Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Miska Peltoniemi

Android-sovelluksen datapohjainen kehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

25.1.2021

Tekijä Otsikko Sivumäärä Aika	Miska Peltoniemi Android-sovelluksen datapohjainen kehitys 45 sivua 25.1.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Jorma Räty

Insinöörityössä otetaan käyttöön Android-sovelluksissa menetelmä, joka mahdollistaa sovellusten käyttöä koskevien kuluttajatietojen keräämisen.

Tieto toimii nykymaailmassa tärkeässä roolissa ohjelmistokehityksessä ja antaa hyvää palautetta siitä, miten sovelluksia pitäisi kehittää. Yrityksellä ei ole tällä hetkellä mitään tapaa kerätä käyttäjädataa sovelluksista.

Android-sovelluksia on kaksi, ja ne toimivat kuvallisina ruokapäiväkirjoina. Sovellukset käyttävät samaa koodipohjaa, mutta ne eroavat toisistaan ominaisuuksiltaan. Työssä esitellään sovellusten kehityksessä käytetyt teknologiat ja arkkitehtuurikomponentit, joita sovellusten käyttöliittymänäkymä ja tietokanta hyödyntävät toiminnassaan.

Työn päätavoitteena on Flurry Analytics SDK -kirjaston integroiminen Android-sovelluksiin, jotta sovelluksista voitaisiin kerätä käyttäjädataa. Flurry Analytics on ilmainen analytiikkatyökalu, joka on suunniteltu keräämään sovelluksista tietoja, joista se palauttaa parhaat mahdolliset raportointitulokset. Kirjaston integrointi ja sen tiedon keräämistä koskevat ominaisuudet käydään työn aikana läpi perusteellisesti.

Tulokseksi kerättiin runsaasti dataa, joka analysointiin Flurryn nettipalvelimen kautta. Tulokset antoivat hyvän kuvan siitä, miten käyttäjät käyttävät sovelluksia, ja antoivat ideoita sovellusten jatkokehitykselle.

Avainsanat	Android, Data, Flurry Analytics

Author	Miska Peltoniemi
Title	Data-based development of an Android Application
Number of Pages	45 pages
Date	25 January 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Jorma Räty, Senior Lecturer

The thesis introduces an Android application method of collecting data on consumer usage.

In today's world, information on applications plays an important role in software development and provides useful feedback on future development. At the starting point the commissioner of the study company had no means of collecting information on users.

The company had two Android applications that act as pictorial food diaries. The applications use the same code base but differ in their features. This thesis presents the technologies used in the development of the applications and the architectural components utilized in their operation.

The main goal of the study was to integrate the Flurry Analytics SDK library into Android applications to collect user data. Flurry Analytics is a free analytics tool designed to collect user data and return the best possible reporting results. The integration of the library and its features for collecting data are thoroughly reviewed.

As a result of the study, a large amount of data was collected from the applications and analyzed through Flurry's web server. The results provided a valuable overview of how the users in question use the applications and ideas for further development.

Keywords	Android, Data, Flurry Analytics

Sisällys

Lyhenteet

ORM	Object-relational mapping. Oliomallin mukaisen esityksen kuvaus relaatiomallin mukaiseksi esitykseksi.
APK	Android package. Android-sovellusohjelmien pakettitiedosto, joka julkaistaan ja asennetaan laitteeseen.
SDK	Software Development Kit. Ohjelmistokehityspaketti, joka on kokoelma ohjelmistokehityksen työkaluja yhdessä asennettavassa paketissa.
DAO	Data Access Object. Objekti, joka tarjoaa abstraktin käyttöliittymän tietyntyyppiselle tietokannelle, jotta se pääsisi siihen käsiksi.
API	Application Programming Interface. Ohjelmointirajapinta, jota käyttäen sovellukset kommunikoivat keskenään.
AVD	Android Virtual Device. Android-virtuaalilaite, joka määrittelee Android-emulaattorissa simuloitavan puhelimen ominaisuudet.

1 Johdanto

Opinnäytetyössä rakennetaan Android-sovelluksiin menetelmä, joka mahdollistaa sovelluksen käyttöä koskevien kuluttajatietojen keräämisen.

Nykypäivän digitaalisessa maailmassa tietoja luodaan, arvioidaan ja päivitetään jatkuvasti. Tieto toimii ohjelmistokehityksessä tärkeässä roolissa, sillä se tarjoaa tarkkaa ja hakukelpoista palautetta, joka auttaa kehittäjää ymmärtämään, miten parantaa tuotettaan. Tuotteen kehityksessä on hyvä ymmärtää perusteellisesti asiakkaiden tavat ja tarpeet, jotta tuote vastaisi juuri sitä, mitä asiakkaat haluavat käyttää. Sen vuoksi yritys näkee datapohjaisen kehityksen tärkeänä mallina sellaiselle sovellusten jatkokehittämiselle, jossa tieto itsessään ohjaa sovellusten kehitystä.

Opinnäytetyön tilaajana toimii yritys, joka toivoo pysyvänsä tuntemattomana. Tällä hetkellä yrityksellä ei ole kovinkaan tarkkaa käsitystä siitä, miten sen asiakkaat käyttävät sovelluksia. Sovellukset eivät käytä Backend-palvelimia tietojen tallentamiseen, joten tietoihin ei pääse mitenkään käsiksi, sillä ne ovat tallessa ainoastaan käyttäjien puhelimissa. Sovellukset ovat pitkälle kehitettyjä, mutta jatkokehityksen kannalta tarvitaan enemmän tietoja käyttäjistä ja heidän tarpeistaan, jotta sovellusten kehitys jatkossa kulkisi oikeaan ja parempaan suuntaan.

Työn tarkoituksena on ottaa Android-sovelluksissa käyttöön Flurry Analytics SDK -kirjasto, joka toimii analytiikkatyökaluna tietojen keräämistä ja niiden analysoimista varten. Kirjasto tarjoaa paljon erilaisia ominaisuuksia tiedon keräämiseen. Tiedot raportoidaan Flurryn palvelimelle analysoitavaksi, ja niistä kerättyjen tulosten avulla yritys pystyy vastaamaan tarpeellisiin kysymyksiin ja käyttämään vastauksia hyvänä mallina sovellusten jatkokehittämiselle.

2 Android-sovellus

Insinöörityössä kehitettiin kahta mobiilisovellusta, jotka ovat Android-käyttöjärjestelmällä toimivia kuvallisia ruokapäiväkirjoja. Sovellukset käyttävät samaa koodipohjaa, mutta ne eroavat toisistaan ominaisuuksiltaan. Sovellukset ovat ladattavissa ilmaiseksi Google Play -kaupasta.

Sovellusten päätarkoituksena on pitää käyttäjä tietoisena syömistavoistaan ja toimia apuna hyvän ruokarytmin noudattamisessa. Ruokapäiväkirjan avulla käyttäjä pystyy näkemään yhdellä silmäyksellä kaikki ateriat, jotka hän on tallentanut sovellukseen. Käyttäjällä on sovelluksissa käytössään jokaiselle päivälle erilaisia ruokaruutuja, joihin hän pystyy lisäämään syömistään aterioista kuvan ja tekemään muistiinpanoja. Ruokapäiväkirjan lisäksi sovelluksesta löytyy myös paljon erilaisia lisäominaisuuksia, kuten päivittäinen vedenjuonnin seuranta ja päivittäisiä vinkkejä terveellisen ruokavalion saavuttamiseen.

Sovellusten ateriamuistutukset auttavat käyttäjää syömään säännöllisesti ja pysymään energisenä koko päivän ajan. Sovellus lähettää tietyin aikavälein käyttäjälleen ilmoituksia, joissa häntä muistutetaan syömisestä ja pyydetään tallentamaan kuva ateriastaan sovellukseen. Ilmoitukset ovat sovelluksessa helposti muokattavissa tai poistettavissa, ja uusia ilmoituksia voi luoda itse lisää.

Sovellus on tällä hetkellä ilmaiseksi käytettävissä, mutta ilmainen versio sisältää tiettyjä rajoituksia. Ilmaisversio rajoittaa useiden ominaisuuksien käyttöä ja sisältää pienemmän historian ja runsaasti mainoksia. Maksavat asiakkaat saavat käyttöönsä Premium-version, joka avaa käyttäjälle kaikki sovellukseen kuuluvat ominaisuudet, kuten laajemman ruokahistorian. Käyttäjä voi ostaa Premium-version kerralla koko vuodeksi tai maksaa kuukausimaksuja.

3 Teknologiat

Tässä luvussa käsitellään Android-sovellusten kehityksessä käytettyä kehitysympäristöä Android Studiota ja sen rakennustyökalua Gradlea. Luvussa tarkastellaan myös Google Play -sovelluskauppaa, joka on vastuussa sovellusten julkaisusta ja hallinnasta. Työn tärkeintä työkalua Flurry Analyticsia käsitellään omassa luvussaan.

3.1 Android Studio

Android Studiota käytetään virallisena kehitysympäristönä Android-käyttöjärjestelmille. Se perustuu IntelliJ IDEA:aan, Java-integroituun ohjelmistokehitysympäristöön, joka toimii koodinmuokkaus- ja kehittämistyökaluna. Ensimmäinen versio Android Studiosta julkaistiin vuonna 2014, ja se korvasi Eclipse Android-kehittämistyökalut. [1.]

Android Studio käyttää Gradle-pohjaista rakennustyökalua. Gradle-rakennusjärjestelmä vastaa koodin kokoamisesta, testaamisesta, käyttöönottamisesta, muuntamisesta ja sovelluksen suorittamisesta laitteilla. Gradlen avulla pystytään automatisoimaan ja hallitsemaan sovelluksen rakennusprosessia. Android Studion mukana tulee esiasennettu Gradle-järjestelmä projektin rakentamista varten, joten kehittäjän ei tarvitse asentaa erillisiä ajonaikaisia ohjelmistoja. [2.] Kun sovellus suoritetaan joko emulaattorilla tai kytketyssä laitteessa, tämä laukaisee automaattisesti projektin ja alkaa rakentaa sitä yhdeksi paketiksi. Näitä paketteja kutsutaan APK-tiedostoiksi.

Android-ohjelmoinnissa tärkeänä osana voidaan pitää siistin ja käyttäjäystävällisen käyttöliittymän luomista. Android Studiossa on mukana suunnittelutyökalu (layout editor), jonka avulla pystytään suunnittelemaan käyttöliittymien asetteluja nopeasti vetämällä käyttöliittymäelementit visuaalisessa editorissa sen sijaan, että käyttöliittymäelementit kirjoitettaisiin XML-tiedostossa käsin. Näitä käyttöliittymäelementtejä voivat olla mm. tekstikentät, kuvaikkunat, listat/luettelot tai napit. Suunnittelutyökalussa voidaan esikatsella aseteltua kokonaisuutta eri Android-laitteilla ja muuttaa dynaamisesti koon asettelua sen varmistamiseksi, että se toimii hyvin erikokokoisilla näytöillä.

Sovellusta voidaan pyörittää Android Studiossa joko käyttämällä tietokoneeseen kytkettyä laitetta tai kehitysympäristöön sisäänrakennettua Android-emulaattoria. Emulaattorin avulla pystytään simuloimaan erilaisia Android-laitteita ilman laitteen fyysistä versiota. Emulaattori tarjoaa lähes kaikki oikean Android-laitteen ominaisuudet ja tekee täyden kopion simuloitavan laitteen ulkoasusta ja käyttöliittymästä. Sovelluksen testaaminen on tietyllä tapaa nopeampaa ja helpompaa emulaattorin avulla. Esimerkiksi datan siirto tapahtuu nopeammin emulaattorin kuin tietokoneeseen kytketyn laitteen avulla. Android Studiolla pystytään luomaan virtuaalisia laitteita Android Virtual Device (AVD) Managerin kautta. AVD-laite sisältää laitteistoprofiilin, järjestelmäkuvan, tallennusalueen, ulkoasun ja kaikki muut tärkeät ominaisuudet Android-laitteen kopioimista varten. [3.] Onkin aina hyvä luoda erilaisia profiileita emuloitaville laitteille, joissa sovellusta halutaan testata.

3.2 APK ja Build

Android Package Kit (APK) on pakettitiedostomuoto, jota Android-käyttöjärjestelmät käyttävät mobiilisovellusten jakamiseen ja asentamiseen. Kun sovellusta halutaan testata tai se halutaan jakaa, Gradle luo Android Studiossa APK-tiedoston ja pakkaa kaikki Android-ohjelman osat yhdeksi sisältöpaketiksi. APK-tiedostot sisältävät kaiken ohjelman koodin, kuten dex-tiedostot, resurssit, varmenteet ja manifestitiedoston. [4.]

Uusissa jakelukelpoisissa APK-tiedostoissa on otettava joitakin asioita huomioon. Kriittinen osa sovelluksen päivitys- ja ylläpitostrategiaa on versiointi. Se auttaa Android-järjestelmiä suojautumaan sovellusten palaamiselta vanhempiin versioihin. Se antaa käyttäjilleen myös tarvittavan informaation sovelluksista heidän Android-laitteillaan. Jotta sovelluksen julkaiseminen palveluissa olisi mahdollista, on kehittäjän nostettava sovelluksen versionumeroa korkeammaksi. Jos versionumero on alempi tai yhtä suuri kuin nykyinen versio, ei sovelluspaketin tarjoaja tunnista sitä päivitykseksi ja sovellus on mahdollista päivittää vasta sen poistamisen jälkeen.

Jokaisella sovelluksella on oltava AndroidManifest.xml-tiedosto juurihakemistossaan. Tiedostossa esitellään olennaiset tiedot Android-rakennustyökalulle, Android-käyttöjärjestelmälle ja sovelluspaketin tarjoavalle. Näitä tietoja ovat paketin nimi, joka

toimii sovelluksen yksilöivänä tunnisteena, sovelluksen komponentit ja kaikki käyttöoikeudet, joita sovellukselta vaaditaan vuorovaikutukseen sen ulkopuolella olevien komponenttien kanssa. Manifestista on mahdollista myös määritellä Android API-version vähimmäistaso, jota sovellus vaatii pyöriäkseen.

Gradle-rakennustyökalupaketin avulla pystytään määrittelemään joustavat mukautetut rakennuskokoonpanot sovellukselle. Gradle Android -laajennus toimii rakennustyöpaketin kanssa ja tarjoaa erityisiä prosesseja ja konfiguroitavia asetuksia, joita tarvitaan sovellusten testaamisessa ja niiden luomisessa. Android-rakennusjärjestelmän joustavuuden ansiosta kehittäjän on mahdollista suorittaa mukautettuja rakennusmäärityksiä muutamatta sovelluksen ytimen lähdetiedostoja. Määritellyillä rakennuskokoonpanoilla voidaan hallita rakennustyyppejä, tuotemakuja (product flavours) ja rakennevariantteja. [5.]

3.2.1 Build Type

Rakennustyypit määrittelevät ominaisuudet, joita Gradle käyttää sovelluksen rakentamisessa ja pakkaamisessa. Kun uusi moduuli luodaan, luo Android Studio automaattisesti testaus (debug)- ja julkaisu (release) -rakennustyypit sovelluksesta. Testaus-rakennustyyppi mahdollistaa virheenkorjausvaihtoehdot kirjoittamalla APK:n vianetsinnälle, kun taas julkaisu-rakennustyyppi allekirjoittaa APK:n avaimen jakelua varten. [6.] Jotta sovellus voitaisiin rakentaa, on valittava ainakin toinen näistä rakennustyypeistä.

