Tampereen ammattikorkeakoulu

# Yleinen monialustainen kuittaussovellus

Joonas Pennanen

OPINNÄYTETYÖ
Huhtikuu 2022

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

PENNANEN, JOONAS:
Yleinen monialustainen kuittaussovellus

Opinnäytetyö 33 sivua, joista liitteitä 0 sivua
Huhtikuu 2022

---

Tutkimusten mukaan maailman väestöstä 63 % käyttää internetiä jollakin laitteella. Näistä viidestä miljardista käyttäjästä noin 92 % käyttää mobiililaitetta. Mobiililaitteilla tunnistautumista käytetään nykyään useissa eri verkkopalveluissa ja niiden tunnistautumistaso ja toiminta vaihtelevat.

Opinnäytetyössä tehtiin Eduix Oy:lle mobiilisovellus, joka toimii kuittaussovelluksena liitännäisohjelmistojen kanssa vahvan tunnistautumisen sijaan. Sovelluksen avulla liitännäisohjelmiston lähettämä pyyntö voidaan vastaanottaa käyttövaltuudet omaavan henkilön mobiililaitteella, jonka käyttäjä voi hyväksyä tai hylätä.

Sovellus suunniteltiin toimimaan natiivisti Android- ja iOS-alustoilla. Molemmille alustoille tehtiin omat versiot. Android-toteutuksessa ohjelmointi suoritettiin Kotlin-ohjelmointikielellä, kun iOS-toteutuksessa käytettiin Swift-ohjelmointikieltä. Molemmat toteutukset käyttävät Googlen Firebase BAAS -pilvipalveluita. Hyväksymispyynnöt välitetään Googlen Firestore messaging -palvelun avustuksella. Sovelluksen tietokanta on toteutettu Googlen Firebase Firestore -tietokannalla.

---

Asiasanat: mobiilisovellus, android, ios, swift, kotlin, google, firebase, firestore, kuittaus, tunnistautuminen

## ABSTRACT

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

PENNANEN, JOONAS
Generic Multiplatform Confirmation Application

Bachelor's thesis 33, appendices 0 pages
April 2021

---

According to research, 63 % of the world’s population is using internet with some type of device. From those 5 billion users 92 % use a mobile device. Authentication is used with various internet services and the level of security and purpose varies.

This thesis presents a mobile application created for Eduix Ltd. The purpose of the application is to receive incoming confirmation requests from pared software services. The application receives requests from pared services to authenticated users’ mobile devices. Authenticated users can accept or deny.

The mobile application was designed to function natively with Android and iOS devices. Native versions were created for both Android and iOS. Android was programmed in Kotlin programming language. The Swift programming language was used with iOS. Both platforms are using the same Google Firebase BAAS cloud service. Confirmation requests are transmitted via the Google Firestore messaging service. Applications database is implemented using Google Firebase Firestore Database.

---

Key words: mobile application, android, iOS, Swift, Kotlin, Google, Firebase, Firestore, confirmation, authentication

**SISÄLLYS**

**ERITYISSANASTO**

| | |
|---|---|
| Android | Android-puhelimien käyttöjärjestelmä |
| iOS | Applen-puhelimien käyttöjärjestelmä |
| NoSQL | Relaatiomallista poikkeava tietokanta |
| SQLite | Sovellukseen linkittyvä tietokannan hallintajärjestelmä |
| BaaS | Backed-as-a-Service, pilvessä toimiva taustajärjestelmäpalvelu |
| Xcode | Applen ohjelmointiympäristö |
| Android Studio | Android-käyttöjärjestelmän ohjelmointiympäristö |
| Funktio | Ohjelman hyödyntämä aliohjelma |
| Metodi | Luokan aliohjelma |

# 1 JOHDANTO

Henkilöllisyyden todentamista sähköisesti voidaan todentaa vahvalla sähköisellä tunnistamisella – kuluttaja voi käyttää turvallisesti sähköisiä palveluita verkossa, kun henkilöllisyyden vahvistaminen tapahtuu vahvan sähköisen tunnistautumisen avulla. Lainsäädäntö vaatii tunnistuspalveluiden käyttöä ja Kyberturvallisuuskeskus seuraa vaatimusten noudattamista ja tarvittaessa antaa määräyksiä, joilla tarkennetaan lakia sekä valvoo rekisteriä, jossa listataan vaatimukset täyttävät tunnistuspalvelun tarjoajat. (Kyberturvallisuuskeskus.fi.) Vastaavanlaista tunnistautumista käytetään nykyään useissa eri verkkopalveluissa ja niiden tunnistautumistaso ja toiminta vaihtelevat.

Maailman väestöstä 63 % käyttää internetiä jollakin laitteella. Näistä viidestä miljardista käyttäjästä 92.4 % käyttää mobiililaitetta. (Datareportal 2022.) Mobiililaitteiden etuna voidaan yleisesti nähdä niiden helppo kuljetettavuus ja käyttö. Mobiililaitteiden turvallisuutta lisää niiden omat tunnistautumiset. Pankkipalveluiden siirtyminen mobiililaitteille on toiminut esimerkkinä tunnistuspalveluille.

Tässä opinnäytetyössä esitellään Eduix Oy:lle tehty opinnäytetyö. Opinnäytetyönä tehtiin mobiilisovellus, joka ei toimi vahvana tunnistautumisena vaan kuittaussovelluksena liitännäisohjelmistojen kanssa. Sovelluksen avulla liitännäisohjelmiston lähettämä pyyntö voidaan vastaanottaa käyttövaltuudet omaavan henkilön mobiililaitteella, jonka käyttäjä voi hyväksyä tai hylätä erillisenä palveluna toimivan sovelluksen hyväksymispyynnön.

Sovellus on tarkoitettu tilanteisiin missä tarvitaan laajempaa päätöksentekokykyä kuin mitä tavallisella ohjelmistolla on tarjota tai nopeuttaa hyväksyntäprosessia siten, että luvanantajan ei tarvitse erikseen kirjautua nettipalveluun hyväksyäkseen pyynnön, vaan se voidaan tehdä suoraan mobiililaitteesta. Vertailtavana sovelluksena voidaan käyttää esimerkiksi Microsoftin Authenticatoria, jolla voidaan hyväksyä kirjautuminen Microsoftin palveluihin.

Eduix Oy on vuonna 1996 perustettu koulutuksen tarpeisiin digitaalisia palveluita tarjoava yritys, jonka asiakkaisiin kuuluu kunta- ja korkeakoulusektorilla toimivia tahoja. Eduixin palveluita ovat muun muassa opetuksenhallintajärjestelmä Peppi, vastausten keräämiseen tarkoitettu E-lomake ja opinnäytetyöprojektien ohjaus- ja hallintajärjestelmä Wihi. (Eduix 2022.) Opinnäytetyössä valmistuvan sovelluksen on tarkoitus olla liitännäiskelpoinen edellä mainittujen palveluiden kanssa.

# 2 TAVOITE JA TARKOITUS

Tavoitteena oli valmistaa sovellus, jolla voidaan vastata hyväksymispyyntöihin mobiililaitteilla. Sovellus vastaanottaa lähettäjän hyväksymispyynnön ja tekee siitä push-ilmoituksen käyttäjän laitteelle. Laite ilmoittaa käyttäjälle vastaanotetusta hyväksymispyynnöstä. Push-ilmoitus avaamalla tai sovelluksen kautta navigoimalla esitetään vastaanottajalle lisätietoa hyväksymispyynnöstä. Hyväksymispyyntö sisältää itse viestin, viestin tunnisteen, viestin otsikon, viestin tyypin, vastaanottajan sähköpostiosoitteen, vastauskentän, lähettäjän tunnisteen ja lähetysajan. Vastaanottajan vastaus välittyy lähettäjän tietokantaan, jolloin lähettäjä saa tiedon vastauksen saapumisesta. Sovellus luodaan Android- ja iOS-alustoille natiivina toteutuksena.

