Mika Repo

Flutterin integrointi olemassa olevaan natiiviin projektiin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

24.4.2022

Tiivistelmä

Tekijä:	Mika Repo
Otsikko:	Flutterin integrointi olemassa olevaan natiiviin projektiin
Sivumäärä:	41 sivua
Aika:	24.4.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Projektipäällikkö Mia Karlsson

---

Tässä insinöörityössä käsitellään menetelmää rakentaa yhteisiä näkymiä iOS- ja Android-projekteihin.

Flutter on avoimen lähdekoodin käyttöliitymäkehys, jota käytettiin insinöörityössä. Sillä pystyy rakentamaan näkymiä, joita sitten iOS- sekä Android-projektit käyttävät. Työssä ohjeistetaan Flutterin asennusta ja sen integroimisesta iOS- ja Android-projekteihin. Kerrotaan myös vinkkejä ja keinoja rakentaa ja käyttää Flutteria. Lopputuloksena tehtiin 7 ”Onboarding”-näkymää Android-sovellukselle.

Lisäksi mietitään Flutterin käyttöönoton kannattavuudesta. Sen hyviä ja huonoja puolia. Sekä missä tilanteessa kannattaa ottaa/suunnitella sen käyttöönottoa.

Avainsanat: Flutter, Android, iOS, projekti, käyttöönotto

Abstract

Author:	Mika Repo
Title:	Deploy Flutter to an Existing iOS/Android Project
Number of Pages:	41 pages
Date:	24 April 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer
	Mia Karlsson, Project Manager

This graduate study deals with the installation of Flutter to existing iOS/Android projects. Pros and cons of flutter deployment are discussed in the paper. Profitability of the entire Flutter deployment in existing projects is also discussed. Implementing the “Onboarding” pages to an existing app is introduced in detail.

Keywords: Flutter, Android, iOS, project, deployment

Sisällys

1 Johdanto 1
2 Työkalut 1
2.1 Flutter 2
2.2 React Native 2
2.3 Visual Studio 2
2.4 Xcode 3
2.5 Android Studio 3
2.6 Swift 3
2.7 Java 3
3 Flutterin integrointi natiiveihin projekteihin 4
3.1 Flutterin asennus 4
3.2 Flutterin integrointi iOS:ään 5
3.3 Flutterin integrointi Androidiin 6
4 Onboarding-tapahtuman toteutus 7
4.1 StatelessWidgetin rakenne ja toiminta näkymässä 8
4.2 StatefulWidgetin rakenne ja toiminta näkymässä 11
4.3 Onboarding-tapahtuman aloittaminen ja näkymän näyttäminen 17
4.4 Kommunikaatio Flutterin ja Androidin välillä 19
4.4.1 Ostotapahtuman toiminta 23
4.4.2 Kolmannen näkymän toiminta 26
4.5 Siirtyminen Flutter-näkymien välillä ja poistuminen FlutterActivitysta 28
4.6 Lokalisaatioiden teko Flutterilla 32
4.7 Kuvien ja fonttien lisääminen Flutteriin 34
4.8 Flurry-seurannan teko Onboarding-tapahtumaan 35
5 Flutterin käytön arviointia 35
5.1 Sovelluksen koko 37
5.2 Responsiivisuus 37
5.3 Kannattavuus 37
5.4 Mielipide Flutterista ja sillä työskentelystä 38
6 Yhteenveto 39
Lähteet 40

1 Johdanto

Insinöörityön tavoitteena oli löytää keino luoda näkymiä, joita sekä iOS että Android pystyvät käyttämään. Ei tarvitsisi luoda näkymiä kahteen kertaan erikseen iOS:lle ja Androidille. Tällä tavalla säästetään runsaasti aikaa. Tähän löytyi ratkaisuna Flutter. Flutteria lähdettiin kokeilemaan ja testaamaan aluksi Androidin puolella. Tarkoituksena on myöhemmin ottaa se käyttöön myös iOS-puolella ja tehdä sillä kaikki uudet tulevat näkymät sovelluksiin.

Insinöörityössä käydään läpi Flutterin asennus ja integrointi iOS:lle ja Androidille. Lisäksi käydään läpi Flutterin käyttöä Android-projektissa. Tuloksena tehtiin Android-sovellukseen ”onboarding”-tapahtuma. ”Onboarding”-tapahtumalla tarkoitetaan tapahtumaa, joka näkyy käyttäjille, kun he lataavat ja avaavat sovelluksen ensimmäisen kerran. Tapahtumassa käyttäjä siirtyy erilaisten näkymien välillä. Näkymissä kerrotaan sovelluksen ominaisuuksista, mainostetaan ”subscription”-tuotetta, jonka käyttäjä voi samalla heti ostaa. Lisäksi mukana on vapaaehtoinen sähköpostin ja nimen antaminen sovelluksen kehittäjille. Kun käyttäjä on mennyt tapahtuman läpi, avautuu hänelle varsinainen sovelluksen etusivu ja hänelle ei ”onboarding”-tapahtumaa enää näytetä.

Työssä kerrotaan Flutterin hyvistä ja huonoista puolista sekä sen käyttöönoton kannattavuudesta jo olemassa oleviin projekteihin.

2 Työkalut

Flutter on kehys, joka otettiin käyttöön toteutuksen ja projektin teon kanssa. Toisena vaihtoehtona oli React Native, joka on tehty samanlaiseen tarkoitukseen kuin Flutter. Kummallakin on mahdollista luoda käyttöliittymiä iOS:lle ja Androidille. Näin syntyy käytännössä niin sanottuja hybridisovelluksia, joissa sama käyttöliittymän koodi toimii kummallakin alustalla. Ei tarvitsisi erikseen tehdä eri alustoille omia käyttöliittymiään.

Muita käytettyjä työkaluja toteutuksen teossa olivat Android Studio ja Visual Studio. Lisäksi kerrotaan hieman Xcodesta.

2.1 Flutter

Flutter on Googlen käyttöliittymäkehys [1]. Sitä käytetään sovellusten kehittämiseen Androidille ja iOS:lle yhdestä koodikannasta [2].

Flutterilla voi siis tehdä näkymiä, joita kummatkin alustat käyttävät. Näkymien tekoa on kannattava käyttää olemassa olevissa natiiveissa sovelluksissa. Ei tarvitse tehdä näkymiä kahteen kertaan sovelluksien Android- ja iOS-versioille. Flutter-koodi varsinaisesti toimii frontendinä eli käyttöliittymänä ja natiivi koodi toimii backendina. Natiivikoodi toimisi tietokantana ja koneistona sovellukselle.

Flutter käyttää Dart-ohjelmointikieltä. Dart-ohjelmointikieli on kehitetty kehittämään erilaisia näkymiä, jotka tulevat toimimaan erilaisissa ympäristöissä. Esimerkkinä on mobiilikehityksessä iOS ja Android. [3.]

2.2 React Native

React Native yhdistää natiivikehityksen parhaat osat Reactiin, joka on JavaScript-kirjasto käyttöliittymien rakentamiseen. React Nativea voi käyttää nykyisissä Android- ja iOS-projekteissa tai voidaan luoda kokonaan uusi sovellus tyhjästä. [4.]

2.3 Visual Studio

Microsoft Visual Studio on Microsoftin valmistama ohjelmisto, jota käytetään erityyppiseen ohjelmistokehitykseen, kuten tietokoneohjelmiin, verkkosivustoihin, verkkosovelluksiin, verkkopalveluihin ja mobiilisovelluksiin. Se sisältää täydennystyökaluja, kääntäjiä ja muita ohjelmistokehitysprosessia helpottavia ominaisuuksia. Visual Studio on ohjelmisto, jonka avulla kehittäjät voivat kirjoittaa ja muokata koodia. [5.]

Flutter-ohjelmistolla ei ole omaa käyttöliittymää koodaamiseen ja kehittämiseen. Tässä työssä päätettiin ottaa Visual Studio Flutter -koodin teon käyttöliittymäksi. Se olisi erillisenä osana natiiveista koodeista. Flutter-koodia olisi mahdollista myös tehdä Android Studiossa. Flutter oli tarkoitus tulla toimimaan Androidin ja iOS:n puolella, minkä takia haluttiin pitää Flutter hieman erillään Android-puolelta.

2.4 Xcode

Xcode on ohjelmisto, jonka avulla kehitetään sovelluksia ja ohjelmia kaikille Applen ympäristöille. Kehittäjät voivat käyttää Xcodea käyttöliittymän suunnitteluun, sovelluskoodin kirjoittamiseen, koodin kääntämiseen, koodin ja sovelluksen testaamiseen ja virheenkorjaukseen [6].

2.5 Android Studio

Android Studio tarjoaa nopeimmat työkalut sovellusten luomiseen kaikentyyppisille Android-laitteille [7]. Kehittäjät voivat käyttää Android Studiota käyttöliittymän suunnitteluun, sovelluskoodin kirjoittamiseen, koodin kääntämiseen, koodin ja sovelluksen testaamiseen ja virheenkorjaukseen.

2.6 Swift

Swift on tehokas ja intuitiivinen ohjelmointikieli iOS:lle, iPadOS:lle, macOS:lle, tvOS:lle ja watchOS:lle [8]. Swift-ohjelmointikieltä käytetään Applen työympäristön ohjelmistojen ja sovellusten kehittämiseen. Swift on pääasiallinen ohjelmointikieli Xcode-ohjelmistossa.