Debug-rakennustyyppi on olennainen osa virheiden etsimiseksi ja niiden korjaamiseksi. Tämä rakennustyyppi takaa sen, että Android Studiossa ovat päällä kaikki virheiden jäljittämiseen tarvittavat työkalut, kuten konsoli, logcat ja sovelluksen profilointiin liittyvät ominaisuudet. Release-rakennustyyppi on se sovelluksen rakennustyyppi, jolla sovellus pakataan julkaistavaksi versioksi sovelluskauppaan. Release-rakennustyypiltä hävitetään kaikki ominaisuudet sovelluksen virheiden jäljittämiseksi.

3.2.2 Product Flavour

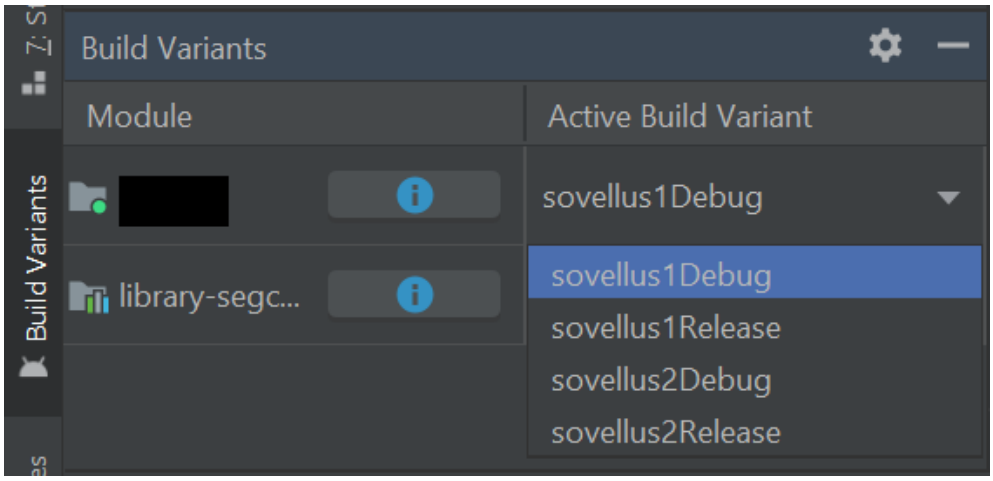
Tuotemaku on variantti, joka edustaa sovelluksessa erilaisia versioita. Se on tehokas ominaisuus Android Studion Gradle-laajennuksessa. Sen avulla on mahdollista luoda räätälöityjä versioita tuotteista, joiden lähdekoodi on yhteinen sovelluksen kaikille versioille. Tuotemakujen avulla pystytään määrittämään erilaisia ominaisuuksia ja laitevaatimuksia, kuten mukautettua lähdekoodia, resursseja ja API-tasoja. [6.] Tuotemaut ovat sovelluksessa valinnaisia, joten kehittäjän on luotava ne itse.

Tuotemaut mahdollistavat useamman erilaisen sovellusversion luomisen yhdellä koodipohjalla. Tämä tarkoittaa sitä, että eri versiolla vältytään erillisen lähdekoodin luomiselta. Eri versiot voivat olla ilmaisia tai maksullisia versioita sovelluksesta tai niistä voidaan rajata tietynlaisia ominaisuuksia ehtojen täyttyessä. Tuotemaut pitävät koodin siistinä ja tekevät siitä helpomman ja nopeamman ylläpitää, sillä kaikki tuotemakuihin liittyvät asiat säilytetään samoissa tiedostoissa.

3.2.3 Build Variant

Build Variant on rakennusvaihtoehto, joka on rakennustyypin ja tuotemaun yhdistelmä ja jota käytetään sovelluksen luomisessa. Rakennusvaihtoehtoa käyttämällä voidaan käsitellä useampaa sovellusta yhdessä projektissa. Rakennusvaihtoehtoja ei määritellä suoraan, vaan ne muodostuvat rakennustyypin ja tuotemaun kautta. Android Studio luo tiettyä rakennusvaihtoehtoa vastaavan APK:n sovellusta rakentaessaan. Rakennusvaihtoehtoja käyttämällä voidaan sovelluksesta rakentaa virheitä korjaava tai julkaistava versio, joka noudattaa tiettyä tuotemakua. Rakennusvaihtoehtoja voidaan luoda joko määrittelemällä uusia tuotemakuja tai rakennustyyppejä. [6.]

Kuvassa 1 on sovelluksesta kaksi eri versiota ”sovellus1” ja ”sovellus2” erilaisina rakennustyyppeinä jakelua ja testaamista varten. Kuten aiemmin todettiin, molemmat sovellukset noudattavat samaa koodipohjaa, mutta niillä on erilaisia ominaisuuksia, joten rakennusvaihtoehtoja käyttäen sovellukset voidaan erotella toisistaan. Ennen APK-tiedoston rakentamista on rakennusvaihtoehdoista valittava toinen, minkä jälkeen Android Studio pakkaa sen ja asentaa sen laitteeseen.



Kuva 1. Rakennusvaihtoehdot, esimerkki kahdesta sovelluksesta

3.3 Google Play

Google Play on markkinoiden johtavin sovellusten latauspaikka Android-alustoilla. Se toimii ”Google-sertifioiduissa” Android-käyttöjärjestelmässä toimivien laitteiden virallisena sovelluskauppana, jossa käyttäjät voivat selata ja ladata sovelluksia, jotka on kehitetty Android-ohjelmistokehityspaketin (SDK) avulla ja julkaistu Googlen kautta. Google Play tarjoaa sekä ilmaisia että maksullisia sovelluksia ja toimii digitaalisena mediakauppana musiikille, kirjoille, elokuville ja televisiosarjoille. Google Play julkaistiin 6. maalikuuta 2012, ja se yhdisti Android-markkinoinnin, Googlen musiikin ja Googlen e-kirjat yhdeksi tuotemerkiksi. [7.]

Google Play Developer Console on sovelluskehittäjille rakennettu kehitysalusta. Se toimii työkaluna Android-sovellusten julkaisemista, seurantaa ja analysointia varten. Alusta tarjoaa paljon erilaisia sovellustietoja, joiden avulla voidaan valvoa sovellusten asennuksia, päivityksiä, ostoja, kaatumisia ja arvosteluja. Kehittäjät pystyvät myös kommunikoimaan alustan kautta ja vastaamaan käyttäjien jättämiin Google Play -arvosteluihin.

Sovellus julkaistaan Google Playihin joko lataamalla APK-tiedosto tai Bundle sovelluksesta kehittäjäkonsoliin. Latauksen jälkeen kehittäjäkonsoli tarkistaa ladatun paketin virheiden varalta. Virhe voi olla esimerkiksi versiointi, jolloin kehittäjän on nostettava sovelluksen versionumeroa korkeammaksi edelliseen julkaisuun verrattuna. Uuden julkaisun yhteydessä kirjoitetaan myös sovellukseen tehdyt päivitykset ja muutokset lyhyenä tekstinä vaihtoehtoisesti useammalla eri kielellä. Teksti antaa käyttäjille tiedot uusista päivityksistä Google Playn kautta.

Sovellus voidaan julkaista joko avoimesti kaikille käyttäjille tai suljetusti määritellyille käyttäjäryhmille. Suljettua julkaisua voidaan kutsua myös sisäiseksi testiversioksi, ja se voi olla alfa- tai betaversion muodossa. Nämä versiot ovat avoimia vain niille käyttäjille, jotka ovat saaneet erillisen kutsun ja jotka voivat ladata kyseiset versiot Google-tunnuksillaan.

Kehittäjäkonsoli toimii alustana sovellusten ostotapahtumille ja niiden määritellylle. Jotta sovelluksen sisäiset ostot olisivat mahdollisia Google Playn kautta, on tuotteet määriteltävä etukäteen kehittäjäkonsolilla. Google Playn laskutuksessa voidaan myydä joko kertakäyttöisiä tuotteita (one-time products) tai tilauksia (subscriptions). Kertakäyttöisellä tuotteella tarkoitetaan tuotetta, joka laskutetaan käyttäjältä vain kerran, eli se on hallittu tuote. Tilaustuotteet ovat taas tuotteita, jotka laskutetaan käyttäjältä toistuvasti. Tuotteiden ostotapahtumia käsitellään sovelluksessa joko käyttämällä Google Play Billing- tai vanhempaa In App Billing -kirjastoa. Uudet tuotteet integroidaan sovellukseen niiden ID-arvojen avulla, joilla tuotteeseen sisältyvät tiedot löydetään.

Yksi tärkeimmistä seikoista konsolin käytössä kehittäjän kannalta on sovelluksen kaatumisten yhteydessä luotujen virheraporttien seuranta. Jos sovelluksessa käytetään ProGuardia APK-tiedostojen optimointiin, voidaan Play Consoleen lisätä sovelluksen julkaisun yhteydessä ProGuard-karttatiedosto. Se tehostaa sovelluksen virheiden ja kaatumisten seuraamista. Virheraporteista nähdään kaikki tarvittavat tiedot kaatumisten syistä ja niiden korjaamisesta. Tällaisia tietoja ovat koodissa tapahtuva virhe/poikkeus, puhelinmalli, sovellus- ja Android-versio sekä kaikki muut oleelliset tiedot, joista on apua virheitä tunnistettaessa.

4 Sovelluksen arkkitehtuuri

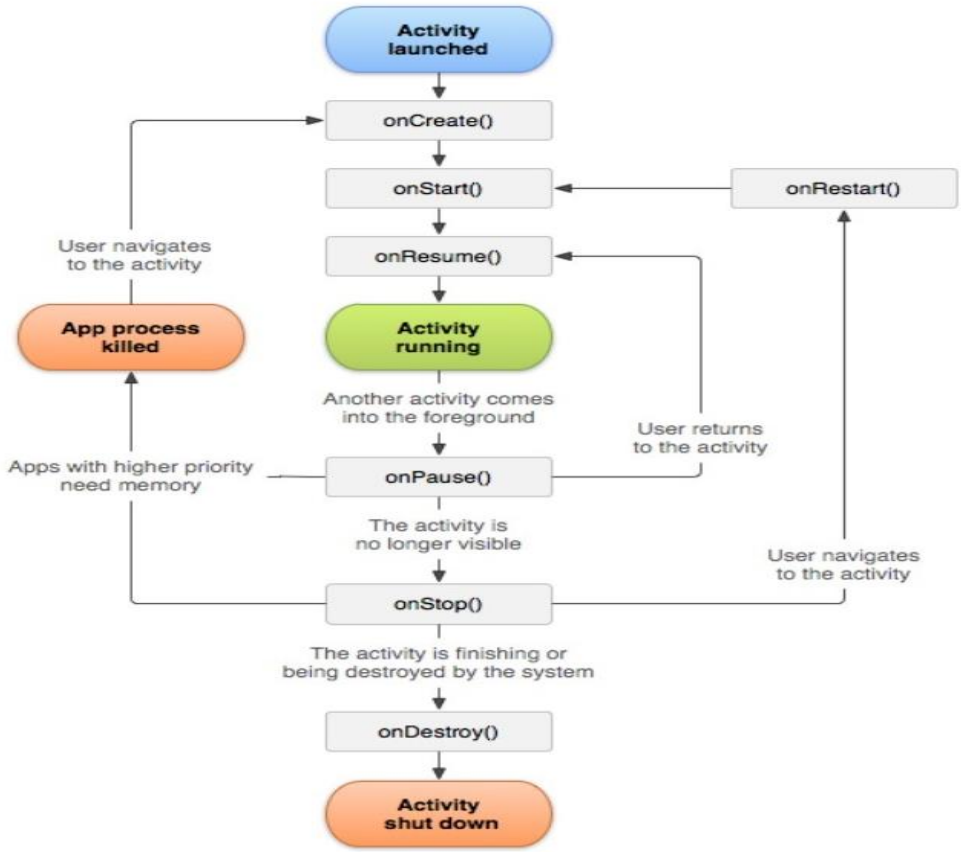
Androidin arkkitehtuurikomponentit ovat kokoelma kirjastoja, joiden avulla voidaan suunnitella vankkoja, testattavia ja ylläpidettäviä sovelluksia. Ne auttavat hallitsemaan käyttöliittymäkomponenttien elinkaarta ja käsittelemään tietojen pysyvyyttä. [8.] Tässä luvussa käsitellään sovelluksissa käytettyjä arkkitehtuurikomponentteja, jotka muodostavat sovellusten käyttöliittymänäkymän ja tietokannan.

4.1 Näkymät: aktiviteetit ja fragmentit

Aktiviteetti on sovelluksen komponentti, joka tarjoaa käyttäjälle toiminnon ja siihen liittyvän käyttöliittymänäkymän. Aktiviteetti tarjoaa sovelluksessa ikkunan, johon sovellus piirtää käyttöliittymänsä. Tämä ikkuna täyttää tyypillisesti koko näytön, mutta se voi olla pienempi osa näyttöä ja voidaan asettaa muiden ikkunoiden päälle. Yleensä yksi aktiviteetti toteuttaa yhden näytön sovelluksessa. Sovellukset sisältävät usein monia näyttöjä, mikä tarkoittaa sitä, että niissä on useampia toimintoja. Sovelluksessa yksi aktiviteetti määritellään usein päätoiminnoksi (MainActivity), joka on ensimmäinen näyttö, joka tulee näkyviin käyttäjän käynnistäessä sovelluksen. Kukin aktiviteetti voi käynnistää toisen aktiviteetin eri toimintojen suorittamista varten. Vaikka kaikki aktiviteetit muodostavat sovelluksessa yhden käyttökokemuksen, ne ovat löysästi sidottuina toisiinsa eli niiden välillä on yleensä minimaalinen riippuvuus. [9.] Aktiviteetit ovat tärkeimpiä peruselementtejä sovelluksessa, ja niihin määritellään kaikki muut toiminnallisuudet.

Aktiviteettien elinkaaren hallinta on tärkeä osa sovelluskehityksessä. Kun käyttäjä navigoi sovelluksessa, sovelluksessa olevat aktiviteetit siirtyvät niiden elinkaaren eri vaiheiden läpi. Aktiviteettiluokka tarjoaa useita takaisinkutsuja (callbacks), joiden avulla aktiviteetti saa tiedon siitä, että sen tila on muuttunut. Takaisinkutsumetodeilla voidaan ilmoittaa, kuinka aktiviteetti käyttäytyy, kun sieltä poistutaan ja sinne palataan takaisin. [10.] Kutsuilla voidaan suorittaa sopiva työ sovelluksen tilan muuttuessa, mikä tekee siitä paljon vahvemman ja suorituskykyisemmän. Elinkaaren hallinnan avulla voidaan muun muassa välttää ylimääräisten järjestelmäresurssien käyttöä, kun sovellus ei ole

aktiivisesti käytössä, välttää erilaisia sovelluskaatumisia ja välttää käyttäjän sovelluksentilan menettämistä sovelluksesta poistuttaessa ja sinne palattaessa.



