Tampereen ammattikorkeakoulu

Videopuhelusovelluksen toteutus WebRTC-teknologialla

Jaakko Saranpää

OPINNÄYTETYÖ
Marraskuu 2023

Tietojenkäsittely, Tradenomi
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely, Tradenomi
Ohjelmistotuotanto

SARANPÄÄ, JAAKKO:
Videopuhelusovelluksen toteutus WebRTC-teknologialla

Opinnäytetyö 47 sivua, joista liitteitä 8 sivua
Marraskuu 2023

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa videopuhelusovellus mobiilialustalle sekä raportoida työn vaiheita ja tuloksia. Sovelluksen tarkoituksena oli mahdollistaa käyttäjälle yhteystietojen hallinta sekä käyttäjien välinen videopuheluyhteys. Projektin motivaationa oli osoittaa mobiilikehityksen ja videopuheluyhteyden toteuttamisen, oikeita työkaluja ja teknologioita hyödyntämällä, olevan mahdollista myös aloittelevalle ohjelmistokehittäjälle.

Mobiilisovelluksen toteuttamisen teknologiaksi valittiin React Native -mobiiliohjelmointikirjasto. Valinta perustui sen helppokäyttöisyyteen sekä samankaltaisuuteen suositun verkkokehityskirjasto Reactin kanssa. Videopuheluyhteys toteutettiin WebRTC-teknologialla, joka mahdollisti suoran yhteyden laitteiden välillä. Sovelluksen tietokanta sekä videopuheluyhteyden luomiseen vaadittu signaalipalvelin toteutettiin Firebase-pilvialustan tarjoamilla palveluilla. Sovelluksen käyttöliittymä suunniteltiin tukemaan sekä tummaa että vaaleaa väriteemaa.

Sovelluksen toteutus alkoi työkalujen ja koodipohjan alustamisella. Kehitystyössä hyödynnettiin erilaisia työkaluja, joista yhtenä oli tekoälyyn perustuva Github Copilot. Sovelluksen ja sen työkalujen alustamisen jälkeen lähdettiin toteuttamaan käyttöliittymää sekä vaadittuja ominaisuuksia. Sovelluksen käyttäjätietojen autentikaatio toteutettiin ulkoista palvelua hyödyntämällä. Yhteystietojen hallintaa varten tehtiin erilaisia näkymiä. Viimeisenä kehitettiin videopuhelunäkymä sekä yhteys osapuolien välillä.

Projektin lopputulokseksi saatiin pääominaisuuksiltaan toimiva sovellus. Sovellus mahdollistaa käyttäjän rekisteröitymisen, yhteystietojen lisäämisen ja selaamisen sekä videopuhelun soittamisen toisille käyttäjille. Projekti osoitti videopuheluyhteyden toteuttamisen sekä mobiilisovelluskehityksen oikeita resursseja käyttämällä olevan oletettua suoraviivaisempaa. Jatkokehitettäväksi jäi puuttuvia käyttöliittymänäkymiä ja itse toteutettu signaalipalvelin.

Asiasanat: videopuhelu, mobiilikehitys, webrtc

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technologies, Bachelor of Business Administration
Software Development

SARANPÄÄ, JAAKKO:
Implementing a Video Call Application Using WebRTC

Bachelor's thesis 47 pages, appendices 8 pages
November 2023

The aim of this thesis was to design and implement a video call application for a mobile platform and to report the results. The purpose of the application was to enable users to manage their contacts and establish a video call connection between users. The motivation behind this project was to prove that creating a mobile application with a video call functionality is achievable for even a beginner software developer.

React Native, a mobile programming library, was chosen as the main technology behind the application, due to its simple syntax and similarity with the popular web development library, React. Functionality of the video call connection was created using WebRTC, which enables a direct connection between devices. The signaling server that is used to create said connections, and the database of the application, were implemented on Google's Firebase cloud platform using its many available services.

The implementation began with the setup of various tools and the codebase, after which began the creation of the user interface and many of the core features of the application. Many different views were created to handle different functions, such as browsing contacts and video calling users. The functionality of the video call was implemented last.

This project resulted in an application with all the planned core features. The project also demonstrated that mobile development and the implementation of a video call connection is not as hard as previously thought.

Key words: video call, mobile development, webrtc

SISÄLLYS

1 JOHDANTO

1.1 Projektin tarkoitus, konsepti ja motivaatio

Älypuhelimien yleistyessä nopeasti ympäri maailmaa kommunikaatio on muuttunut huomattavasti nopeammaksi ja ainakin vaivattomammaksi. Tämä teknologinen edistys on kasvattanut ihmisten odotuksia helposta ja kustannustehokkaasta viestinnästä. Tästä johtuen älypuhelimille ladattavien videopuhelusovellusten markkinat ovat laajentuneet merkittävästi, ja tarve entistä paremmille ja tehokkaammille sovelluksille kasvaa jatkuvasti.

Tämän projektin päämääränä on suunnitella ja kehittää mobiilialustalle helppokäyttöinen videopuhelusovellus, joka vastaa käyttäjien tarpeisiin. Samalla projektista syntyy raportti, joka toimii ohjelmistokehittäjille suuntaa antavana ohjeena vastaaviin projekteihin. Raportti tarjoaa kuvauksen sovelluksen kehitysprosessista, jotta ohjelmistokehittäjät voivat hyödyntää sitä omissa projekteissaan.

Videopuhelusovelluksen kehittäminen eroaa merkittävästi monista yleisimmistä ohjelmistoprojekteista. Tämä projekti vaatii ymmärrystä tietoverkkojen toiminnasta ja erilaisten verkkoprotokollien käytöstä. Videopuhelusovelluksen ohjelmointi saattaa vaikuttaa ensisilmäyksellä valtavalta haasteelta, joka kuuluu vain suurien ohjelmistotalojen kokeneiden ammattilaisten toimenkuvaan. Projektissa halutaan tutkia, onko ennakko-oletus totta, ja osoittaa videopuhelun toteuttamisen olevan oikeita teknologioita ja työkaluja käyttämällä oletettua yksinkertaisempaa.

Toinen tärkeä motivaattori tämän projektin taustalla on tarjota lukijalle mahdollisuus tutustua erilaisiin teknologioihin ja oppia niiden käytöstä projektin kehittämisen yhteydessä. Projektissa esitellään ja hyödynnetään erilaisia teknologioita sekä työkaluja, ja niiden käyttöä opitaan käytännön kautta. Tämä mahdollistaa paitsi videopuhelusovelluksen luomisen, myös laajentaa yleistä teknistä osaamista ja ymmärrystä.

1.2 Projektin rajaukset

Projektin määrittely ja vaatimusten rajaaminen on tärkeä vaihe ohjelmistokehityksessä, ja sen laadulla on suuri vaikutus projektin onnistumiseen. Projektin huono määrittely voi johtaa suunnitteluongelmiin, väärinymmärryksiin ohjelmistokehittäjien ja toimeksiantajan välillä sekä lopulta epätyydyttävään tuotteeseen. Yleensä määrittelyssä mietitään esimerkiksi sovelluksen käyttötilannetta, käyttötapaa sekä sen oletettua kohdeyleisöä. Yksi keskeisimmistä tekijöistä tämän projektin kannalta on käyttäjäkohderyhmä. Määrittelyn yhteydessä vielä tärkeämpi osuus on itse sovelluksen toiminnallisuuksien rajaaminen ja niihin sitoutuminen.

Älypuhelimien yleistyessä sovelluksen käyttäjänä voisi mahdollisesti olla kuka tahansa iästä tai asuinpaikasta riippumatta. Mahdollinen käyttäjäryhmän monimuotoisuus edellyttää huolellista harkintaa ja suunnittelua, jotta lopputuloksena olisi laadukas ja saavutettava käyttökokemus.

Toiminnalliset vaatimukset muodostavat sovelluksen ytimen ja määrittävät sen perustoiminnot. Ne antavat sovellukselle sen käyttötarkoituksen ja tekevät siitä käyttökelpoisen. Ilman näitä vaatimuksia sovellus olisi tyhjä ja merkityksetön. Tämän projektin toiminnallisiksi vaatimuksiksi kuuluivat seuraavat asiat.

Yksi tärkeimmistä ominaisuuksista käyttäjätilejä hyödyntävässä sovelluksessa on se, että käyttäjän täytyy pystyä luomaan tili syöttämällä vaaditut yhteystiedot sekä luomalla tietoturvallinen salasana. Tämän sovelluksen vaatimiin yhteystietoihin kuuluivat puhelinnumero, sähköposti sekä käyttäjän etunimi ja sukunimi.