## 2.1 Toiminnalliset vaatimukset

Työn pitää täyttää seuraavat vaatimukset:

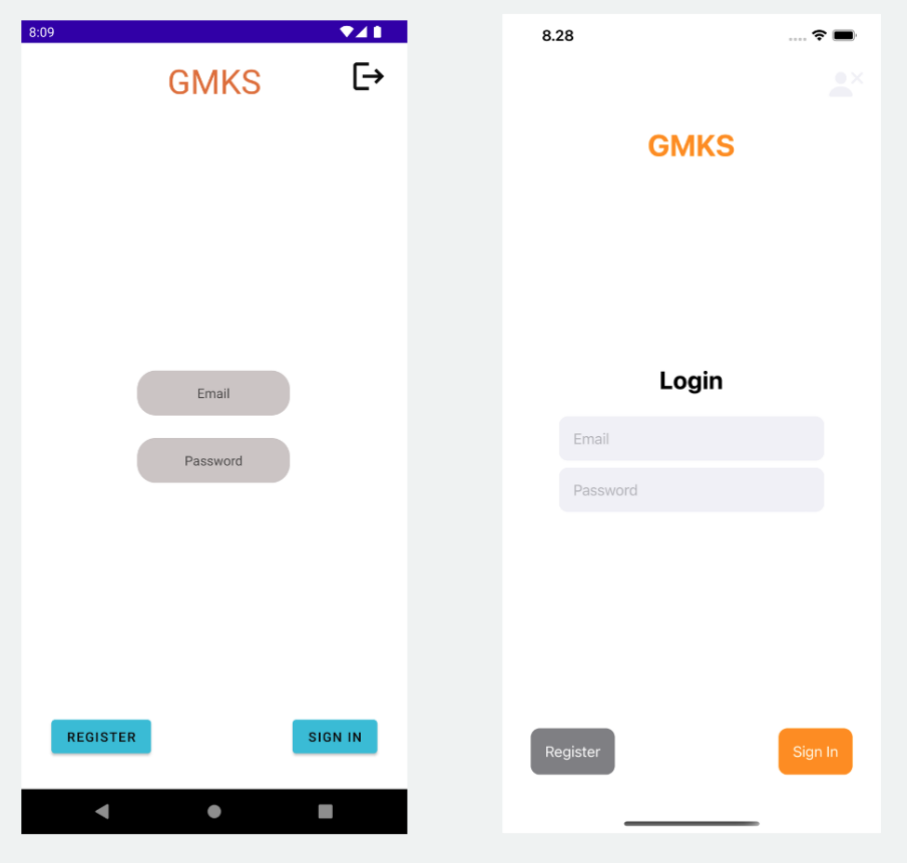
1. Sovelluksen pitää toimia Androidilla ja iOS:lla
2. Molempien alustojen sovelluksen tulee olla natiivi toteutus
3. Sovelluksen pitää pystyä vastaanottamaan push-ilmoituksia
4. Sovelluksen pitää pystyä vastaamaan vastaanotettuun push-ilmoitukseen
5. Vastausvaihtoehdot ovat ”kyllä” ja ”ei”
6. Vastauksen pitää päätyä sen lähettäjälle
7. Viestin pitää tallentua laitteen tietokantaan

## 2.2 Käyttöliittymä

Sovelluksen käyttöliittymä yritettiin tehdä mahdollisimman yksinkertaiseksi ja käyttää sellaisia ulkoasuratkaisuja, jotka ovat mobiililaitteiden käyttäjille jo entuudestaan tuttuja. Sovellus koostuu toistaiseksi kolmesta näkymästä.

### 2.2.1 Kirjautumisnäkymä

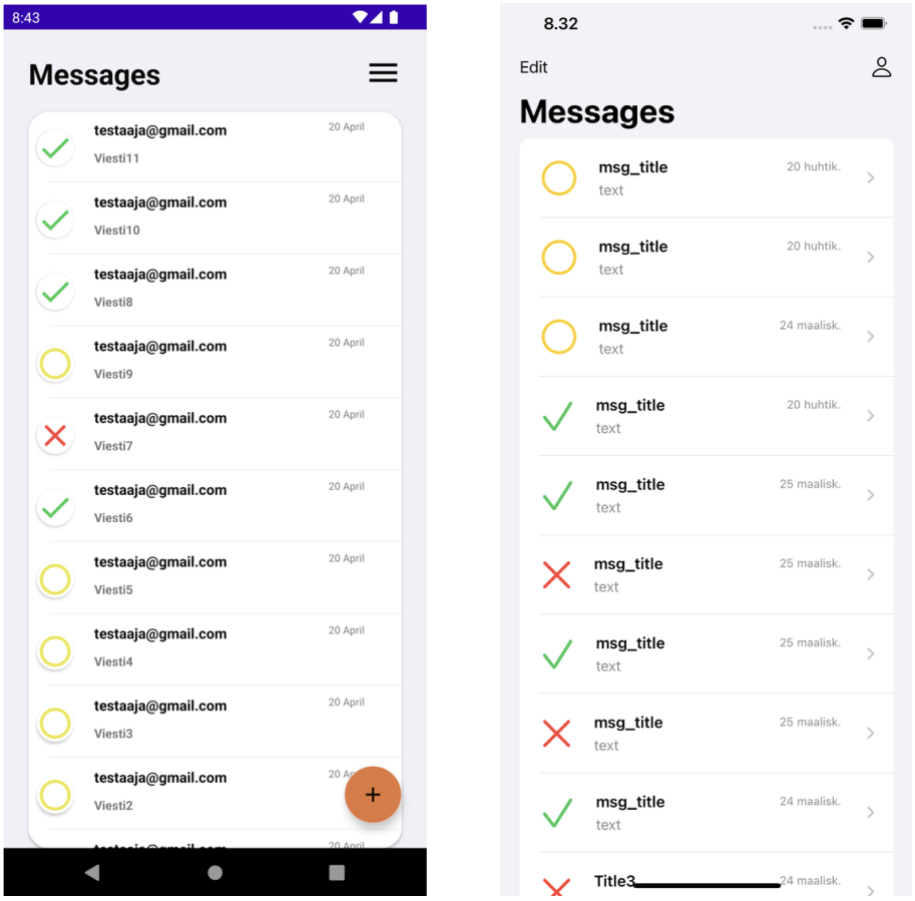
Sisäänkirjautumisessa on tekstinsyöttökentät sähköpostille sekä salasanalle. Alempana on kaksi painiketta; toinen rekisteröinnille ja toinen sisäänkirjautumiselle. Sisäänkirjautumisnäkymät kuvassa 1.




KUVA 1. Android- (vas.) ja iOS-kirjautumisnäkymä (oik.)

### 2.2.2 Kontaktinäkymä

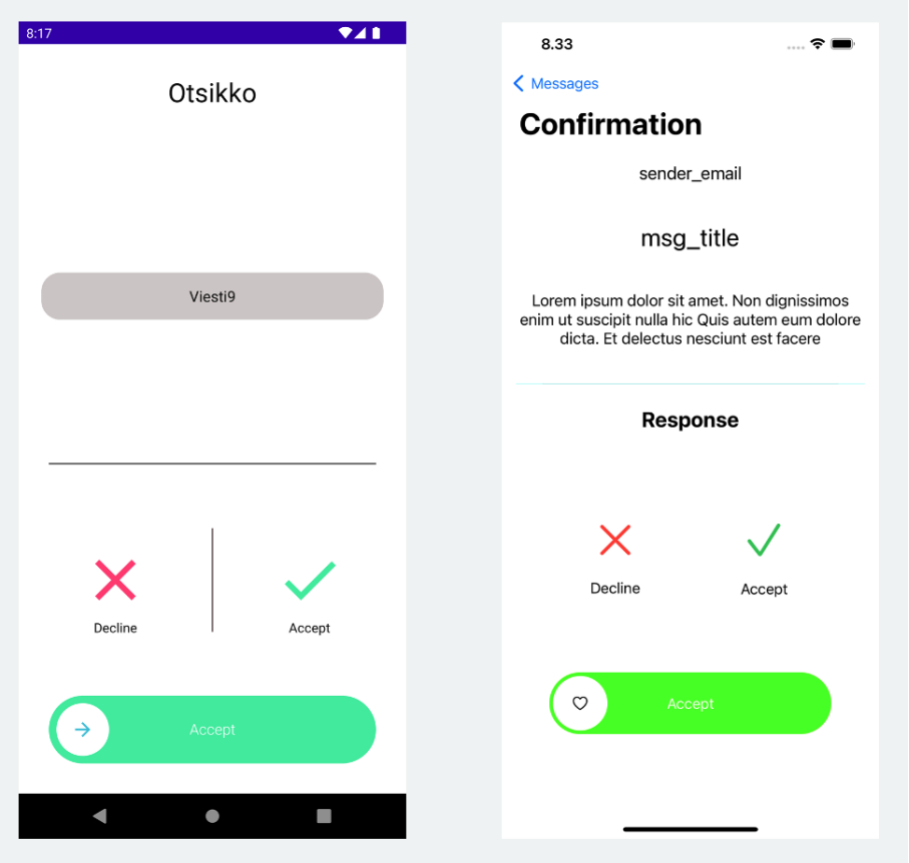
Kontaktinäkymässä (KUVA 2) on lista hyväksymispyynnöistä. Listan vasemmassa laidassa näkyvä ikoni kertoo hyväksymispyynnön tilan. Keltainen ympyrä tarkoittaa vastaamatonta pyyntöä, punainen raksi evättyä pyyntöä ja vihreä hyväksymismerkki hyväksyntää. Listassa näytetään myös lähettäjän sähköpostiosoite, viesti ja viestin saapumisajankohta. Viestiä painamalla pääsee tarkastelemaan saapunutta hyväksymispyyntöä. Kontaktinäkymä on saman tyyppinen kuin Whatsappin tai Outlookin viestinäkymä.