Kuva 2. Aktiviteetin elinkaari [10]

Elinkaari sisältää kuusi erilaista takaisinkutsumetodia, kuten kuvasta 2 näkyy. Järjestelmä käyttää kutakin metodia, kun aktiviteetti päättyy uuteen tilaan. Metodit ovat onCreate(), onStart(), onResume(), onPause(), onStop() ja onDestroy(). Kolmea ensimmäistä metodia kutsutaan aktiviteetin käynnistyksen ja luomisen yhteydessä. Niiden avulla käynnistetään kaikki toiminnallisuudet, joita aktiviteetti vaatii toimiakseen. Metodi onCreate() tulisi suorittaa vain kerran koko aktiviteetin elinkaaren aikana, sillä se vastaa aktiviteetin peruslogiikasta. Loppuja metodeja kutsutaan silloin, kun sovellukseen halutaan palauttaa oikea tila. Järjestelmä kutsuu kolmea viimeistä metodia, kun aktiviteettia aletaan purkaa käyttäjän poistumisen yhteydessä. Kun purkaminen on vain osittaista eli ei tuhota aktiviteettia kokonaan, kutsutaan vain osaa metodeista. Tuolloin

aktiviteetti jää muistiin, ja käyttäjä pystyy palaamaan siihen joko toisesta aktiviteetista tai sovelluksesta. Kun käyttäjä palaa kyseiseen aktiviteettiin, se jatkuu samasta kohdasta, johon käyttäjä sen lopetti.

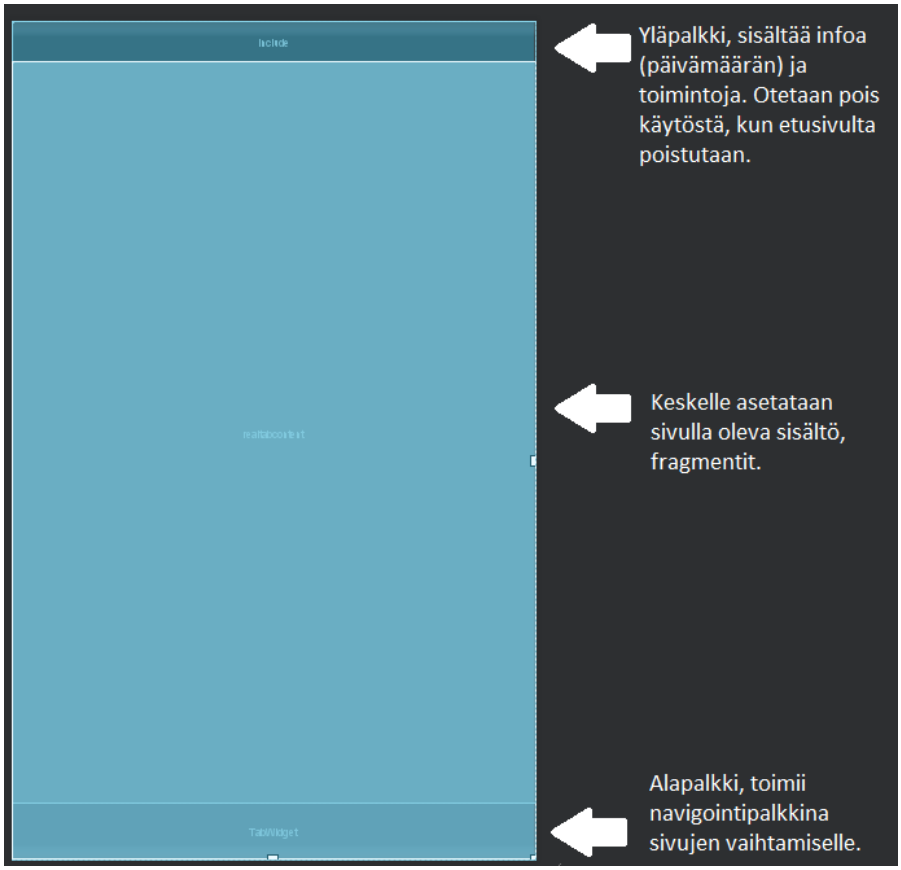
Fragmentti on pieni palanen aktiviteettia, ja se mahdollistaa modulaarisemman aktiviteetin suunnittelun. Fragmentit edustavat toimintaa ja pientä osaa näkymästä, joita käytetään käyttöliittymän rakentamisessa joko yhdistämällä yksi tai useampi fragmentti kiinni aktiviteettiin. [11.] Fragmentit ovat niin sanottuja lapsikomponentteja, joiden avulla rakennetaan käyttöliittymä ja sen toiminnot isäntäaktiviteetille.

Fragmentit ovat uudelleenkäytettäviä luokkia, joilla on omat ulkoasut ja oma elinkaaren hallinta. Fragmentit on aina pidettävä kiinni aktiviteetissa, koska ne eivät pysty toimimaan ilman isäntäaktiviteettia. Isäntäaktiviteetin elinkaari vaikuttaa suoraan fragmenttien elinkaareen. [11.] Kun aktiviteetti on käynnissä, kutakin fragmenttia pystytään hallitsemaan yksilöllisesti joko lisäämällä tai poistamalla fragmentteja. Fragmentteja pystytään myös pinoamaan aktiviteetissa eli aina, kun uusi fragmentti avataan, se työnnetään pinorakenteen päällimmäiseksi. Pinon päällimmäisenä olevassa fragmentissa on silloin se näkymä, jonka käyttäjä näkee puhelimessaan. Pinorakenteen ansiosta käyttäjä voi perua fragmenttitapahtumia painamalla Takaisin-painiketta, jolloin pinon päällimmäisenä oleva fragmentti poistetaan ja siirrytään taaksepäin edelliselle fragmentille.

Fragmentit mahdollistavat dynaamisen ja joustavan tavan käyttöliittymän suunnittelussa. Jokainen fragmentti tulisi suunnitella modulaariseksi ja uudelleenkäytettäväksi käyttöliittymäkomponentiksi, jotta niitä voitaisiin sisällyttää useisiin eri aktiviteetteihin useassa eri paikassa. [11.] Aktiviteetin ulkoasu on helposti muokattavissa ajon aikana, kun se on jaettu fragmenteiksi. Fragmentit helpottavat käyttöliittymäkomponenttien yhdistämistä ja vaihtamista ja antavat siihen enemmän tilaa, jolloin monimutkainen näkymähierarkian hallitseminen jää pois.

Insinöörityössä kehitettävä sovellus koostui pääasiassa yhdestä pääaktiviteetista (MainActivity), joka toimi pohjana kaikille käyttöliittymäkomponenteille. Kun sovellus käynnistyi, tämä aktiviteetti luotiin. Se tuhottiin vasta, kun sovellus suljettiin. Pääaktiviteetin ulkoasu koostui kolmesta erilaisesta tyhjästä ikkunasta, joihin kiinnitettiin

moduuleita sovelluksen pyöriessä (ks. kuva 3). Näitä moduuleita ovat aktiviteetin yläreunassa oleva yläpalkki, ruudun keskellä oleva ikkuna fragmentille ja alareunassa oleva sovelluksen navigointipalkki. Navigointipalkki on osa aktiviteettia, ja se pysyy elossa koko aktiviteetin elinkaaren ajan.

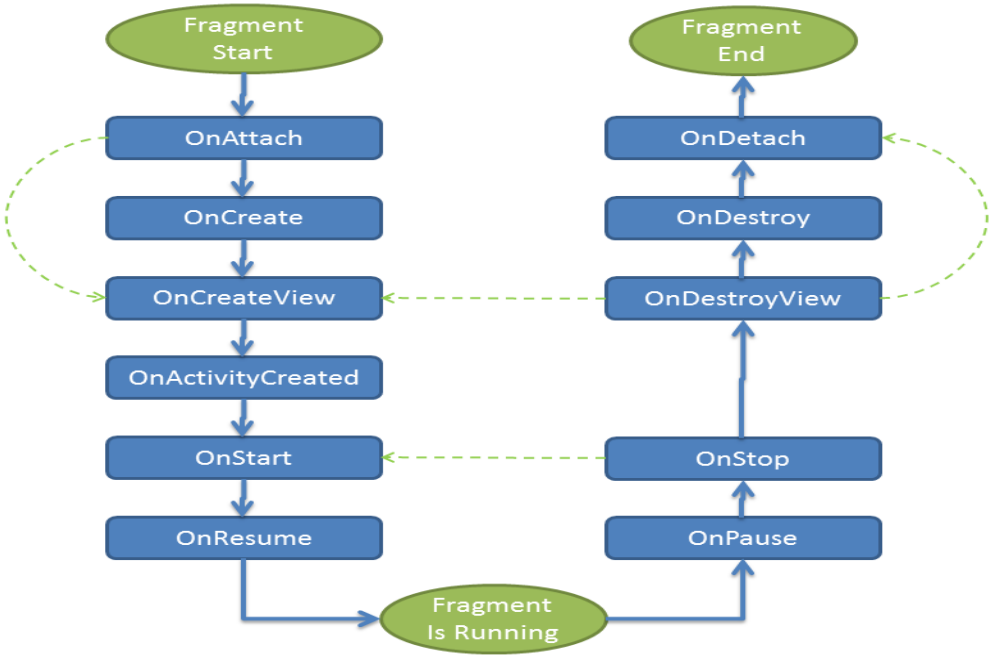


Kuva 3. Pääaktiviteetin moduulit

Käyttäjä pystyy muuttamaan pääaktiviteetin näkymän ulkoasua valitsemalla navigointipalkilta sivun. Kun käyttäjä navigoi uudelle sivulle, pääaktiviteetissa olevat ikkunat päivittyvät. Aktiviteetin yläpalkki päivittyy vastamaan sivua, jolla käyttäjä sillä hetkellä on, ja sivun keskellä oleva fragmentti korvataan uudella fragmentilla. Kun fragmentti vaihtuu, vanha fragmentti tuhotaan, jotta sovellus ei keräsi liikaa muistia ylimääräisistä fragmenteista. Käyttäjä pystyy myös avaamaan uusia näkymiä eli fragmentteja sivuilla. Nämä fragmentit asetetaan pääaktiviteetin päälle, jolloin ne toimivat kokonaan uusina näkyminä.

Sovelluksen eri sivujen fragmentit voivat sisältää ViewPager-objektin (widget). ViewPager mahdollistaa yksinkertaisen navigoinnin fragmenttien välillä joko vasemman tai oikean pyyhkäisyn avulla. ViewPageria käytetään useimmiten fragmentin ohella, mikä onkin kätevä tapa hallita kunkin sivun elinkaarta. ViewPager toimii FragmentPagerAdapterin kanssa, joka kontrolloi olemassa olevia ja näkyvänä olevaa fragmenttia. Sen avulla luodaan useammasta eri data-aineistosta fragmentteja, jotka asetetaan pyyhkäisyn päähän toisistaan. Tällöin näkyvän fragmentin eri puolilla on muita fragmentteja.

ViewPageriä hyödynnetään sovelluksessa eri näkymissä, jotta saadaan luotua helposti selattavia sivuja eri data-aineistoilla (kuten eri päivien ateriat). Se luo näkymään kolme samanlaista fragmenttia eri datalla. Näkyvä fragmentti on niistä keskimmäinen ja sen molemmin puolin ovat odottavassa tilassa olevat fragmentit. Kun käyttäjä pyyhkäisee jompaankumpaan suuntaan, näkyviin tulee toinen fragmentti, ja edellinen siirtyy odottavaan tilaan. Eri data-aineistoilla olevia fragmentteja luodaan ja tuhotaan sitä mukaa, kun käyttäjä siirtyy fragmentista toiseen. Näkyvän fragmentin molemmin puolin on aina ainoastaan yksi fragmentti odottavassa tilassa. Tällä tavoin vältytään liialliselta muistin varaamiselta, ja sivujen selaaminen on nopeaa ja tehokasta.



Kuva 4. Fragmentin elinkaari [12]

Fragmenteilla on myös oma elinkaarensa, vaikka ne ovat sidoksissa ne omistavan isäntäaktiviteetin elinkaareen (ks. kuva 4). Fragmentit sisältävät aktiviteetin peruselinkaaren menetelmät ja menetelmät, jotka liittyvät vuorovaikutukseen aktiviteetin ja käyttöliittymän luomisen kanssa. Kun fragmentti luodaan ja kiinnitetään aktiviteettiin, se käy läpi kaikki aloitusmetodit onAttach(), onCreate(), onCreateView(), onAcitvityCreated(), onStart() ja onResume(). OnAttach()- ja onCreate()-metodit ovat niin sanottuja alustusmetodeja, joissa kaikki käyttöliittymän ulkopuoliset muuttujat alustetaan ja kaikki tarvittava data käyttöliittymän luomiselle haetaan. OnCreateview()-metodissa luodaan fragmenttiin liittyvä käyttöliittymä ja kaikki siihen liittyvät toiminnallisuudet, kuten napit, tekstikentät ja muut komponentit. OnStart()-metodi tekee fragmentin käyttäjälle näkyväksi ja onResume()-metodi aloittaa vuorovaikutuksen käyttäjän kanssa.

Kun fragmenttia ei enää käytetä, käydään läpi sen käänteiset tuhoamismenetelmät. Menetelmien ansiosta fragmentti pystytään asettamaan odottavaan tilaan tai se pystytään tuhoamaan kokonaan. Näitä menetelmiä ovat onPause(), onStop(), onDestroyView() ja onDestroy(). Kun fragmentin näkymä halutaan säilyttää sieltä poistuttaessa, käydään läpi onPause()- ja onStop()-metodit. Kun näkymä vaihtuu

kokonaan ja se halutaan tuhota, käydään läpi onDestoyView()-metodi. Kun käyttäjä palaa tämän jälkeen odottavassa tilassa olevaan fragmenttiin, on näkymä luotava uudelleen onCreateView()-metodin avulla. Jos fragmentti halutaan tuhota kokonaan, käytetään kaikkia edellä mainittuja tuhoamismenetelmiä.

Fragmenttien vaihtaminen ja hallitseminen tapahtuu FragmentManagerin kautta. Sen avulla pystytään avaamaan, sulkemaan ja muokkaamaan fragmenttien transaktioita. Kun sovelluksessa avataan uusia fragmentteja pääaktiviteetin kautta, ne pinoutuvat päällekkäin, jolloin alemmat fragmentit jäävät tauolle ja päällimmäiset fragmentit ovat aktiivisia. Odottavassa tilassa olevat fragmentit eivät kuitenkaan koskaan tuhoudu täysin, vaan ainoastaan näkymä tuhoutuu, mutta sen sisällä oleva data säilyy. Kun fragmenttiin palataan, se pystytäänkin palauttamaan sen alkuperäiseen tilaan datan avulla. Kun poistutaan pinon päällimmäiseltä fragmentilta, voidaan dataa lähettää sen ulkopuolelle, minkä jälkeen se tuhoutuu kokonaan.