Käyttäjän täytyy myös voida kirjautua sisään käyttääkseen sovellusta luotuaan ensin käyttäjätilin rekisteröitymällä. Sisäänkirjautumisen täytyy olla yksinkertainen sekä tietoturvallinen. Näkymän täytyy ilmoittaa käyttäjälle, jos kirjautuminen epäonnistuu, sekä viedä käyttäjä sovellukseen, mikäli kirjautuminen onnistuu.

Sovelluksessa toisille käyttäjille soittaminen tapahtuu yhteystietolistan kautta. Yhteystiedot ovat toisia käyttäjätilejä, jotka käyttäjä on lisännyt itselleen sovelluksesta. Yhteystietoja lisätään antamalla käyttäjätilin puhelinnumero lisäyskenttään, jolloin jos puhelinnumerolla löydetään käyttäjätili, lisätään se yhteystietolistaan.

Lisättyjä yhteystietoja täytyy voida hallita. Vaikka yhteystiedon nimeä tai puhelinnumeroa ei voi muuttaa, on tärkeää pystyä esimerkiksi hiljentämään halutun yhteystiedon soittojen aiheuttamat ilmoitukset tai estämään puhelut kokonaan. Näitä asetuksia varten tarvitsemme asetusnäkymän, johon navigointi tapahtuu esimerkiksi yhteystietoa painamalla listassa. Tästä näkymästä onnistuu myös yhteystiedon poistaminen.

Videopuhelu on tämän sovelluksen tärkein ominaisuus. Käyttäjän täytyy pystyä soittamaan yhteystiedoilleen sekä ääni- että videopuheluita. Videopuhelun aikana käyttäjät voivat kuulla ja nähdä toisensa reaaliaikaisesti. Käyttäjät voivat myös hallita oman kameransa ja mikrofoninsa käyttöä kesken puhelun. Puhelun saa lopetettua kumpi tahansa osapuoli painamalla punaista painiketta käyttöliittymässä.

Sovelluksella on toiminnallisten vaatimusten lisäksi ei-toiminnallisia vaatimuksia. Ei-toiminnalliset vaatimukset eivät suoraan liity sovelluksen toimintoihin, mutta vaikuttavat kuitenkin suuresti sovelluksen suunnitteluun ja teknisiin päätöksiin. Näistä suurin on julkaisualusta; sovellus halutaan julkaistavan iOS-käyttöjärjestelmää pyörittäville mobiililaitteille, tarkemmin Applen iPhone-älypuhelimille. Tämä vaikuttaa teknologisiin valintoihin, kuten ohjelmointikieliin ja kehitystyökaluihin.

2 PROJEKTIN TEKNOLOGIAT

2.1 React Native

React Native on *Reactiin* pohjautuva ohjelmointikirjasto, joka mahdollistaa natiivin koodin kirjoittamisen JavaScriptillä monelle eri alustalle. Facebookin luoman React Nativen yleisin käyttökohde on mobiilisovellusten kehittäminen, mutta se mahdollistaa kehittämisen myös esimerkiksi tvOS ja Android TV -käyttöjärjestelmille. Kerran kirjoitettu sovellus kääntyy monelle eri alustalle. (Fraktio n.d.; React Native n.d.)

Mobiilisovelluksen kehittäminen Android- ja iOS-alustoille on aikaisemmin vaatinut kehittäjiltä kahden eri sovelluksen tekemisen, sillä natiivi Android-sovellus ei toimi iOS-alustalla ja toisinpäin. React Native kääntää JavaScript-ohjelmointikielen käyttöjärjestelmän natiiviksi koodiksi, jolloin säästytään kahden eri sovelluksen tekemiseltä.

React Native, kuten React, on JavaScriptillä kirjoitettava komponenttipohjainen ohjelmointikirjasto. Komponentit ovat sovelluksen rakennuselementtejä, jotka muodostavat sen käyttöliittymän. Ne ovat itsenäisiä koodinpätkiä, jotka toimivat kuten JavaScript-funktiot. Näissä komponenteissa määritellään, miten käyttöliittymän palaset näytetään ja miten ne reagoivat käyttäjän syötteisiin. Komponentit voivat sisältää toisia komponentteja sekä logiikkaa, joka määrittää, miten sovellus käyttäytyy. React Nativen komponentit perustuvat samoihin perusperiaatteisiin kuin React-komponentit, mutta ne eroavat toteutukseltaan. React Native -komponentit eivät palauta HTML-elementtejä, kuten React, vaan ne tuottavat natiiveja käyttöliittymäkomponentteja, jotka toimivat suoraan kohdealustan, kuten Android tai iOS, kanssa (React Components n.d.). React Nativen komponenttien ulkoasut tyylitellään käyttäen JavaScriptiä, mutta tyylien nimet ja arvot usein vastaavat CSS-tyylikieltä (Style n.d.).

React Nativen komponenttien tilanhallinta perustuu Reactin tavoin state-tilamuuttujiin, jotka sisältävät tietoa komponentin tilasta. State voi sisältää esimerkiksi käyttäjän tekstikenttään antamat syötteet tai listan haetuista

yhteystiedoista. Staten muuttaminen aiheuttaa komponentin uudelleen renderöinnin, mikä tarkoittaa käyttöliittymän päivittämistä vastaamaan uusinta tilaa. Normaalit JavaScript-muuttujat eivät aiheuta uudelleenrenderöintiä, joten komponenttien logiikkaa voidaan toteuttaa ilman käyttöliittymän päivitystä, ja vasta esimerkiksi raskaan laskelman jälkeen voidaan sen tulos päivittää käyttöliittymään muuttamalla sen tilaa. State eli tila mahdollistaa dynaamisen ja muutoksiin reagoivan käyttöliittymän luomisen. (State: A Component's Memory n.d.)

Facebook julkaisi React Nativen vuonna 2015 avoimen lähdekoodin projektina. Facebookin päätettyä sovelluksensa julkaisusta mobiililaitteille, valittiin natiivisovelluksen rakentamisen sijaan HTML5-pohjaisen mobiilisivun. Tämä ratkaisu ei kestänyt, ja vuonna 2013 Facebook-kehittäjä Jordan Walke keksi tavan luoda natiiveja komponentteja JavaScriptiä käyttäen, ja täten syntyi React Native. Ohjelmointikirjasto tehtiin julkiseksi vuonna 2015, ja vain kolme vuotta myöhemmin tämä avoimen lähdekoodin projekti oli toiseksi suurin projekti GitHubissa. (Budziñski 2022.) Tällä ohjelmointikirjastolla on luotu maailman suosituimpia mobiilisovelluksia, kuten Facebook, Microsoft Teams sekä Discord (Who is using React Native n.d.).

2.2 WebRTC

Web Real Time Communication (*WebRTC*), on Googlen vuonna 2011 julkaisema avoimen lähdekoodin projekti, joka tarjoaa rajapinnan video- ja äänisyötteen siirtämiseen kahden laitteen välillä ilman dataa välittävää palvelinta. WebRTC mahdollistaa *peer-to-peer*-tiedonsiirron ilman ulkoisten ohjelmien tai lisäosien asentamista (WebRTC API 2023). WebRTC toimii useimmilla selaimilla sekä natiivialustoilla. Yleisin käyttökohde WebRTC:lle on videopuhelu- ja etäkokousominaisuudet. Sitä käytetään esimerkiksi Microsoft Teams ja Google Meet -sovelluksissa. Tässä projektissa sitä käytetään videopuhelusovelluksen toteuttamiseen.

WebRTC toimii luomalla yhteyden kahden tai useamman laitteen välille ja lähettämällä dataa, kuten videosyötettä, laitteiden kesken. Tämän yhteyden

luomista varten tarvitaan kuitenkin signaalipalvelin, joka välittää tarvittavaa tietoa laitteelta toiselle ja takaisin. Näiden tietojen avulla laitteet pystyvät välittämään keskenään dataa. WebRTC ei siis toimi täysin ilman palvelimia, mutta itse tiedonsiirtoon laitteiden välillä ei palvelimia tarvita.