2.7 Java

Java on yleiskäyttöinen, luokkapohjainen olio-ohjelmointikieli, joka on suunniteltu vähäisempään toteutusriippuvuuteen [9]. Java on ohjelmointikieli Android Studiossa, mitä kehittäjät käyttävät ohjelmistojen ja sovellusten kehittämiseen.

3 Flutterin integrointi natiiveihin projekteihin

3.1 Flutterin asennus

Flutter asennetaan Flutter-ohjelman sivulta. Siellä myös voi valita, mitä editoria voi käyttää Flutter-näkymien tekoon. Flutterilla ei ole itse minkäänlaista käyttöliittymää vaan pitää käyttää jotain muuta.

Vaihtoehtoina sen käyttöön sivulla annetaan Android Studio ja IntelliJ, Visual Studio Code ja Emacs. Päädyimme käyttämään Visual Studio Codea. Kun tarkoituksena Flutter-näkymiä on käyttää kummassakin Android:issa ja iOS:ssä. Ajatuksena oli, että on helpompaa tehdä erillisellä ohjelmistolla kuin esimerkiksi Android Studiolla ja sieltä sitten kiinnittää iOS:ään ja Xcode:een. Työssä haluttiin pitää Flutter-projekti hieman erillisenä elementtinä, jota kummatkin käyttävät.

Flutter ladataan zip-tiedostona, joka sitten puretaan haluttuun sijaintiin tiedostoissa. Esimerkkinä annetaan ”development”-kansioon purkaminen. Sitten ehdotetaan lisäämään ”flutter tool” valitulle reitille.

Polun päivittäminen tai lisääminen tapahtuu tekemällä tai avaamalla rc-tiedosto. Varsinaisesti se on ”$HOME/.bash_profile”- tai ”$HOME/.bashrc”-nimellä, jos käytössä on Bash-komentoikkuna. Jos on Z-komentoikkuna, käytetään ”$HOME/.zshrc”-nimeä. Sitten kun tiedosto on luotu, sen sisään kirjoitetaan kuvan 1 mukainen lauseke. Kirjoituksen ja tallennuksen jälkeen käynnistetään komentoikkunassa ”source $HOME/.<rc tiedosto>” päivittääksesi nykyisen ikkunan. Voidaan tarkistaa vielä, kun päivitetään nykyinen ikkuna komennoilla ”$ echo $PATH” ja ”$ which flutter”.

```
.zshrc
export PATH="$PATH:[PATH_OF_FLUTTER_GIT_DIRECTORY]/bin"
```

Kuva 1. Lauseke, joka kirjoitetaan .zshrc-tiedoston sisään. [PATH_OF_FLUTTER_GIT_DIRECTORY] kohdalle kirjoitetaan reitti Flutter-sijaintiin.

Kaiken tämän jälkeen komentoikkunassa laitetaan ja käynnistetään komento ”$ flutter doctor”. Komento tarkistaa käytössä olevan ympäristön ja raportoi komentoikkunaan (kuva 2). Raportista näkee, mitä ohjelmistoja puuttuu ja mitä pitää vielä ladata.

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.0.6, on macOS 11.5.2 20G95 darwin-x64, locale
    fi-FI)
[✓] Android toolchain - develop for Android devices (Android SDK version 31.0.0)
[✓] Xcode - develop for iOS and macOS
[✗] Chrome - develop for the web (Cannot find Chrome executable at
    /Applications/Google Chrome.app/Contents/MacOS/Google Chrome)
    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[!] Android Studio (version 2020.3)
    ✗ Unable to find bundled Java version.
[!] Connected device
    ! No devices available

! Doctor found issues in 3 categories.
```

Kuva 2. Tiedot ”flutter doctor” -komennosta. Näyttää, missä on ongelmia ja puuttuuko jokin osa.

3.2 Flutterin integrointi iOS:ään

Flutter-projektin integrointi jo olemassa olevaan Xcode-projektiin tarvitsee aikaisemmin tehdyt asennukset ja Xcode pitää olla myös asennettuna. Flutter tukee iOS 9.0:aa ja sitä jälkeisempiä versioita [10].

Seuraavaksi luodaan Flutter-projekti komentoikkunassa komennolla valittuun sijaintiin ”flutter create –template module (valittu projektin nimi esimerkiksi

”my_flutter”)”. Myös projektin pystyy luomaan käyttämällä Visual Studion komentoikkunaa tai itsessään Visual Studiota. Kun projekti on luotu, pitää suorittaa komento ”flutter build ios”, joka luo .ios/-kansion projektiin.

Varsinaisesti projektin luonnin jälkeen aletaan miettimään sen kiinnitystä Xcode-projektiin. Xcode-projektin integroinnissa on kaksi vaihtoehtoa. Ensimmäinen vaihtoehto on Flutterin lisäys käyttäen CocoaPodsia. Tarvittavat lausekkeet lisätään Podfile-tiedostoon, joka löytyy iOS-projektin kansiosta. Jos sitä tiedostoa ei ole olemassa projektissa, voi sen luoda. Tosin CocoaPods pitää olla asennettuna koneelle ennen kuin Podfile-tiedosto luodaan. Toinen vaihtoehto on Flutterin rakenteiden eli frameworkien lisäys iOS-projektiin. Xcodesta löytyy projektin ”targetista” lista frameworkeille. Sinne lisätään tarvittavat Flutter-rakenteet.

3.3 Flutterin integrointi Androidiin

Flutter-käyttöliittymän voi integroida olemassa olevaan Android studio -projektiin lähdekoodina Gradle aliprojektina tai AAR:na [11]. Vaihtoehtoja on kaksi tässä tapauksessa. AAR-vaihtoehdon kohdalla suoritetaan komento ”flutter build aar” Flutter-projektin kansiossa. Komentoikkuna tulostaa koodin pätkiä, jotka pitää sitten lisätä Android-projektiin. Aliprojektina Flutter lisätään Android-projektin settings.gradle-tiedostoon. Sinne tiedostoon lisätään tarvittavat koodin pätkät.

Flutter tukee tällä hetkellä vain AOT (ahead-of-time) -käännettyjä kirjastojen rakentamista arkkitehtuureille x86_64, armeabi-v7A j arm64-v8a [11]. Flutter projektin kiinnityksessä pitää myös huomioida, että on käytössä Java 8. Varsinaisesti Flutter Android engine käyttää Java 8:n ominaisuuksia.

AAR-vaihtoehdon kohdalla ongelmana ilmestyi, että ”flutter build aar” -komento ei toiminut. Ratkaisuksi tähän ongelmaan löytyi komennon suoritus Virtual Studion komentoikkunassa. Sitä aikaisemmin yritettiin Macin komentoikkunassa. Visual Studiossa nähdään, kun sitä komentoa suoritetaan, komentoikkuna muuttuu bash-komentoikkunaksi (kuvat 3 ja 4). Normaalisti se on zsh. Voi olla

mahdollista, että komento ei toimi zsh-komentoikkunassa, joka on Macissa oletuskomentoikkuna.

Kuva 3. Normaalisti on zsh-komentoikkuna.

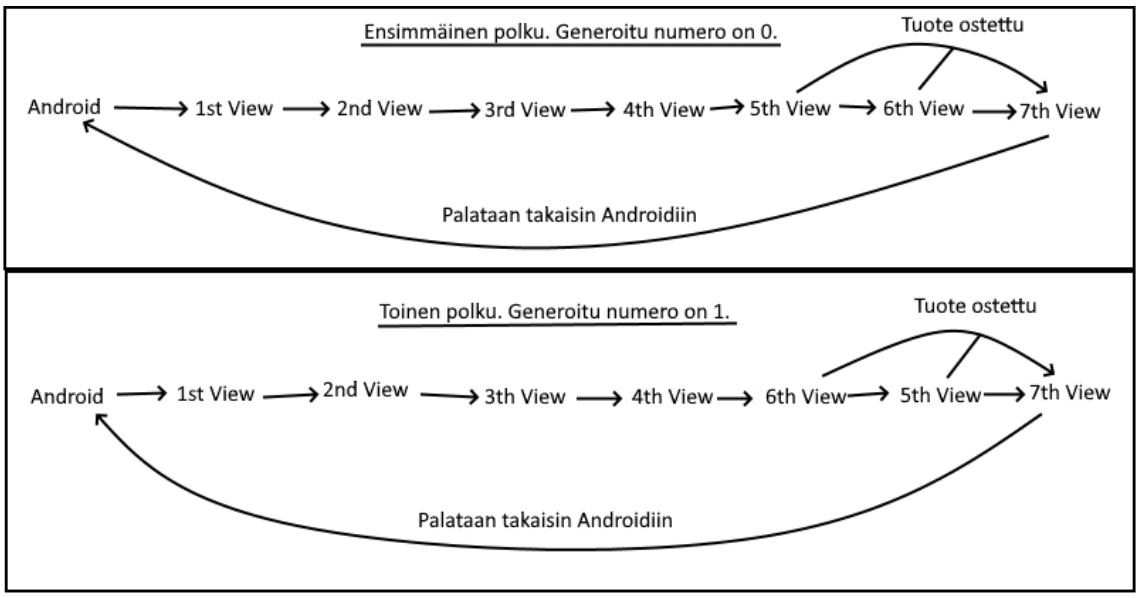
Kuva 4. Kun komentoa suorittaa, se muuttuu bash-komentoikkunaksi.

4 Onboarding-tapahtuman toteutus

Seuraavana on kysymyksiä, joita heräsi ennen Flutterin käyttöönottoa ja käytönoton jälkeen.

- Miten rakennetaan Flutter-näkymä? Toisin sanoen mikä on Flutter-näkymän rakenne?
- Miten siirrytään Flutter-näkymien välillä?
- Miten poistutaan Flutter-näkymistä takaisin natiiviin koodiin?
- Miten Flutter kommunikoi Android-natiivin koodin kanssa?
- Miten lisätään kuvia projektiin?
- Miten lisätään fontti/fontteja projektiin?
- Miten tehdään lokalisaatio projektiin?

Kuvassa 5 on kaksi polkua, joita käyttäjä voi kulkea onboarding-tapahtuman läpi. Kerrotaan polun valinnasta myöhemmin. Siinä samalla kerrotaan generoidusta numerosta. Onboarding-tapahtumassa on 7 eri näkymää, joiden läpi käyttäjä kulkee. Läpikäynnin jälkeen käyttäjä palaa takaisin Androidin koodiin ja näkymiin. Varsinaisesti käyttäjä ei voi kulkea näkymissä taaksepäin vaan voi ainoastaan kulkea seuraavaa näkymään. Tuotteen ostosta myös kerrotaan myöhemmin.



Kuva 5. Navigaatiokaavio onboarding-tapahtuman poluista.

4.1 StatelessWidgetin rakenne ja toiminta näkymässä

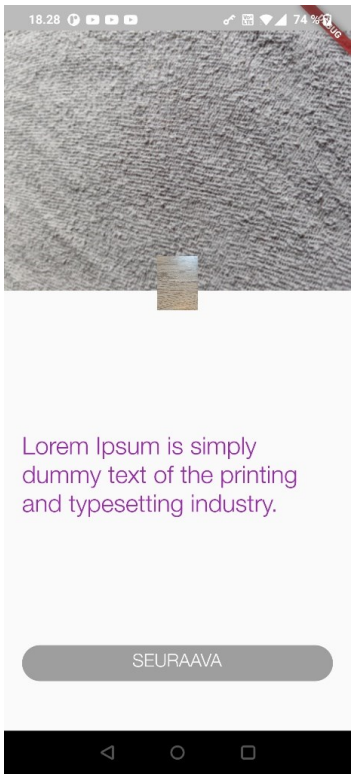
Flutter-näkymä rakennetaan widgettien avulla. Widgettejä on kahdenlaista: StatelessWidget ja StatefulWidget. StatelessWidget ei voi muuttaa sen tilaa sovelluksen käydessä, mikä tarkoittaa se ei voi varsinaisesti uudelleen piirtää itseään [12]. Kun StatelessWidget luodaan, sen näkymä ei muutu luonnin jälkeen.

Kuva 6 ja esimerkkikoodi 1 liittyvät ensimmäiseen onboarding-tapahtuman näkymään. Esimerkkikoodissa 1 widget nimeltä ”OnboardingView1” on kooltaan koko puhelimen näytön kokoinen. ”OnboardingView1” sisältää kolme widgetiä,

jotka ovat näkymässä yhdessä sarakkeessa. Tarkemmin sanoen ne ovat allekkain näkymässä menemättä toistensa päälle.