KUVA 2. Android- (vas.) ja iOS-kontaktinäkymä (oik.)

### 2.2.3 Viestinäkymä

Viestinäkymässä (KUVA 3) esitetään lähettäjän sähköpostiosoite ja itse viesti, joiden alapuolella on painikkeet: ”hylkää” ja ”hyväksy”. Kumpaa tahansa painiketta painamalla niiden alapuolelle ilmestyy liu`utin, millä varsinainen vastaus vahvistetaan. Tällä vältetään vahinkopainallukset. Vahvistuksen jälkeen liu`utin palaa takaisin paikoilleen ja sovellus ohjaa itsensä takaisin kontaktinäkymään. Jos hyväksymispyyntöön on jo vastattu, voi viestiä edelleen tarkastella, mutta näkymän painikkeet ovat lukittuja.




KUVA 3. Android- (vas.) ja iOS-viestinäkymä

# 3 SOVELLUKSESSA KÄYTETTÄVÄT TEKNOLOGIAT

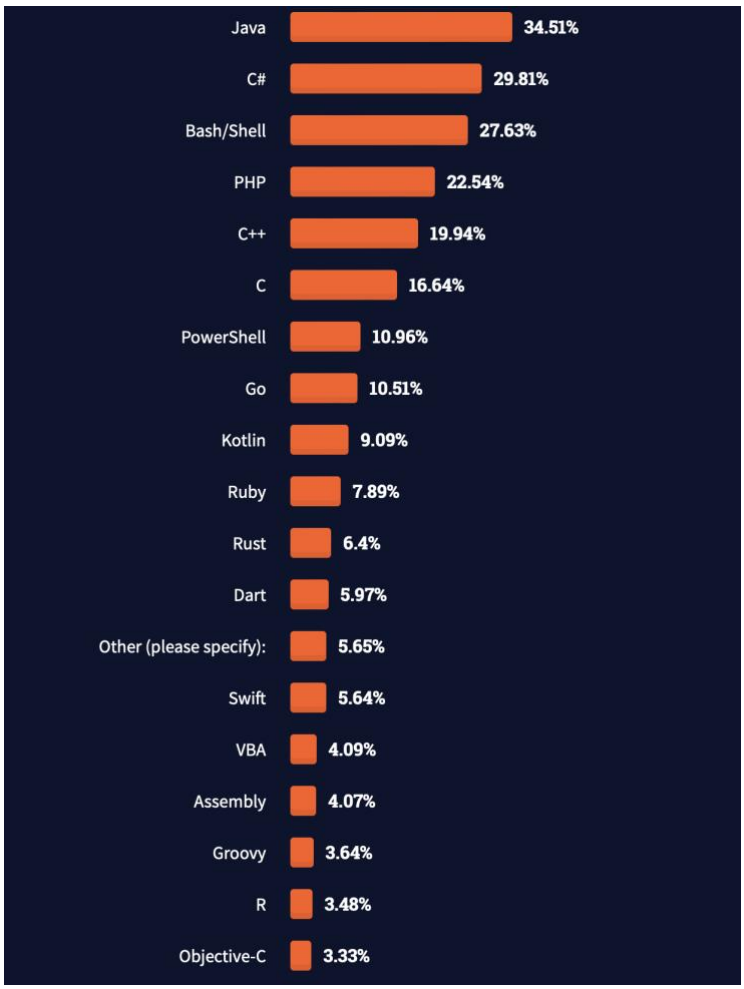
Työssä käytettävät teknologiat pohjautuvat laitevalmistajien tai käyttöjärjestelmäkohtaisiin kokonaisuuksiin, kuten tabletteihin, matkapuhelimiin ja älykelloihin. Mobiililaitteen käyttöjärjestelmä tarjoaa ohjelmoijalle tai loppukäyttäjälle mobiiliteknologioita esimerkiksi GPS-sijainnin, tekstiviestin tai push-ilmoitukset.

## 3.1 Kotlin

Kotlin mainostaa itseään modernina, ytimekkäänä ja turvallisena ohjelmointikielenä (Kotlin 2022). Lisäksi se on Googlen suosittelema kieli Android-kehitykseen (Developer Android 2022). Vaihtoehtona Kotlinille olisi ollut Java, jota suosii huomattavasti suurempi osa ohjelmoijista Kotliniin verrattuna (KUVIO 1).

## 3.2 Swift

Swift on taas yleiskäyttöön suunnattu ohjelmointikieli, jolla on moderni lähestymistapa turvallisuuteen, suorituskykyyn ja ohjelmistojen suunnittelumalleihin. Swiftin tarkoituksen on korvata c-tyyppiset kielet kuten C, C++ ja Objective-C (Swift 2022). Swiftin on tarkoitus korvata Objective-C Applen sovelluskielenä ja ammattilaisten joukossa sillä on jo suurempi käyttäjäkunta (KUVIO 1).




KUVIO 1. Suosituimmat ohjelmointikielet (Stack Overflow Developer Survey 2021)

## 3.3 Keskeiset sovelluskehykset ja kirjastot

Sovelluksessa käytettiin erilaisia ohjelmointia kirjastoja ja kehyksiä. Oikein valitut kirjastot helpottavat sovelluksen kirjoittamista ja parhaassa tapauksessa tuovat lisää suorituskykyä, vakautta ja helpottavat koodin luettavuutta.

### 3.3.1 SwiftUi

SwiftUi on Applen kehittämä käyttöliittymäkehys. SwiftUi tarjoaa näkymät, kontrollit ja pohjan käyttöliittymän tekemiseen (Developer Apple 2022b). SwiftUi:llä voidaan rakentaa natiiveja käyttöliittymiä sovelluksille, jotka toimivat iOS:llä, iPadOS:llä, watchOS:llä, tvOS:llä tai macOS:llä (Swift By Sundell 2022).

Vaihtoehtona SwiftUi:lle olisi ollut esimerkiksi UiKit. SwiftUi ja UiKit ei sulje toisiaan pois vaan niitä voidaan hyödyntää molempia saman sovelluksen sisällä.

SwifUi:n haasteena on yhteensopivuus Applen vanhempien laitteiden kanssa, sillä sitä voidaan käyttää sovellusten tekemiseen laitteille, jotka käyttävät iOS 13-versiota tai uudempaa. iOS 13 julkaistiin kesäkuussa 2019. Lisäksi siitä on vähän tietoa tai apua saatavilla muilta kehittäjiltä esimerkiksi Stack Overflow -sivustolta. (Progtrails 2022.) Tällä hetkellä uusin iOS-versio on 15.4.

### 3.3.2 Core Data ja Room

Core Data on Applen sovelluskehys, jolla voidaan tallentaa ja poistaa dataa laitemuistiin. Core Data ei itsessään ole tietokanta kuten MongoDB tai SQL. Sen sijaan se hyödyntää iOS-laitteisiin sisältyvää SQLite-relaatiotietokantaa (Better Programming 2022).