4.2 Tietokanta: SQLite ja Room

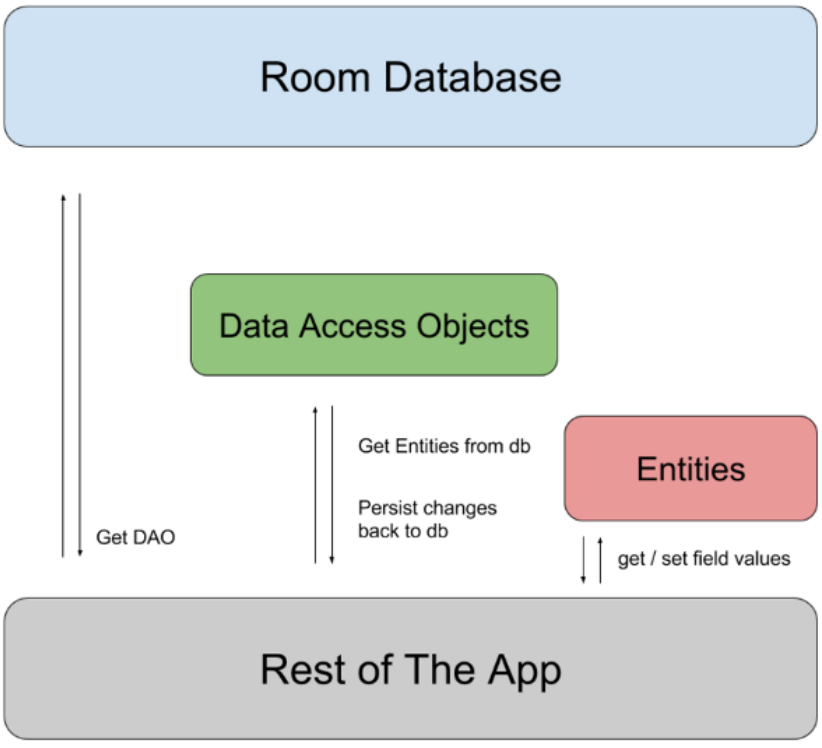
SQLite on avoimen lähteen SQL-tietokanta, joka tallentaa sovelluksen tietoja puhelimen sisällä olevaan tekstitiedostoon. Android-laitteet sisältävät oletuksena SQLite-tietokannan toteutuksen. SQLite tukee kaikkia relaatiotietokannan ominaisuuksia, eikä tietokantaan pääsyä varten tarvitse luoda mitään yhteyksiä, koska yhteys tietokantaan luodaan automaattisesti. [13.] Kaikki sovelluksessa olevat tiedot, kuten aterioiden valokuvat, muistiinpanot, merkinnät ja ravintoarvot, tallennetaan SQLite-tietokantaan.

Room persistance libary on Android-arkkitehtuurikomponentti, joka helpottaa työskentelyä SQLiteDatabase-objektien kanssa. Room tarjoaa abstraktiokerroksen SQLiten päälle ja antaa sujuvan pääsyn tietokantaan hyödyntäen SQLiten koko tehoa. Tämä helpottaa työskentelyä sovelluksessa olevien SQLiteDatabase-objektien kanssa vähentämällä koodin määrää ja tarkistamalla SQL-kyselyt sen käännöshetkellä. [14.]

Room on ORM eli Object Relational Mapping -kirjasto, joka kartoittaa tietokantaobjektit Java-objekteiksi. ORM tekee tietokannan rakentamisen ja sen muuttamisen helpoksi ja yksinkertaiseksi ja muutettaessa tietokantaobjekteja Java-objekteiksi vältetään monimutkaisen ja pitkän raa'an koodin kirjoittaminen. Päinvastoin kuin SQLite Room on myös rakennettu toimimaan LiveDatan kanssa tietojen tarkkailua varten. [14.] Tämä mahdollistaa käyttöliittymän/näkymien tarkkailemisen ja päivittämisen, kun tietokannassa tapahtuu muutoksia.

Roomilla on kolme pääkomponenttia:

1. Entity, joka edustaa tietokantataulua
2. DAO, joka antaa sovellusliittymän tietojen lukemiselle ja kirjoittamiselle
3. Database, joka edustaa tietokannan haltijaa.



Kuva 5. Roomin arkkitehtuurikomponentit sovelluksessa [14]

Kuvassa 5 sovellus käyttää Room-tietokantaa saadakseen kyseiseen tietokantaan kuuluvat tietojenkäsittelyobjektit eli DAOt. Objektien avulla entiteetit saadaan tietokannasta ja niihin tehdyt muutokset pysytään tallentamaan tietokantaan. Lopuksi sovellus käyttää entiteettejä hakemaan ja asettamaan arvoja tietokannan taulukoista. [15.]

Entity-luokat määrittävät tietokannan taulukon. Luokkaan määritellään @Entity-anotaatio ja siihen kuuluvan taulukon nimi, minkä ansiosta luokka pystytään yhdistämään taulukkokokonaisuudeksi. Room luo jokaiselle luokalle taulukon, joka sisältää tämän merkinnän. Jokaiselle Entity-luokalle on myös määriteltävä PrimaryKey eli pääavain, joka toimii yksilöivänä tunnisteena taulukon riveille. Taulukossa voi olla vain yksi pääavain, jota käytetään tunnisteena datan nopeaan jäsentämiseen taulukossa. Pääavaimen arvot voivat olla joko numeraalisia tai merkkijonoarvoja. Ne eivät voi olla nolla-arvoja ja jokaisen tietorivin on sisällettävä yksilöllinen arvo. Jokaiselle taulukon sarakkeelle määritellään muuttujat. Oletuksena sarakkeiden datatyyppi ja nimi määrittyvät muuttujien tyypin ja nimen perusteella, mutta nimiä voidaan myös muokata @ColumnInfo-anotaation avulla.

```
@Entity(tableName="goal_progress")
public class GoalProgress {

    @ColumnInfo(name="goal_id")
    @PrimaryKey(autoGenerate = true)
    public long id;

    @ColumnInfo(name="goal_day")
    public int day;

    @ColumnInfo(name="goal_month")
    public int month;

    @ColumnInfo(name="goal_value")
    public Integer value;

    @ColumnInfo(name="day_notes")
    public String notes;

    @ColumnInfo(name="day_water")
    public Integer water;
```

```
    public GoalProgress(int day, int month, int value){
        this.day= day;
        this.month=month;
        this.value = value;
    }

    public int getDay() { return day; }

    public void setDay(int day) { this.day = day; }

    public int getMonth() { return month; }

    public void setMonth(int month) { this.month = month; }

    public Integer getValue() { return value; }

    public void setValue(Integer value) { this.value = value; }

    public String getNotes() { return notes; }
```

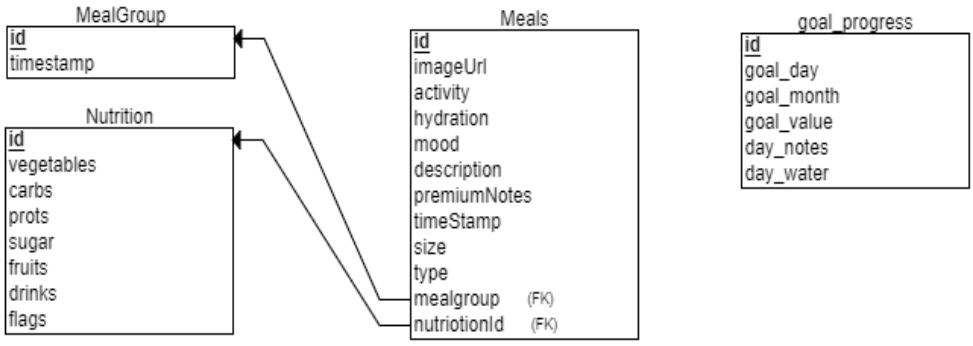
Kuva 6. Esimerkki Entity-luokka

Kuvassa 6 on entiteettiluokka, joka määrittelee tavoitteiden etenemistä koskevan taulukkorakenteen sovelluksessa. Taulukolla ei ole suhteita muihin sovelluksessa oleviin taulukoihin. Luokka sisältää @Entity-anotaation ja taulukon nimen ja kaikki siihen kuuluvat sarakkeet, jotka on nimetty @ColumnInfo-anotaatioilla. Luokan pääavain on

nimetty ID-arvoksi, ja se käyttää automaattisesti generoituvaa arvoa jokaiselle taulukon riville. ID-arvon generointi alkaa arvosta yksi ja jokaisen uuden lisäyksen myötä se kasvaa yhdellä. Luokalla on konstruktori sekä setterit ja getterit muuttujien asettamiselle ja palauttamiselle.

Relaatiotietokanta muodostuu useammasta eri relaatiosta, joiden välillä on kytköksiä [16]. Ulkoisten avainten avulla pystytään luomaan yhteyksiä entiteettien välille. Ulkoinen avain on kenttä yhdessä taulukossa ja viittaa toisen taulukon alkuperäiseen pääavaimeen. Taulukon indeksit ovat myös tärkeitä monirelaatiotaulukoille varsinkin silloin, kun taulujen välillä on yhteyksiä. Ne tehostavat kyselyjen suorittamista ja nopeuttavat rivien hakua taulukosta.

Kuvassa 7 näkyvät sovellukseen kuuluvat neljä eri tietokantataulukkoa: ateria-, ateriaryhmä-, ravintoaine- ja tavoitteiden seuranta -taulukot. Tavoitteiden seuranta -taulukko eroaa muista taulukoista siten, ettei sillä ei ole minkäänlaisia sidoksia muihin taulukoihin. Ateriataulukko on suhteessa kahteen muuhun taulukkoon (ateriaryhmä- ja ravintoainetaulukot). Ateriataulukolle on määritelty ulkoiset avaimet mealgroup (FK) ja nutritionId (FK), joilla viitataan kummankin taulukon pääavaimiin. Aterian ja ateriaryhmän välille on määritelty yksi-moneen-suhde, jossa useammalla aterialla voi olla sama ateriaryhmä. Aterian ja ravintoaineen välille on määritelty yksi-yhteen-suhde, jossa yhtä ateriaa kohti voi olla vain yksi ravintoainearvo.



Kuva 7. Tietokantakaavio sovelluksen taulukoista

Kuvassa 8 näkyvät ateria-entiteettiluokalle määritellyt yhteydet ateriaryhmä- ja ravintoaineluokkiin. Indeksit on määritelty @Index-anotaatioden avulla ja, niistä näkyy, mihin taulukon sarakkeeseen ne viittaavat. Ravintoaineindeksille on asetettu yksilöivä indeksi (unique=true), joka varmistaa sen, että taulukon eri tietoriveillä ei ole identtisiä avainsanoja. Tämä tarkoittaa sitä, että yhdellä aterialla voi olla vain yksi ravintoainearvo, mutta useammalla aterialla voi olla useampia ateriaryhmiä. Luokan ulkoiset avaimet on merkitty @ForeignKey-anotaatioden avulla, jotka sisältävät yhdistettävän entiteettiluokan taulun nimen, yhdistettävän sarakkeen nimen ja ulkoisen avaimen nimen sitä kantavassa luokassa.

```
@Entity(tableName = "meals",
        indices = {@Index(value="mealgroup"),
                   @Index(value="nutrition_id", unique=true)},
        foreignKeys = {@ForeignKey(entity = MealGroup.class,
                parentColumns = "mg_id",
                childColumns = "mealgroup"),
                @ForeignKey(entity=Nutrition.class,
                parentColumns="id",
                childColumns="nutrition_id",
                onDelete = CASCADE)
        })
```

Kuva 8. Esimerkki entiteettiluokan yhteydet

DAO eli tietojenkäsittelyobjekti on suunnittelumalli, jonka avulla erotetaan tietokantaan kuuluvat operaatiot muusta ohjelmalogiikasta omaan luokkaansa. DAO on Roomin pääkomponentti, jossa määritellään kaikki ne menetelmät, jotka tarjoavat pääsyn sovelluksen tietokantaan. [17.] Jokaiselle tietokantaan kuuluvalle entiteettiluokalle kirjoitetaan DAO-luokka, joka sisältää metodit erilaisia tietokantakyselyitä varten (esimerkiksi lisäämiseen, muokkaamiseen ja poistamiseen kuuluvat operaatiot).

Kuvassa 9 on abstrakti DAO-luokka, joka sisältää kaikki tavoitteiden etenemisen lisäämiseen, muuttamiseen tai poistamiseen liittyvät tietokantaoperaatiot. Jokainen metodi on merkittävä käyttäen @Insert-, @Update- tai @Delete-anotaatioita. Kaikilla merkinnöillä on oma tarkoituksensa ja niihin pystytään asettamaan konfliktinkäsittelystrategioita, kuten Replace-strategia. Tämän strategian mukaisesti vanha tavoite korvataan uudella, jos tavoite on jo asetettu ja se yritetään asettaa

uudelleen. Luokkaan kuuluvat myös hakukyselyt, jotka merkitään @Query-anotaation avulla. Kyselyt sisältävät SQL-kyselylauseet, jotka palauttavat tietokannasta halutut arvot.

```
@Dao
public abstract class GoalProgressDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    public abstract void insertGoalProgress(GoalProgress progress);

    @Update
    public abstract void updateGoalProgress(GoalProgress progress);

    @Delete
    public abstract void deleteGoalProgress(GoalProgress progress);

    @Query("SELECT * FROM goal_progress WHERE goal_day=:day AND goal_month=:month")
    public abstract GoalProgress getProgress(int day, int month);

    @Query("SELECT * FROM goal_progress WHERE goal_month = :month")
    public abstract List<GoalProgress> getProgressFromMonth(int month);

    @Query("SELECT day_water FROM goal_progress WHERE goal_day=:day AND goal_month=:month")
    public abstract Integer getWaterFromDay(int day, int month);

}
```

Kuva 9. Tavoitteiden etenemisen DAO-luokka

Room-tietokanta rakennetaan tietokantaluokan avulla, joka toimii Room-tietokannan (RoomDatabase) aliluokkana. Luokka sisältää tietokannan haltijan ja toimii pääyhteyspisteenä sovelluksen jatkuvaan relaatiotietoon. Tietokantaluokka on abstrakti luokka, jossa määritetään kaikki entiteetit eli taulukot, jotka kyseiselle tietokannalle halutaan luoda, ja tietojenkäsittelyobjektit, joiden avulla taulukoihin päästään käsiksi tietokantakutsujen avulla.

Kuvassa 10 on abstrakti tietokantaluokka, joka laajentaa RoomDatabase-luokan. Luokkaan on merkitty @Database-anotaatio ja lista entiteettiluokista, jotka muodostavat tietokannan. Kuvassa 11 määritellään abstraktit metodit DAO-luokista, jotka vastaavat luokassa olevia entiteettejä ja palauttavat ne luokat, jotka on merkitty @DAO-anotaatiolla. Kuvasta 12 näkyy, että luokka sisältää Room-tietokannan rakentamiseen staattisen metodin, joka palauttaa tietokantaolion.

```
@Database(entities = {MealGroup.class, Meal.class,
        Nutrition.class, GoalProgress.class}, version =6)
@TypeConverters({Converters.class})
public abstract class MyDatabase extends RoomDatabase {
```

Kuva 10. Tietokantaluokka, entiteetit

```
    public abstract MealGroupDao mealGroupDao();
    public abstract MealDao mealDao();
    public abstract NutritionDao nutritionDao();
    public abstract GoalProgressDao progressDao();
```

Kuva 11. Tietokantaluokka, tietojenkäsittelyobjektit

```
public static MyDatabase buildDatabase(final Context appContext){
    context = appContext;
    try {
        return Room.databaseBuilder(appContext, MyDatabase.class, DATABASE_NAME)
                .addCallback(onCreate(db) → {
                        super.onCreate(db);
```

Kuva 12. Tietokantaluokka, tietokannan rakentamien

5 Työn toteutus

On olemassa monia mobiilisovellusten analysointityökaluja, jotka tarjoavat erilaisia toimintoja ja joihin liittyy erilaisia etuja ja haittoja, jotka pitää ottaa huomioon tiedon analysoinnissa. Päätin käyttää työssäni Flurry Analytics SDK -kirjastoa, koska se tarjoaa parhaat edut ja vastaa yrityksen tarpeita.