```
class OnBoardingView1 extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   setValueViewFirst();
   debugPrint('$view6thFirst');
   if (flurryCheckDone1 != true) {
     if (view6thFirst == true) {
       //flurryCheckB1();
     } else {
       //flurryCheckA1();
     }
     //flurryAorB();
   }
   flurryCheckDone1 = true;
   return Scaffold(
     //body: Image(alignment: Alignment.topCenter),
     body: SafeArea(
       child: Column(
         mainAxisAlignment: MainAxisAlignment.spaceBetween,
         crossAxisAlignment: CrossAxisAlignment.stretch,
         children: <Widget>[
           FirstPart(),
           SecondPart(),
           ThirdPart()
         ]
       )

     ),
   );
 }
}
```

Esimerkkikoodi 1. Ensimmäisen näkymän koodi. Sisältää kolme osaa, jotka kaikki ovat StatelessWidgettejä.



Kuva 6. Onboarding-tapahtuman ensimmäinen näkymä. 1st View.

”OnboardingView1” sisältää funktion nimeltä ”setValueViewFirst”, jossa tehdään 50/50 random -numeron generoiminen. Näkymien järjestys onboarding-tapahtumassa ovat erilaiset riippuen kumpi arvo generoidaan. Generoitu arvo voi olla 0 tai 1. Varsinaisesti näkymien järjestyksen muutoksilla katsotaan, missä reitissä käyttäjä ostaa tuotetta enemmän ja missä reitissä enempi käyttäjiä kulkee läpi onboarding-tapahtuman. Eli seurataan, kumpi reitti toimii paremmin. Seurantaan käytetään Flurrya. Flurrysta kerrotaan myöhemmin enemmän.

```
int randomNumber = random.nextInt(2);
```

Esimerkkikoodi 2. Näkymän luonnissa tehdään random-numeron generoiminen.

```
setValueViewFirst() {
 if (randomNumber == 0){
   view6thFirst = false;
 } else {
   view6thFirst = true;
 }
 debugPrint('Random number is: $randomNumber');
}
```

Esimerkkikoodi 3. Funktiossa päätetään kuudennen ja viidennen näkymän järjestystä käyttäen aikaisemmin generoitua numeroa.

Esimerkkikoodissa 4 ”FirstPart” koostuu kahdesta kuvasta, jotka on asetettu pinoon. Pienempi kuva tulee osittain isomman kuvan päälle. ”SecondPart” sisältää näkymän keskellä olevan tekstin. Näkymän kolmas ja alimmaisin osa koostuu napista, jolla käyttäjä siirtyy seuraavaan onboarding-näkymään.

```
class FirstPart extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Container(
     alignment: Alignment.topCenter,
     child: Stack(
       children: [
         Image.asset("images/testLautasKuva.jpg"),
         Container(
           width: double.infinity,
           margin: EdgeInsets.only(top: 250),
           height: 60,
           child: Center(
             child: Image.asset("images/testKuvaLogo.jpg"),
           ),
         ),
       ],
     )
   );
 }
}
```

Esimerkkikoodi 4. Ensimmäisen näkymän yläosa. Sisältää kaksi kuvaa.

4.2 StatefulWidgetin rakenne ja toiminta näkymässä

StatefulWidget eroaa StatelessWidgetistä sillä tavoin, että sen sisältö voi muuttua. Sen tila voi muuttua ja se voidaan uudelleen piirtää useaan kertaa sovelluksen käydessä. Widget jakautuu kahteen osaan. Ensimmäisessä osassa on ainoastaan createState-metodi, jolla luodaan toinen osa. Toinen osa on State,

jossa rakennetaan käyttäjälle näytettävä osa. Uudelleen piirtäminen tehdään setState-metodilla, joka sijaitsee State-osan sisällä. Uudelleen piirtämisen voi tehdä automaattisesti tai manuaalisesti käyttäjän tekojen mukaisesti.

Kuvan 7 ja esimerkkikoodin 5 kohdalla on tehty kuvakaruselli keskelle näkymää, mikä on StatefulWidget. Siihen keskiosaan kuuluu kuva ja kuvan alla olevat pallot. Kuva vaihtuu 5 sekunnin välein, ja pallojen väri vaihtuu riippuen, mikä kuva on näytillä. Joka kuvassa on yksi erivärinen pallo. Kun kaikki kuvat on käyty läpi, alkaa se ensimmäisestä kuvasta uudelleen. Tässä tapahtuu automaattinen uudelleenpiirtäminen. Käyttäjä ei voi sen toimintaan mitenkään vaikuttaa. Muut osat näkymästä on StatelessWidgetejä.

Kuva 7. Viides onboarding-näkymä. 5th View. Keskellä löytyy kuvakaruselli.

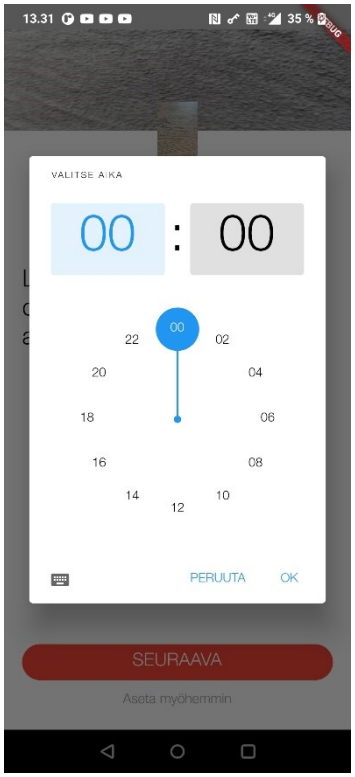
```
class ThirdPart extends StatefulWidget {
 @override
 _ReviewsView createState() => _ReviewsView();
}

class _ReviewsView extends State<ThirdPart>{
  @override
 void initState() {
   startTimer();
   super.initState();
 }

 startTimer() {
   if (_timer != null) {
     _timer.cancel();
   }

   _timer = Timer.periodic(timeout, (timer) {
     updateCounter();
   });

 }

 void updateCounter() {
   if (reviewIndex < maxValue) {
     setState(() {
       reviewIndex++;
       if (reviewIndex == 1) {
         ball1 = ballGrey;
         ball2 = ballGreen;
         ball3 = ballGrey;
         ball4 = ballGrey;
       } else if (reviewIndex == 2) {
         ball1 = ballGrey;
         ball2 = ballGrey;
         ball3 = ballGreen;
         ball4 = ballGrey;
       } else if (reviewIndex == 3) {
         ball1 = ballGrey;
         ball2 = ballGrey;
         ball3 = ballGrey;
         ball4 = ballGreen;
       }
     });
   } else {
     setState(() {
       reviewIndex = 0;
       ball1 = ballGreen;
       ball2 = ballGrey;
       ball3 = ballGrey;
       ball4 = ballGrey;
     });
   }
 }

 @override
 Widget build(BuildContext context) {
   return Container(
     padding: EdgeInsets.only(left: 50, right: 50),
     child: Column(
```

```
        children: [
          reviews[reviewIndex],
          Padding(
            padding: EdgeInsets.only(left: 80, right: 80),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                ball1,
                ball2,
                ball3,
                ball4,
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

Esimerkkikoodi 5. Viidennen näkymän keskiosan koodi, joka muodostaa kuvakarusellin.