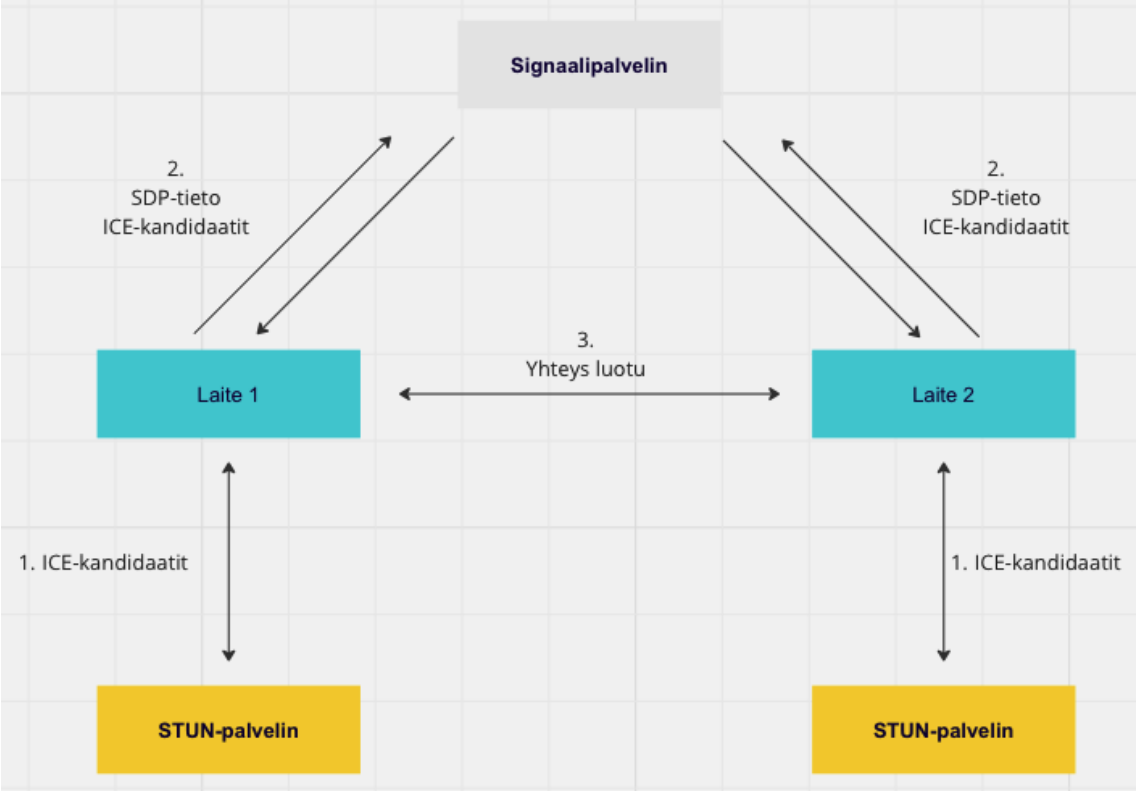
Saadakseen yhteyden laitteiden välille täytyy signaalipalvelimen ensin välittää kahdenlaista tietoa laitteiden välillä. Toinen näistä on Session Description Protocol (SDP), joka on yhteydelle oleellista tietoa laitteen antamasta syötteestä sisältävä tietoformaatti. SDP sisältää esimerkiksi tiedon käyttäjän mediasyötteestä, syötteen tiedostomuodosta sekä siirtoprotokollasta, jota käyttämällä data tullaan välittämään kun yhteys on luotu (Handley 2006, 5). SDP-tieto tarvitaan kaikilta yhdistettäviltä laitteilta, sillä jokaisen laitteen konfiguraatio voi olla täysin eri.

SDP-tiedon lisäksi yhteyden muodostamiseksi tarvitaan laitteiden *ICE*-kandidaatit, joiden avulla laitteet voivat yhdistää suoraan toisiinsa. ICE, eli Interactive Connectivity Establishment, on protokolla, joka mahdollistaa parhaimman mahdollisen reitin löytämisen laitteesta toiseen yhdistämiselle. ICE-kandidaatit sisältävät laitteen tiedonsiirto-osoitteita, joihin kuuluu muun muassa sen julkinen IP-osoite ja portti.

Nykyään palomuurit suojelevat jokaista laitetta erilaisilta verkkohyökkäyksiltä. Palomuurien takia laitteen osoite ei ole julkinen, vaan verkkoyhteydet liikkuvat reitittimen läpi. *NAT*, eli Network Address Translation, on prosessi, joka antaa yhden julkisen IP-osoitteen kaikille sitä käyttävän reitittimen takana oleville laitteille. Suurin osa kotona käytettävistä WiFi-reitittimistä ovat NAT-reitittimiä. Tästä syystä ICE-kandidaatteja ei pysty luomaan suoraan laitteesta, vaan ne täytyy pyytää ulkoisilta palvelimilta.

STUN, eli Session Traversal Utilities for NAT, on protokolla, joka palauttaa tiedon laitteen julkisesta osoitteesta. STUN-palvelimen avulla luodaan lista potentiaalisista ICE-kandidaateista, joista WebRTC valitsee parhaan signaalipalvelimen välityksellä. Yhdistettävien laitteiden SDP-tiedoilla sekä ICE-kandidaateilla saadaan viimein luotua laitteiden välinen yhteys (katso kuvio 1).

Joidenkin reitittimien palomuurien rajoitteiden takia reititin hyväksyy yhteyksiä vain jo aikaisemmin yhdistetyiltä laitteilta, joten STUN-palvelimen sijaan käytetään *TURN*-palvelinta, joka välittää tiedon laitteiden välillä. Tässä projektissa kuitenkin käytetään STUN-palvelimia yhteyksien luomiseen.



KUVIO 1. Peer-to-peer-yhteyden luonti vaiheittain WebRTC:n avulla.

2.3 Firebase

Firebase on Googlen tarjoama *BaaS*-pilvialusta, joka tarjoaa erilaisia palveluita ja rajapintoja sovelluskehityskäyttöön. BaaS, eli Backend-as-a-Service, on palvelu, joka tarjoaa käyttäjälleen backend-toimintoja. Näitä palveluita ovat esimerkiksi tietokannat sekä käyttäjien autentikaatio. Tässä luvussa syvennytään Firebaseen sekä sen tarjoamiin palveluihin.

Sovellusten tietoturvavaatimusten ja monimutkaisuuden kohotessa on perinteisen backend-taustajärjestelmän rakentamisesta tullut usein liian vaativaa ja aikaa vievää etenkin pienelle kehitystiimille. Tästä syystä pilvipalveluiden

käyttö on yleistynyt merkittävästi. Pilvipalvelut tarjoavat ratkaisuja esimerkiksi palvelimen hostaukseen sekä datan säilyttämiseen poistaen tarpeet fyysisille palvelinlaitteille ja kiintolevyille. Useimmat pilvipalvelut tarjoavat myös dynaamisen *pay-as-you-go*-maksutavan, jolloin käyttäjä maksaa vain käyttämistään resursseista. Tämä mahdollistaa halvan käytön pieniin projekteihin, jolloin periaatteessa kuka tahansa voi luoda ohjelmistoja ja sovelluksia ilman ulkoista rahoittajaa.

Firebasen eräs suosituimmista palveluista on *NoSQL-dokumentti-tietokanta Firestore*, joka tarjoaa reaaliaikaisen, automaattisesti skaalaavan tietokantaratkaisun. Firestore toimii kätevästi asennettavan *SDK*:n (Software Development Kit) kautta, jolloin kehittäjät pystyvät rakentamaan kutsuja tietokantaan suoraan frontend-tasolta. Tämä helpottaa ja nopeuttaa huomattavasti sovelluskehitystä. Firestore on dokumenttipohjainen tietokanta, joka säilyttää dataa eri lailla kuin tavanomaisemmat relaatiotietokannat. Dokumenttipohjaiset tietokannat ovat NoSQL-tietokantoja, jotka säilyttävät dataa avain–arvo-pareissa. Avaimena on yleisesti säilytettävän arvon nimike, ja arvona voi olla esimerkiksi tekstiä, numeroita, tai listoja. (Schaefer n.d.). Firestorea käytetään tämän projektin tietokantana.

Käyttäjien salasanojen säilyttämisessä ja autentikaatiossa voi mennä kokemattomalla koodarilla monta asiaa vakavasti väärin. Salasanojen vääränlainen säilytys altistaa käyttäjät usein tietoturvariskeille. Vuonna 2022 tehdyn kyselyn mukaan jopa 62 % vastaajista käyttää samaa tai lähes samaa salasanaa sekä työpaikalla että kotona (LastPass 2022, 3). Jos edes yhden sovelluksen käyttäjätiedot ja salasanat leviävät julkisuuteen huonon tietoturvan takia, voivat monet yritykset ja käyttäjätunnukset olla tietoturvauhan alla. Tästä syystä kehittäjiä kehotetaan käyttämään ulkoista autentikointipalvelua. *Firebase Authentication* on palvelu, joka tarjoaa salasanojen säilyttämisen sekä käyttäjätietojen luomisen, hallinnan ja autentikaation tietoturvallisesti. Firebase Authentication mahdollistaa käyttäjätunnusten rekisteröimisen monella eri kirjautumistavalla ja hoitaa myös tässä projektissa käyttäjätunnusten rekisteröinnin ja autentikaation.

Muita Firebasen tarjoamia palveluita on esimerkiksi Cloud Storage, joka mahdollistaa tiedostojen, kuten kuvien ja videoiden, säilyttämisen pilvialustalla. Firebase Cloud Functions tarjoaa serverless-tietojenkäsittelymalliin perustuvia funktioita, ja Firebase Hosting mahdollistaa pilvipohjaisen palvelinhostauksen.

