, joka ladataan App Center Testiin sovelluksen testien ajamiseksi pilvipohjaisesti fyysisillä laitteilla.

3.3.2 Kielituen toteuttaminen

Sovelluksen kielituki toteutettiin hyödyntämällä .NET:in omaa resurssihallintaa (Kuva 14). Resurssihallinta mahdollistaa kulttuuripohjaisten resurssien käytön sovelluksen ajon aikana. Jokaista tuettua kieltä kohden luotiin oma resurssitiedosto (.resx), joka sisältää avain-arvo parin jokaista sovelluksen tekstielementtiä kohden. Resurssitiedostot käännetään C# olioiksi ja sisällytetään osaksi sovelluksen koodia, jolloin niiden käyttäminen ajon aikana on nopeampaa.

Sovelluksen käynnistyessä ensimmäistä kertaa oletuskulttuuriksi asetetaan käyttöjärjestelmän kulttuuri. Mikäli kulttuurin kieltä ei tueta sovellus siirtyy oletuksena käyttämään englannin kieltä. Käyttäjä voi vaihtaa sovelluksen kielen asetukset-sivulla. Kielen vaihtaminen ei vaadi sovelluksen käynnistämistä uudelleen vaan resurssimanagerin ansiosta käännökset voidaan hakea uudelleen eri kielelle heti.

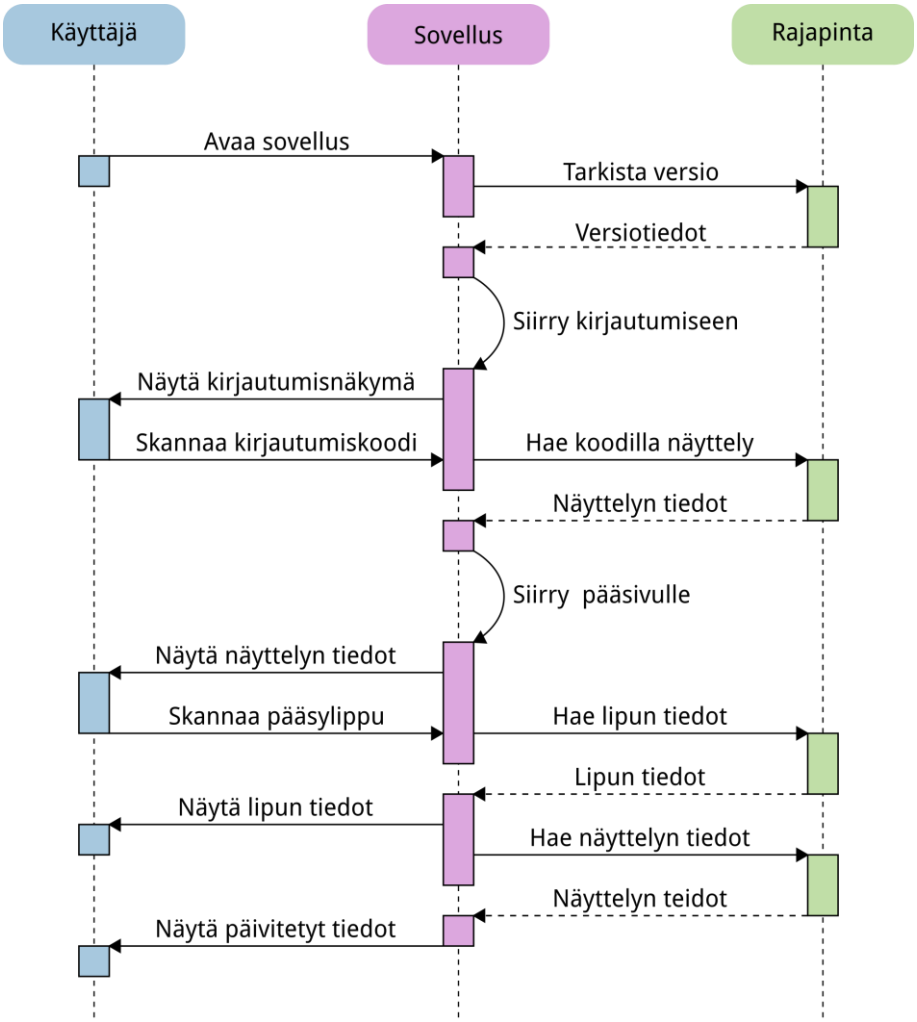


Kuva 14 Kielien dynaaminen käsittely resurssitiedostojen avulla

Käännösten hallinta hyödyntää Xamarin.Forms:in datasidontaominaisuutta yhdistämällä resurssimanagerin antaman käännöksen referenssin käyttöliittymän elementtiin. Kun kieli halutaan vaihtaa, tekstiä sisältävien käyttöliittymän elementtien tulee päivittyä. Näytettyjen tekstien päivitys tapahtuu vaivattomasti päivittämällä sidokset, jolloin käyttöliittymä hakee sidoksen avulla oikean käännöksen.