5.1 Flurry Analytics

Flurry on Yhdysvalloissa toimiva yritys, joka tarjoaa erilaisia analytiikkaratkaisuja mobiilikäyttäjien käyttäytymisen ja tapojen seurantaan mobiilisovelluksissa. Se pyrkii

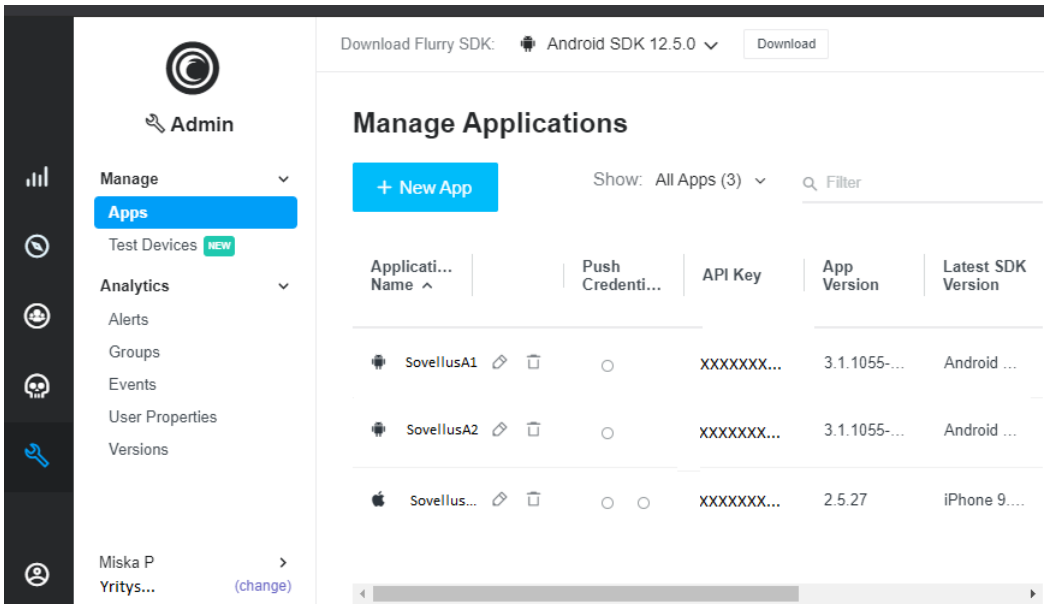
optimoimaan kehittäjien, markkinoijien ja kuluttajien mobiilikokemuksen tarjoamalla erilaisia palveluita sovellusten rakentamiseen, mittaamiseen ja mainostamiseen. Yahoo osti Flurryn vuonna 2014. Flurry Analytics on tällä hetkellä yksi markkinoiden johtavimmista analytiikkatuotteista, jonka tavoitteena on antaa yritykselle parempi käsitys kuluttajien käyttäytymisestä mobiilisovelluksissa. [18.]

Flurry Analytics SDK on ilmainen työkalu, joka tarjoaa kehittäjälle tavan seurata kuluttajan toimintoja ja käyttäytymistä, kun he käyttävät mobiilisovellusta. Se tarjoaa kehittäjälle joukon analytiikkatyökaluja, jotka on suunniteltu antamaan parhaat mahdolliset raporttitulokset kerätyistä data-aineistoista ja sisältää ominaisuuksia käyttäjän segmentointiin, myyntisyklin hallintaan (consumer conversion funnel) ja sovelluskannan analysointiin. Data-aineistot ja niistä saadut raportit ovat helposti luettavissa Flurryn nettisivuilta (https://www.flurry.com/) selaimen kautta.

5.2 Flurryn integrointi Android-sovellukseen

Flurry SDK:n integrointi Android-sovellukseen on helppo ja suoraviivainen prosessi. Flurryn alusta tukee useamman eri Android-sovelluksen analysoimista, mutta kukin sovellus on kuitenkin integroitava alustalle erikseen.

Sovellus rekisteröidään Flurryyn nettisivuilta löytyvän Admin-paneelin kautta (ks. kuva 14). Uuden sovelluksen rekisteröinti vaatii järjestelmävalvojan roolin eikä uusia sovelluksia ole mahdollista lisätä alustalle ilman sitä. Kun uusi sovellus on lisätty Flurryyn, saadaan sovelluksen Flurry-sovellusliittymäavain (API Key). Jokaisella sovelluksella on uniikki sovellusliittymäavain, jonka SDK vaatii, kun sitä käytetään.



Kuva 14. Flurryn Admin-paneelin sovellussivu

Flurry SDK voidaan asentaa Android-sovellukseen kahdella eri tavalla, jotka vaativat AndroidManifest.xml-tiedoston konfiguroimisen. Tiedostossa on sallittava kaikki luvat, jotka mahdollistavat verkon yhteyden muodostamisen ja analytiikkatietojen lähettämisen Flurry SDK:n palvelimelle. Tiedostossa on sallittava luvat android.permission.INTERNET ja android.permission.ACCESS_NETWORK_STATE.

Flurry integroidaan sovellukseen joko manuaalisesti tai JCenter-repositorion kautta. SDK-paketti ladataan manuaalisesti Flurryn nettisivuilta Admin-paneelilta, kuten kuvan 14 yläreunasta näkyy. Ladatut SDK-paketit (kun versio on ylempi kuin 12.0.0) ovat Android-arkisto eli AAR-tiedostot, jotka ovat nimeltään FlurryAnalytics-x.y.z.aar. Ladattu AAR-tiedosto siirretään Android-projektin libs-kansioon, minkä jälkeen kirjasto otetaan käyttöön Gradlen-rakennustiedoston (build.gradle) dependensseissä.

AAR-tiedosto lisätään projektin libs-kansioon Android Studiossa valitsemalla File > New > New module > Import .JAR/.AAR Package, minkä jälkeen valitaan AAR-tiedosto. Kirjasto toteutetaan rakennustiedoston dependensseissä joko kirjoittamalla se itse tai lisäämällä valikon kautta File > Project Structure > Dependencies > Module dependency ja valitsemalla tiedosto libs/FlurryAnalytics-x.y.z.aar (ks. kuva 15).

```
124 ▶ dependencies {
125       implementation project(path: ':flurryAnalytics_12.5.0')
126       // ...
```

Kuva 15. Lisätty Flurry Analytics AAR-tiedosto build.gradle-rakennustiedoston dependensseissä

Suositeltavampi tapa asentaa Flurry SDK on JCenter-repositorin kautta, koska prosessi on paljon yksinkertaisempi eikä vaadi säätöä tiedostojen tai kansioiden kanssa. Asennus tapahtuu Android-projektin build.gradle-rakennustiedoston kautta. Rakennustiedostossa on ilmoitettava tietovarasto (repository), josta SDK ladataan. Tämä vaatii jcenter()-notaation lisäämisen rakennustiedoston repositories-kohtaan. Tämän jälkeen asetetaan rakennustiedoston dependenssin toteutukseksi ladattava kirjasto muodoltaan com.flurry.android:analytics:x.y.z (ks. kuva 16).

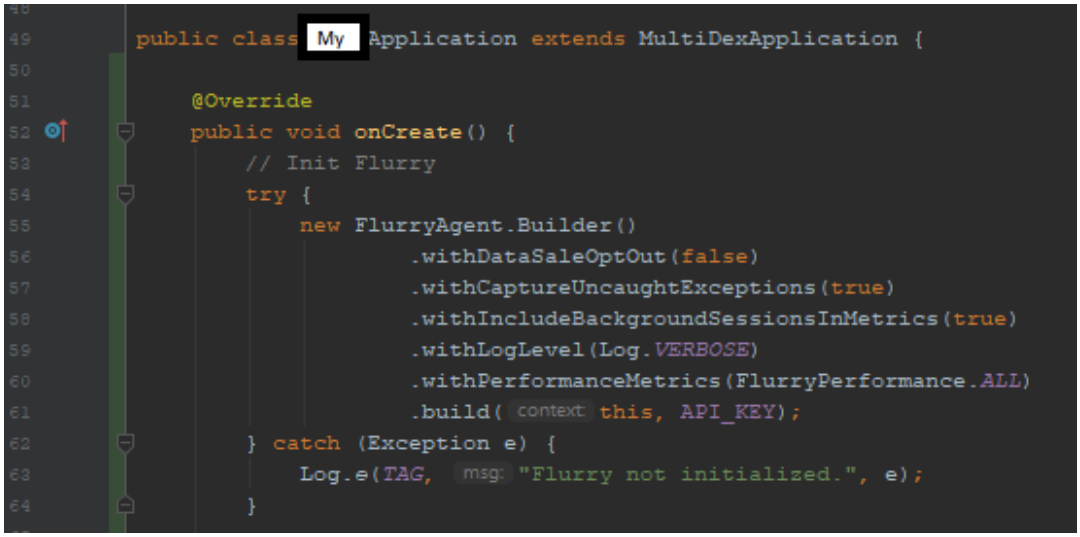
```
124     repositories {
125         jcenter()
126         // ...
127     }
128
129 ▶   dependencies {
130         implementation 'com.flurry.android:ads:12.5.0'
131         // ...
```

Kuva 16. Flurry Analytics SDK:n asentaminen JCenerin kautta build.gradle-rakennustiedostossa

Flurry SDK:n asennuttua sovellukseen tulee FlurryAgent-luokka, joka sisältää kaikki menetelmät Flurryn rakentamiselle ja tiedon keräämiselle sovelluksen käyttäjistä. FlurryAgent on alustettavissa milloin tahansa sovelluksen pyöriessä, mutta se on suositeltavaa alustaa heti sovelluksen käynnistyttyä ennen pääaktiviteetin rakennusta tai sen aikana. Sovellusluokka on hyvä paikka FlurryAgentin alustukselle, koska luokka instantisoidaan ennen kuin kaikki muut aktiviteetit tai muut sovellusobjektit on luotu. Tämän jälkeen FlurryAgent voidaan palauttaa sen sovelluskontekstissa.

Kuvassa 17 näkyy sovellusluokka nimeltä MyApplication, jossa Flurry SDK alustetaan FlurryAgent.Builer()-kutsun avulla. Alustuksessa on määriteltävä projektin sovellusliittymäavain, joka on välttämätön, jotta Flurry pystyisi vastaanottamaan tietoja

sovelluksesta. Builder sisältää useita erilaisia menetelmiä FlurryAgentin asetusten säätämiselle.



Kuva 17. Flurryn alustus sovellusluokassa

5.3 Flurry-tapahtumat

Flurryn avulla voidaan luoda mukautettuja tapahtumia seuraamaan tiettyjä toimintoja, joita käyttäjät suorittavat sovellusta käyttäessään. Näitä toimintoja voivat olla esimerkiksi nappien painallukset, ostosten tekeminen tai sovelluksesta poistuminen. Jokainen tapahtuma luodaan erikseen, ja se sisältää yksilöivän tunnisteen tapahtumanimelle ja siihen kuuluvat parametrit. Flurryn avulla voidaan seurata enintään 500 yksilöllistä tapahtumanimeä kullakin sovelluksella. Kun sovellus saavuttaa 500 tapahtuman rajan, ei uusia tapahtumia enää seurata. [19.]

Projektissa suunniteltiin luokka nimeltä FlurryUtils (ks. kuva 18), joka sisältää kaikki metodit erilaisten tapahtumien seuraamiseksi. Luokassa olevat metodit ovat staattisia metodeja public-näkyvyydellä ja suoraan käytettävissä eri puolilla sovellusta luokan nimen kautta, esimerkiksi FlurryUtils.metodinNimi(). Metodeja kutsutaan kaikkialla sovelluksessa eri toimintojen kautta.

```
/**
 * Class that contains static logging methods for events we want to keep track of
 * with Flurry.
 */
public class FlurryUtils {
```

Kuva 18. FlurryUtils-luokka, joka sisältää metodit tapahtumille

Tapahtumia voidaan luoda FlurryAgentin logEvent()-metodin avulla. Metodista on olemassa erilaisia variantteja, jotka ottavat vastaan erilaisia syötteitä. Tapahtumat voivat sisältää parametreja tai toimia ilman parametreja. Yksinkertaisimmillaan tapahtumalla on määriteltynä ainoastaan nimi ilman mitään parametreja, jolloin saadaan mitattua, kuinka monta käyttäjää on suorittanut tietyn toiminnan. Kuvassa 19 näkyy yksinkertainen tapahtumametodi, joka mittaa käyttäjien tekemiä painalluksia mainoksissa.

```
24      public static void adClicked(){
25          MyApplication.getAppExe().networkIO().execute(()->{
26              if(BuildConfig.BUILD_TYPE=="release") {
27                  FlurryAgent.logEvent( s: "Ad_Clicked");
28              }
29          });
30      }
```

Kuva 19. adClicked()-metodi mainosten painalluksien seuraamiseksi

Parametrit ovat tärkeä osa tapahtumien seurantaa, koska niiden avulla saadaan informatiivisempaa tietoa ja voidaan tarkastella helposti tapahtumien eri ominaisuuksien jakautumista. Jokaisella tapahtumalla voi olla enintään 10 parametria, ja kuhunkin parametriin voi liittyä ääretön määrä arvoja. [19.] Parametreilla varustettuja tapahtumia voidaan lisätä FlurryAgentin logEvent()-metodiin käyttämällä hajautustauluja. Hajautustauluun on lisättävä kaksi tyyppiparametria, jotka ovat avainmuuttujan arvo eli parametrin nimi ja parametrille lisättävä arvo. On tärkeää huomioida myös se, että parametrien on oltava tyypiltään merkkijonoarvoja, joten kaikki numeraaliset arvot on muutettava String-tyyppisiksi ennen kuin ne voidaan lisätä hajautustauluun. Parametrit lisätään logEvent()-metodiin asettamalla siihen ensin tapahtumanimi ja tämän jälkeen hajautustaulu, joka sisältää asetetut parametrit.

```
384     public static void dayAction(String action, String version){
385         if(BuildConfig.BUILD_TYPE=="release"){
386             MyApplication.getAppExe().networkIO().execute(()->{
387                 String event = "dayAction" + version;
388                 Map<String, String> basicParams = new HashMap<>();
389                 basicParams.put("Action", action);
390                 FlurryAgent.logEvent(event, basicParams);
391             });
392         }
393     }
```

Kuva 20. dayAction()-tapahtumametodi parametrilla

Kuvassa 20 näkyy dayAction()-tapahtumametodi, jolle asetetaan kaksi merkkijonoa: (String) toiminta ja (String) versio. Metodi luo tapahtumia Flurryyn, ja tapahtumien parametrien avulla pystytään seuraamaan käyttäjien tekemiä toimintoja sovelluksen pääsivulta. Metodissa muodostetaan parametreille ensimmäiseksi tapahtumanimi ja sen jälkeen hajautustaulu. Tapahtumanimi muodostuu yhdistämällä olemassa oleva merkkijono ”dayAction” syötettyyn merkkijonoon ”versio”, jotta pystyttäisiin luomaan kahdella eri sovelluksella omat tapahtumat (esimerkiksi dayActionSovellus1 tai dayActionSovellus2). Metodissa määritellään String-tyyppinen hajautustaulu nimeltä basicParams, jonne parametrin nimeksi syötetään Action ja parametrin arvo. Kun tapahtumanimi on luotu ja parametrit on asetettu hajautustauluun, ne syötetään FlurryAgentin logEvent()-metodiin.