Käyttäjä pystyy valitsemaan kellonajan sovelluksesta tulevalle muistutukselle kuvien 8 ja 9 sekä esimerkkikoodin 6 mukaisesti. Kun käyttäjä painaa näkymän keskellä löytyvää kellonaikaa, avautuu hänelle uusi näkymä, jossa hän voi asettaa kellonajan. Ajan valinnan jälkeen ja näkymästä poistuttua tehdään sovelluksessa uudelleen piirtäminen, josta käyttäjän valitsema kellonaika tulee vanhan kellonajan tilalle.

Kuva 8. Neljäs Onboarding-näkymä. 4th View.



Kuva 9. Käyttäjä voi näkymästä valita ajan.

```
class ThirdPart extends StatefulWidget {
 @override
 _TimePickerState createState() => _TimePickerState();
}

class _TimePickerState extends State<ThirdPart> {
 @override
 void initState() {
   final dt = DateTime(_year, _month, _day, DateTime.now().hour,
DateTime.now().minute);
   final format = DateFormat.jm();
   final format1 = DateFormat('HH:mm');
   _timeController.text = format1.format(dt);
   super.initState();
 }

 Future<Null> _selectedTime(BuildContext context) async {
   final TimeOfDay picked = await showTimePicker(
     context: context,
     initialTime: selectedTime,
   );
   if (picked != null) {
     selectedTime = picked;
     final dt = DateTime(_year, _month, _day, selectedTime.hour, se-
lectedTime.minute);
     final format = DateFormat.jm();
     final format1 = DateFormat('HH:mm');
     setState(() {
       _timeController.text = format1.format(dt);
     });
   }
 }

 @override
 Widget build(BuildContext context){
   //setValueViewFirst();
   return Container(
     padding: EdgeInsets.only(left: 70, right: 70),
     child: Column(
       crossAxisAlignment: CrossAxisAlignment.stretch,
       mainAxisAlignment: MainAxisAlignment.center,
       children: [
         Text(
           AppLocalizations.of(context).test_teksti_9,
           textAlign: TextAlign.center,
           style: TextStyle(color: Colors.black, fontSize: 20.0),
         ),
         ButtonTheme(
           minWidth: 200,
           height: 50,
           child: TextButton(
             style: ButtonStyle(
               shape: MaterialStateProperty.all<RoundedRectangleBor-
der>(
                 RoundedRectangleBorder(
                   borderRadius: BorderRadius.circular(18.0),
                   side: BorderSide(color: Colors.black),
                 ),
               ),
             ),
```

```
            onPressed: (){
              _selectedTime(context);
            },
            child: Text(
              _timeController.text,
              style: TextStyle(color: Colors.grey, fontSize: 20.0),
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```

Esimerkkikoodi 6. Neljännen näkymän keskiosan koodi. Käyttäjä pystyy vaihtamaan ja valitsemaan kellonajan.

4.3 Onboarding-tapahtuman aloittaminen ja näkymän näyttäminen

Esimerkkikoodin 7 mukaisesti lisätään FlutterActivity AndroidManifest.xml-tiedostoon Android Studiossa. FlutterActivityn suorittamisessa on kolme erilaista tapaa. Esimekkikoodissa 8 on ensimmäinen tapa tehdä se. Koodin pätkä lisätään startActivityn sulkeiden sisään. Siinä aloitetaan FlutterActivity oletustavalla. Toinen tapa nähdään esimerkkikoodissa 9. Siinä luodaan uusi FlutterEngine startActivityn yhteydessä.

```
 <activity
   android:name="io.flutter.embedding.android.FlutterActivity"
   android:theme="@style/MainTheme"
   android:configChanges="orientation|keyboardHidden|key-
board|screenSize|locale|layoutDirection|fontScale|screenLayout|den-
sity|uiMode"
   android:hardwareAccelerated="true"
   android:windowSoftInputMode="adjustResize"
   android:screenOrientation="portrait"
   />
```

Esimerkkikoodi 7. Koodin pätkä AndroidMainfest.xml-tiedotosta

```
FLutterActivity.createDefaultIntent(currentActivity)
```

Esimerkkikoodi 8. Ensimmäinen tapa käynnistää ”FlutterActivity”.

```
startActivity(
 FlutterActivity
```

```
  .withNewEngine()
  .initialRoute(”/my_route”)
  .build(currentActivity)
);
```

Esimerkkikoodi 9. Toinen tapa käynnistää “FlutterActivity”.

Kolmas tapa käynnistää FlutterActivity on luomalla ja alustamalla FlutterEngine ennen startActivityä. Esimerkkikoodeissa 10 ja 11 tehdään FlutterEnginen alustaminen ja startActivity käyttäen sitä alustettua FlutterEngineä. Onboarding-tapahtuman toteutuksessa käytettiin kolmatta tapaa. Valintaan vaikutti se, että FlutterEngineä tarvitaan myös methodChannelin kanssa Android-puolella, minkä takia alustettiin ja luotiin FlutterEngine ennen startActivityä. methodChannelista kerrotaan lisää myöhemmin.

```
flutterEngine = new FlutterEngine(this);

flutterEngine.getDartExecutor().executeDartEntrypoint(
       DartExecutor.DartEntrypoint.createDefault()
);

FlutterEngineCache.getInstance().put("my_engine_id", flutterEngine);
GeneratedPluginRegistrant.registerWith(flutterEngine);
```

Esimerkkikoodi 10. FlutterEnginen luontia ja alustamista.

```
startActivity(
       flutterActivity.withCachedEngine("my_engine_id").build(this)
);
```

Esimerkkikoodi 11. Kolmas tapa käyttäen aikaisemmin luotua FlutterEngineä.

Kun FlutterActivity käynnistetään Androidin puolelta, Flutterin puolella suoritetaan ensimmäisenä esimerkkikoodi 12 main.dart-tiedostossa. Siinä koodissa suoritetaan StatelessWidget nimeltä MyApp, joka näkyy kokonaisuudessaan esimerkkikoodissa 13. MyAppissa alustetaan tuetut lokalisaatiot (niistä lisää myöhemmin) ja fontFamily, jota kaikki Flutterin näkymät käyttävät. MyApp suorittaa OnboardingView1-widgetin, joka on ensimmäinen Onboarding-näkymä. OnboardingView1 nähdään MyAppin home-arvossa. Käyttäjälle tulee näkyviin ensimmäinen Onboarding-näkymä.

```
void main() => runApp(MyApp());
```

Esimerkkikoodi 12. Suoritetaan ensimmäisenä Flutter-koodissa.

```
class MyApp extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     builder: (context, child) {
       return MediaQuery(
         child: child,
         data: MediaQuery.of(context).copyWith(textScaleFactor: 1.0),
       );
     },

     localizationsDelegates: const [
       AppLocalizations.delegate,
       GlobalMaterialLocalizations.delegate,
       GlobalWidgetsLocalizations.delegate,
       GlobalCupertinoLocalizations.delegate,
     ],
     supportedLocales: const [
       Locale('en', ''),
       Locale('fi', ''),
       Locale('sv', ''),
       Locale('de', ''),
       Locale('es', ''),
       Locale('pt', ''),
       Locale('ru', ''),
       Locale('fr', ''),
     ],
     theme: ThemeData(fontFamily: 'CustomFont'),
     home: OnBoardingView1(),
   );
 }
}
```

Esimerkkikoodi 13. StatelessWidget nimeltä MyApp.

4.4 Kommunikaatio Flutterin ja Androidin välillä

Ensimmäisenä tehdään esimerkkikoodien 14 ja 15 mukaiset muuttujat. Android-puolella se on String ja Flutterin puolella se on MethodChannel. Ne pitää olla arvoltaan samanlaiset, koska niitä arvoja käytetään kanavana Flutterin ja Androidin välillä. Kuvassa 10 on kuudes onboarding-näkymä, jossa käyttäjälle kerrotaan erilaisista ominaisuuksista ja mitkä niistä on ilmaisessa versiossa ja mitkä

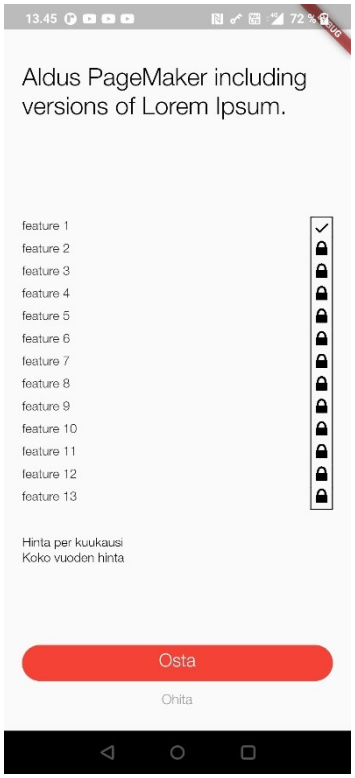
ovat avautuu tilaustuotteen oston jälkeen. Näkymässä näytetään myös tilaustuotteen hinta. Käyttäjä pystyy ostamaan tuotteen ”osto”-painikkeesta, tai hän voi jatkaa ilmaisversiolla painamalla ”ohita”-painiketta.