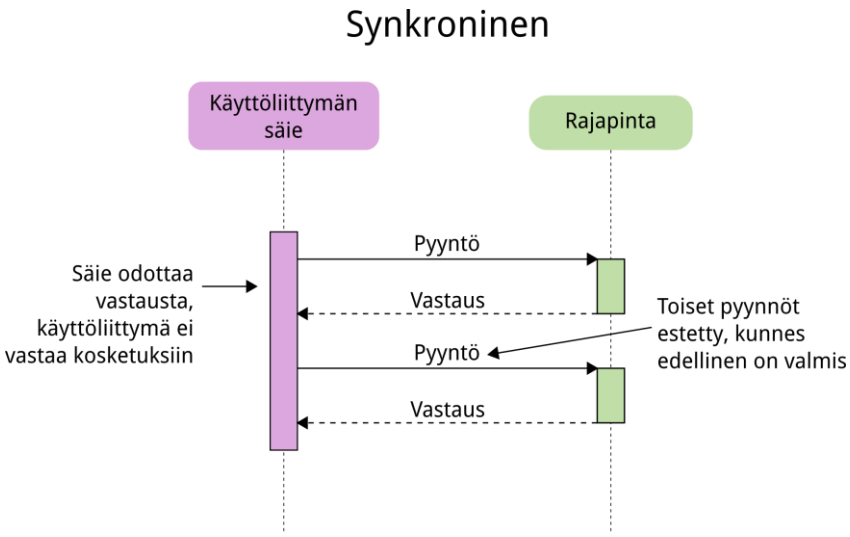
3.3.3 Integrointi taustapalvelimeen

Rajapinnan käytöstä huolehtii JSON-projektin koodi. Rajapinnan käyttö perustuu JSON datan siirtämiseen HTTPS yhteyden välityksellä. Kaikki rajapinnan kutsut käsitellään asynkronisesti sovelluksen pääsäikeen vapaana pitämiseksi.



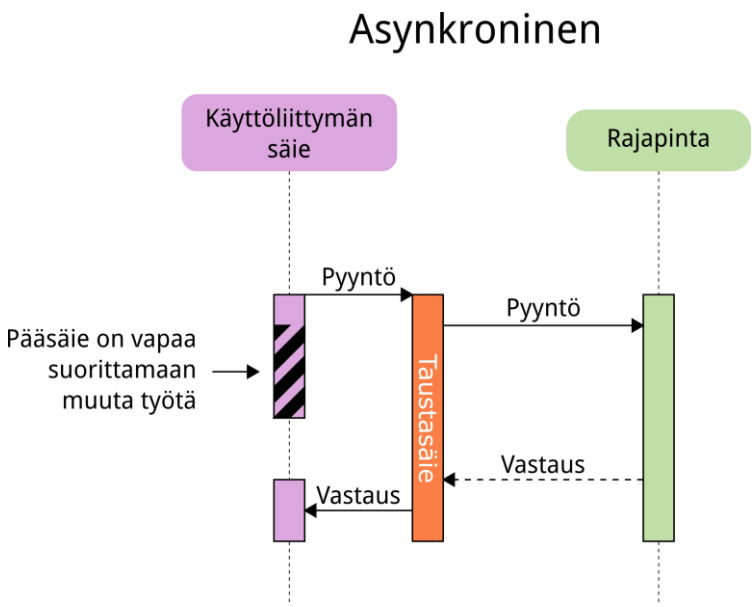
Kuva 15 Sovelluksen ydinominaisuuden, lipun lukemisen, sekvenssikaavio käyttäjän, sovelluksen ja rajapinnan välillä

Käyttöliittymän päivitys tapahtuu vain sovelluksen pääsäikeessä. Jos rajapintaa kutsutaan synkronisesti (Kuva 15), käyttöliittymän säie jää odottamaan kutsun valmistumista ennen kuin näyttöä voidaan päivittää. Kutsujen odottaminen jäädyttää käyttöliittymän, joka haittaa sovelluksen käyttökokemusta. Myös käyttöjärjestelmä seuraa pääsäikeen toimintaa. Jos käyttöjärjestelmä huomaa pääsäikeen olevan pysähtyneenä liian pitkään, Android näyttää käyttäjälle ilmoituksen ja mahdollisuuden sulkea sovellus. iOS voi suoraan kaataa sovelluksen, jos se on järjestelmän mielestä liian pitkään jäätyneenä. Joten pääsäikeen esteettömänä pitäminen on hyvin tärkeää sovelluksen toiminnan kannalta.



Kuva 16 Synkronisen rajapintakutsun sekvenssikaavio

Lipuntarkistussovelluksessa kaikki rajapintakutsut tapahtuvat asynkronisesti. Asynkroninen tapahtumaketju ei pysäytä käyttöliittymää (Kuva 16), vaan kutsun käsittely siirretään taustasäikeelle, joka odottaa rajapinnan vastausta. Kun vastaus saapuu, pääsäie käsittelee vastauksen ja päivittää käyttöliittymää tarvittaessa. Taustasäikeitä luodaan tarpeen mukaan lisää. Olemassa olevia säikeitä käytetään uudelleen.



Kuva 17 Asynkronisen rajapintakutsun sekvenssikaavio

3.4 Käyttöliittymän rakenne

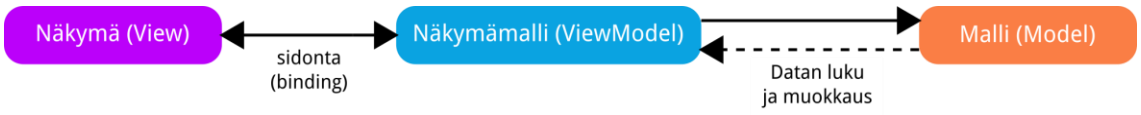
Xamarin.Formsin käyttöliittymän rakenne muistuttaa HTML-sivua, jossa komponentit ja näkymät ovat puumaisessa hierarkiassa keskenään. Käyttöliittymän näkymät määritellään käyttäen XAML (Extensible Application Markup Language) merkkauskieltä. Jaetussa koodissa määritellyn käyttöliittymän ulkoasu on kaikilla kohdealustoilla samantyylinen lukuun ottamatta natiiveja komponentteja, kuten esimerkiksi monivalintalistoja ja nappeja. Liitteessä 4 on vierekkäin sovelluksen päänäkymä kolmessa eri ympäristössä. Kaikissa ympäristöissä sovelluksen käyttöliittymä näyttää samalta.