3 TEKNINEN SUUNNITTELU

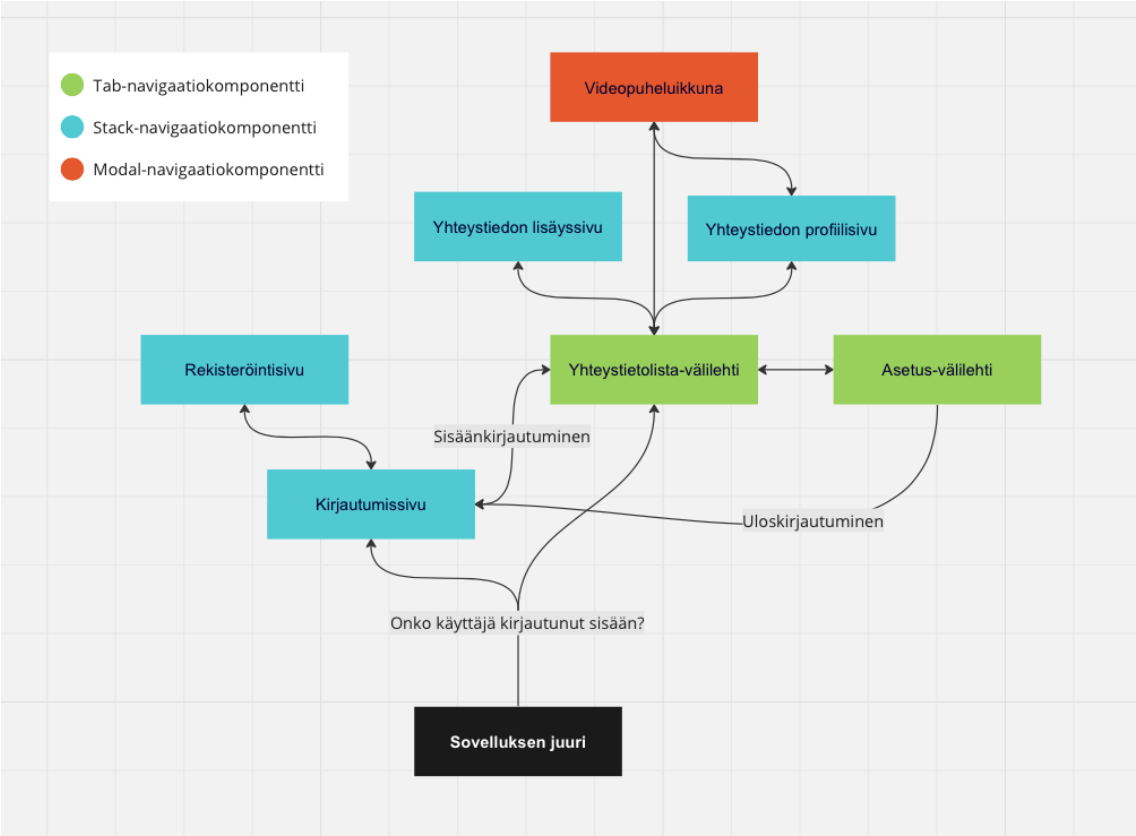
3.1 Sovelluksen rakenne ja komponentit

Sovelluksen arkkitehtuurilla ja rakenteella on merkittävä vaikutus sen suorituskykyyn ja ylläpidettävyyteen. Tässä luvussa tarkastellaan sovelluksen yleistä rakennetta sekä komponentteja, joista rakentuvat sovelluksen erilaiset toiminnallisuudet.

Sovelluksen käyttöliittymän toteuttamiseksi valittiin React Native -mobiiliohjelmointikirjasto. Sovelluksen käyttöliittymä koostuu useista erilaisista käyttöliittymäkomponenteista, jotka on toteutettu React Nativen tarjoamista komponenteista ja räätälöity palvelemaan sovelluksen käyttötarkoitusta. Pienemmistä komponenteista, kuten painikkeet, tekstikentät ja kytkimet, rakentuu kokonaisia käyttöliittymänäkymiä, kuten kirjautumissivu ja yhteystietolista. Yksinkertaiset komponentit voivat taipua moneen käyttötarkoitukseen. Hyvin tehtyinä komponentit ovat monikäyttöisiä, jolloin samaa komponenttia voi hyödyntää monessa eri näkymässä.

Sovelluksen eri näkymät ja niiden väliset siirtymät hoidetaan navigaatiokomponenteilla. Tässä projektissa käytetään *React Navigation* -kirjastoa, joka tarjoaa muun muassa välilehti- sekä ikkunakomponentteja. Näitä komponentteja hyödyntämällä saadaan aikaan sulavasti navigoitava käyttöliittymä.

Sovelluksen navigaation rakenne on järkevää suunnitella perusteellisesti ennen sovelluksen kehittämisen aloittamista, sillä navigaation järjestelyä voi olla haastava muuttaa jälkikäteen. Tämän sovelluksen navigaatio rakentuu kahdesta välilehdestä, joiden sisällä on sivuja, sekä videopuheluikkunasta, joka avautuu täysin käyttöliittymän päälle (katso kuvio 2). React Navigation tarjoaa näitä vastaavat komponentit: *tab*-, *stack*- ja *modal*-komponentit.



KUVIO 2. Sovelluksen navigaation ja siirtymien rakenne.

Videopuhelun toteutus rakentuu WebRTC-teknologian avulla. *React-Native-WebRTC*-apukirjaston avulla saadaan toteutettua suoratoistona kulkeva video- ja äänisyöte. Videopuhelunäkymän päälle rakentuu käyttöliittymä, joka mahdollistaa kameran ja mikrofonin tilan hallinnan sekä puhelun lopettamisen.

Firebase-pilvipalvelun tarjoamaa Firestore-tietokantaa hyödynnetään käyttäjätietojen ja yhteystietojen säilyttämisessä sekä kahden osapuolen välisen videopuheluyhteyden luomisessa. Firestoren tietokantakomponentit päivittävät käyttäjän tekemiä muutoksia tietokantaan sekä noutavat sieltä esimerkiksi käyttäjän yhteystiedot.

Käyttäjän kirjautumistietojen säilyttäminen tavallisessa tietokannassa voi aiheuttaa sovellukseen merkittäviä tietoturvariskejä. Tätä varten tarvitaan palvelu, joka hoitaa käyttäjätilien hallinnan, salasanojen säilyttämisen sekä tietoturvallisen autentikoinnin sisäänkirjautumistilanteessa. Firebase Authentication -palvelu tarjoaa komponentit juuri tähän tarpeeseen.

3.2 Käytetyt työkalut

3.2.1 Mobiilisovelluskehitys

Mobiilisovelluksen kehittämisen mahdollistavat useat työkalut, joiden avulla ohjelmistokehittäjä kirjoittaa koodia ja testaa sovelluksen toimivuutta. Seuraavaksi esitellään neljä keskeistä työkalua tämän sovelluksen toteuttamiseksi.

Visual Studio Code (*VSCode*) on laajalti käytetty koodieditori, joka mahdollistaa kehittäjälle lukuisten eri ohjelmointikielten kirjoittamisen yhdellä editorilla ilman teknologiakohtaisia kehitysympäristöjä. VSCode tukee myös käyttäjien tekemiä laajennuksia, jotka tarjoavat esimerkiksi syntaksin tarkistuksen sekä muita kehittäjää avustavia toimintoja. VSCode on päässyt suureen suosioon ohjelmistokehittäjien keskuudessa. Stack Overflow kyselyssä jopa 74 % äänestäjistä luokitteli sen suosikkikoodieditorikseen (Stack Overflow 2022). Tähän suosioon VSCode on päässyt olemalla ilmainen, kevyt sekä modulaarinen. Nämä ominaisuudet tekevät VSCodesta sopivan kaikentasoisille koodareille.

XCode on Applen kehittämä *integroitu kehitysympäristö* (IDE), joka on suunniteltu erityisesti iOS-sovellusten kehittämiseen. XCode tarjoaa työkalut ja emulaattorit kaikkeen iOS-kehittämiseen. Valitettavasti XCode on tarjolla ainoastaan Applen MacOS-tietokoneille. Tässä projektissa XCodea käytetään projektin koodipohjan alustamiseen sekä sovelluksen koontiversioiden luontiin.

iOS Simulator on osa XCode-kehitysympäristöä ja mahdollistaa iOS-käyttöjärjestelmän ajamisen suoraan tietokoneelta. Simulator tarjoaa emulaattorit kaikille iOS-laitteille mahdollistaen sovelluksen testaamisen esimerkiksi eri näyttöko'oilla.