```
private static final String CHANNEL = "com.Flutter/main";
```

Esimerkkikoodi 14. Kanavan arvo Androidin puolella.

```
const platform = const MethodChannel("com.Flutter/main");
```

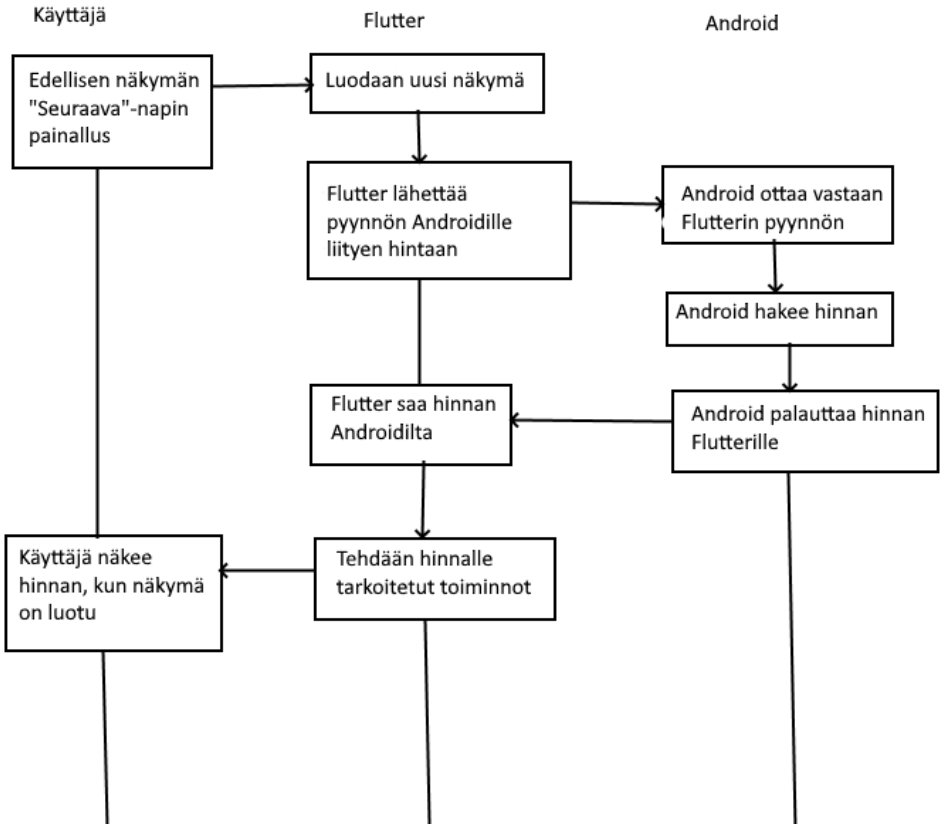
Esimerkkikoodi 15. Kanavan arvo Flutterin puolella.



Kuva 10. Kuudes onboarding-näkymä. 6th View.

Näkymässä näytettävät hinnat haetaan Androidilta käyttäen methodChannelia. Kuvassa 11 näytetään, miten se yksinkertaisesti tapahtuu. Siinä myös nähdään, miten Android ja Flutter kommunikoivat toistensa kanssa. Esimerkkikoodeissa 16 ja 17 sisältää hintojen hakuun ja näyttämiseen käyttäjälle Flutterin puolelta. Lähetetään Androidille pyyntö ja Android palauttaa arvon. Palautetulle arvolle

tehdään tarvittavat toiminnot, jotka tässä tapauksessa on hinnan näyttäminen käyttäjälle.



Kuva 11. Kaavio Flutterin ja Androidin kommunikaatiosta hinnan näytön suhteen.

```
class PriceText extends StatefulWidget {
 @override
 PriceTextState createState() => PriceTextState();
}

class PriceTextState extends State<PriceText> {

 @override
 void initState() {
   super.initState();
   getPrice1();
   getPrice2();
 }

 getPrice1() {
   getPrice1Data().then((String data) {
     if (data == "" || data == null) {
       return;
     }
     debugPrint(data);
     setState(() {
       price1 = data;
     });

   });
 }

 getPrice2() {
   getPrice2Data().then((String data) {
     if (data == "" || data == null) {
       return;
     }
     debugPrint(data);
     setState(() {
       price2 = data;
     });

   });
 }

 @override
 Widget build(BuildContext context) {
   return Container(
     margin: EdgeInsets.only(top: 30),
     child: Column(
       children: [
         Row(
           children: [
             Text(
               AppLocalizations.of(context).sixth_onboard-
ing_view_text2 + " ",
               style: TextStyle(color: Colors.black),
             ),
             Text(
               price1,
               style: TextStyle(color: Colors.black),
             ),
           ],
         ),
         Row(
```

```
            children: [
              Text(
                AppLocalizations.of(context).sixth_onboard-
ing_view_text3 + " ",
                style: TextStyle(color: Colors.black),
              ),
              Text(
                price2,
                style: TextStyle(color: Colors.black),
              ),
            ],
          ),
        ],
      ),
    );
  }
  }
```

Esimerkkikoodi 16. Hintaan liittyviä koodin pätkiä.

```
Future<String> getPrice1Data() async {
 return await platform.invokeMethod('Price1') ?? "";
}
```

Esimerkkikoodi 17. Pyydetään Androidilta hinnan arvoa.

4.4.1 Ostotapahtuman toiminta

Kommunikaatiota käytetään onboarding-näkymien ostotapahtumassa. Esimerkkikoodeissa 18, 19, 20, 21, 22 ja 23 nähdään ostotapahtumaan liittyviä koodin pätkiä Flutterilta ja Androidilta. Esimerkkikoodissa 18 on MethodChannel Androidin puolelta. MethodChannelin toimintaa tullaan tarvitsemaan aikaisemmin luotua ja alustettua FlutterEngineä. Siinä myös käytetään kanavana CHANNEL-arvoa, joka luotiin edellä esitetystä esimerkkikoodista 14.

Esimerkkikoodissa 21 on widgetti ostonapista, josta painamalla käynnistetään koko ostotapahtuma. Flutter käyttää MethodChannelin invokeMethod-metodia, jossa arvona on buyProductA6thView tai buyProductB6thView. Arvoon vaikuttaa ensimmäisessä näkymässä tehty random-numeron generoiminen, josta päätetään missä järjestyksessä näkymät näytetään käyttäjälle. Lisäksi Androidille lähetetään String-arvo ”6th Buy Pressed”. String-arvoa käytetään Androidin puolella Flurryn seurantaan (tästä lisää myöhemmin).

Androidilla MethodChannelin sisällä on if-lausekkeita, jotka ottavat vastaan erilaisia Flutterin invokeMethodeita. Esimerkkikoodissa 19 katsotaan, onko Flutterin invokeMethod arvoltaan buyProductA6thView. Jos näin on, aloitetaan ostotapahtuma Androidin puolelta. Käyttäjälle ilmestyy silloin Googlen ostonäkymä, joka on käynnistetty Androidin koodin puolelta. Samaan aikaan Flutter odottaa vastausta Androidilta, minkä Android sitten lähettää käyttäjän käytyä ostotapahtuman loppuun. Ostotapahtuman lopussa mennään esimerkkikoodiin 20, jossa palautettaan arvo Flutteriin käyttäen success-metodia. Flutter silloin jatkaa toimintaansa riippuen palautetusta arvosta. Jos tuote ostettiin, siirrytään viimeisimpään onboarding-näkymään. Jos käyttäjä keskeytti tai perui ostotapahtuman, pysytään kuudennessa näkymässä. Käyttäjä voi perumis- tai keskeytystapauksessa jatkaa seuraavaan näkymään painamalla kuudennen näkymän ”ohita”-painiketta. Samalla tavalla toimii ostotapahtuma edellä mainitussa viidennessä onboarding-näkymässä.

```
new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessen-
ger(), CHANNEL).setMethodCallHandler(new MethodChannel.MethodCallHan-
dler() {
   @Override
   public void onMethodCall(@NonNull MethodCall call, @NonNull Method-
Channel.Result result) {
       if (call.method.equals("endActivityA")) {
           FlurryUtils.onBoardingChoiceA("No sign up");
           System.out.println("endActivityA sisällä");
           onSkipRegistration();
       }
```

Esimerkkikoodi 18. MethodChannel Androidin puolella.

```
else if (call.method.equals("buyProductA6thView")) {
   String text = call.argument("BuyA");
   FlurryUtils.onBoardingBuyActionsA(text);
   activity.AorBMain = "A2";
   System.out.println("buyProductA sisällä!!!");
   myResult = result;
   buyPressed(1);
}
```

Esimerkkikoodi 19. Ostotapahtumaan liittyvä koodin pätkä Androidin puolelta

```
private void myCustomMethod(boolean a) {
myResult.success(a);
}
```

Esimerkkikoodi 20. Suoritetaan, kun ostotapahtuma lopetetaan Androidin puolelta. Palautetaan Flutteriin arvo succes-metodilla.

```
class ThirdPart1 extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Container(
     height: 40,
     margin: EdgeInsets.only(bottom: 0),
     padding: EdgeInsets.only(left: 20, right: 20),
     child: TextButton(
       style: ButtonStyle(
         backgroundColor: MaterialStateProperty.all(Colors.red),
         shape: MaterialStateProperty.all<RoundedRectangleBorder>(
           RoundedRectangleBorder(
             borderRadius: BorderRadius.circular(20.0)
           ),
         )
       ),
       onPressed: (){
         buyPressed(context);
       },
       child: Text(
         AppLocalizations.of(context).test_teksti_12,
         style: TextStyle(color: Colors.white, fontSize: 20.0),
       ),
     ),
   );
 }
}
```

Esimerkkikoodi 21. Kuudennen näkymän ostotapahtuman painike.