SQLite-tietokantoja Android-laitteilla voidaan hallita Room-kirjastolla. Room ei ole sovelluskehys, vaan kirjasto ja se kuuluu Android Jetpackiin, mikä on kokoelma eri kirjastoja Android-kehitykseen (Android Developer 2022b).

Datamalleja hyödynnetään tiedon käsittelyssä ja sen säilömisessä puhelimen omaan muistiin, jolloin data on saatavilla vaikka nettiyhteyttä ei olisi.

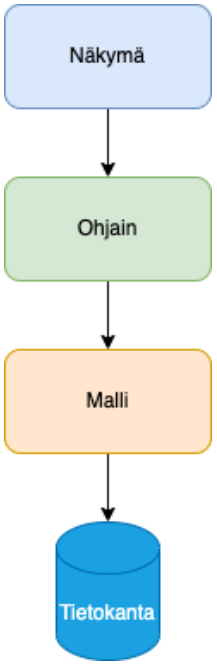
## 3.4 Arkkitehtuuri

Mobiilisovelluksen rakentamista ja suunnittelua voidaan selkeyttää sovelluksen arkkitehtuurilla. Sovellusarkkitehtuuri on kokonaisuudessaan laajempi käsitys, jota ohjaavat toiminnalliset vaatimukset, laatuvaatimukset sekä rajoitteet. Nyt tarkastellaan sovelluksen suunnittelumalleja, rakennetta ja niitä palikoita mistä sovellus rakentuu ja miten ne keskustelevat keskenään sekä tiedon liikettä niiden välillä. Selkeä arkkitehtuuri auttaa muita kehittäjiä ymmärtämään sovelluksen toimintaa, mikä mahdollistaa helpomman jatkokehityksen ja virheidenpaikantamisen.

### 3.4.1 Malli-Näkymä-Ohjain Model-View-Controller (MVC)

Model-View-Controller (MVC) kuvaillaan olevan ”perinteinen ohjelmistokehityksen suunnittelumalli, jonka tärkein tehtävä on eriyttää sovelluksen esitys- ja logiikkakerros toisistaan ja delegoida sovelluksen sisäisiä vastuita eri osa-alueille” (Hurja 2022). Malli, näkymä ja ohjain ovat vuorovaikutuksessa keskenään. Vaikka näiden toiminnot toimivat erikseen, ovat ne kuitenkin toisistaan riippuvaisia. Tämä tarkoittaa sitä, että muutokset esimerkiksi mallissa saattavat aiheuttaa muutoksia myös näkymässä. (Hurja 2022.)

Näkymä pitää sisällään sen osan sovelluksesta, minkä käyttäjä näkee ja jonka kanssa se pystyy vuorovaikuttamaan. Ohjain tarjoilee näkymän käyttäjälle näytettävät tiedot ja mallille käyttäjän näkymässä tehdyt muutokset, jotka tallennetaan tietokantaa. Malli kuvaa niitä tietoja mitä käsitellään ja tallennetaan tietokantaan, esimerkiksi viestin- tai käyttäjän tiedot (KUVIO 2).



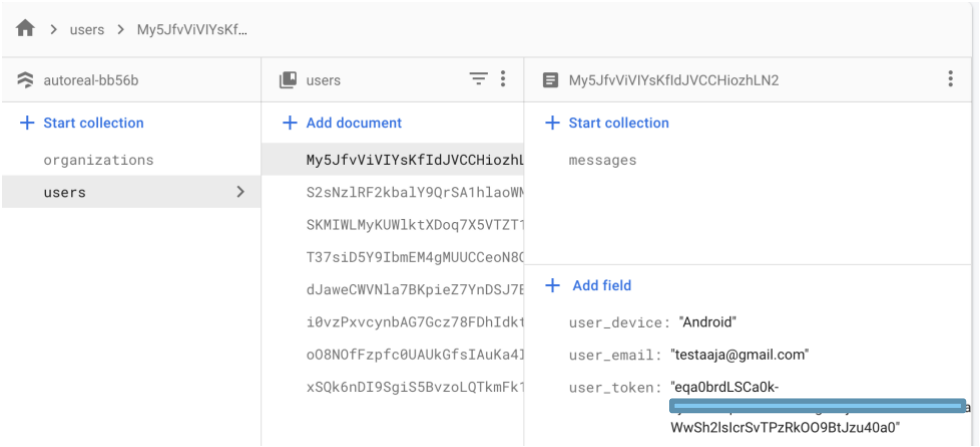

KUVIO 2. MVC-arkkitehtuuri

### 3.4.2 Firebase Firestore

Firestore Database on pilvessä pyörivä NoSQL dokumenttipohjainen tietokanta, joka antaa varastoida, lukea, kirjoittaa ja synkronoida dataa mobiili- ja nettiapplikaatioille (Firebase Google 2022).

Firestoressa on selainpohjainen käyttöliittymä, mikä helpottaa palvelun käyttöönottoa ja hallinnointia. Siitä on apua myös kehittämisessä ja virheiden etsimisessä. Dataa ja sen muuttumista pystyy seurata reaaliajassa selaimella. Tietoja voi myös muokata selaimesta ja tietokannalle voi asettaa erilaisia sääntöjä, kuten luku- ja kirjoitusoikeudet.

Firestoren tietokanta koostuu kolmesta kentästä: Kokoelma (Collection), Dokumentti (Document) ja Kenttä (Field). Lisäksi Dokumentin sisään voidaan luoda uusi kokoelma, jota kutsutaan alikokoelmaksi (subcollection), jolla voi taas olla omat kentät (KUVA 4).




KUVA 4. Firestore-tietokannan rakenne

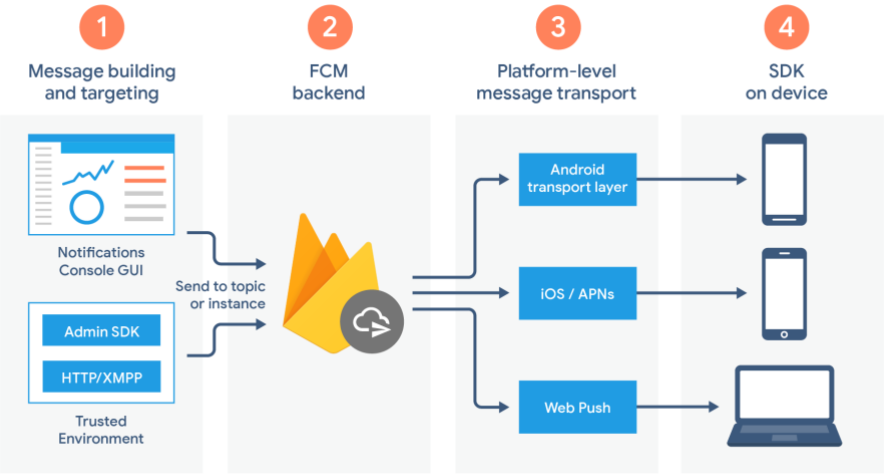
### 3.5 Push-ilmoitukset

Push -ilmoitukset ovat kuitattavia näyttöön ilmestyviä viestejä, jotka esitetään selaimen tai mobiililaitteen näytölle (Vwo, 2022). Google tarjoaa ratkaisun Firebase -pilvipalveluillaan. Firebaseen kuuluu joukko erilaisia palveluita, joita voi tarvittaessa ottaa mukaan sovellukseen. Tässä sovelluksessa on käytössä

Firebasen autentikaatio, push-ilmoitukset sekä Firestore-tietokanta. Valitut palvelut ovat yhteensopivia Androidiin, iOS:lle ja webbikäyttöliittymälle.

### 3.5.1 Firestore Cloud Messaging (FCM)

FCM:llä voidaan lähettää kohdennettuja ilmoituksia käyttäjille. Googlen FCM-backend (KUVIO 3, kohta 2) huolehtii viestin välityksestä oikeaan mobiililaitteeseen. Viesti voidaan lähettää kahdella eri tavalla: nimettyyn ryhmään (topic), jolloin kaikki ryhmän jäsenet saavat ilmoituksen laitteeseensa tai suoraan yksittäiseen laitteeseen laitetunnuksen (FCM-token) perusteella. Ryhmäilmoitusominaisuutta voidaan tarvita sovelluksen jatkokehityksessä, mutta ei vielä tässä kohtaa. Tässä työssä käytetään lähetystä laitetunnuksella.