Tapahtumiin voidaan lisätä myös ajanseuranta, joka antaa tiedon tapahtuman keskimääräisestä kokonaispituudesta istuntokohtaisesti ja käyttäjittäin. Tapahtuman kesto voidaan kaapata yhdessä tapahtuman ja sen parametrien kanssa yhden totuusarvon kanssa. Ajastettu tapahtuma on hyvä lopettaa sopivassa paikassa, koska muuten se päättyy vasta istunnon päätyttyä eli käyttäjän poistuttua sovelluksesta. [19.]

```
public static void mealOpened(String meal){
    if(BuildConfig.BUILD_TYPE=="release") {
        MyApplication.getAppExe().networkIO().execute(()->{
            Map<String, String> mealParams = new HashMap<>();
            mealParams.put("Meal", meal);
            FlurryAgent.logEvent( s: "mealOpened", mealParams,  b: true);
            /**
             * Timed event can be stopped by calling
             * FlurryAgent.endTimedEvent("mealOpened");
             */
        });
```

Kuva 21 mealOpened()-tapahtumametodi ajanseurannalla

Kuvassa 21 näkyy mealOpened()-tapahtumametodi, johon on asetettu ajanseuranta. Metodi ottaa talteen aterian nimen ja aloittaa ajanseurannan, kun käyttäjä painaa sovelluksen pääsivun ruokaruutua. Tapahtuman ajastin lopetetaan käyttäjän poistuttua ateriasivulta eli fragmentin tuhouduttua. Ajanseuranta voidaan asettaa tapahtumaan määrittämällä logEvent()-metodin kolmas totuusarvo (boolean) todeksi (true). Metodin kutsuttua aloitetaan ajanseuranta ja se lopetetaan, kun endTimedEvent()-metodi kutsutaan.

```
158         @Override
159         public void onCreate(Bundle savedInstanceState) {
160             // Paljon muuta koodia ennen...
161
162             // Kutsutaan Flurry tapahtuma
163             FlurryUtils.mealOpened(mMealType.toString());
```

```
316         @Override
317         public void onDestroy() {
318             // Lopetetaan tapahtuman ajan seuranta
319             FlurryAgent.endTimedEvent( s: "mealOpened");
320             //   ...
321
322             super.onDestroy();
```

Kuva 22 Esimerkki Flurryn tapahtumakutsut

Kuvassa 22 on esimerkki ajanseurannan mealOpened()-tapahtumametodin käytöstä sovelluksen fragmentilla. Tapahtumaa kutsutaan fragmentin luomisen yhteydessä sen onCreate()-metodissa, jossa tapahtumalle syötetään parametrit. Koska metodi sijaitsee staattisessa luokassa nimeltä FlurryUtils, sitä on kutsuttava luokan nimen kautta. Tapahtuman ajanseuranta alkaa heti, kun tapahtumaa kutsutaan, ja se mittaa, kuinka kauan käyttäjä pysyy fragmentilla. Käyttäjän poistuessa fragmentilta lopetetaan ajanseuranta fragmentin onDestroy-metodissa.

5.4 Flurryn muut lisäominaisuudet

Tapahtumien lisäksi Flurry tarjoaa kehittäjille useita muita lisäominaisuuksia, joiden avulla käyttäjästä voidaan saada parempi käsitys. Näitä lisäominaisuuksia on muun muassa käyttäjän sijainnin seuraaminen, sivunäkymien seuraaminen, tulojen seuraaminen ja laitetietojen seuraaminen (kuten laitenimi, laiteosat ja Android-versio). [20.] Tässä työssä ei käsitellä lisäominaisuuksia tarkemmin, koska tällaisten asioiden seuraaminen ei ole sovellusten kannalta tärkeää. Tietojen kaappauksessa on kuitenkin noudatettava Flurryn käyttöehtoja: esimerkiksi henkilökohtaisten tunnistetietojen seuraaminen on ehdottomasti kiellettyä.

Flurry tarjoaa kehittäjille tietoja sovelluksen kaatumisista, poikkeuksista ja virheistä. Nämä tiedot kirjataan ylös sovelluksen käytön aikana reaaliajassa, jotta syyt kaatumisille olisivat selvitettävissä ja korjattavissa mahdollisimman nopeasti. Sovellus ei vaadi erillistä kutsua kaatumisen lähettämiseksi palveluun. Tämä tarkoittaa sitä, että tiedot kaatumisista virtaavat sovelluksesta automaattisesti järjestelmään. Flurryn kaatumisanalytiikka tukee ProGuard-kartoitustiedostojen automaattista lataamista, ja virheraportit ovat luettavissa Flurryn kaatumisanalyytikoista. Kun halutaan tiedot kiinni jääneistä poikkeuksista (exceptions) ja kirjattavista virheistä (errors) silloin, kun sovellus ei kaadu (esimerkiksi kun virhe siepataan), ne voidaan ottaa talteen vaihtoehtoisesti menetelmäkutsulla. [21.]

Poikkeukset ja virheet voidaan siepata FlurryAgentin onError-metodin avulla. Metodiin asetetaan kolme merkkijonotyyppistä parametria (virheen nimi, viesti ja poikkeus).

Kuvassa 23 on esimerkki poikkeuksen kirjaamisesta Flurryyn, kun se siepataan try-catch-lohkorakenteen sisällä.

```
@Override
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    try{
        viewModel.getMealModel(getPageIndex()).
                observe(getViewLifecycleOwner(), this::updateView);
    }catch (Exception e){
        FlurryAgent.onError( s: "MealFragment",  s1: "Meal viewModel", e);
    }
}
```

Kuva 23. Flurryn poikkeusmenetelmän kutsu

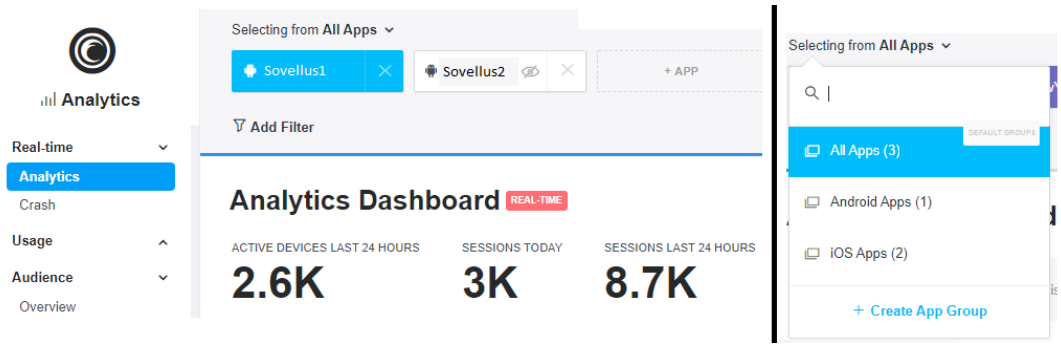
6 Tulokset

6.1 Flurry dashboard

Flurry-istunnot ja kerätyt data-aineistot ovat katseltavissa Flurryn kojelaudan hallintapaneelilta. Flurry SDK kommunikoi palvelimen kanssa vain kahdesti istunnon aikana. Ensimmäinen kommunikointi tapahtuu, kun istunto aloitetaan ja istunnon aikaleima asetetaan, uusi käyttäjä lasketaan tai nykyinen käyttäjä päivitetään aktiiviseksi. Toinen kommunikointi tapahtuu, kun istunto päättyy, ja kaikki tapahtumatiedot lähetetään palvelimelle yhdessä erässä. Tapauksissa, joissa tapahtumatietoja ei lähetetä palvelimelle, kutsutaan ”keskeneräiseksi istunnoksi”. Tähän voi olla syynä verkkoyhteyden puuttuminen tai sovelluksen taustallaolo. Tässä tapauksessa tapahtumadata tallentuu laitteen levylle, ja se lähetetään, kun sovellus käynnistetään seuraavan kerran. [22, viimeinen vastaus.] Tapahtumadatan tallennuttua palvelimelle voi kulua useampi tunti ennen kuin ne ovat nähtävissä kojelaudalta.

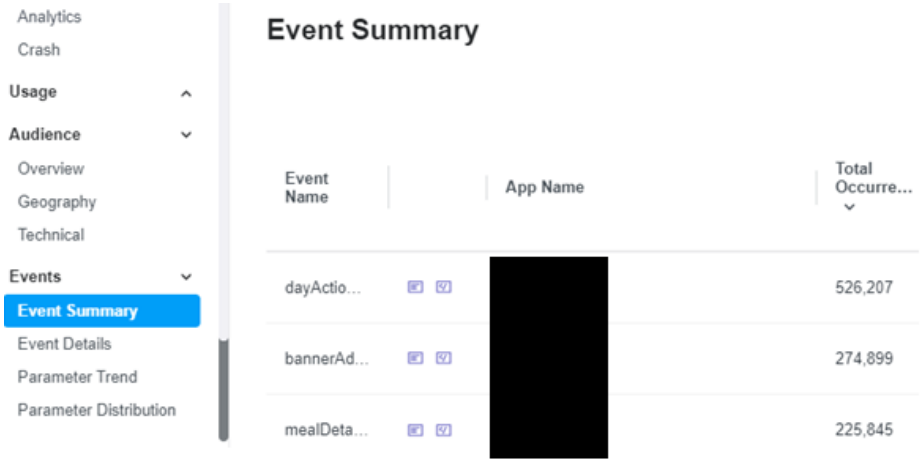
Flurryn verkkosivuille kirjautumisen jälkeen analytiikan hallintapaneeli on ensimmäinen asia, jonka kehittäjä näkee. Hallintapaneelista voidaan nähdä, hallita ja seurata kaikkia luotuja tapahtumia, istuntoja, yleisöä, kaatumisia ja sovelluksen käyttöä.

Hallintapaneelilla on tehokkaat kontekstiohjaimet, joiden avulla voidaan tarkastella sovellusten tietoja halutulla tavalla. Paneelilla on mahdollista asettaa kontekstilaatikkoon samaan aikaan useampi kuin yksi sovellus (ks. kuva 24, vasen puoli). Kontekstilaatikoissa olevat sovellukset voidaan asettaa näkyviksi tai näkymättömiksi; näkyvät sovellukset ovat värillisiä. Sivun yläreunassa on myös avattava pudotusvalikko ”All Apps”, josta hallitaan sovelluksia ja sivun kontekstia (kuva 24, oikea puoli). Asetettu sovelluksen konteksti tai sovellusten kontekstit hallitsevat tietoja, jotka tulevat sivulle näkyviin. [23.]



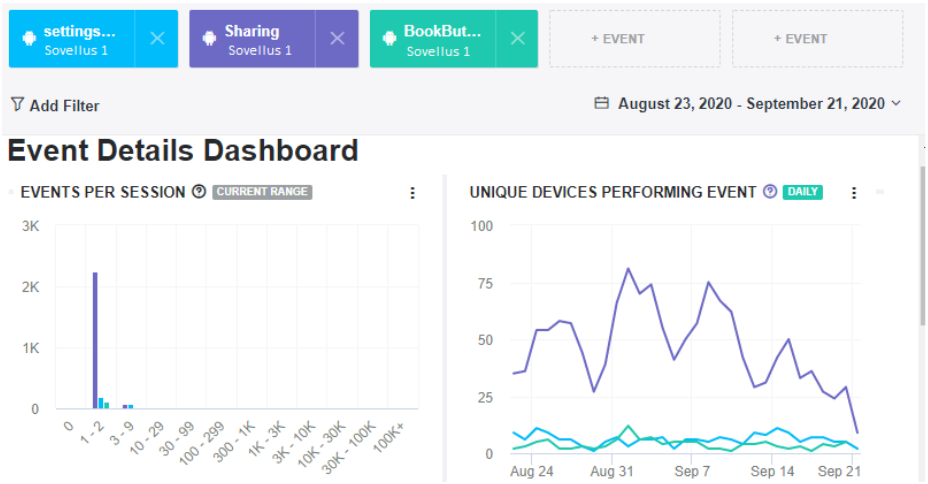
Kuva 24. Flurryn hallintapaneelin kontekstiohjaimet

Flurryn hallintapaneelin tapahtumiin sisältyvät tapahtumien yhteenveto-, tapahtumatieto-, parametritrendi- ja parametrien jakelu -näkymät. Tapahtumien yhteenvetonäkymä tarjoaa taulukkonäkymän kaikista valituista sovelluskokoelman tapahtumista (ks. Kuva 24). Taulukko sisältää tiedot esiintyneiden tapahtumien kokonaismäärästä sekä istunto- ja laitekohtaiset keskiarvot. Taulukkonäkymästä valittu tapahtuma pystytään linkittämään tapahtumatieto- tai parametrien jakelu -sivuille. [24.]



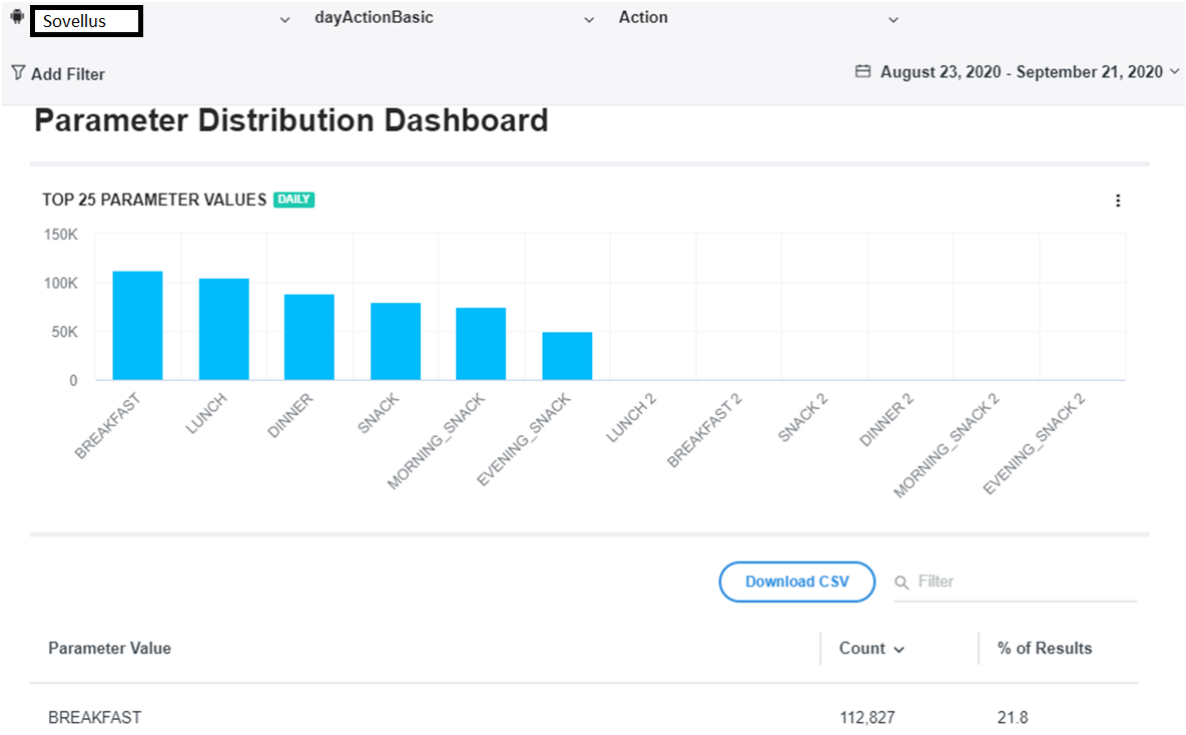
Kuva 24. Flurryn tapahtumien yhteenvetonäkymä

Flurryn tapahtumatietohallintapaneeli tarjoaa jokaiselle merkitylle tapahtumalle erilaiset perustiedot. Hallintapaneelissa on useita erilaisia kaavionäkymiä sovellusten tallentamista tapahtumatiedoista (ks. kuva 25). Kaavioiden ansiosta on helppo nähdä ja vertailla useamman eri tapahtuman perustietoja asetetun aikavälin aikana. Näitä perustietoja ovat muun muassa tapahtumien päivittäiset lukumäärät, tapahtumien jakautuminen istuntoa kohti, tapahtuman keskimääräinen kesto ja laitekohtaiset tiedot. [24.] Kaavioiden taustalla olevat tiedot ovat ladattavissa hallintapaneelilta CSV-tiedostoihin, joita voidaan avata useimpien taulukkolaskentaohjelmien avulla (kuten Microsoft Excel).