XAML rakenteet käännetään kompilaatio vaiheessa C# koodiksi, joka nopeuttaa käyttöliittymän toiminta poistamalla tarpeen kääntää näkymiä kesken sovelluksen suorituksen. Kuitenkin kehitysvaiheessa XAML-koodia voidaan muokata kesken sovelluksen ajon jättämällä XAML-koodin kääntämisvaihe pois. Tätä kutsutaan Hot Reload -ominaisuudeksi. Hot Reload mahdollistaa nopeamman käyttöliittymän kehityksen, kun XAML-koodin muokkaukset näkyvät heti laitteen ruudulla.

Kun sovellus luo näytettävän sivun, ensimmäisenä ajetaan käyttöliittymän graafiset elementit luova koodi. Seuraavaksi suoritetaan taustakoodi (code behind),

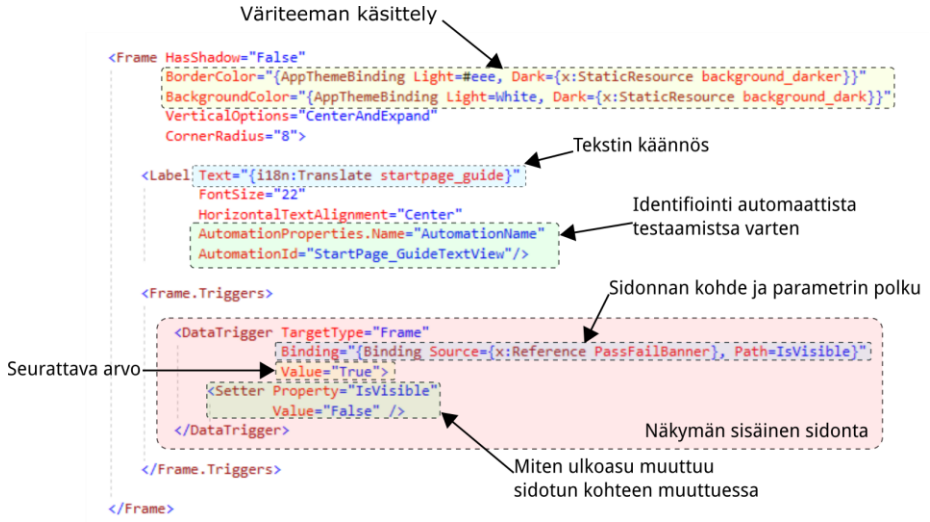
joka voi muokata luotua käyttöliittymää ennen sen esittämistä käyttäjälle. Taustakoodi voi sisältää myös interaktiivisien elementtien, kuten painonappien, toiminnallisuutta käsittelevän koodin. Pääsääntöisesti kuitenkin interaktiivisen koodin paikka on näkymämallissa.

Käyttöliittymän näkymien päivitys perustuu MVVM (Model-View-ViewModel) rakenteeseen (Kuva 18). MVVM eristää näkymän ja näkymän logiikan toisistaan riippumattomiksi, joka mahdollistaan niiden uudelleenkäyttämisen ja helpomman testaamisen. Näkymän logiikkakoodi ei ole osa näkymää, vaan se on oma komponenttinsa, jonka osat sidotaan näkymään.



Kuva 18 Model-View-ViewModel (MVVM) rakenne

Mallin päivitys laukaisee tapahtuman näkymämallissa, joka puolestaan sidonnankautta kertoo näkymälle, että sen on päivitettävä muutoksia koskevat osat käyttöliittymässä. Näkymässä hyödynnettiin myös sisäisiä sidoksia (Kuva 19), jolloin näkymän sisäiset osat sidotaan toisiinsa. Esimerkiksi sovelluksen etusivun ohjeteksti piilotetaan, kun virheilmoitus tulee näkyviin (Kuva 19).



Kuva 19 Aloitussivun ohjetekstin näkymän määrittelevä XAML-koodifragmentti

3.4.1 Käyttöliittymän teemoitus

Käyttöliittymän teema tarkoittaa käyttöliittymän elementtien visuaalista ulkoasua. Lipuntarkistussovelluksen teema seuraa pääsääntöisesti natiivien elementtien ulkoasua, mutta käyttöliittymän painonapit on teemoitettu näyttämään samalta molemmilla kohdealustoilla. Sovelluksen teema sisältää myös sen käyttämät värit ja niiden dynaamiset muutokset sovelluksen käyttöympäristön muuttuessa.