```
buyPressed(BuildContext context) {
 _buyBtnPressed().then((bool data) {
   if (data == null) {
     return;
   }
   debugPrint('$data');
   if (data == true) {
     if (firstView6th == true) {
       setValueSkip5th();
     }
     Navigator.push(
       context,
       MaterialPageRoute(
         builder: (context) {
           return OnBoardingView7();
         }
       )
     );
   } else {
     return;
   }
 });
}
```

Esimerkkikoodi 22. Flutterin buyPressed-funktiossa tehdään toiminnot Androidin palauttaman arvon mukaan.

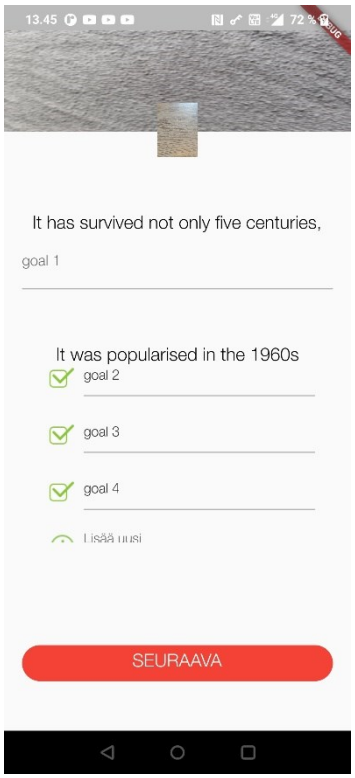
```
Future<bool> _buyBtnPressed() async {
 if (firstView6th == true) {
   return await platform.invokeMethod('buyProductB6thView', {'BuyB':
"6th Buy Pressed"}) ?? "";
 } else {
   return await platform.invokeMethod('buyProductA6thView', {'BuyA':
"6th Buy Pressed"}) ?? "";
 }
}
```

Esimerkkikoodi 23. Ostotapahtumaan liittyvä MethodChannelin invokeMethod-metodi. Tämä koodi on Flutterin puolella.

4.4.2 Kolmannen näkymän toiminta

MethodChannelillä voidaan lähettää dataa Androidille ilman, että Android palauttaisi mitään arvoa takaisin. Kuvassa 12 on kolmas onboarding-näkymä, jossa näkymässä on ylhäällä kaksi kuvaa, keskellä on listView ja alhaalla on ”seuraava”-nappi. listViewiin on listattu tekstikenttiä, joihin käyttäjä voi kirjoittaa tekstiä, ja checkBoxeja. listView on skrollattava ja sen lopussa on nappi, josta

käyttäjä voi lisätä uuden checkBoxin ja tekstikentän. Esimerkkikoodissa 24 lähetetään koko listViewin tekstikenttien arvot Androidille. _goalsListSend suoritetaan, kun käyttäjä on näkymässä painanut ”seuraava”-nappia. Androidin puolella suoritetaan esimerkkikoodi 25. Koodissa talletetaan saadun listan arvot Androidin tietokantaan. Tässä tilanteessa ei ole success-metodia, joka voisi palauttaa mitään arvoa eikä Flutter odota, että Android palauttaisi mitään arvoa.



Kuva 12. Kolmas onboarding-näkymä. 3rd View.

```
Future<void> _goalsListSend() async {
 for(int i = 0; i <= textControllers.length - 2; i++){
   String tempolaryString = textControllers[i].text;
   goalList.insert(i, tempolaryString);
   debugPrint(goalList[i]);
 }
 try {
   final String result = await platform.invokeMethod('goalList',
{'goalsList': goalList, 'titleGoal': titleController.text});
   debugPrint('Result: $result');
 } catch(e) {
   debugPrint("Error: '${e.message}'.");
 }
}
```

Esimerkkikoodi 24. Lähetetään lista Androidille.

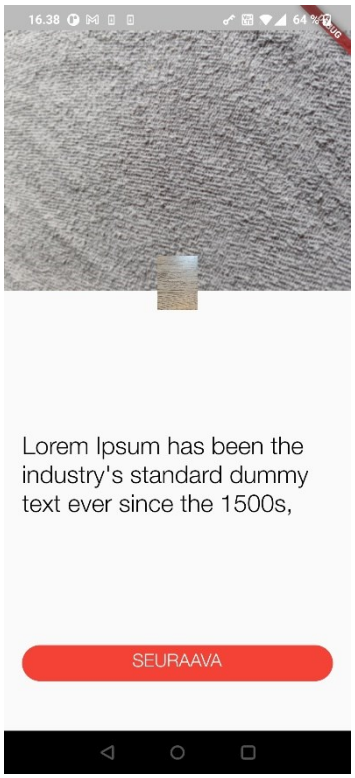
```
else if(call.method.equals("goalList")){
   List<String> goals = call.argument("goalsList");
   String title = call.argument("titleGoal");

   ProfileSettings settings = ProfileSettings.fromContext(getApplica-
tionContext());

   settings.goalsSetMainGoal(title);

   String[] benefits = new String[goals.size()];

   System.out.println(title);
   for(int i = 0; i < goals.size(); i++) {
       System.out.println(goals.get(i));
       benefits[i] = goals.get(i);
   }

   settings.goalsSetBenefits(benefits);

}
```

Esimerkkikoodi 25. Android ottaa listan vastaan ja tekee siihen liittyviä toimintoja.

Aikaisemmin mainitussa neljännessä onboarding-näkymässä lähetetään käyttäjän asettama aika Androidille. Se tehdään samalla tavalla kuin kolmannessa näkymässä, että käyttäjän siirtyessä seuraavaan näkymään lähetetään kellonaika Androidille ilman, että palautetaan mitään arvoa takaisin.

4.5 Siirtyminen Flutter-näkymien välillä ja poistuminen FlutterActivitysta

Kuvassa 13 on onboarding-tapahtuman toinen näkymä. Se on samankaltainen näöltään kuin ensimmäinen näkymä. Keskellä löytyvä teksti on ainoa eroavaisuus näkymien välillä. Esimerkkikoodissa 26 on toisen näkymän onPressed-funktio, joka suoritetaan, kun käyttäjä painaa näkymän ”seuraava”-nappia. Funktiossa suoritetaan uuteen näkymään siirtyminen, joka on varsinaisesti uuden näkymän avaaminen edellisen päälle. Näkymät kasautuvat pinoon.



Kuva 13. Toinen onboarding-näkymä. 2nd View.

```
onPressed: (){
         Navigator.push(
           context,
           MaterialPageRoute(
             builder: (context) {
               return OnBoardingView3();
             }
           )
         );
       },
```

Esimerkkikoodi 26. ”Seuraava”-napin onPressed-funktio.

FlutterActivitysta poistuminen tapahtuu viimeisimmällä onboarding-näkymällä, josta on kuva 14 alhaalla. Näkymässä käyttäjä voi antaa tietonsa sovelluksen yritykselle. Tietojen antaminen on vapaaehtoista. Käyttäjä voi halutessaan ainoastaan laittaa check-merkin käyttöehtojen hyväksymiseen ja painaa ”jatka”-nappia. Jos ei ole check-merkkiä käyttöehtojen hyväksynnässä, tulee alert popup-viesti käyttäjälle. Popup-viestissä kerrotaan hyväksymään käyttöehdot. Alert popup on pieni nelikulmainen näkymä, joka ilmestyy käyttäjälle nykyisen näkymän

päälle. Siinä on yleensä jokin teksti ja ”ok”-nappi. Näkymä poistuu nappia painamalla.

Kuva 14. Seitsemäs ja viimeinen onboarding-näkymä. 7th View.

Toinen tapa päästä eteenpäin käyttäjänä on kirjoittaa tietonsa tekstikenttiin ja laittaa kumpaakin check-laatikkoon check-merkit. Toisessa check-laatikon tekstissä kysytään, haluaako käyttäjä tervetuloaviestin. Viesti lähetetään käyttäjän sähköpostiin, jonka takia käyttäjällä pitää olla vähintään etunimi ja sähköposti kirjattuna. Jos niitä ei kirjattu, tulee siitä alert popup-viesti käyttäjälle. Popupissa kerrotaan, mitkä osat tekstikentistä puuttuu. Kun käyttäjällä on kaikki tiedot kirjattu, check-merkit laitettu ja ”jatka”-painiketta painettu, hänelle tulee vielä esimerkkikoodin 27 mukainen alert popup -viesti, josta popupin nappia painamalla alkaa koko tapahtumasta poistuminen takaisin Android-koodiin. Kaikki näkymät poistetaan pop-funktioilla yksikerrallaan. Esimerkkikoodissa 28 on toisen näkymän pop-funktio. Kaikilla onboarding-näkymien tiedostoilla on samankaltaiset

pop-funktiot. Lopuksi tehdään SystemNavigator.pop(), josta poistutaan kokonaan FlutterActivitysta. Kaikki käyttäjän kirjoittamat tiedot lähetetään Android-koodiin käyttäen methodChannelia.