Kuva 25. Flurryn tapahtumatietohallintapaneeli

Flurryn parametrien jakelu -sivulla tapahtumista voidaan tarkastella valittujen parametrien arvojen jakaumaa (ks. kuva 26). Sivun yläreunasta löytyvät ohjaimet tapahtuman ja niiden parametrien määrittelylle, jotka määrittelevät sivulle piirrettävät arvot. Parametrien arvot piirretään näkymän yläpuolella olevaan kaavioon ja alapuolella olevaan taulukkoon. Jokaisella parametrilla on kaikista tuloksista sekä lukumääräiset että prosentuaaliset arvot.



Kuva 26. Flurryn parametrien jakelu -sivu

6.2 Istunnot ja tapahtumat

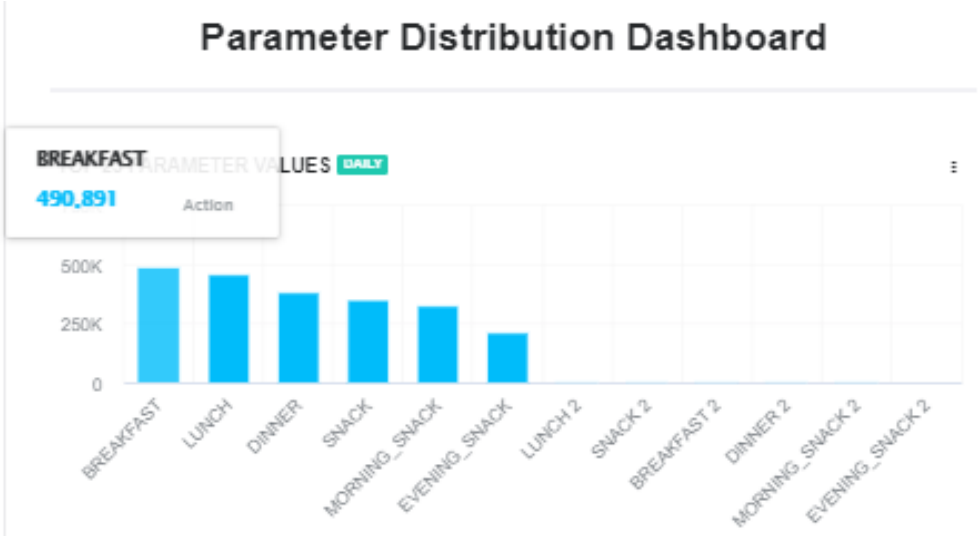
Android-sovelluksiin kehitettiin useita kymmeniä tapahtumia, joita seurattiin noin puolen vuoden ajan. Kun sovelluksen sisällä tapahtui yksi tai useampi tapahtuma tai kaatuminen, istunnosta saatu data kirjattiin automaattisesti Flurryyn yrityksen toiminnan hallintapaneelille. Kirjatun datan avulla pystyttiin seuraamaan, kuinka monella laitteella Flurryyn raportoitiin ja kuinka usein. Sovelluksista saatiin kerättyä puolen vuoden aikana runsas data-aineisto. Hallintapaneelilta näkyy, että istuntoja oli puolen vuoden aikana

yhteensä noin 2,2 miljoonaa 36 500 eri laitteessa. Eli Flurryyn raportoitiin päivittäin noin 11 000 istuntoa 2 300 laitteesta.

Sovellusten fragmenteille asetettiin tapahtumia erilaisten tietojen hakemista varten. Talteen otettiin seuraavia tietoja: valinnat, nappien painallukset, käyttäjien syöttämät arvot, sivuilla vietetty aika ja erilaiset kulkureitit (esim. miten käyttäjä pääsi fragmentilta A fragmentille C). Saatujen tietojen avulla pyrittiin vastaamaan erilaisiin kysymyksiin ja saamaan tieto siitä, mitkä asiat ja ominaisuudet olivat yhteisiä eri käyttäjille. Saatujen tietojen ansiosta pystyisimme parantamaan sovelluksen käytettävyyttä ja päättämään, miten kehitystä jatkettaisiin eteenpäin.

Työssä luotiin runsaasti tapahtumia, joista annan seuraavaksi muutaman esimerkin. Tapahtuma nimeltä dayActionBasic kerää sovelluksen pääsivun kuudelta eri ateriaruudulta tapahtumaparametriksi sen ruudun nimen, jonka käyttäjä valitsi sovellusta käyttäessään. Tämän parametriarvon avulla pystyttiin määrittelemään ateriat, joita käyttäjät valitsivat eniten tai vähiten, ja se, kuinka usein ja kuinka monta ateriaa käyttäjät kuvasivat päivittäin. Puolen vuoden aikana tapahtumaa kutsuttiin yhteensä 2,25 miljoonaa kertaa eli päivittäin noin 12 000 kertaa. Tulokset saatiin aikavälillä 1.3.–1.9.2020.

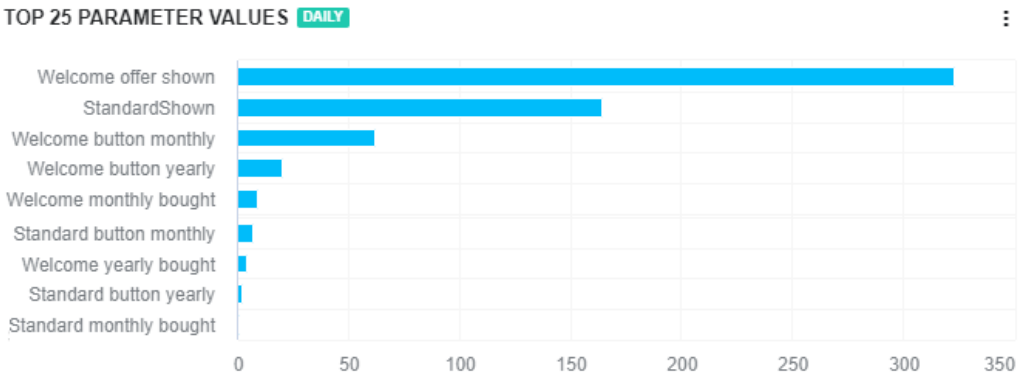
Kuvassa 27 on pylväsdiagrammi tapahtuman dayActionBasic tuloksista. Jokaisella ateriaruudulla on mahdollista lisätä kaksi ateriaa, joten tulokset sisältävät yhteensä 12 erilaista parametriarvoa. Esimerkiksi aterialla aamupala on parametriarvot BREAKFAST ja BREAKFAST 2. Tuloksista näkyy, että toisen aterian lisääminen yhdelle ateriaruudulle on lähes olematon arvo eli käyttäjät eivät käytä sovelluksen tätä ominaisuutta. Sovelluksen käytetyimmät ateriat ovat aamupala, lounas ja illallinen, joista aamupala voittaa ylivoimaisesti arvolla 490 891. Päivän aikana syötävät välipalat sijoittuvat näiden aterioiden alapuolelle, eikä välipalalla ei ole kovinkaan suurta eroa illalliseen.



Kuva 27. Tapahtuman dayActionBasic tulokset

Kuvassa 28 on diagrammi tapahtumasta offerAll, jota koskevat tulokset kerättiin aikavälillä 1.8.–1.9.2020. Sovelluksessa on kahdeksan offer-nimistä tapahtumaa, joista offerAll pitää sisällään seitsemän muun tapahtuman tulokset. Tapahtuman tarkoituksena on kerätä sovelluksen kahdelta eri myyntisivulta käyttäjän tekemiä toimintoja. Kun sovelluksen ilmaisversion käyttäjä yrittää käyttää Premium-ominaisuutta, hänelle näytetään toinen näistä myyntisivuista. Käyttäjän tekemät toiminnot kirjataan Premium-ominaisuutta vastaavaan tapahtumaan ja tapahtumaan offerAll. Tapahtumat ovat melko uusia, joten ne eivät sisällä vielä paljon dataa.

Tapahtuman tuloksiin kerättiin arvot Welcome offer shown ja StandardShown aina, kun jompikumpi myyntisivuista avautui käyttäjälle. Myyntisivuilla on kaksi nappia, joista käyttäjä pystyy ostamaan Premiumiin joko kuukausi- tai vuositilauksen. Nappien painallusten arvot ja ostot kerätään tapahtuman tuloksiin heti, kun käyttäjä on maksanut tilauksen. Saatujen tulosten avulla oli mahdollista laskea, kuinka usein käyttäjät ostavat Premiumin myyntisivulta ja päätellä seitsemästä muusta tapahtumasta, millä ominaisuudella on suurin vaikutus ostojen tekemiseen. Esimerkiksi Welcome Offer -myyntisivu näytettiin 328 kertaa ja kuukausittainen osto suoritettiin 9 kertaa eli kuukausitilaus tehtiin 2,8 %:n varmuudella aina, kun myyntisivu näytettiin.



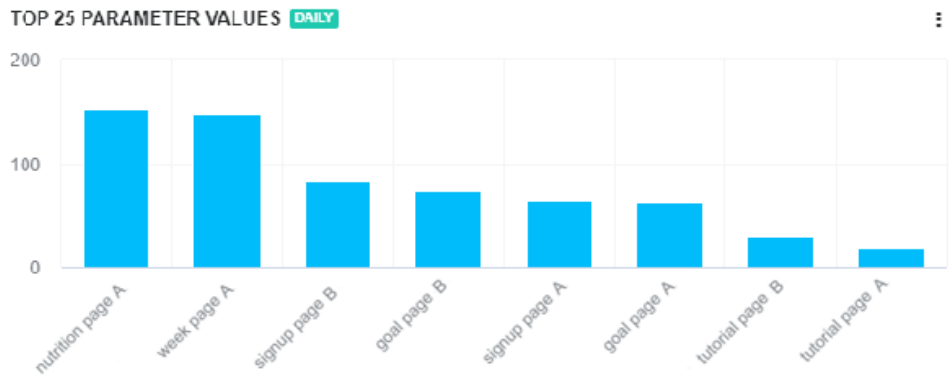
Kuva 28. Tapahtuman offerAll tulokset

OnBoarding eli perehdytys on mekanismi, jonka avulla sovellus esitellään sitä ensimmäistä kertaa käyttävälle henkilölle, jotta hän saisi tarvittavat taidot ja tiedot sen käyttämiseksi. Perehdytyksessä on tarkoitus esittää sovellusta koskevat perustiedot, kiinnittää huomiota sovelluksen huomionarvoisiin ominaisuuksiin ja kuvata kaikki vaaditut tai suositellut vaiheet, jotka käyttäjän tulisisi käydä läpi, kun hän käyttää sovellusta ensimmäistä kertaa. [25.]

Perehdytys sovellukseen tapahtuu WelcomeActivity-nimisen aktiviteetin kautta. Aktiviteetti sisältää ViewPager-objektin, johon kiinnitetään useita erilaisia perehdytykseen tarkoitettuja fragmentteja. Perehdytys tapahtuu useammalla eri askeleella: yksi fragmentti on aina käyttäjän nähtävissä ja hän siirtyy fragmentista toiseen pyyhkäisemällä. Fragmentit kertovat käyttäjille kaiken tarvittavan tiedon sovelluksen käytöstä ja sen ominaisuuksista. Yksi fragmenteista sisältää rekisteröintisivun.

Sovelluksiin suunniteltiin kaksi vaihtoehtoista reittiä käyttäjien perehdyttämistä varten. Reittien erona on fragmenttien erilainen ulkonäkö ja niiden erilainen järjestys. Perehdytykselle suunniteltiin A/B-testaus eli jaettu testaus. Tämä tarkoittaa prosessia, jossa käyttäjille näytetään satunnaisesti kahden tai useamman sivun muunnelma, minkä jälkeen määritellään analyysin avulla, mikä muunnelmista tuotti parhaimmat tulokset [26]. Tulosten avulla määritellään, kumman perehdytysmuunnelman isompi käyttäjäryhmä läpäisi ja kummalla muunnelmalla oli enemmän rekisteröityneitä käyttäjiä.

Kuvassa 29 esitetään tapahtuman onBoadingSteps-tulokset, jotka kerättiin ajanjaksolla 1.8.–1.9.2020. Parametri merkitään Flurryyn heti, kun käyttäjä on siirtynyt yhdeltä perehdytyksen askeleelta seuraavalle. Tulokset sisältävät perehdytyksen muunnelmat A ja B, joista A sisältää viisi perehdytysaskelta ja B kolme perehdytysaskelta. Muunnelma A alkaa ravintoainesivulta ja päättyy tutoriaalisivulle, kun taas muunnelma B alkaa rekisteröintisivulta ja päättyy tutoriaalisivulle. Muunnelma A aloitettiin 152 kertaa ja päätettiin 18 kertaa, kun taas muunnelma B aloitettiin 84 kertaa ja päätettiin 30 kertaa. Tulosten perusteella muunnelma B on parempi, koska perehdytysmuunnelman A käy läpi kokonaan 11,8 % käyttäjistä, kun taas perehdytyksen B käy läpi 36 % käyttäjistä. Tuloksissa olisi otettava huomioon myös, että käyttäjät voivat ohittaa perehdytyksen minkä tahansa askeleen aikana.



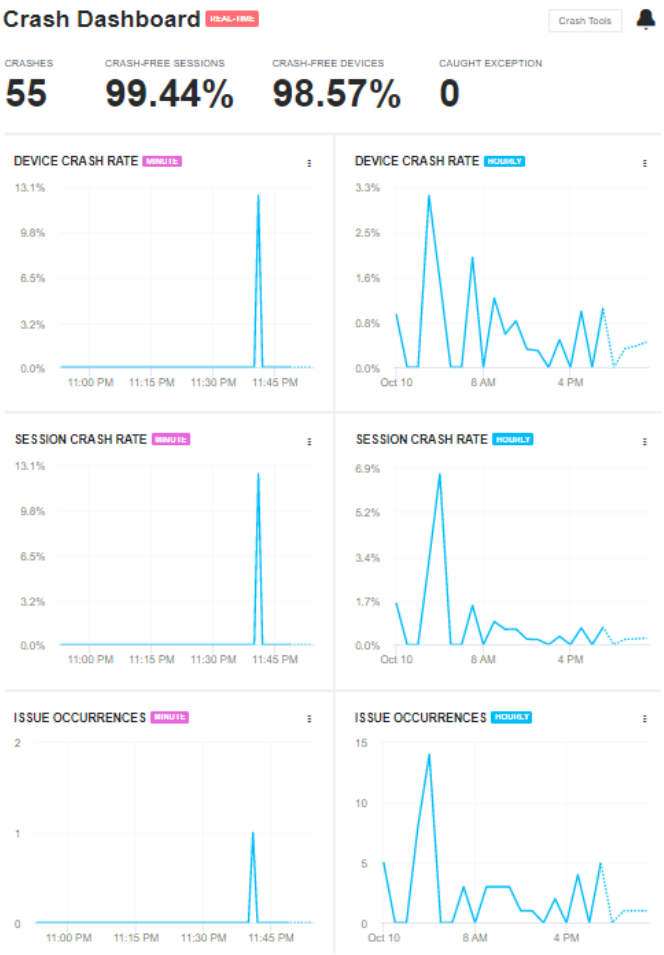
Kuva 29. Tapahtuman onBoardingSteps tulokset

6.3 Kaatumiset

Flurryn kaatumisten reaaliaikaiselta hallintapaneelilta näkee reaaliajassa tietoja sovelluksen kaatumisista, poikkeuksista ja virheistä. Kaatumisia varten on kaksi ajanjaksoa, jotka määrittelevät sen, mitä tietoja paneelilla näytetään. Nämä ajanjaksot ovat minuutti minuutilta viimeisen tunnin ajalta ja tunnista tuntiin viimeisten 24 tunnin ajalta. Tiedot kaatumisista esitetään kummankin ajanjakson osalta kolmessa erilaisessa kaaviossa. Ensimmäinen kaavioista on laitteiden vaikutusaste eli laitteiden määrä

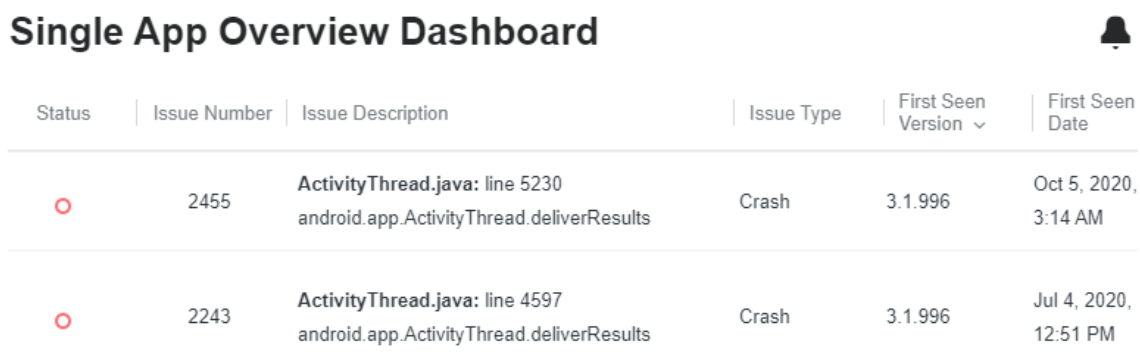
jaettuna ajanjakson niiden aktiivisten laitteiden määrällä, joilla ongelma esiintyi. Toinen kaavio on istuntojen vaikutusprosentti eli istuntojen määrä jaettuna ajanjaksolla tapahtuneiden niiden istuntojen määrällä, joissa ongelma esiintyi. Kolmas kaavioista on ongelmien esiintymälaskuri eli ajanjaksolla tapahtuneiden ongelmien määrä. [27.]