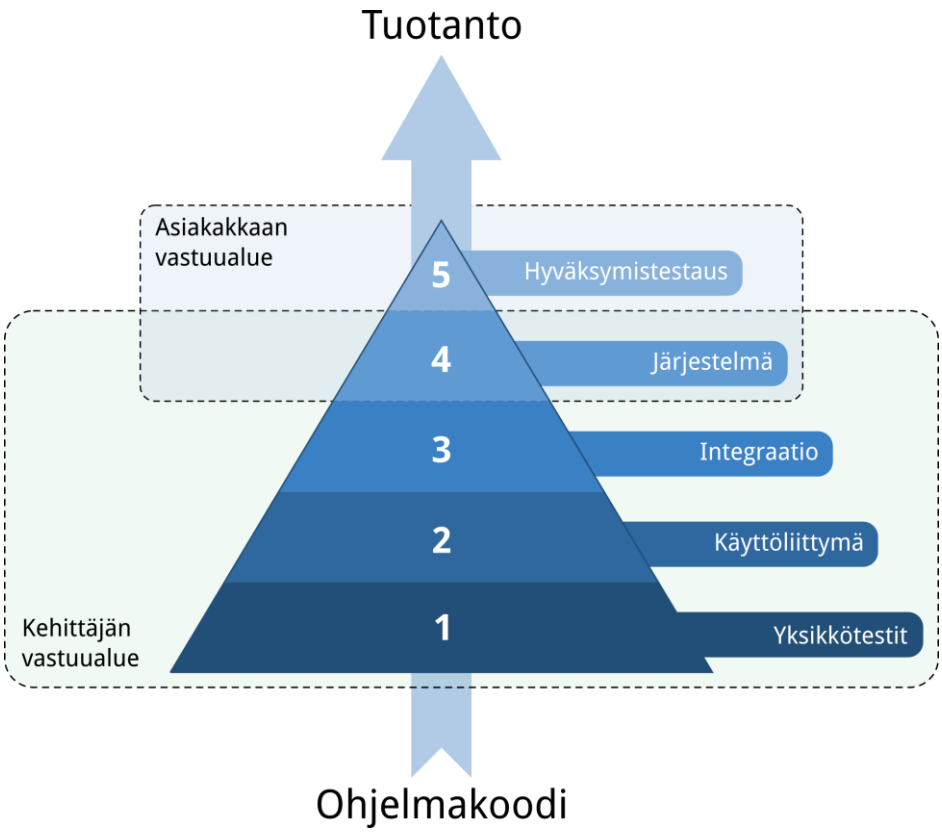
Sovellukseen implementointiin tumma- ja vaaleaväriteema. Xamarin.Formsin versio 4.6.0.967 lisäsi mahdollisuuden dynaamiseen teeman vaihtoon sovelluksen ajon aikana. Tumman teeman toimiminen vaatii Android version 10 (API 29) tai uudemman, iOS-laitteen tulee olla päivitetty iOS versioon 13. Sovelluksen käyttöjärjestelmän minimiversio on kuitenkin huomattavasti vanhempi. Jos laite ei tue tummaa teemaa, sovellus oletuksen käyttää vaaleaa teemaa ja toimii normaalisti.

3.5 Kehitystyön aikainen laadunvarmistaminen

Kehitystyön aikainen sovelluksen jatkuva testaaminen on tärkeä osa kehitystyön kulkua. Jokainen muutos sovelluksen toimintatapaan tai ominaisuuden lisäys on testattava koko sovelluksen toimivuuden varmistamiseksi. Yksinkertaisimmillaan testaaminen voidaan toteuttaa kehittäjän toimesta sovelluksen koodauksen yhteydessä, kuitenkin paremman testauskattavuuden saavuttamiseksi kehitystyössä hyödynnettiin muitakin testaustapoja.

Sovelluksen testaaminen jakautuu testiasteisiin (Kuva 20), jotka kerrosmaisesti testaavat sovellusta (Cohn M. s. 310–320). Asteet jakautuvat kehittäjän ja asiakkaan vastuualueiden kesken. Kehittäjä vastaa suurimmasta osasta sovelluksen laadun testaamisesta ja asiakas vastaa sovelluksen lopullisesta hyväksymistestaamisesta.

Testiasteet mukailevat Mike Cohn:in kirjassa Succeeding with Agile esitetyn testauspyramidin ideaa, jossa nopeasti toteutettavat ja hyvin eristetyt testit muodostavat pohjan suuremmille kokonaisuuksille.



Kuva 20 Sovelluksen ohjelmakoodi käy läpi usean eri testiasteen ennen tuotantoon pääsemistä

Yksikkötestit takaavat yksittäisien luokkien ja logiikan toiminnan sovelluksen sisällä. Käyttöliittymätestit toteavat käyttöliittymän toimivuuden ja, että kaikki visuaaliset elementit ovat esteettömästi näkyvillä laitteen näytöllä. Integraatiotasolla kaikki sovelluksen osat tuodaan yhteen ja niiden kokonaisuuden toimivuus todetaan ennen järjestelmätestaamiseen siirtymistä. Järjestelmätasolla sovelluksesta ladataan testiversioita asiakkaan kokeiltavaksi. Asiakkaan palautteet ja ideat kerätään, ja otetaan huomioon jatkokehityksessä. Viimeisenä askeleena asiakas kertoo, onko sovellus heidän mielestään julkaisukelpoinen. Hyväksymisen jälkeen sovellus siirretään tuotantoon julkisesti asennettavaksi.