```
showAlertDialog4(BuildContext context) {
 Widget okButton = TextButton(
   onPressed: (){
     Navigator.of(context).pop();
     pop1(context);
     pop2(context);
     pop3(context);
     pop4(context);
     if (skip5th == false) {
       pop5(context);
     }
     if (skip6th == false) {
       pop6(context);
     }
     pop7(context);
     SystemNavigator.pop();
   },
   child: Text("Ok",
   style: TextStyle(color: Colors.white),
   ),
   style: ButtonStyle(backgroundColor: MaterialStateProperty.all(Col-
ors.red)),
 );

 AlertDialog alert = AlertDialog(
   content: Text(AppLocalizations.of(context).test_teksti_4),
   actions: [
     okButton,
   ],
 );

 showDialog(
   context: context,
   builder: (BuildContext context) {
     return alert;
   }
 );

}
```

Esimerkkikoodi 27. Alert popup -viestin koodi.

```
pop2(BuildContext context) {
 Navigator.pop(context);
}
```

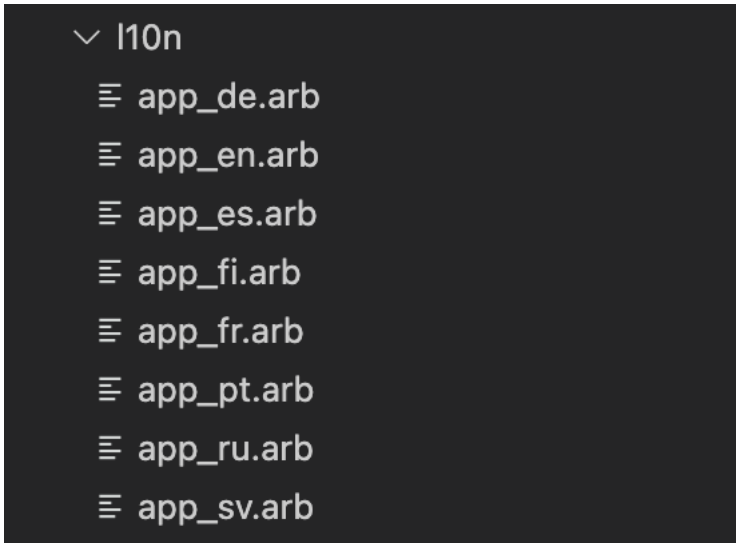
Esimerkkikoodi 28. Toisen näkymän pop-funktio.

4.6 Lokalisaatioiden teko Flutterilla

Tehdään lokalisaatioita Flutter-projektiin. Ensimmäiseksi lisätään uusi tiedosto Flutter-projektin sisään nimellä ”l10n.yaml”. Tiedoston sisään lisätään esimerkkikoodin 29 mukainen pätkä. Flutter käyttää .arb-tiedostoja lokalisaatioon. Jokaisella halutulla kielellä pitää olla oma .arb-tiedostonsa. Kuvassa 15 nähdään kaikki lokalisaatiot, joita onboarding-tapahtuman sovelluksessa käytetään.

```
arb-dir: lib/l10n
template-arb-file: app_en.arb
output-localization-file: app_localizations.dart
```

Esimerkkikoodi 29. Koodin pätkä l10n.yaml-tiedostoon.



Kuva 15. Kansion l10n sisältö.

Lokalisaatioiden tekstit .arb-tiedostojen sisällä on esimerkkikoodien 30 ja 31 muodossa. Esimerkkikoodissa 30 on englannin kielen lokalisaatio ja esimerkkikoodissa 31 suomen kielen lokalisaatio. Lokalisaatiot alustetaan Flutterin main.dart-tiedossa. Sinne lisätään esimerkkikoodin 32 koodirivit riippuen, mille kielille halutaan sovellus kääntää. Onboarding-tapahtumassa ne on lisätty main.dart-tiedoston ”MyApp”-widegetin sisään. Esimerkkikoodissa 33 näytetään

lokalisaatioita käyttöä näkymän tekstin kääntämiseen. Koodissa on ensimmäisen näkymän keskellä sijoitetun tekstin koodi. Varsinaisesti lokalisaatiotiedostoista löytyy kaikilla kielillä kohta ”test_teksti_1”.

```
“helloWorld”:”Hello World!”,
```

Esimerkkikoodi 30. Englanninkielinen lokalisaatio.

```
“helloWorld”:”Hei Maa!”,
```

Esimerkkikoodi 31. Suomenkielinen lokalisaatio.

```
localizationsDelegates: const [
       AppLocalizations.delegate,
       GlobalMaterialLocalizations.delegate,
       GlobalWidgetsLocalizations.delegate,
       GlobalCupertinoLocalizations.delegate,
     ],
     supportedLocales: const [
       Locale('en', ''),
       Locale('fi', ''),
       Locale('sv', ''),
       Locale('de', ''),
       Locale('es', ''),
       Locale('pt', ''),
       Locale('ru', ''),
       Locale('fr', ''),
     ],
```

Esimerkkikoodi 32. Lokalisaatioiden alustus.

```
class SecondPart extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Container(
     padding: EdgeInsets.all(20),
     alignment: Alignment.center,
     child: Text(
       AppLocalizations.of(context).test_teksti_1,
       textDirection: TextDirection.ltr,
       style: TextStyle(color: Colors.purple, fontSize: 27.0),
     ),
   );
 }
}
```

Esimerkkikoodi 33. Ensimmäisen onboarding-näkymän keskellä löytyvän tekstin lokalisaation teko.

4.7 Kuvien ja fonttien lisääminen Flutteriin

Kuvien ja fonttien lisäämisen tekoon voi Flutter-projektissa lisätä kaksi uutta kansiota nimeltä ”images” ja ”fonts”. Tarvittavat kuvat lisätään images-kansioon ja fontit lisätään fonts-kansioon. Seuraavaksi lisätään esimerkkikoodi 34 pubspec.yaml-tiedostoon. pubspec.yaml-tiedoston pitäisi olla Flutter-projektissa valmiina. Kuvat lisätään ”assets”-kohtaan, kuten esimerkkikoodissa on tehty. Fonteilla on oma kohta tiedossa lisäystä varten.

```
assets:
   - images/check_box_selected.png
   - images/check_box.png
   - images/carousel_1_large.png
   - images/carousel_2_large.png
   - images/carousel_3_large.png
   - images/carousel_4_large.png
   - images/green-ball.png
   - images/gray-ball.png
   - images/check.png
   - images/lock.png
   - images/plus_mark.png
   - images/testLautasKuva.jpg
   - images/testKuvaLogo.jpg
   - images/testPieniLautasKuva.jpg

fonts:
   - family: CustomFont
     fonts:
       - asset: fonts/CustonFontName.otf
```

Esimerkkikoodi 34. Kuvat ja fontit pubspec.yaml-tidostossa.

Fontti alustetaan main.dart-tiedostossa. Onbaording-tapahtumassa se tapahtuu ”MyApp”-widgetissä, kuten edellä esitetystä esimerkkikoodista 35 nähdään. Esimerkkikoodissa 35 näytetään pelkän fontin lisäämisen kohtaa onboardingin main.dart-tiedostosta. Kaikki tekstit käyttävät asetettua fonttia automaattisesti. Esimerkkikoodissa 36 on lisätyn kuvan näyttäminen näkymässä. Esimerkkikoodi on osa edellä näytettyä esimerkkikoodia 4.

```
theme: ThemeData(fontFamily: 'CustomFont'),
```

Esimerkkikoodi 35. Fontin asetus Flutter-projektiin main.dart-tiedostossa.

```
Image.asset("images/testKuvaLogo.jpg"),
```

Esimerkkikoodi 36. Lisätyn kuvan käyttäminen.

4.8 Flurry-seurannan teko Onboarding-tapahtumaan

Viimeisempänä tehdään Flurry-seuranta Onboarding-tapahtumaan. Flurry on mobiilisovellusten analytiikka-alusta [13]. Flurry-seurannalla nähdään, kuinka moni on nähnyt tietyn näkymän tai kuinka moni on painanut tiettyä nappia. Onboarding-tapahtumassa pystytään seuraamaan, kumpi reiteistä on parempi. Varsinaisesti kummasta reitistä enemmän käyttäjät painavat ostonappia.

Androidin puolella on jo olemassa Flurry-seuranta. Ajatuksena on käyttää Androidin puoleista Flurrya. Kaikki Flurry-seurantaan liittyvät tiedot lähetetään Androidin puolelle methodChannelia käyttäen. Esimerkkikoodissa 37 on ensimmäisen näkymän yhden reitin Flurry-tiedon lähetys Androidille. Androidin puolella tehdään varsinaiset Flurry-kirjaamiset Flutterista saaduilla tiedoilla.