KUVIO 3. FCM-arkkitehtuuri (Developer Android 2022c)

FCM-token on laitekohtainen tunniste, minkä perustelle Firebase Cloud Messaging osaa välittää push-ilmoituksen oikeaan laitteeseen. FCM-token voidaan tallentaa Firestore -tietokantaan käyttäjän tietoihin, jolloin se on haettavissa ilmoituksen lähettämistä varten.

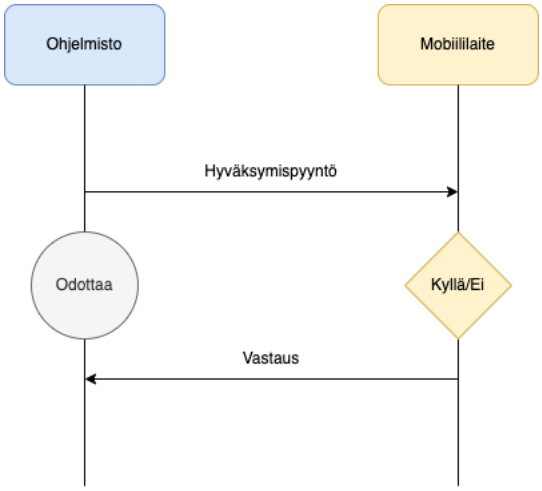
FCM-token voi muuttua, joten Google on varautunut tähän tarjoamalla metodin, mikä seuraan tokenia ja sen muuttuessa sovelluksen on osattava korvata vanha token uudella.

### 3.5.2 Apple Push Notification service (APNs)

APNs on Applen iOS:lle, watchOS:lle, tvOS:lle ja macOS:lle suunnattu lujatekoinen, turvallinen ja erittäin suorituskykyinen push-ilmoituspalvelu (Apple 2022d). Applen laitteisiin ei ole mahdollista kehittää push-ilmoitustoimintoja ilman APNs:ää. Firestore Cloud Messaging käyttö iOS-laitteella vaativat push-ilmoitusta varten APNs-tokenin. Tokenin saa käyttöön rekisteröimällä sovellus Applen Push Notification palveluun. Rekisteröintiä varten pitää luoda maksullinen tili. Rekisteröinnin jälkeen voidaan luoda avain, jonka liittäminen Firebase-projektiin antaa oikeudet push-ilmoitusten käyttöön. Push-ilmoitukset eivät toimi Xcoden emulaattoreille vaan vaativat oikein Applen mobiililaitteen.

# 4 YLEINEN MONIALUSTAINEN KUITTASSOVELLUS (YMK)

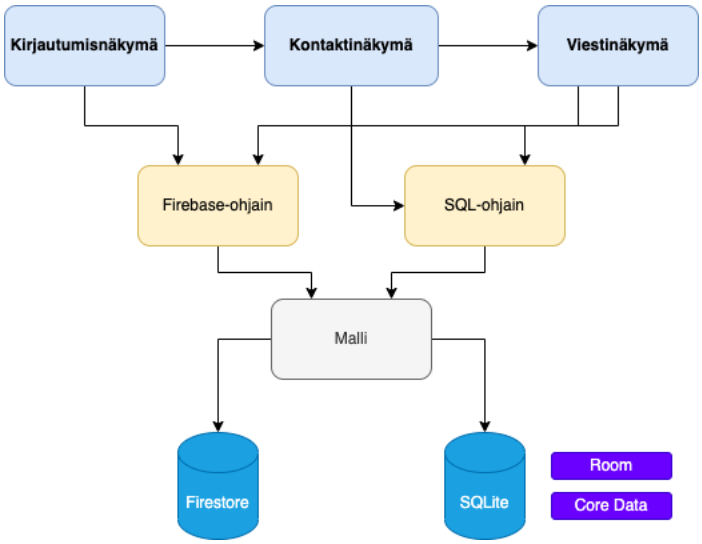
YMK on mobiilisovellus Android- ja iOS-alustoille. Sovelluksen tarkoitus on mahdollistaa hyväksymispyyntöihin vastaamisen nopeasti, vaivattomasti ja turvallisesti. Sovellukseen luodaan käyttäjätili, joka luo tarvittavat tunnisteet hyväksymispyynnön lähettäjän ja vastaanottajan välille. Käyttäjän ollessa kirjautuneena sovellukseen hyväksymispyyntö tulee mobiililaitteeseen push-ilmoituksena, jolloin vastaanottajan puhelin ilmoittaa saapuneesta viestistä. Ilmoituksen voi avata suoraan tai sovelluksen kontaktit -valikosta. Hyväksymispyyntö tallentuu lähettäjän Firestore-tietokantaan ja vastaanottajan mobiililaitteen tietokantaan. Hyväksymispyynnössä näkyy lähettäjä, pyynnön otsikko ja saateviesti. Hyväksymispyyntöön on kaksi vastausvaihtoehtoa: ”kyllä” ja ”ei” (KUVIO 4).




KUVIO 4. Hyväksymispyynnön yleiskaavio

Vastauksen jälkeen hyväksymispyyntö palautetaan lähettäjälle vastauksen kanssa. Vastauksen saavuttua takaisin lähettäjälle lähettäjän ohjelmisto jatkaa toimintaansa saadun vastauksen mukaisesti. Mikäli käyttäjä ei ole kirjautunut sovellukseen, hyväksymispyyntö tallennetaan mobiililaitteen muistiin ja se on saatavilla, kun sisäänkirjautuminen tapahtuu.

Sovelluksen rakenne on molemmilta toteutuksilta kuvion 5 mukainen. Käyttöliittymä koostuu kolmesta eri näkymästä, jotka ovat yhteydessä ohjaimiin: Firebase ja SQL. Molemmat ohjaimet käyttävät yhdenmukaista mallia tiedon tallentamiseen. Pyyntöjen tiedot tallennetaan Firestoren tietokantaa ja/tai mobiililaitteen kiintolevylle.




KUVIO 5. Sovelluksen yleinen rakenne

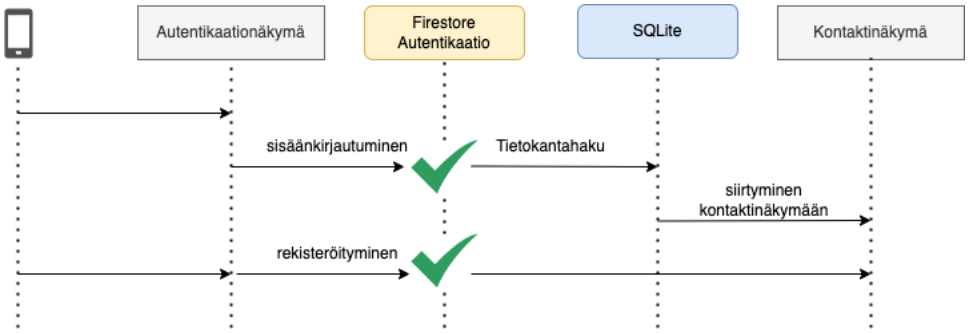
## 4.1 Sovelluksen ohjelmointikielet

Molemmille alustoille haluttiin valita moderni ohjelmointikieli oppimisen ja tulevaisuudennäkymien takia, jonka seurauksena Android-versioissa päädyttiin Kotliniin ja iOS-versiossa Swiftiin. Lisäksi Kotlin ja Swift ovat aloittelijaystävällisempiä verrattuna Javaan ja Objective-C:hen.

## 4.2 Autentikaatio

Käyttäjän ja mobiililaitteen tunnistautumista varten käytetään Firebasen sähköposti-autentikaatiopalvelua (KUVIO 6). Käyttäjä luo tunnukset sähköpostiosoitteella ja salasanalla. Sähköpostiosoite toimii käyttäjän tunnisteena hyväksymispyyntöä lähetettäessä: käyttäjän tunniste haetaan Firestoren tietokannasta sähköpostiosoitteen perustella. Käyttäjän tunnisteella

taas haetaan FCM-token, minkä perusteella hyväksymispyyntö ohjataan oikeaan laitteeseen. Tunnuksen luominen vaatii sähköpostiosoitteen olevan muotoa xxx@xxx.x ja salasanan on oltava vähintään kuusi merkkiä pitkä. Hyväksytyn sisäänkirjautumisen jälkeen tehdään tietokantahaku mobiililaitteen kiintolevyltä ja siirrytään kontaktinäkymään.