3.5.1 Käyttöliittymän automatisoitu testaaminen

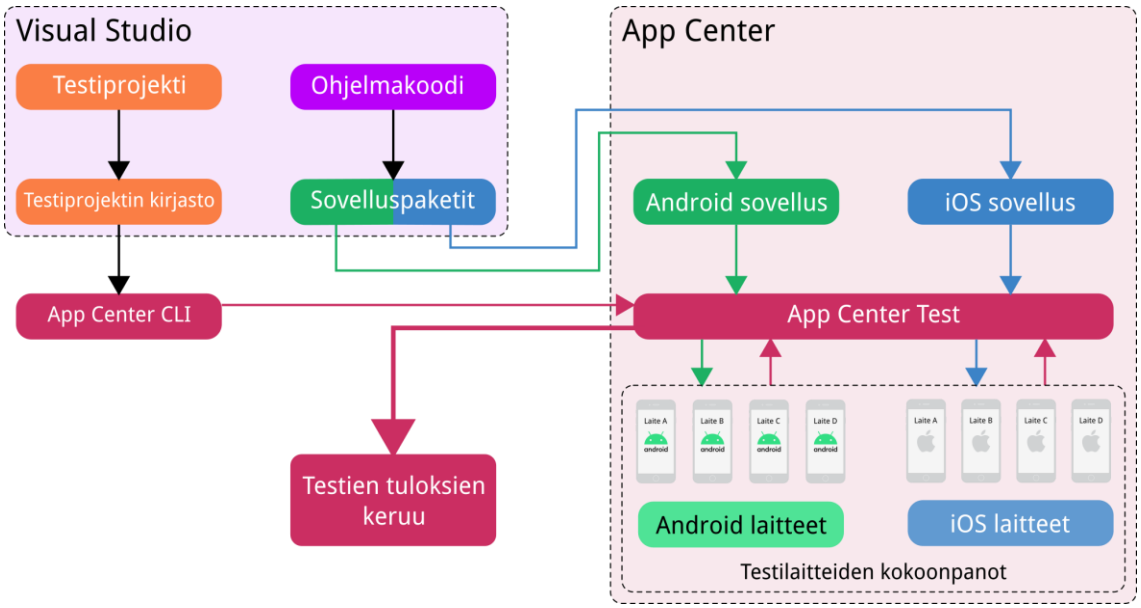
Mobiililaitteille kehittäminen asettaa suuren painoarvon käyttöliittymän toimivuuden varmistamiselle. Laitteen näytöt voivat erota toisistaan resoluutiollaan, kuvasuhteellaan ja koollaan. Mobiilisovelluksen käyttöliittymä tulee rakentaa tavalla, joka on käyttökokemukseltaan toimivia mahdollisimman suurella osalla kohteena olevista mobiililaitteista. Ideaalisesti käyttöliittymä toimisi kaikilla laitteilla virheettömästi, mutta käyttöliittymän testaaminen kaikilla yhteensopivilla laitteilla on lähes mahdotonta.

Automatisoituja käyttöliittymäntestejä varten on oma projektinsa, joka käännetään kirjastoksi. (Kuva 21) Kirjaston voi ajaa paikallisesti kehityskoneeseen kytketyillä laitteilla ja emulaattoreilla. Paikallisella testien ajamisella todetaan kehitettyjen testien kattavuus ja, että testikirjaston ajo onnistuu ennen sen siirtämistä pilveen.

Käyttöliittymän toimivuuden testaamiseen suurella kirjolla laitteita käytettiin Microsoftin App Center palvelua. App Center tarjoaa kehittäjille mahdollisuuden integroida käyttöliittymän toimintaa tutkivia yksikkötestejä, jotka suoritetaan pilvipalvelun kautta fyysisillä Android ja iOS laitteilla.

Automaattisten testejä varten App Center:iin luotiin ensin Android ja iOS sovellukset, joihin ladattiin käännetyt testiversiot. Automaattiset testit tarvitsevat myös kohdelaitekonfiguraatiot, joka määrittää, millä laitteilla testit ajetaan.

Paikallisesti toimivaksi todettu testiprojekti ladattiin App Center:in komentokehotetyökalun (CLI) avulla App Center:iin, jossa se suoritetaan testiprojektissa määritetyillä alustoilla.



Kuva 21 App Center -palvelun avulla toteutetun testaamisen rakenne

Testien suorittaminen tapahtuu automaattisesti kaikilla testiin valituilla laitekonfiguraatioilla. Jokaisesta testivaiheesta kerätään dataa, kuten esimerkiksi ruudunkaappaus ennen testiä ja sen jälkeen.

3.5.2 Rajoitettujen julkaisujen käyttö

App Centerin yksi keskeisiä ominaisuuksia on testiversioiden levittäminen valituille käyttäjille. Testiversioista kerätään käyttödataa App Centeriin, joka auttaa ongelmien etsimisessä ja ratkomisessa. Käyttäjät jaetaan alustapohjaisesti ryhmiin. Lipuntarkistussovellusta varten luotiin yksi testaajaryhmä per alusta (Android ja iOS), joille jaettiin hajautettua testaamista varten testiversio sovelluksesta.

3.6 Sovelluksen julkaiseminen

Koronatilanteen vuoksi lähes kaikki koiranäyttelyt sovelluksen kehityksen aikana oli peruttu (Showlink). Tilanteen takia sovelluksen julkaisua siirretään, kunnes koiranäyttelyitä voidaan taas järjestää normaalisti. Sovelluksen julkaisun vaiheet on kuitenkin suunniteltu ja ne esitellään seuraavissa luvuissa.