Kuvassa 30 näkyy Flurryn kaatumisten reaaliaikainen hallintapaneeli. Sivun yläosassa ovat perustiedot viimeisten 24 tunnin aikaisista sovellusten kaatumisista eli kaatumisten lukumäärä, kaatumattomien istuntojen ja kaatumattomien laitteiden prosenttiosuudet ja kiinniotettujen poikkeusten lukumäärä. Kuten kuvasta käy ilmi, sovelluksen kaatumisia tapahtui tänä aikana yhteensä 55 kappaletta. Perustietojen alapuolella ovat kahdelta eri ajanjaksolta kaaviot, jotka sisältävät tiedot laitteiden vaikutusasteesta, istuntojen vaikutusprosentista ja ongelmien esiintymälaskurista.



Kuva 30. Kaatumisten reaaliaikainen hallintapaneeli

Reaaliaikaiselta hallintapaneelilta (ks. kuva 30) pääsee tarkastelemaan kaatumisten virheraportteja oikean yläkulman kaatumistyökalu Crash tools -napista valitsemalla sivun Yhden Sovelluksen Yleiskatsastus (Single App Overview). Yhden Sovelluksen Yleiskatsastus -sivu antaa yhteenvedon sovellukseen vaikuttavista ongelmista. Sivu sisältää taulukon kaikista sovellukseen vaikuttavista ongelmista ja tiedot, joissa eritellään ongelmatyypit kaatumisten, kiinni jääneiden poikkeusten ja kirjattavien virheiden välillä. Taulukossa esitetään koko ajanjakson aikana tapahtuneet ainutlaatuiset ongelmat ja monessa eri sarakkeessa kuvataan ongelmaan liittyviä tietoja. Näitä tietoja ovat muun muassa ongelman numero, kuvaus ja päivät sekä versiot ensimmäisestä ja viimeisestä todetusta ongelmasta. Kuvassa 31 näkyy, miltä Yhden Sovelluksen Yleiskatsastus -sivun ongelmataulukko näyttää.



Kuva 31. Yhden Sovelluksen Yleiskatsastus -sivun taulukko

Napsauttamalla ongelmaa Yhden Sovelluksen Yleiskatsaus -sivun taulukossa pääsee Ongelman Tiedot -sivulle. Sivulla on tietoja tiettyyn ongelmaan liittyvistä seikoista, kuten erilaisia esiintymismittareita, laitteeseen ja Android-versioon liittyviä tietoja ja virheraportti esiintyneestä ongelmasta. Ongelman tiedot -sivulta löytyvät kaikki tiedot, jotka auttavat kehittäjää löytämään ongelman perimmäisen syyn ja helpottavat sovelluksessa olevien virheiden korjaamista.

Kuvassa 32 näkyy Ongelman Tiedot -sivu sovelluksen fragmentilla tapahtuvasta kaatumisesta. Sivun yläosassa ovat kaaviot sovellusversioista, käyttöjärjestelmistä, laitamalleista ja maista, joissa kaatuminen on tapahtunut. Sivun alaosasta löytyy virheraportti, joka kertoo kaatumisten syyt ja paikan sovelluksen koodissa. Tässä tapauksessa kaatumisen syynä oli poikkeus NullPointerException.

Kuva 32. Ongelman Tiedot -sivu sovelluksen kaatumisesta

7 Yhteenveto

Opinnäytetyö alkoi tilanteesta, jossa yrityksellä ei ollut mitään keinoa kerätä kahden Android-sovelluksen käyttöä koskevia kuluttajatietoja. Työssä kehitettiin yrityksen tarpeita vastaava menetelmä käyttäen Flurry Analytics -analytiikkatyökalua.

Android-sovelluksiin integroitiin Flurry Analytics SDK, jonka avulla yritys sai käyttöönsä joukon analytiikkatyökaluja. Sovelluksiin määriteltiin useita erilaisia tapahtumakutsuja ja virheiden menetelmäkutsuja, jotta kerätystä data-aineistosta saataisiin koottua parhaat mahdolliset raporttitulokset. Kerätty data-aineisto analysoitiin Flurryn nettisivujen kautta,

sillä sivut tarjoavat sovellusta koskevia perustietoja, taulukkonäkymiä ja erilaisia mittareita, joiden avulla voidaan seurata käyttäjien tapoja ja sovelluksen suorituskykyä.

Flurryyn kerättiin puolen vuoden aikana noin 2,2 miljoonaa istuntoa, jotka sisälsivät runsaasti tapahtumadataa. Kerätyistä data-aineistoista yritys sai hyviä ideoita Android-sovellusten parantamiseksi ja kehittämiseksi edelleen sekä käyttäjien käyttökokemuksen ja tuotteen laadun parantamiseksi. Ostoista ja myyntisivuilta kerätyt tiedot ovat hyvin tärkeitä sijoittajille ja myynnin parantamiseksi.

Flurryn kaatumisanalytiikat tarjosivat runsaasti tietoa sovellusten sisäisistä kaatumisista, virheistä ja poikkeuksista. Reaaliaikainen hallintapaneeli varmisti sen, että kehittäjä oli aina tietoinen sovelluksen sisäisistä ongelmista, jotta ne saataisiin korjattua mahdollisimman nopeasti. Flurry tarjosi myös mahdollisuuden kaapata sovellusten virheitä try/catch-lohkorakenteiden sisältä menetelmäkutsuilla, joita Google Developer Consolen virheenraportointi ei tukenut.

Fluryyn tutustuminen ja sen käyttöönotto vei työn alussa melko paljon aikaa. Kirjastoa koskevia ohjeita tai dokumentaatiota ei löytynyt juuri muualta kuin Yahoon omilta verkkosivuilta. Kehittäjän on hyvä tutustua dokumentaatioon huolella, jotta hän on tietoinen siitä, mitä ominaisuuksia kirjasto tukee ja mitä se ei tue. Dokumentaation ansiosta Flurryn integrointi sovelluksiin oli helppo ja suoraviivainen prosessi. Flurryn käyttöönotossa vei eniten aikaa mukautettujen tapahtumien logiikan ja luokittelun suunnittelu ja kehittäminen. Tapahtumien nimeäminen ja parametrit on suunniteltava hyvin, jotta ne olisivat helposti ymmärrettävissä ja muistettavissa. Hyvän rakenteen ansiosta data-aineistot ovat helposti luettavissa Flurryn kojelaudalta.

Flurry oli erittäin joustava työkalu ja tarjosi monipuolisia ratkaisuja käyttäjien seuraamiseksi. Hyvinä puolina Flurryssa olivat monipuoliset tavat kerätä tietoa käyttäjistä (kuten tapahtumat, monetisaatio ja kaatumiset). Paras ominaisuus kirjastossa olivat mukautetut tapahtumat, jotka antoivat kehittäjälle täyden vapauden räätälöidä menetelmäkutsut keräämään juuri sellaisia tietoja, joita yritys halusi.

Hyvien puolten ohella löytyi Flurrysta parannettavaakin. Kirjasto ei sisältänyt omaa analytiikkaa A/B-testausmuunnoksille, joita projektissa käytettiin melko paljon. Kehittäjän

oli suunniteltava logiikka itse olemassa olevien menetelmien avulla. Lisäksi Flurryn verkkosivujen kojelauta oli hieman hämmentävä, mikä voi tuottaa ongelmia ensikertaiselle käyttäjälle. Tietyt seikat ovat myös hieman piilossa kojelaudalla, ja käyttöliittymässäkin on parantamisen varaa. Kehittäjillä pitäisi olla enemmän vapautta muokata UI-elementtejä, kuten tietojen näkyvyyksiä ja ryhmittelyä.

Ennen työn aloittamista kävin yrityksen kanssa läpi useita eri analytiikkatyökaluvaihtoehtoja. Flurry Analyticsin ohella muita analytiikkatyökaluja olivat muun muassa Google Analytics, MixPanel ja Firebase. Nämä analytiikkatyökalut ovat kaikki yhtä suosittuja ja sovellusten omistajat käyttävät niitä yhä enemmän. Niiden joukosta parhaimmiksi osoittautuivat Flurry ja Google Analytics. Flurry Analytics keskittyy ensisijaisesti mobiilisovelluksiin (kattavat mobiilityökalut ja mittarit), kun taas Google Analytics ensisijaisesti verkkosivustoihin. Molemmista työkaluista puuttui edistyneitä ominaisuuksia, kuten yksittäisten käyttäjien seuranta. Koska eri analytiikkatyökalut tarjoavat erilaisia ominaisuuksia, voisi tulevaisuudessa olla hyvä ottaa käyttöön useampi analytiikkatyökalu, jotta yritys löytäisi niiden joukosta sen, joka on sille mieluisin. Tässä tapauksessa Flurry Analytics vastasi parhaiten yrityksen tarpeita ja onnistui saatujen tulosten perusteella hyvin tehtävässään.

Lähteet

1 Android Studio. Wikipedia. Verkkoaineisto. <https://en.wikipedia.org/wiki/Android_Studio> Luettu 5.5.2020.

2 Introduction to Gradle for Android Studio. Gradlen esittely. Verkkoaineisto. <https://www.studytonight.com/android/introduction-to-gradle> Luettu 5.5.2020.

3 Run apps on the Android Emulator. Android Developers-dokumentaatio. Verkkoaineisto. <https://developer.android.com/studio/run/emulator> Luettu 5.5.2020.

4 What is an APK file and how to install APKs on Android?. Verkkoaineisto. Verkkoaineisto. <https://www.nextpit.com/android-for-beginners-what-is-an-apk-file#what> Luettu 5.5.2020.

5 Configure Build Variant. Android Developers-dokumentaatio. Verkkoaineisto. <https://developer.android.com/studio/build/build-variants> Luettu 5.5.2020.

6 Configure your build.Android Developers-dokumentaatio. Verkkoaineisto. <https://developer.android.com/studio/build> Luettu 5.5.2020.

7 Google Play. Wikipedia. Verkkoaineisto. <https://en.wikipedia.org/wiki/Google_Play> Luettu 5.5.2020.

8 Android Architecture Components. Android-arkkitehtuurikomponentit. Verkkoaineisto. <https://developer.android.com/topic/libraries/architecture> Luettu 2.11.2020.

9 Introduction to Activities. Android Developers-dokumentaatio. Verkkoaineisto. <https://developer.android.com/guide/components/activities/intro-activities> Luettu 15.5.2020.

10 Understand the Activity Lifecycle. Android Developers-dokumentaatio. Verkkoaineisto. <https://developer.android.com/guide/components/activities/activity-lifecycle> Luettu 15.5.2020.

11 Fragments. Android Developers-dokumentaatio. Android Developers-dokumentaatio. <https://developer.android.com/guide/components/fragments> Luettu 15.5.2020.

12 Fragment Lifecycle. Android Developers-dokumentaatio. Verkkoaineisto. <https://docs.microsoft.com/en-us/xamarin/android/platform/fragments/creating-a-fragment> Luettu 15.5.2020.

13 Android – SQLite Database. SQLite-dokumentaatio. Verkkoaineisto. <https://www.tutorialspoint.com/android/android_sqlite_database.htm#:~:text=SQLite%20is%20a%20opensource%20SQL,all%20the%20relational%20database%20features.> Luettu 20.7.2020.

14 Using Room Database | Android Jetpack. 23.2.2019. Verkkoaineisto. <https://medium.com/mindorks/using-room-database-android-jetpack-675a89a0e942> Luettu 23.7.2020.

15 Save data in local database using Room. Room-dokumentaatio. Verkkoaineisto. <https://developer.android.com/training/data-storage/room> Luettu 23.7.2020.

16 What is a Relational Database?. AWS määritelmä. Verkkoaineisto. <https://aws.amazon.com/relational-database/#:~:text=A%20relational%20database%20is%20a,be%20represented%20in%20the%20database.> Luettu 23.7.2020.

17 Data Access Objects — DAO in Room. DAO-dokumentaatio. Verkkoaineisto. <https://medium.com/mindorks/data-access-objects-dao-in-room-3d108d6b4b54> Luettu 23.7.2020.

18 Flurry (company). Wikipedia. Verkkoaineisto. <https://en.wikipedia.org/wiki/Flurry_(company)> Luettu 10.9.2020.

19 Custom Events with Flurry Analytics for Android. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/events/android/> Luettu 2.8.2020.

20 Advanced Features for Flurry Analytics with Android. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/technicalquickstart/android/> Luettu 2.8.2020.

21 Crash Analytics Android Instrumentation. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/trackcrashes/android/> Luettu 2.8.2020.

22 Hunter. Flurry parameters not showing up. StackOverFlow-vastaus. <https://stackoverflow.com/questions/29810753/why-are-my-new-flurry-parameters-not-showing-up> Luettu 20.8.2020.

23 Context. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/lexicon/contextblocks/> Luettu 2.8.2020.

24 Events for Flurry Analytics. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/lexicon/eventreporting/> Luettu 2.8.2020.

25 Introduce first-time users to your app. Verkkoaineisto. <https://developer.android.com/training/tv/playback/onboarding> Luettu 6.8.2020.

26 A/B Testing Guide. A/B tutoriaali. Verkkoaineisto. <https://vwo.com/ab-testing/> Luettu 14.9.2020.

27 Crash Analytics. Flurry-dokumentaatio. Verkkoaineisto. <https://developer.yahoo.com/flurry/docs/analytics/crash/> Luettu 6.8.2020.