KUVIO 6. Autentikaation sekvenssikaavio

### 4.2.1 FCM-Token

Ilmoituksien välittämiseen vaadittava FCM-token noudetaan sille tarkoitetulla metodilla. Token noudetaan kirjautumisen yhteydessä. Kuvassa 5 on esillä createNewToken-metodi. Rivillä 64 m metodi pyytää tokenia Firebasesta. Tokenin saatua, se tallennetaan käyttäjän tietokantaan.

```
63    fun createNewToken() {
64        FirebaseMessaging.getInstance().token.addOnSuccessListener { token: String ->
65            if (!TextUtils.isEmpty(token)) {
66                Log.d(TAG, msg: "retrieve token successful : $token")
67                this.token = token
68                FirestoreApi().updateUserToken(token)
69            } else {
70                Log.w(TAG, msg: "token is null")
71            }
72        }.addOnFailureListener { e: Exception ->
73            Log.v(TAG, msg: "retrieve token error", e)
74        }
75    }
```

KUVA 5. FCM-tokenin hakeminen Android-toteutuksessa

Vastaavasti kuvan 6 Swift-koodi pyytää tokenin ja ilmoittaa virheestä, jos jotain menee pieleen. Muussa tapauksessa se välittää tiedot tietokantatallennuksen tekevälle funktiolle.

```
150             Messaging.messaging().token { token, error in
151                 if let error = error {
152                     print("Error fetching FCM registration token: \(error)")
153                 } else if let token = token {
154                     print("FCM registration token: \(token)")
155                     self.setDocument(aEmail: self.email, aToken: token, aUserId:
                            String(result!.user.uid))
156                 }
157             }
```

KUVA 6. FCM-tokenin hakeminen iOS-toteutuksessa

Huomioitavaa on, että token voi muuttua itsekseen ja sitä varten Google on tehnyt kuuntelijametodin nimeltä onNewToken, mikä aktivoituu uudesta tokenista. Uuden tokenin käsittely jää ohjelmoijan vastuulle. Tässä sovelluksessa asia on ratkaistu kutsumalla updateUserToken-metodia (KUVA 5, rivi 68), joka päivittää uuden tokenin vanhan tilalle Firestoren tietokantaan.

### 4.2.2 Push-ilmoitusten hallinta (Android)

Sovellukseen tehtiin toiminto, jolla voi lähettää push-ilmoituksia, vaikka sovelluksen päätarkoitus on niiden vastaanottaminen ja vastaaminen. Tämä ominaisuus on lähes välttämätön kehittämisen kannalta. Vaihtoehtoisesti push-ilmoituksia voidaan lähettää Firestoren konsolista. Push-ilmoituksen lähettämistä varten tehtiin kolme dataluokkaa. PushNotification -dataluokka (KUVA 7) pitää sisällään push-ilmoituksen mukana tulevat tiedot (NotificationData) sekä tiedon (FCM-token) kenelle viesti ollaan välittämässä.

```
3   data class PushNotification(
4       val data: NotificationData,
5       val to: String,
6       val priority: String
7   )
```

KUVA 7. PushNotification-dataluokka

NotificationData-dataluokka sisältää viestin otsikon, viestin tekstin sekä Body-dataluokan (KUVA 8).

```
3   data class NotificationData (
4       val title: String,
5       val message: String,
6       val body: Body
7   )
```

KUVA 8. Notification-dataluokka

Body sisältää tarvittavat tiedot viestin vastaanottamiseen ja vastauksen lähettämiseen (KUVA 9).

```
3   data class Body (
4       val senderToken: String,
5       val userEmail: String,
6       val userId: String,
7       val timeStamp: String,
8       val messageId: String = "",
9       val messageType: String ="",
10      val response: String = "10",
11      val isRead: Boolean = false
12  )
```

KUVA 9. Body-luokan sisältö

Kun tarvittavat tiedot push-ilmoituksen lähettämistä varten on täytetty, kutsutaan RetrofitInstance -luokka (KUVA 10), jonka tehtävä on rakentaa verkkopyyntö Firebasen rajapintaan. RetrofitInstance -luokka sisältävä Companion Object on Kotlinin tapa luoda Singleton -luokka, jolloin siitä on olemassa vain yksi toteutus, mutta sen ominaisuuksia voidaan käyttää kutsumalla luokkaa. Verkkopyynnön lähettämiseen tarvitaan URL-osoite ja palvelinavain, jotka saadaan sovelluksen rekisteröinnin yhteydessä Googlen Firebase -konsolista. Pyyntö sisältää push-ilmoituksen tiedot.

```
class RetrofitInstance {
    companion object {
        private val retrofit by lazy {
            Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        }
        val api by lazy {
            retrofit.create(NotificationAPI::class.java)
        }
    }
}
```

KUVA 10. RetrofitInstance-luokka

Mikäli Retrofitin verkkopyyntö onnistuu, tallennetaan lähetetty viesti lähettäjän tietokantaan Firestoreen odottamaan vastausta.

```
interface NotificationAPI {

    @Headers( ...value: "Authorization: key=$SERVER_KEY", "Content-Type:$CONTENT_TYPE")
    @POST( value: "fcm/send")
    suspend fun postNotification(@Body notification: PushNotification) :
            Response<ResponseBody>
}
```

KUVA 11. NotificationApi on yhteydessä Firebase messaging-palveluun

Push-ilmoituksen vastaanottamiseksi Google tarjoaa onMessageReceived-metodin, joka tarkkailee tulevia viestejä (KUVA 12).

```
override fun onMessageReceived(message: RemoteMessage) {
    super.onMessageReceived(message)

    val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE)
            as NotificationManager

}
```

KUVA 12. onMessageReceived-metodi

Viestin saapuessa käytetään Androidin NotificationManager-luokkaa. Luokan tarkoituksena on ilmoittaa käyttäjälle mobiililaitteen tausta-ajon tapahtumasta. Ennen viestin esittämistä käyttäjälle se tallennetaan laitteen tietokantaan.

Vastaanotetusta viestistä rakennetaan push-ilmoitus, joka antaa käyttäjälle äänimerkin ja ilmoituksen näytölle tai ikonin näytön ilmoituspalkkiin. Ilmoituksen avaaminen siirtää sovelluksen Viestinäkymään (KUVA 3).

Sovelluksen luonteen takia vastausvaihtoehtoja on kaksi: ”kyllä” ja ”ei”. Vastaus lähetetään suoraan Firestoren tietokantaan, jolloin se muokkaa lähettäjän alkuperäisen hyväksymispyynnön kenttiä. Tietokantaan ei siis luoda uutta kirjausta, vaan päivitetään jo olemassa olevaa kirjausta push-ilmoituksen tunnuksen ja lähettäjän tunnuksen perusteella. Jatkokehitettäväksi jää, miten vastausta odottava sovellus saa tiedon siitä, että sen tietokantaa on muokattu.

### 4.2.3 Push-ilmoitusten hallinta (iOS)

iOS -versioon ei erikseen tehty mahdollisuutta lähettää hyväksymispyyntöjä, koska sovelluksen tarkoitus on vastaanottaa pyyntöjä ja vastata niihin. Viestin vastaanottamiseksi on valmiiksi tarjolla AppDelegate-luokka, mikä pitää huolen Firebasen konfiguroimisesta sovelluksen käynnistyessä. AppDelegate-luokkaan on luotu laajennusluokkia (extension class), jotka vastaanottavat tulevat viestit riippuen siitä onko sovellus aktiivinen vai tausta-ajossa. Vastaanotettu viesti välitetään rivin 141 (KUVA 13) handleReceivedMessage-funktiolle, mikä purkaa viestin tiedot ja tekee pyynnön mobiililaitteen tietokantaan viestin tallentamiseksi. Rivin 142 completionHandler luo näytölle ilmestyvän ilmoituksen, äänen ja kuvan.

```
135 extension AppDelegate: UNUserNotificationCenterDelegate {
136     func userNotificationCenter(_ center: UNUserNotificationCenter,
137                                 willPresent notification: UNNotification,
138                                 withCompletionHandler completionHandler: @escaping
                                        (UNNotificationPresentationOptions)
139                                 -> Void) {
140         let receivedMessage = notification.request.content.userInfo
141         handleReceivedMessage(receivedMessage: receivedMessage)
142         completionHandler([.banner, .list, .sound])
143     }
144
145 }
```

KUVA 13. Viestinvastaanottaja-funktio iOS

## 4.3 Tiedon tallentaminen

Sovellukset käyttävät tietojen tallentamiseen kahta eri tietokantaa. Ulkoista Firebase-tietokantaa ja mobiililaitteen sisäistä kiintolevyä.