```
Future<void> flurryCheckA1() async {
 try{
   final String result = await platform.invokeMethod('FlurryStepA',
{'Step': "1st onBoardingView"});
   debugPrint('Result: $result');
 } on PlatformException catch (e) {
   debugPrint("Error: '${e.message}'.");
 }
}
```

Esimerkkikoodi 37. Lähetetään Flurry-tietoja Androidille.

5 Flutterin käytön arviointia

Seuraavaksi tarkastellaan havaittuja hyviä ja huonoja puolia Flutterin käytöstä jo olemassa olevaan iOS/Android-projekteihin. Suraavana löytyy kaksi listaa niistä. Parista kohtaa on kirjoitettu tarkemmin myös alaotsikoissa.

Hyviä puolia:

- Jos sovellus on tarkoitus toimia kummallakin Androidilla ja iOS:ällä, ei tarvitse tehdä näkymiä kahteen kertaan, kun kummatkin voi käyttää Flutterin näkymiä.

- Myös näkymät tulevat olemaan samanlaisia, kun ne kummatkin käyttävät Flutteria. Ei tule erilaisuuksia iOS ja Android sovellusten välillä.

- Google tukee Flutteria. Tosin sanoen sillä on jatkuva pitkäaikainen tuki. [14.]

- Dart-koodikieli on yksinkertainen, mutta tehokas ja täydellinen kieli, joka on verrattavissa Swiftiin ja Javaan [15].

Huonoja puolia:

- Sovelluksen koko kasvoi huimasti Flutterin käyttöönoton myötä. Ainakin Androidin kohdalla halutaan mahdollisimman pieniä sovellus kokoja.

- Ajan käyttö perehtymiseen. Flutter käyttää dart-koodikieltä. Se ei ole varmaan kauhean tuttua koodikieltä. Aikaa tulee kulumaan sen oppimiseen ja käyttöönottamiseen.

- Näkymien asetteluvirheet (”layout errors”) voivat olla todella hämmentäviä ja epäselviä. Natiivissa koodissa iOS:llä ja Androidilla nämä ovat yleensä selvemmin ymmärrettäviä [15].

- Näkymissä olevat ongelmat tulevat vaikuttamaan kumpaakin sovellukseen, jos on kummatkin iOS ja Android versiot sovelluksesta. Jos se ei käyttäisi Flutteria vaan natiivista koodia, iOS ongelmat vaikuttavat ainoastaan iOS-sovellukseen ja Android-ongelmat ainoastaan Android-sovellukseen.

5.1 Sovelluksen koko

Flutter kasvattaa sovelluksen kokoa suuresti. Androidin kohdalla löytyi keino saada sitä hieman pienemmäksi: sovelluksen rakentaminen App bundlena Apk:n sijaan. App bundle pienentää sovelluksen kokoa, mikä sitten voi auttaa korjaamaan Flutterista aiheutuneen sovelluksen koon kasvamisen. Varsinaisesti tämä yksi huono puoli voidaan estää Androidin kohdalla.

5.2 Responsiivisuus

Flutterilla voidaan rakentaa reponsiivisia näkymiä. Miksi tätä ei ole lisätty hyvään puoleen? Natiivilla koodilla iOS:llä ja Androidilla voidaan myös rakentaa responsiivisia näkymiä. Flutterin kohdalla näkymä tarvitsee vaan rakentaa kerran, ja se toimii kummallakin.

5.3 Kannattavuus

Jos olemassa olevasta sovelluksesta ei ole Android- ja iOS-versioita, ei kannata Flutteria käyttöönottaa ollenkaan. Natiivilla koodilla on silloin parasta tehdä. Poikkeuksena voisi olla juuri aloitettu tai aloitettavissa oleva uusi sovellus iOS:lle tai Androidille. Uuden sovelluksen kohdalla tosin tulevaisuutta miettien, että tuleeko sovellus tulevaisuudessa olemaan kummallakin iOS:llä ja Anroidilla. Tosin sanoen se on käyttöönotettava vain, jos on tai halutaan sovelluksesta versio iOS:lle ja Androidille.

Flutter-perehtymiseen tulee menemään aikaa ja varsinkin alkuvaiheessa Flutterilla kehitys tulee olemaan hidasta. Kuinka paljon aikaa halutaan käyttää tai kuinka paljon aikaa on varsinaisesti mahdollista käyttää Flutterin käyttöönottoon, ovat kysymyksiä, joihin on hyvä löytää vastaus ennen kuin lähdetään tekemään Flutterin käyttöönottoa.

Flutterin käyttöönoton kannattavuudessa täytyy ottaa huomioon, mitä osioita tai tulevia osioita tehdään Flutterilla. Varsinaisesti ei ole kannattavaa lähteä kokonaan muuttumaan olemassa olevia sovelluksia Flutteriin. Flutterilla voi tehdä tulevat osiot ja uudet näkymät, jolloin ne voidaan tehdä vaan kerran. Varsinaisesti vanhojen näkymien muuttaminen Flutter-näkymiin on suurimalta osin ajan haaskausta. Poikkeuksena voivat olla näkymät, jotka ovat ongelmallisia, ja muuttaminen Flutter-näkymää korjaisi ongelman.

Flutter kannattaa ottaa käyttöön olemassa olevissa sovelluksissa, jos on suunnitteilla uusia näkymiä, jotka vaikuttavat iOS- ja Android-sovelluksiin. Vaikka aikaa kuluu perehtymiseen ja käyttöönottamiseen, tulee säästymään aikaa suunniteltujen näkymien teossa, kun ne tehdään vaan kerran.

5.4 Mielipide Flutterista ja sillä työskentelystä

Alussa Flutterin asennuksessa ja kiinnityksessä kummallekin iOS- ja Android-projekteihin oli hankalaa. Siinä taisi kestää n. viikon verran, kunnes sain Flutterin esimerkki sivun näkyviin kummallakin. Androidin kohdalla se bash-ongelma selvisi vaan siten, että satuin tekemään komennon Visual Studion -komentoikkunassa. Koska Flutter-asennuksessa kesti noin kauan, minulle annettiin tehtäväksi tehdä iOS Onboarding -näkymät natiivilla tavalla. Ei Flutteria otettu käyttöön iOS:n onboarding-tapahtuman rakentamiseen. Siinä katosi varsinaisesti yksi syistä Flutterin käyttöönottoon.

Sen teon jälkeen aloin tekemään sitä Onboardingia Androidille Flutterilla. Aluksi se oli hyvin hidasta, kun varsinaisesti opettelin sen käyttöä samalla, kun tein niitä näkymiä. Jos tarvitsi tietoa jostakin asiasta tai korjata ongelman, löytyi se suurimmalta osin helposti Googlea käyttämällä. Flutterilla näkymän teko on aika helppoa mielestäni. Tosin uskon iOS:n natiivilla tavalla näkymän tekeminen on helpompaa. Siihen varmaan vaikuttaa kokemukseni iOS-näkymien parissa. Android-näkymien teosta natiivilla tavalla minulla ei ole paljon kokemusta.

Loppujen lopuksi Flutter on loistava työkalu näkymien tekoon. Siinä vaan kuluu aikaa käyttöönottaa se ja opetella sen käyttämistä. Jos on näkymiä, joita tiedetään tulevan kumpaakin iOS- ja Android-versioon sovelluksesta, voidaan ne rakentaa Flutterin avulla. Pitää myös olla melkein sata prosenttisesti varma, että ne rakennetaan Flutterin avulla. Muuten Flutterista ei varsinaisesti ole mitään hyötyä.

6 Yhteenveto

Päätettiin ottaa Flutter-työkaluksi onboarding-näkymien tekoon. Asennettiin ja integroitiin Flutter olemassa olevaan sovellukseen. Lähdettiin rakentamaan Flutterilla onboarding-tapahtumaa sovelluksen Android-versiolle.

Tuloksena syntyi sovellukselle Onboarding-tapahtuma, jossa on 7 eri näkymää. Näkymät on lokalisoitu eri kielille eikä niiden toiminnassa ole ongelmia. Flutterin onboarding-tapahtuma oli hetken aikaa käyttäjillä käytössä, mutta loppujen lopuksi se päätettiin ottaa pois käytöstä. Suurimpana syynä oli sovelluksen koon kasvaminen. Sovelluksen koko halutaan pitää mahdollisimman pienenä.

Suunnitelmissa on ottaa Flutterin onboarding-tapahtuma käyttöön uudelleen. Sovelluksen kokoa saatiin pienennettyä siivoamalla turhia osia pois ja muuttamalla kaikki sovelluksen kuvat .webp-muotoon.

Lähteet

1 Flutter kotisivu. Verkkoaineisto. <https://flutter.dev> Luettu 31.10.2021.

2 Flutter (software). Verkkoaineisto. https://en.wikipedia.org/wiki/Flutter_(software) Luettu 31.10.2021.

3 Dart overview. Verkkoaineisto. https://dart.dev/overview Luettu 31.10.2021.

4 React Native. Verkkoaineisto. https://reactnative.dev/ Luettu 16.1.2022.

5 Visual Studio. Verkkoaineisto. https://www.incredibuild.com/integrations/visual-studio Luettu 31.10.2021.

6 Xcode. Verkkoaineisto. https://appleinsider.com/inside/xcode Luettu 31.10.2021.

7 Android Studio. Verkkoaineisto. https://developer.android.com/studio Luettu 31.10.2021.

8 Swift. Verkkoaineisto. https://developer.apple.com/swift/ Luettu 31.10.2021.

9 What is Java? Definition, Meaning & Features of Java Platforms. Verkkoaineisto. https://www.guru99.com/java-platform.html Luettu 31.10.2021.

10 Intergrate a Flutter module into your iOS project. Verkkoaineisto. https://flutter.dev/docs/development/add-to-app/ios/project-setup Luettu 14.10.2021.

11 Intergrate a Flutter module into your Android project. Verkkoaineisto. https://flutter.dev/docs/development/add-to-app/android/project-setup Luettu 14.10.2021.

12 Flutter: Stateful vs Stateless Widget. Verkkoaineisto. https://medium.com/flutter-community/flutter-stateful-vs-stateless-db325309deae Luettu 29.10.2021.

13 Flurry kotisivu. Verkkoaineisto. https://www.flurry.com Luettu 24.4.2022

14 The Good and the Bad of Flutter App Development. Verkkoaineisto. https://www.altexsoft.com/blog/engineering/pros-and-cons-of-flutter-app-development/ Luettu 29.10.2021.

15 Flutter: the good, the bad and the ugly. Verkkoaineisto. https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9 Luettu 29.10.2021.

Ei liitteitä.