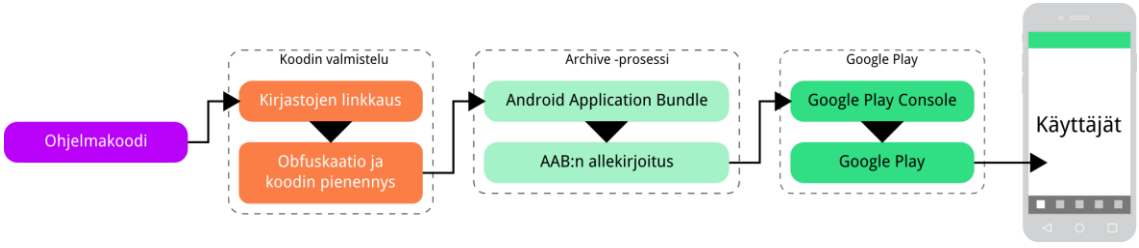
Julkaisuun asti sovelluksen kehitys on ollut yhtenäistä molempien kohdealustojen kesken. Julkaisuvaiheessa alustat erkanevat toisistaan, sillä Android ja iOS -sovellusten julkaiseminen sovelluskauppoihin eroavat toisistaan. Molemmilla on oma tapansa toimittaa sovelluspaketti käyttäjien asennettavaksi.

Sovelluksen testiversioiden julkaisu App Center -palveluun on käytännössä identtinen sovelluskauppaan julkaisun kanssa. Erona on iOS versiossa käytettävät sertifikaatit.

3.6.1 Android-sovelluksen valmistelu

Android-sovelluksen koodia suojataan konfiguroimalla ProGuard, joka obfuskoi ja pienentää Android-projektin Java koodia. ProGuard pienentää koodia poistamalla käyttämättömiä luokkia, kenttiä, metodeja ja attribuutteja. Android sovelluksissa on 64 000 metodireferenssin raja, jonka välttämiseksi hyödynnetään ProGuardia.

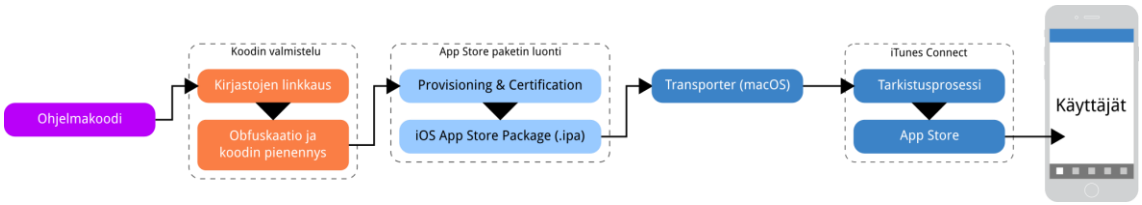
Sovellus toimitetaan Google Play Consoleen AAB (Android App Bundle) tiedostomuodossa (Kuva 22). App Bundle sisältää sovelluksen koodin ja resurssit, joista Google Play generoi asennettavat sovelluspaketit (.apk). Sovelluspaketin generoinnin siirtäminen Google Playn puolelle mahdollistaa sovelluspaketin optimoinnin sitä lataavalle laitteelle. App Bundle allekirjoitetaan paikallisella salaisella avaimella, jonka julkinen pari on Google Playn palvelimella. Allekirjoituksen perusteella Google Play varmistuu App Bundlen aitoudesta.



Kuva 22 Android-sovelluksen julkaisemisen askeleet

3.6.2 iOS-sovelluksen valmistelu

Androidista poiketen iOS-sovelluksen koodi ei vaadi obfuskaatiota, sillä sovelluksen koodi kääntyy suoraan natiiviksi binäärikoodiksi. Natiivibinääri paketoidaan yhteen sovelluksenresurssien kanssa iOS App Store Package (.ipa) tiedostoksi (Kuva 23). Ipa-tiedosto toimitetaan iTunes Consoleen App Transporter ohjelman avulla.



Kuva 23 iOS-sovelluksen julkaisun kulku

3.6.3 Sovelluskauppojen valmistelu

Käyttäjän ensimmäinen kosketus sovellukseen ennen sen asentamista on sovelluskaupassa sovelluksen tietosivu. Tietosivu sisältää informaatiota sovelluksen tarkoituksesta, viimeisimmän version muutoksista sekä sovelluksen toimintoja esittäviä kuvia.

Kauppasivujen varten laaditaan sovelluksen kuvaukset. Kuvaus kertoo lyhyesti, mikä sovelluksen tarkoitus on ja, mitä sillä voi tehdä. Seuraavaksi sovelluksesta tarvitaan näytönkaappauksia muutamasta ominaisuudesta. Apple on hyvin tarkka, että ladatut kuvat vastaavat haluttua resoluutiota. Google on hyvin väljä kuvien vaatimuksien suhteen.

3.6.4 Julkaisun koordinointi

Sovellusta ei julkaistu tämän opinnäytetyön aikana, sillä koronarajoituksien mahdollistama lisäkehitysaika halutaan hyödyntää mahdollisimman syvällisesti. Kuitenkin sovelluksen julkaisua varten suunniteltiin askeleet, joiden avulla sovellus saadaan asennettavaksi molemmille alustoille yhtäaikaisesti.