Sovelluksen toteuttaminen alusta loppuun on vaativa tehtävä, joten siihen käytettävien työkalujen merkitys kehitysprosessissa on valtava. Näiden

työkalujen avulla pystytään varmistamaan tehokas kehittäminen ja testaaminen sekä helpottamaan vianetsintää.

Node tai Node.js on asynkroninen, tapahtumapohjainen JavaScript-ympäristö, joka perustuu Google Chromen V8 -JavaScript moottoriin. Vaikka JavaScript lähtökohtaisesti on selainpohjainen ohjelmointikieli, Node tarjoaa ympäristön, jossa JavaScriptiä voidaan suorittaa myös palvelinpuolella. Vaikka Nodea ei suoraan tämän mobiilisovelluksen kehityksessä tarvitakaan, tarjoaa se monia työkaluja JavaScript-pohjaisten sovellusten tekemiseen. Näistä suurimpana ja tärkeimpänä on *Node package manager* (*npm*). Npm on maailman suurin ohjelmistorekisteri, ja se sisältää yli 800 000 ohjelmistopakettia. Npm mahdollistaa näiden pakettien jakamisen ja asentamisen projektiin vain yhtä terminaalikomentoa käyttämällä. (What is npm n.d.)

Npm mahdollistaa lukuisten eri ohjelmistokirjastojen asentamisen ja hallinnan myös tässä projektissa. Npm asentaa halutun paketin projektiin käyttämällä komentoa `npm install <paketin nimi>`. Asennetun paketin ohjelmistokoodi talletetaan *node_modules*-kansioon projektin juuressa, ja paketin tiedot tallentuvat *package.json*-tiedostoon (katso kuva 1), jotta niiden asentaminen olisi helppoa vaikka kirjastot poistettaisiin projektin tiedostoista. Kirjastojen uudelleenasennus tapahtuu komennolla `npm install`.
Asennettujen pakettien käyttö koodissa tapahtuu tuomalla paketin halutut komponentit tiedostoon käyttämällä import-avainsanaa ja asennetun paketin ja sen komponentin nimeä:

```
import { <komponentin nimi> } from ”<paketin nimi>”
```

KUVA 1. Ohjelmistokirjastojen nimet ja versionumerot näkyvät kätevästi package.json-tiedostosta.

3.2.2 Tekoälypohjainen avustus

Projektin kehittämisen tehokkuuden ja lopputuloksen laadun varmistamiseksi ohjelmointivaiheessa hyödynnetään tekoälypohjaisia työkaluja. Tekoäly pystyy auttamaan koodaria esimerkiksi vianetsinnässä sekä nopeuttamaan yksinkertaisten ja toistuvien koodipätkien kirjoittamista.

Github Copilot on tehokas tekoälypohjainen työkalu, joka on erityisesti suunniteltu olemaan kuin koodarin työkaveri, joka auttaa kirjoittamaan koodia tehokkaasti sekä ehdottaa ratkaisuja. Copilot on maksullinen, mutta opiskelijoille se tarjotaan ilmaiseksi. Copilot integroituu suoraan VSCodeen laajennuksena ja analysoi kirjoitettua koodia reaaliajassa. Copilot ehdottaa kirjoitetulle lausekkeelle loppua toimien kuin koodin *autocomplete.*

Vaikka Copilot voi kirjoittaa kokonaisia funktioita ja toiminnallisuuksia, löysin itse eniten hyötyä siitä toistuvien koodinpätkien nopeassa kirjoittamisessa. Tästä esimerkki on React Nativen *useState*-funktioilla luotavat tilamuuttujat (katso kuva 2).

KUVA 2. Github Copilotin ehdottama koodi tilamuuttujan luomiseen.

ChatGPT on chat-pohjainen tekoälyavustaja, joka pystyy teoriassa neuvomaan missä aihealueessa vain. ChatGPT perustuu OpenAI:n GPT-3.5-teknologiaan, ja vaikka maksullinen versio tarjoaa uudemman ja älykkäämmän GPT-4.0:n, riittää ilmaisversio varsin hyvin useimmissa käyttötarpeissa. GPT on saanut suurta suosiota vuonna 2023 helposti saavutettavan ja älykkään chatin ansiosta, mutta ehkä eniten hyötyä ChatGPT:stä ovat saaneet ohjelmistokehittäjät, sillä GPT osaa kirjoittaa mitä tahansa ohjelmointikieltä sekä ratkaista vikoja ja ongelmia annetusta koodista. Lisäksi ChatGPT pystyy suunnittelemaan monimutkaisia ratkaisuja esimerkiksi videopuhelusovelluksen toteuttamiseksi.

Tässä projektissa ChatGPT auttoi paljon ohjelmointiprojektin alustamisessa, sillä iOS-sovelluksen luominen monine konfigurointeineen tuottaa haasteita kokeneellekin koodarille. Hyötyä löytyi myös vianetsinnässä, mutta toimivaa koodia GPT ei aina osaa kirjoittaa. Tämän takia sitä on hyvä käyttää enemmänkin auttavana työkaluna, kuten Github Copilotia, eikä kaikkitietävänä koodigeneraattorina, sillä se ei sitä ole.

3.3 Käyttöliittymäsuunnittelu

3.3.1 Käyttöliittymä

Käyttöliittymän suunnittelun merkitys ohjelmistoprojektissa on suuri. Käyttöliittymä on käyttäjän ensimmäinen ja usein ainoa kosketuspinta sovellukseen. Hyvä käyttöliittymä ja sen ulkoasu vaikuttaa sekä itse sovelluksen, mutta myös koko yrityksen imagoon.

Sovelluksen käyttäjät ärsyyntyvät helposti huonosti suunnitellusta käyttöliittymästä. Kun painike ei teekään sitä mitä käyttäjä odottaa sen tekevän, voi hän kyllästyä sovellukseen jo ensikättelyssä. Hyvä käyttöliittymä parantaa käyttäjän mielikuvaa sovelluksesta, ja tämä usein tuo sovellukselle hyviä arvosteluja, jotka taas tuovat lisää asiakkaita. Käyttöliittymän ollessa intuitiivinen ja looginen käyttää on käyttäjä todennäköisesti tehokkaampi käyttämään sovellusta. Hyvä käyttötehokkuus sovelluksissa voi yritystoiminnassa olla jopa rahan arvoinen asia.

Mobiilisovellusmarkkinoiden kasvaessa kehitetään koko ajan hienompia ja parempia käyttöliittymiä. Hyvän ulkoasun kehittäminen antaa sovellukselle paremman mahdollisuuden menestyä markkinoilla muiden sovelluksien seassa. (How can UI/UX improve your business 2023.)

Minkälainen sitten on hyvä käyttöliittymä? Hyvän ja huono käyttöliittymän ero on siinä, kuinka helppoa käyttäjän on saavuttaa tavoitteensa: hyvän käyttöliittymän tulisi olla helposti ymmärrettävä, johdonmukainen, reagoiva sekä saavutettava, ja sen täytyisi tarjota palautetta käyttäjälle heidän sitä käyttäessään (Niebla 2023).

3.3.2 Ulkoasu

Käyttöliittymän ulkoasuun vaikuttavat esimerkiksi painikkeiden sijainnit, tekstien fontit, näkymien taustavärit sekä lukuisat muut asiat. Näiden asioiden suunnitteleminen etukäteen voi helpottaa myös kehittäjän työtä, sillä hyvän ulkoasun ollessa valmiiksi määriteltynä tarvitsee koodarin keskittyä vain sen toteuttamiseen, jolloin aikaa jää enemmän toiminnallisuuksien kehittämiseen.

Ulkoasua suunniteltaessa on usein hyvä hakea inspiraatiota muista samankaltaisista sovelluksista. Käyttöliittymän ulkoasua suunniteltaessa usein pidetään mielessä brändin visuaaliset ominaisuudet, etenkin väri sekä logot. Tämä projekti tehtiin oppimistarkoitukseen, joten brändin värit sai valita oman mielen mukaan. Tämän sovelluksen ulkoasun suunnittelussa haettiin inspiraatiota musiikin suoratoistopalvelu Spotifyn mobiilisovelluksesta. Spotifystä otettiin mallia esimerkiksi painikkeiden ulkoasussa sekä värimaailmassa (katso liitteet 1 ja 2).

Käyttöliittymän responsiivisuus on myös olennainen osa käyttöliittymän suunnittelua. Responsiivisuus mahdollistaa sovelluksen käytön eri laitteilla ja näyttöko'oilla. Responsiivisuuden toteuttaminen vaatii paljon testailua kehittäjiltä, mutta iOS-simulaattorin lukuisat eri laite-emulaattorit auttoivat tässä.