### 4.3.1 Firestore Firebase

Firestoren Firebase toimii ulkoisena tietokanta, minne tallennetaan käyttäjän tiedot. Hyväksymispyynnön lähettäjällä (ohjelmisto) on vastaavat käyttäjätiedot kuin tavallisella käyttäjällä, joille hyväksymispyyntöjä lähetetään (KUVA 14 JA KUVA 15).

```
fun newUser(email: String, token: String, userID: String) {
    val userTemplate = hashMapOf(
        "user_email" to email,
        "user_device" to "Android",
        "user_token" to token
    )
    userCollectionRef
        .document(userID)
        .set(userTemplate)
        .addOnSuccessListener { documentReference ->
            Log.d(TAG, msg: "DocumentSnapshot added with ID: $documentReference")
        }
        .addOnFailureListener { e ->
            Log.w(TAG, msg: "Error adding document", e)
        }
}
```

KUVA 14. Käyttäjän lisääminen Firestore-tietokantaan Kotlin-kielellä

Sovellus luo uuden käyttäjän Firestore-tietokantaa kutsumalla newUser-metodia. Metodi saa käyttäjän sähköpostiosoitteen, FCM-tokenin ja käyttäjätunnuksen parametrina. Parametrina saadut tiedot asetetaan ennalta määriteltyyn datamalliin. Saatujen parametrien lisäksi merkitään Android-toteutuksessa käyttäjän laitemalliksi (user_device) Android- ja iOS-toteutuksessa iOS:n eriävän push-ilmoituksen JSON-rakenteen takia.

```
42      public func setDocument(aEmail: String, aToken: String, aUserId: String) {
43          let userRef = Firestore.firestore().collection("users")
44          userRef.document(aUserId)
45              .setData([
46                  "user_email": aEmail,
47                  "user_device": "iOS",
48                  "user_token": aToken
49              ]) {
50                  err in
51                  if let err = err {
52                      Logger.log(.error, "Error writing document: \(err)")
53                  } else {
54                      Logger.log(.success, "Document successfully written!")
55                  }
56              }
57      }
```

KUVA 15. Käyttäjän lisääminen Firestore-tietokantaan Swift-kielellä

Jokaisesta uudesta hyväksymispyynnöstä tulee uusi merkintä Firebase messages -alikokoelmaan (KUVA 16).

```
message: "text"
message_id: "13[redacted]ed"
message_title: "msg_title"
message_type: "RequestMessage"
receiver_email: "testaaja@gmail.com"
response: "10"
sender_id: "7sl[redacted]D2"
time: "2022-03-22T21:18:06.437"
```

KUVA 16. Messages-alikokoelman kentät

Hyväksymispyynnön tunnisteena toimii yksilöllinen tunniste, joka luodaan viestin lähetyksen yhteydessä. Samat tiedot lähetetään hyväksymispyynnön yhteydessä mobiililaitteen käyttäjälle. Mobiililaitteen käyttäjä muokkaa response-kenttää. Ykkönen (1) tarkoittaa hyväksymistä ja nolla (0) kieltäytymistä (KUVA 17).

```
CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
    try {
        userCollectionRef
            .document(aSenderId)
            .collection( collectionPath: "messages").document(aMessageId)
            .set(messageTemplate as Map<String, Any>).await()
    } catch (e:Exception){
        withContext(Dispatchers.Main) {  this: CoroutineScope
            Log.d(TAG,  msg: "searchTeamEvents: ", e)
        }
    }
}
```

KUVA 17. Pyynnön tallentaminen Firestore-tietokantaan Kotlin -kielellä

Vastauksessa käytetään hyväksymispyynnön alkuperäisiä tietoja. Käyttäjä valitsee vastauksen, jonka jälkeen lähettäjä etsitään Firestoren tietokannasta käyttäjä -tunnisteen perusteella ja vastaanotettu viesti etsitään viestin tunnisteen perusteella. Vastaus korvaa response -kentän alkuperäisestä tietokantakirjauksesta (KUVA 18 ja KUVA 19).

```
CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
    try {
        val querySnapshot = userCollectionRef
            .document(aSenderId)
            .collection( collectionPath: "messages")
            .get()
            .await()
        for (document in querySnapshot.documents){
            if(document.data?.getValue( key: "message_id") == aMessageId){
               userCollectionRef
                    .document(aSenderId)
                    .collection( collectionPath: "messages")
                    .document(aMessageId)
                    .update( field: "response", aResponse)
                    .await()
            }
        }
    } catch (e:Exception){
        withContext(Dispatchers.Main) {  this: CoroutineScope
            Log.d(TAG,  msg: "ResponseMessage: ", e)
        }
    }
}
```

KUVA 18. Vastauksen tallentaminen Firestore Firebaseen Kotlin-kielellä

Android-toteutuksessa (KUVA 18) vastausviestin lähettäminen on ympäröity Kotlin Coroutinella. Tarkoituksena on siirtää tietokantakysely taustasäikeelle, jolloin pääsäikeellä pyörivän käyttöliittymän ei tarvitse odottaa kyselyn valmistumista. Dispatcherillä määritellään millä säikeellä operaatio suoritetaan. Nyt käytössä on IO (input/output), joka on varattu tiedon siirräntää varten.

```
59      public func responseMessage(aUserId: String, aMessageId: String, aResponse: String){
60          let userRef = Firestore.firestore().collection("users")
61
62          userRef.document(aUserId)
63              .collection("messages")
64              .whereField("message_id", isEqualTo: aMessageId)
65              .getDocuments() { (querySnapshot, err) in
66                  if let err = err {
67                      Logger.log(.error, "Error getting documents before update: \(err)")
68                  } else {
69                      for document in querySnapshot!.documents {
70                          Logger.log(.action, "Document.data: \(document.data()) ")
71
72                          document.reference.updateData(["response": aResponse])
73                      }
74                  }
75              }
76      }
```

KUVA 19. Vastauksen tallentaminen Firestore Firebaseen Swift-kielellä

### 4.3.2 Mobiililaitteiden tietokannat

Core Data ja Room ovat käytössä vastaanotetun hyväksymispyynnön tallentamisessa, poistamisessa ja muokkaamisessa. Mikäli mobiililaitteen käyttäjä ei ole kirjautunut sisään, ei push-ilmoitusta toimiteta. Sen sijaan se vain tallennetaan SQLite-tietokantaan Core Datan ja Roomin avustuksella. Tämän seurauksena vastaanotetut hyväksymispyyntökyselyt ovat näkyvissä Kontaktinäkymässä (KUVA 2) siinä vaiheessa, kun käyttäjä kirjautuu sovellukseen sisään.

Vastauksen lähettämisen jälkeen muokataan Firestoren tietokannan lisäksi myös mobiililaitteen tietokantaa. Tietokantaan merkitään vastauskenttä (response) ja päivitetään kellonaika vastaamaan lähetysajankohtaa. Tämän seurauksena

sovellukset huomaavat muutokset tietokannasta ja päivittävät kontaktinäkymässä esillä olevan viestin tilan (ympyrä, X-merkki, oikein-merkki).

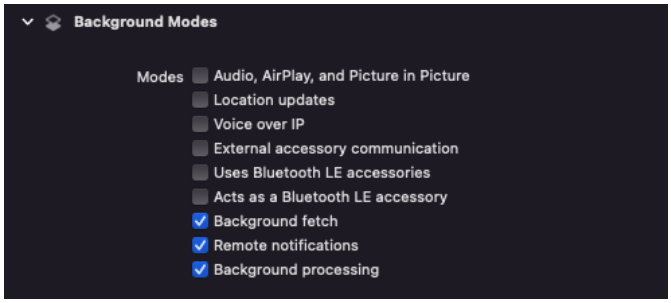
### 4.4 Tausta-ajo

Ilmoitusten on tultava perille, vaikka ohjelma on taustalla tai näyttö lukittu. Androidilla sovelluksen AndroidManifest.xml-tiedostoon on lisätty palvelu (KUVA 20), mikä antaa sovellukselle oikeuden toimia taustalla, vaikka sovellus on suljettu tai mobiililaitteen näyttö lukittu. Palvelun sisältämä directBootAware mahdollistaa sovelluksen itsenäisen käynnistymisen, kun mobiililaite uudelleen käynnistetään.

```
<service
    android:name=".FirebaseService"
    android:exported="false"
    android:directBootAware="true"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
    </intent-filter>
</service>
```

KUVA 20. Palvelu tausta-ajo Androidilla

iOS-versiossa tausta-ajo otetaan käyttöön asentamalla projektiin Background Modes -paketti Xcoden ympäristöstä. Tätä käyttötarkoitusta varten paketista on otettu käyttöön kuvan 30 ominaisuudet.