Julkaisun koordinoimista varten asetetaan päivämäärä, jolloin sovellus tulee olla asennettavissa sovelluskaupoista. Tarkkaa kellonaikaa ei voida taata, sillä Google suorittaa tarkistuksia ja asennuspakettien generointia sovelluspaketin julkiseksi asettamisen jälkeen. Googlen tekemät toimenpiteet kestävät yleensä muutaman tunnin. Applen App Storessa sovellus pääsääntöisesti näkyy muutaman minuutin jälkeen, kun sovellus on hyväksytty Applen toimesta.

Applen hyväksymää sovellusta ei kuitenkaan tarvitse julkaista heti, vaan se voidaan asettaa odottamaan kehittäjän toimenpidettä. Koska Androidin julkaisussa on muutaman tunnin epävarma aika, iOS sovellus laitetaan julkiseksi noin yhden tunnin Androdia jäljessä. Näin sovellus tulee molemmille alustoille julkiseksi arviolta tunnin sisällä toisistaan.

4 JULKAISUN JÄLKEINEN TYÖ

Mobiilisovelluksen kehitystyö jatkuu julkaisun jälkeen. Keskeinen osa seurantaa on sovelluksen virheiden ja kaatumisien keruu. Kerätty data helpottaa ongelmatilanteiden toistamista ja korjaamista huomattavasti.

4.1 Käytön seuranta

Testaamisessa käytetty App Center -palelua käytetään myös käytön seurantaan ja sovelluksen kaatumisien seurantaan. Seurannassa otetaan huomioon käyttäjien yksityisyys ja vain välttämätön data kerätään analysoitavaksi. Kaatumisien selvitystyössä tärkeimmät tiedot ovat käytetty sovelluksen versio ja käyttölaitteen kokoonpano (käyttöjärjestelmän versio ja laitteen malli). Näiden tietojen perusteella on mahdollista luoda kaatumista vastaavat olosuhteet ongelman toistamiseksi ja korjaamiseksi. App Center:illä kerätyn datan perusteella on mahdollista arvioida kohdealustojen käytön jakauma ja, millä käyttöjärjestelmäversioilla sovellusta käytetään. Kun tehdään päätöksiä sovelluksen minimiversion korotuksista, seurannan datalla nähdään tarkasti, kuinka moneen käyttäjää päätös vaikuttaa.

4.2 Käyttäjäpalaute

Käyttäjäpalautteen seuranta antaa suoran kuvan sovellukseen kohdistuvista mielikuvista ja odotuksista. Palautteen seuraaminen auttaa priorisoimaan korjauksia ja uusien ominaisuuksien sonnittelua. Palautetta tullaan keräämään suoraan asiakkaalta sosiaalisenmedian välityksellä, käyttäjiltä valituissa testinäyttelyissä, sekä sovelluskauppojen arvioteksteistä.

4.3 Virheiden korjaus

Sovelluksen ongelmia ja virheitä korjataan julkaisemalla päivityksiä ensimmäisen version julkaisun jälkeen. Korjauspäivityksen julkaisu tapahtuu identtisesti ensimmäisen version julkaisun tyyliin. Päivityksen yhteydessä sovelluskaupan tietosivulle kirjoitetaan kiteytetty lista korjauksista ja muutoksista.

5 YHTEENVETO

Tässä opinnäytetyössä kehitettiin monialustainen mobiilisovellus Android ja iOS alustoille käyttäen Xamarin.Forms ohjelmistokehystä ja sen apukirjastoja. Kehityksessä hyödynnettiin suuresti Microsoftin App Center -palvelua, jota käytetiin automaattisien käyttöliittymä testien suoritukseen, testiversioiden jakeluun, sekä käyttödatan keräämiseen.

Käyttöliittymän kehityksessä huomattiin, kuinka syvällinen suunnittelu ennen kehityksen aloittamista säästää aikaa ja vaivaa myöhemmin projektin aikana. Vektorigrafiikan muokkaaminen vie huomattavasti vähemmän aikaa kuin kirjoitetun koodin säätäminen, kun sovelluksen ulkoasua halutaan muuttaa. Selkolukuisista käyttöliittymän leiskoista on helppo tehdä interaktiivisia prototyyppejä, jotka tuovat esiin käyttöliittymän ongelmia jo ennen koodin kehityksen aloittamista. Myös Hot Reload -ominaisuuden tärkeys käyttöliittymien nopeassa kehityksessä korostui.

Koronarajoituksien vuoksi testijakson koiranäyttelyt oli peruttu (Showlink), joten testaamista ei voitu suorittaa oikeassa näyttely-ympäristössä. Perutuksista huolimatta sovelluksen testaaminen voitiin kuitenkin suorittaa keinotekoisessa ympäristössä oikealla näyttelyaineistolla. Rajoituksista johtuen sovelluksen julkaisua siirrettiin myös myöhäisemmäksi. Siirron tuoma lisäaika käytetään kehityksen jatkamiseen ja syvällisempään testaamiseen. Yhtenä mahdollisena lisäominaisuutena on push-viestien integrointi sovellukseen, jotka tukevat näyttelyiden sisäistä informaation kulkua.