3.3.3 Teemat

Nykypäivänä suuri osa merkittävistä sovelluksista, nettisivuista ja jopa kokonaisista käyttöjärjestelmistä tarjoavat mahdollisuuden käyttää niin sanottua *dark mode* -teemaa, jolloin sovelluksen normaalisti vaaleat värit vaihdetaan tummiin ja teksti käännetään mustasta valkoiseksi. Dark modesta on tullut suosittu, sillä jopa 82,7 % Thomas Steinerin tutkimukseen vastanneista kertoi käyttävänsä käyttöjärjestelmänsä tarjoamaa tummaa teemaa (Steiner 2019). Tämän projektin sovellukseenkin haluttiin siis luoda mahdollisuus vaihtaa vaalean ja tumman teeman välillä.

Tumman ja vaalean teeman luominen onnistuu määrittämällä värit erikseen teemoihin. Näiden välillä vaihtaminen onnistuu kuuntelemalla käyttöjärjestelmän rajapinnasta tulevaa teematietoa, jolloin sovelluksen teema vaihtuu samalla kun koko käyttöjärjestelmän teemaa vaihdetaan. Tämä mahdollistaa myös sen, että sovelluksen teema vaihtuu käyttöjärjestelmässä määritetyn kellonajan mukaan.

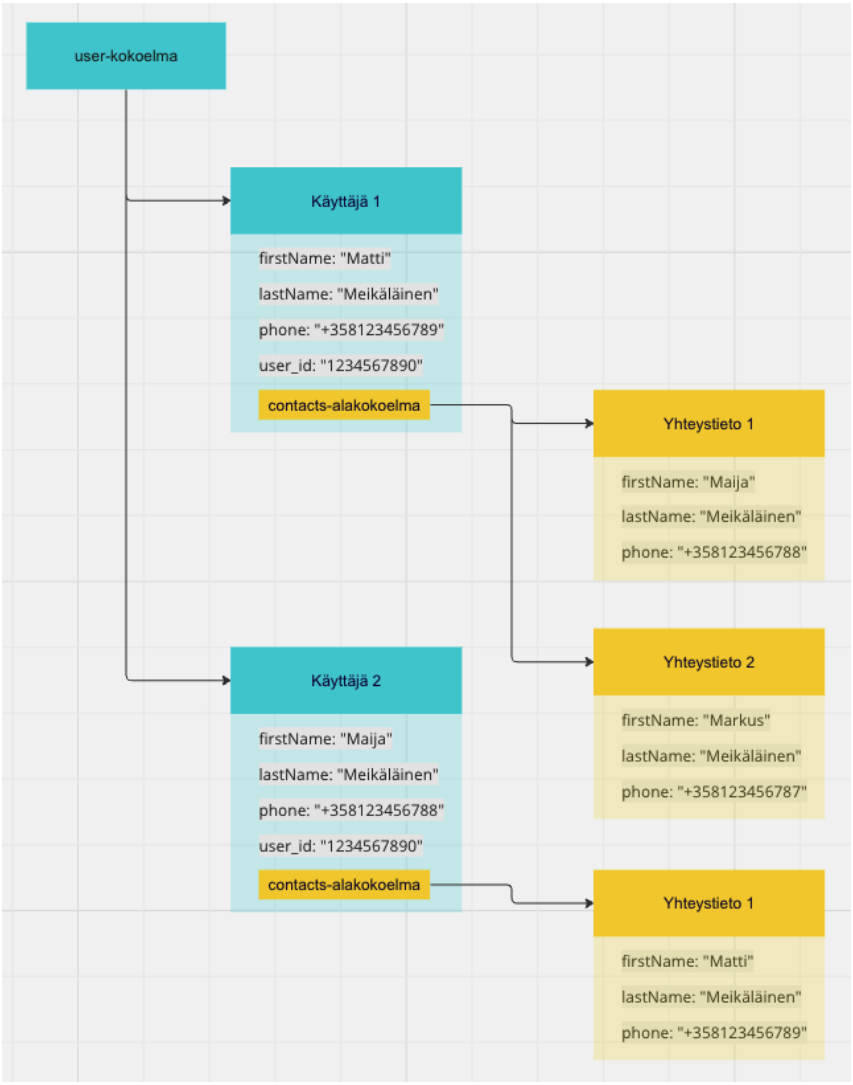
3.4 Tietokantamalli

Tietokantamalli on tietokantatauluista ja niiden yhteyksistä muodostuva rakenne, joka näyttää, miten tietokannan eri taulut ja niiden data ovat kytköksissä toisiinsa. Tietokantamalli suunnitellaan ennen projektin kehittämisen aloitusta, mikä helpottaa datan organisointia ja mahdollistaa tehokkaamman tietojenkäsittelyn. Hyvin suunniteltuun tietokantamalliin voi myöhemminkin lisätä tauluja sekä relaatioita niiden välille.

Videopuhelusovelluksen tietokantamallissa täytyi ottaa huomioon esimerkiksi käyttäjät ja yhteystiedot sekä niiden väliset relaatiot. Myös itse videopuhelu tarvitsee tietokantaa säilyttääkseen puheluyhteydelle tärkeitä tietoja. Projektissa käytetään Firebasen tarjoamaa Firestore-tietokantaa. Firestore on dokumenttipohjainen NoSQL-tietokanta, joka yleisemmän relationaalisuuden sijaan tarjoaa dokumenttikokoelmia (*collections*) sekä niiden alakokoelmia (*subcollections*). Näitä kokoelmia hyödyntämällä taulujen sijaan saadaan rakennettua yksinkertainen tietokantamalli, josta dataa saadaan haettua yhdellä haulla ilman datan yhdistelemistä.

Käyttäjiä varten luodaan kokoelma *user*, jonka alle luodaan alakokoelma *contacts*, jonka sisälle talletetaan käyttäjän yhteystiedot. Tällä rakenteella käyttäjän yhteystiedot ovat suoraan käyttäjän alapuolella tietokannassa, jolloin käyttäjää haettaessa sovellukseen saadaan myös sen lisäämät yhteystiedot. Tämä tarkoittaa myös sitä, että käyttäjän yhteystiedot ovat täysin riippumattomia toisista käyttäjistä, joten käytännössä käyttäjä voisi lisätä yhteystiedoksi henkilön, joka ei ole sovellukseen rekisteröitynyt.

User-kokoelmaan tallennetaan jokaisesta käyttäjästä etunimi, sukunimi, puhelinnumero sekä Firebase Authenticationia vastaava käyttäjätunniste, jolla osataan yhdistää oikea kirjautumistieto oikeaan käyttäjään tietokannassa. User-kokoelman contacts-alakokoelmaan tallennetaan jokaista yhteystietoa vastaava puhelinnumero sekä etunimi ja sukunimi. Kuviossa 3 tietokantamallin rakenne.



KUVIO 3. Käyttäjien ja yhteystietojen tietokantamalli.

Rooms-kokoelma sisältää puheluyhteyden luomiseen vaadittavat ns. huoneet, jotka sisältävät soittajan sekä vastaajan SDP-tekstidatan, jonka avulla luodaan yhteys vastaanottajan laitteeseen. Kokoelma sisältää myös laitteiden ICE-kandidaatit. Tätä tietokantakokoelmaa pystytään käyttämään sovelluksen signaalipalvelimena, sillä tietokannan reaaliaikainen yhteys mahdollistaa puheluyhteyden luomiselle tarvittavien tietojen hakemisen molemmille osapuolille. Tällöin siis tietokantayhteys yhdistää puhelun osapuolet toisiinsa.

4 SOVELLUKSEN KEHITYS

4.1 Sovelluksen pohja

4.1.1 Projektin alustus

Sovelluksen kehitysprosessin ensimmäinen vaihe on projektin alustus, jolla viitataan tarvittavien työkalujen asentamiseen, sovelluksen pohjakoodin luomiseen sekä erilaisten konfiguraatioiden tekemiseen. Videopuhelusovelluksen alustuksessa tehtiin seuraavat asiat.

React Native -sovelluksen koodipohjan luominen vaatii muutaman riippuvuuden asentamisen. Näiden asentaminen tapahtui komentoriviltä käsin. Asennettavista riippuvuuksista oleellisimpia olivat seuraavat:

- Node.js, jonka avulla asennetaan ohjelmistokirjastoja, luodaan uusi projekti sekä ajetaan testiympäristöä.
- *Ruby*, joka on Pythonin kaltainen ohjelmistokieli, jota käytetään joidenkin React Nativen riippuvuuksien hallintaan.
- *CocoaPods*, jolla hallitaan iOS-riippuvuuksia.

Riippuvuuksien asennuttua luotiin koodipohja React Native -sovellukselle. Tämä tapahtui Noden mukana tulevalla *npx*-komentorivityökalulla ajamalla komentoriviltä `npx react-native@latest init Videopuhelusovellus`.