KUVA 30. Tausta-ajoon vaadittavat ominaisuudet (iOS)

# 5 POHDINTA

Opinnäytetyöhön sisällytetyt toiminnalliset vaatimukset 1–7 ovat täyttyneet ja sovellus toimii molemmilla alustoilla oikein, pieniä poikkeuksia lukuun ottamatta. Sovellus ei tällaisenaan kykene toimimaan oikealla tavalla, koska hyväksymispyyntöjen lähettäjäohjelmistoa ei ole vielä tehty. Firestoren palveluita voidaan ottaa samalla tavalla käyttöön webbiympäristössä kuin tässä työssä esitellyt sovellukset. Todennäköisen vaihtoehtona on luoda React.js-käyttöliittymä ja Node.js:llä backend, millä pyyntöjen lähettäminen onnistuu molemmille alustoille.

Lisäksi käyttäjälle pitää kehittää mahdollisuus käyttää sovellusta useammalla mobiililaitteella tai nettikäyttöliittymästä. Tämä voidaan ratkaista siten, että käyttäjän Firestoren tietokantaan luodaan lista FCM-tokeneista ja hyväksymispyyntö lähetetään jokaiseen tokeniin, jolloin hyväksymispyyntö lähtee kaikkiin niihin laitteisiin, jotka ovat rekisteröity samalle käyttäjälle. Tämän jälkeen riittäisi, että yhdestä laitteesta kuittaa hyväksymispyynnön.

Teknologiavalintojen kanssa ei tullut ongelmia. Googlella on todella kattava dokumentaatio heidän tarjoamiin pilvipalveluihin. Sen sanottua dokumentaatiosta löytyi useamman kerran vanhentunutta tietoa. Siihen voi vaikuttaa nopeasti kehittyvät kielet ja sovelluskehykset. Firestore-palveluiden käyttöönotto on myös tehty helpoksi Android Studiossa ja Xcodessa. Koska SwiftUi ja Kotlin ovat verrattain uusia kieliä, löytyy niistä vähemmän tietoa ja käyttäjäkokemuksia kuin esimerkiksi Javasta ja Objective-C:stä.

Koska aiempaa kokemusta Android-kehityksestä oli opiskelujen ajalta kahdenkurssin verran, työ aloitettiin tekemällä ensin Android-versio. iOS-toteutus luotiin sitten Android-toteutuksen pohjalta ja tämä toimi mielestäni oikein hyvin. iOS-toteutuksessa pystyin ohittamaan aiempia ongelmia ja luomaan ehkä sujuvamman rakenteen sovellukseen.

Ohjelmointikielinä Swift ja Kotlin ovat pitkälti samanlaisia. Kehittämisen suurin ero tulee, kun ottaa käyttöön SwiftUi-kehyksen. Käyttöliittymien rakentaminen,

muokkaaminen ja toiminnalliseksi tekeminen on ohjelmoijaystävällisempää verrattuna Androidiin. Tekemisen helppous ja kehyksen valmiit tyylit vähentävät tarvetta ulkoasun hienosäätämiseen. SwiftUi:stä ja Reactista voi löytää jonkin verran samankaltaisuuksia, mikä ainakin itselläni helpotti SwiftUi:n sisäistämistä React-taustan takia. Android-sovelluksen tekeminen natiivina on välillä tuskallista. Se, mikä SwiftUi:llä vaatii muutaman rivin koodia, voi vaatia Androidilla 50 riviä. Kontaktinäkymässä esilläoleva lista hyväksymispyynnöistä on hyvä esimerkki tästä. Jetpack Compose on työkalu Androidin käyttöliittymän tekemiseen, mikä vastaisi ohjelmointityyliltään SwiftUi:ta. Tästä syystä vertailu SwiftUi:n ja Androidin välillä ei ole tasapuolista.

Lopuksi voi miettiä, onko järkevää tehdä samaa sovellusta kahta kertaa, kuten nyt kahdelle eri alustalle. Opinnäytetyötä ja oppimiskokemusta ajatellen kyllä, ehkä. Yrityksen kannalta voisi olla järkevää käyttää kehystä tai teknologiaa, millä voidaan rakentaa sovellus Androidille sekä iOS:lle yhdellä kertaa. Kovassa suosiossa ovat React Native ja Flutter, kun puhutaan hybridisovelluksista. React Nativen suosio selittyy varmasti osin sillä, että webbiohjelmoijien kynnys siirtyä mobiilikehitykseen on matalampi, koska sitä voi tehdä tutulla JavaScript -ohjelmointikielellä ja varsinkin jos React.js on jo tuttu. Flutter on taas Googlen kehittämän sovelluskehys, mitä ohjelmoidaan Dart-kielellä. Google on myös kehittänyt Kotlin Multiplatformia, millä voidaan tehdä Android- ja iOS-sovelluksia.

**LÄHTEET**

Better Programming, 2022. An Introduction to Core Data. Luettu. 25.3.2022. https://betterprogramming.pub/a-light-intro-to-core-data-part-un-e344f9d1528

Clearbridge Moblie, 2022. Luettu. 11.3.2022. https://clearbridgemobile.com/benefits-of-native-mobile-app-development/

Datareportal, 2022. Luettu. 27.4.2022. https://datareportal.com/global-digital-overview

Developer Android, 2022a. Android's Kotlin-first approach. Luettu. 10.3.2022. https://developer.android.com/kotlin/first

Developer Android, 2022b. Save data in a local database using Room. Luettu. 25.3.2022. https://developer.android.com/training/data-storage/room

Developer Android, 2022c. FCM Architectural Overview. Luettu. https://firebase.google.com/docs/cloud-messaging/fcm-architecture

Developer Apple, 2022a. APNs Overview. Luettu. 15.3.2022. https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html

Developer Apple, 2022b. SwiftUi. 25.3.2022. https://developer.apple.com/documentation/swiftui

Developer Apple, 2022c. Core Data. Luettu. 25.3.2022. https://developer.apple.com/documentation/coredata

Developer Apple, 2022d. APNs Overview. Luettu. 31.3.2022. https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html

Eduix Oy, 2022. Luettu 27.4.2022. https://eduix.fi/

Firebase Google, 2022. Cloud Firestore. Luettu. 15.3.2022. https://firebase.google.com/products/firestore

Hurja. MVC for dummies: malli, näkymä ja ohjain -arkkitehtuuri web-sovelluksessa. Luettu. 4.4.2022. https://www.hurja.fi/blogi/mvc-for-dummies-malli-nakyma-ja-ohjain-arkkitehtuuri-web-sovelluksissa/

Kotlin. 2022. Why Kotlin. Luettu. 11.3.2022. https://kotlinlang.org/

Kyberturvallisuuskeskus. 2022. Sähköinen tunnistautuminen. Luettu. 27.4.2022. https://www.kyberturvallisuuskeskus.fi/fi/toimintamme/saantely-ja-valvonta/sahkoinen-tunnistaminen

Progtrails. 2022. Why you still shouldn’t build business app with SwiftUi. Luettu. 25.3.2022. https://prograils.com/swift-ui-b2b-app-2022#old

Stack Overflow. Developer Survey 2021. Luettu 4.4.2022. https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof

Swift. 2022. About Swift. Luettu. 8.3.2022 https://www.swift.org/about/

Swift By Sundell, 2022. Luettu. 25.3.2022 https://www.swiftbysundell.com/discover/swiftui/

Vwo, 2022. What Are Push Notifications?. Luettu 10.4.2022 https://vwo.com/push-notifications/