LÄHTEET

Taba S.E.S., Keivanloo I., Zou Y., Ng J., Ng T. (2014) An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality. In: Casteleyn S., Rossi G., Winckler M. (eds) Web Engineering. ICWE 2014. Lecture Notes in Computer Science, vol 8541. Springer, Cham. https://doi.org/10.1007/978-3-319-08245-5_22

Microsoft. 2020a. What is Xamarin? https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (viitattu 30.4.2021)

Eisfeld H., Kristallovich F. (2020) The Rise of Dark Mode: A qualitative study of an emerging user interface design trend. DIVA. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1464394&dswid=6411

Saarelma O., Värisokeus ja poikkeava värinäkö, Lääkärikirja Duodecim. Duodecim Terveyskirjasto, https://www.terveyskirjasto.fi/dlk00347 (viitattu 30.4.2021)

S. Aytac (2017) Using Color Blindness Simulator During User Interface Development for Accelerator Control Room Applications. DESY, Hamburg, Germany. https://accelconf.web.cern.ch/ICALEPCS2017/papers/thsh103.pdf (viitattu 3.5.2021)

Showlink. Showlink näyttelyt 2021. https://www.showlink.fi/show/koiranayttelypalvelut/nayttelyt2021/ (viitattu 3.5.2021)

Android Studio (v 4.0.1), Android Platform/API Version Distribution. 2021. https://developer.android.com/studio

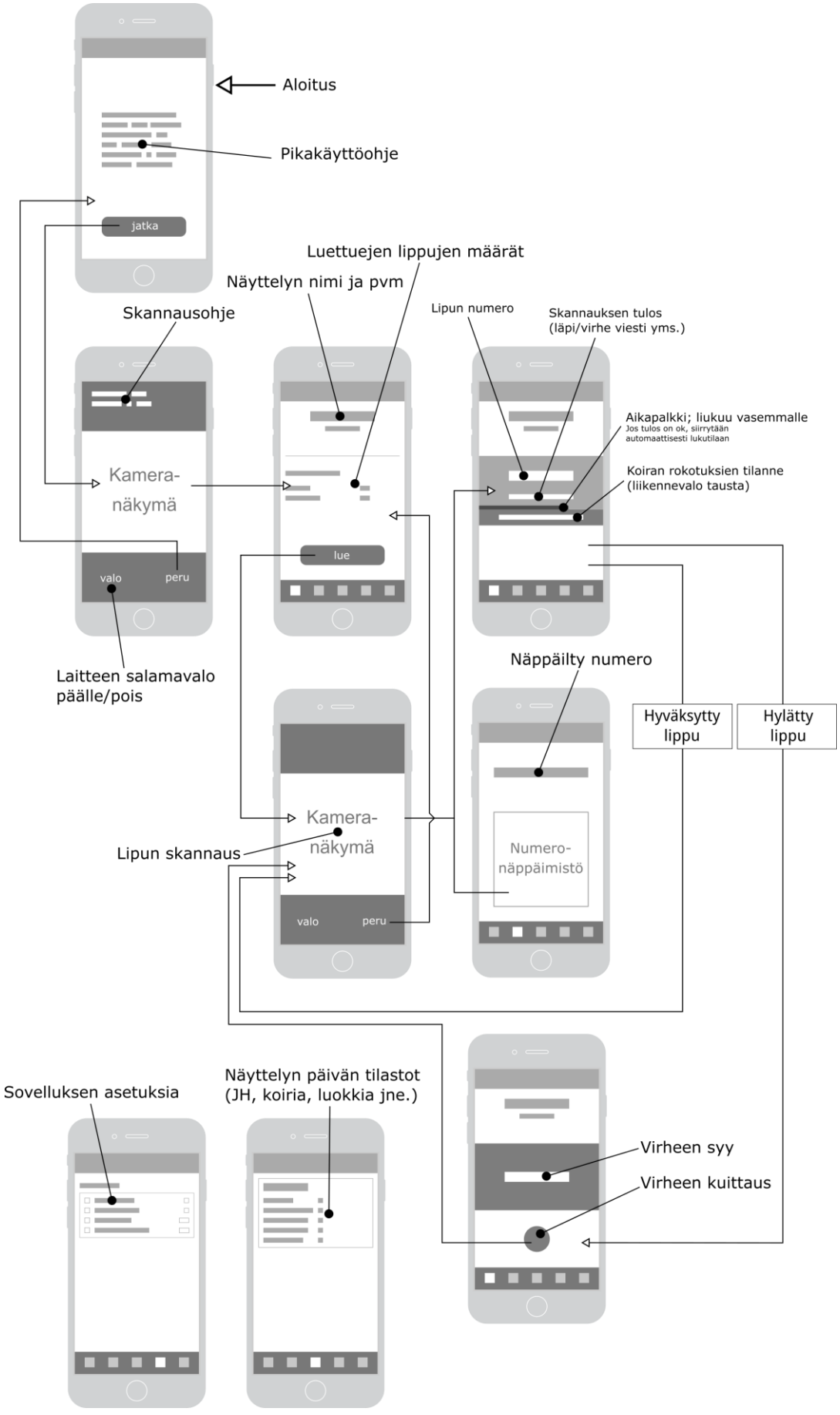
Android Developer. Run apps on the Android Emulator. 2021. https://developer.android.com/studio/run/emulator

Cohn M., (2009) Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional.

LIITTEET

Liite 1. Sovelluksen rautalankamallit muokattu vastaamaan puna- ja viherso-keutta sekä heikkoutta.

Liite 2. Sovelluksen käyttöliittymän rautalankamalli selityksineen



Liite 3. Sovelluksen kaikkien näkymien leiskat vaalealla- ja tummalla teemalla

Liite 4. Sovelluksen pääsivu esitetty Android emulaattorilla, fyysisellä iOS-laitteella ja iOS-simulaattorilla