Koodipohjan luomisen jälkeen aloitettiin sovelluksen toiminnallisuuteen vaadittavien ohjelmistokirjastojen asennus. Näiden asennus tapahtuu käyttämällä Noden mukana asennettua npm-komentorivityökalua, joka hakee ja asentaa pyydettävän ohjelmistokirjaston npm-rekisteristä sekä asentaa sille vaadittavat riippuvuudet projektiin. Ohjelmistokirjastojen asennus tapahtui komennolla `npm install <paketin-nimi>`.

Asennettuihin kirjastoihin kuuluivat muun muassa seuraavat:

- *react-native-firebase*, joka tarjoaa funktioita ja komponentteja Firebase-pilvipalvelun ja React Native -sovelluksen väliselle tiedonsiirrolle
- *react-native-webrtc*, jonka tarjoamilla komponenteilla saadaan luotua videopuheluyhteys laitteiden välille
- *react-native-vector-icons*, joka tarjoaa sovelluksen käyttöliittymässä käytettyjä ikoneja eli pieniä kuvakkeita
- *react-navigation*, joka tarjoaa komponentit erilaisille navigaationäkymille ja toiminnallisuuden niiden välillä siirtymiseen

Koodipohjan luonnin ja tarvittavien kirjastojen asennuksen jälkeen kokeiltiin, käynnistyykö sovellus. Aluksi pystytettiin tarvittavat testiympäristöt, jotka olivat iOS Simulator sekä fyysinen iOS-laite, sillä emulaattori ei tarjonnut kaikkia tarvittavia rajapintoja kunnollisen videopuhelun soittamiseksi.

Projektin asennus fyysiselle laitteelle vaati Apple-kehittäjätilin, joka täytyi rekisteröidä Applen kehittäjäsivuilla. Fyysisenä laitteena käytettiin iPhone 12 mini -älypuhelinta. Puhelin täytyi myös asettaa kehittäjätilaan, mikä tapahtui iPhonen asetuksista. Tämän jälkeen laite liitettiin kehityskäytössä olevaan tietokoneeseen sekä asetettiin aktiiviseksi XCoden koontiasetuksissa. Testiympäristön ollessa valmis saatiin uusi sovellus asennettua puhelimelle painamalla run-painiketta XCodessa.

iOS Simulatorin käynnistäminen oli jonkin verran helpompaa, mutta ei tarjonnut samanlaista kosketuspintaa kuin fyysinen laite. Emulaattorin ajaminen tapahtui XCoden koontiasetuksissa valitsemalla laite lukuisista iOS Simulator -laitteista. Emulaattorilaitteeksi valittiin iPhone 13, sillä sen suurempi näyttökoko käytössä olevaan fyysiseen laitteeseen verrattuna antoi laajemman kuvan sovelluksen ulkonäöstä. Painamalla run-painiketta käynnistyi valittu emulaattori sekä sovellus.

4.1.2 Käyttöliittymän alustaminen

Sovelluksen ominaisuuksia voi olla hankala testata ilman kunnon käyttöliittymää, joten sen kehittäminen aloitettiin seuraavaksi. Käyttöliittymä on hyvä tehdä ennen

sovelluksen toiminnallisia ominaisuuksia ja logiikkaa. Se antaa kehittäjälle selkeän käsityksen odotetusta toiminnallisuudesta sekä jotain, mihin voi palata koodatessa järjestelmää. Käyttöliittymä usein määrittää sen, minkälaisia toimintoja taustajärjestelmältä vaaditaan. Tämän lisäksi se helpottaa sovelluksen testausta kehittämisen aikana. Ennen sovelluksen toiminnallisuuden varsinaista kehittämistä luotiin sovelluksen navigointi sekä alustettiin teemat, jotka helpottavat myöhempää käyttöliittymäkehitystä.

Navigaatio oli järkevintä tehdä ensimmäisenä, sillä sen kautta päästiin eri näkymiin testaamaan erilaisia toiminnallisuuksia helposti. Navigaatio toteutettiin React Navigation -ohjelmistokirjastoa hyödyntäen (katso luku 3.1). Eri näkymien ja niiden välisten siirtymien suunnitelmaa seuraten saatiin nämä ohjelmoitua melko helposti.

Ensin luotiin kaksi eri navigaatiopinoa, joista toinen hoitaisi sisäänkirjautumis- ja rekisteröintinäkymän ja toinen vastaisi itse sovelluksen sisällöstä. Näiden kahden navigaatiopinon välillä vaihdeltaisiin sen perusteella, onko käyttäjä kirjautunut sovellukseen sisään vai ei (katso kuva 3).

```
166   // Jos käyttäjä on kirjautunut sisään, näytetään sovelluksen pääsivu, muuten kirjautumisnäkymä
167   return (
168     <NavigationContainer>
169       {user ? <NavigationTabs /> : <AuthenticationStack />}
170     </NavigationContainer>
171   );
```

KUVA 3. Sovelluksen juuri, joka määrittää näytettävän navigaatiopinon käyttäjän autentikaatiotilan perusteella.

Sisäänkirjautumisen jälkeen käyttäjä ohjataan *NavigationTabs*-autentikaationäkymälle, joka sisältää kaikki itse sovelluksen näkymät ja niiden väliset siirtymälogiikat (katso kuva 4). React Navigationin tab -komponentteja käytettiin sovelluksen alareunassa olevan navigaatiopalkin luomiseen. Tabit eli välilehdet ohjaavat stack-komponenttia käyttäviin näkymiin. Näkymien välinen siirtymä tapahtui helposti kutsumalla funktiota `navigation.navigate()` antaen parametriksi halutun navigaationäkymän nimi (katso liite 3).

```
function NavigationTabs({theme}) {
  return (
    <Tab.Navigator
      initialRouteName="Home"
      screenOptions={
        {
          // Navigointipalkin tyyli ja muita asetuksia
        }
      }>
      <Tab.Screen
        name="Home"
        component={HomeView}
        options={
          {
            // Navigointivälilehden nimi ja ikoni
          }
        }
      />
      <Tab.Screen
        name="Settings"
        component={SettingsView}
        options={
          {
            // Navigointivälilehden nimi ja ikoni
          }
        }
      />
    </Tab.Navigator>
  );
}
```

KUVA 4. Tab-navigaation rakenne, joka ohjaa käyttäjän *HomeView*- ja *SettingsView*-näkymiin.

Stack- ja tab-navigaatiokomponentteihin liitettiin component-parametriin uusi komponentti, joka sisältäisi navigaatiota vastaavan näkymän. Navigaatiokomponenttien ohjaamiin näkymiin oli nyt helppo ohjelmoida kaikki näkymälle tarpeelliset ominaisuudet ja ulkoasut, sillä näkymien pohja oli valmis.

Ennen kuin käyttöliittymäkehitystä oli järkevää jatkaa pidemmälle, alustettiin sovelluksen teemojen tiedot sekä toiminnallisuus. Teemoille tehtiin oma tiedosto, joka sisältäisi teemoille annetut tiedot, kuten fonttikoot ja -tyylit, sekä teemakohtaiset muuttujat, kuten taustavärit. Tätä teematiedostoa käyttämällä käyttöliittymän koodissa oli komponenttien tyylien yhtenäisyyden pitäminen helppoa. Jos värejä tai fontteja päätettäisiin vaihtaa myöhemmin, tapahtuisi se kätevästi tätä tiedostoa muokkaamalla (katso kuva 5).

```
const darkTheme = {
  colors: {
    primary: '#00A676',
    primaryLight: '#005940',
    secondary: '#276fbf',
    secondaryLight: '#174375',
    background: '#171717',
    text: '#fff',
    textSecondary: '#a5a5a5',
  },
  text: {
    headerTextSize: 18,
    bodyTextSize: 14,
    font: 'Futura',
  },
  opacity: {
    disabled: 0.33,
  },
  size: {
    margin: 10,
    padding: 10,
    borderRadius: 10,
  },
};
```

KUVA 5. Esimerkki teemojen väri-, koko- ja fonttiarvoista.

Teemojen täytyi vaihtua käyttöjärjestelmän teeman muutoksien perusteella tumman ja vaalean välillä. Tämä onnistui React Nativen *useColorScheme*-funktiota käyttämällä, joka palauttaa arvon ”dark” tai ”light” käyttöjärjestelmän oman väriteeman perusteella. Tätä arvoa käyttämällä tehtiin oma *useTheme*-funktio, joka antaa sitä kutsuvalle komponentille oikean teeman tiedot (katso kuva 6). Komponentissa useTheme-funktiota kutsuttiin luomalla siitä muuttuja `const theme = useTheme()`. Teeman arvoja komponentin tyyleissä käytettiin viittaamalla muuttujaan teeman sisältä, esimerkiksi `theme.colors.primary`.

```
const useTheme = () => {
  const colorScheme = useColorScheme(); // Kutsutaan käyttöjärjestelmän teemaa

  return colorScheme === 'dark' ? darkTheme : lightTheme; // Palautetaan teeman tiedot käyttöjärjestelmän teeman perusteella
};

export default useTheme;
```

KUVA 6. UseTheme-funktion koodi, joka palauttaa oikean teeman sitä kutsuvalle komponentille.

4.1.3 Käyttäjän autentikaatio

Sovelluksen toiminnallisuudelle elintärkeä ominaisuus on käyttäjien luominen sekä niiden sisäänkirjautuminen. Käyttäjän autentikaation tila ohjaa sen kirjautumisnäkymän ja itse sovelluksen välillä, se on siis tärkeä tehdä ennen muiden ominaisuuksien luomista.

Sovelluksen autentikointi on tietoturvan kannalta järkevä tehdä ulkoisen autentikaatiotarjoajan kautta, sillä kokematon kehittäjä ei välttämättä tiedä tietoturvallisen sisäänkirjautumisen ja käyttäjätietojen tallentamisen viimeisimpiä vaatimuksia. Tämän takia projektissa käytettiin Firebase Authentication -palvelua, joka tarjosi helpon rajapinnan käyttäjien luontiin ja autentikointiin.

Sisäänkirjautuminen tapahtui kutsumalla Firebase Authenticationin tarjoamaa *signInWithEmailAndPassword*-kirjautumisfunktiota, joka tarkastaa, vastaako annettu sähköpostiosoite ja salasana olemassa olevaa käyttäjää. Jos funktion käyttämä rajapinta palauttaa myönteisen vastauksen, päästetään käyttäjä sisälle sovellukseen. Jos taas ei, näytetään käyttäjälle teksti ”invalid username or password”.

Käyttäjän rekisteröinti toimi hyvin samalla tavoin, kutsumalla *createUserWithEmailAndPassword*-funktiota, joka luo Firebase Authenticationin järjestelmään uuden käyttäjätiedon. Uuden käyttäjätiedon luomisen jälkeen lisätään myös tietokannan user-tauluun rivi, joka sisältää käyttäjän nimen ja puhelinnumeron sekä Authentication-käyttäjätietoon yhdistävän tunnisteen *uid* (katso kuva 7*)*.

```
161  const onHandleSignup = async () => {
162    // Tarkistetaan että sähköposti ja salasana on annettu ja että salasanat täsmäävät
163    if (email !== '' && password !== '' && passwordAgain === password) {
164      auth()
165        .createUserWithEmailAndPassword(email, password)
166        .then(user => {
167          // Kutsutaan funktiota, joka luo käyttäjän myös tietokantaan
168          createUser(user.user);
169        })
170        .catch(err => console.log(err));
171    }
172  };
173
174  // Luodaan käyttäjä tietokantaan
175  const createUser = async user => {
176    if (user) {
177      await firestore().collection('user').doc(user.uid).set({
178        uid: user.uid,
179        email: user.email,
180        createdAt: firebase.firestore.FieldValue.serverTimestamp(),
181        firstName: firstName,
182        lastName: lastName,
183        phone: phone,
184      });
185    }
186  };
```

KUVA 7. Käyttäjän rekisteröinnin funktiot. *OnHandleSignup* luo käyttäjätiedon Firebase Authentication -järjestelmään, minkä jälkeen *createUser* luo käyttäjän tietokantaan.

4.2 Sovelluksen toiminnallisuuden toteutus

4.2.1 Yhteystietojen käsittely

Sovelluksen toiminnallisuus videopuheluiden soittamiseen vaatii, että käyttäjä pystyy valitsemaan yhteystiedoista henkilön kenelle soittaa. Näitä yhteystietoja täytyy pystyä hallitsemaan, lisäämään sekä selailemaan ja etsimään käytännöllisesti.

Sovelluksen etusivuksi sisäänkirjautumisen onnistuttua tuli näkymä, joka sisältää listan kaikista yhteystiedoista. Tässä näkymässä täytyi pystyä myös hakemaan, mikä nopeuttaisi halutun henkilön löytämistä, mikäli lista olisi pitkä. Näkymän sisältö rakennettiin käyttämällä React Nativen tarjoamaa *SectionList*-komponenttia, jonka avulla voi luoda jaoteltuja listoja. Komponentti on hyödyllinen etenkin, jos listan rivit halutaan esittää otsikoiduissa osioissa. Tässä

tapauksessa yhteystiedot jaettiin osioihin etunimen ensimmäisen kirjaimen perusteella, jotta listan lukeminen olisi sujuvaa käyttäjälle (katso kuva 8).

```
// 1. Järjestä yhteystiedot aakkosjärjestykseen
const sortedContacts = filteredContacts.sort((a, b) =>
  a.firstName.localeCompare(b.firstName),
);

// 2. Luo osiot aakkosjärjestyksessä
const sections = sortedContacts.reduce((acc, contact) => {
  const firstLetter = contact.firstName.charAt(0).toUpperCase();
  // Luo osio aakkosen mukaan, jos sitä ei ole vielä olemassa, muuten lisää yhteystieto osioon
  if (!acc[firstLetter]) {
    acc[firstLetter] = {
      title: firstLetter,
      data: [contact],
    };
  } else {
    acc[firstLetter].data.push(contact);
  }
  return acc;
}, {});

// 3. Muuta objekti taulukoksi
const result = Object.values(sections);

// 4. Aseta taulukko tilamuuttujaan
setContactSectionList(result);
```

KUVA 8. Hakusanalla suodattamisen jälkeen yhteystietolista jaotellaan aakkosjärjestyksessä osioihin etunimen ensimmäisen kirjaimen perusteella.

Yhteystietojen lisäämiselle luotiin näkymä, johon käyttäjä pääsee painamalla plus-painiketta yhteystietolistan yläkulmasta. Näkymässä käyttäjä syöttää henkilön puhelinnumeron kenttään ja lisää tämän yhteystiedoksi. Käyttäjän tiedot haetaan puhelinnumeron perusteella tietokannasta, ja palautuvat arvot lisätään käyttäjän contacts-alakokoelmaan.

Yhteystietoa pystyy myös hallinnoimaan painamalla sen kohdalta yhteystietolistassa, mikä navigoi käyttäjän ContactView-näkymään. Tässä näkymässä pystyy esimerkiksi poistamaan yhteystiedon listasta sekä estämään yhteystiedolta tulevat ilmoitukset. Tämä näkymä tarjoaa tilaa monelle ominaisuudelle, joita voidaan lisätä jatkokehityksessä.

4.2.2 Videopuhelun soittaminen

Videopuheluominaisuuden toteuttaminen oli koko projektin tärkein osuus. WebRTC-teknologiaa hyödyntäen toteutettiin kahden käyttäjän välinen

videopuheluyhteys. Puheluyhteyden luomiseksi tarvittiin signaalipalvelin, joka välittäisi SDP-tiedon sekä ICE-kandidaatit puhelun osapuolille (katso luku 2.2). Signaalipalvelin toteutettiin Firestore-kokoelmalla, johon talletettiin tarvittavat tiedot jokaista puheluyhteyttä kohden (katso luku 3.4).

Puhelun alkaessa ja siihen vastattaessa luodaan uusi *RTCPeerConnection*-olio STUN-palvelimen palauttamilla ICE-kandidaateilla (katso kuva 9). RTCPeerConnection huolehtii laitteiden yhdistämisestä. STUN-palvelimena käytettiin tässä projektissa Googlen ilmaiseksi tarjoamaa palvelinta. Tämän lisäksi luodaan uusi mediasyöte-tilamuuttuja, jota näytetään käyttäjän omana videosyötteenä sekä lisätään RTCPeerConnection-objektiin. Mediasyöte saadaan laitteesta kutsumalla `mediaDevices.getUserMedia(constraints)`, jossa constraints on lista halutuista muuttujista, kuten videon resoluutio.

```
112    const servers = {
113      iceServers: [
114        {
115          urls: ['stun:stun1.l.google.com:19302'],
116        },
117      ],
118      iceCandidatePoolSize: 10,
119    };
120
121    // Luodaan RTCPeerConnection
122    peerConnection.current = new RTCPeerConnection(servers);
```

KUVA 9. Luodaan peerConnection, joka hoitaa yhteyden luomisen ja ylläpidon. Käytössä on Googlen ilmaiseksi tarjoama STUN-palvelin.

PeerConnection-olion luomisen jälkeen tehtiin toiminnallisuus yhteyden luomiseen. Soittajan mediasyöte lisätään peerConnection-objektiin, minkä jälkeen saadut ICE-kandidaatit lisätään signaalipalvelimen *callerCandidates*-kokoelmaan. Tämän jälkeen soittajan SDP-tieto lisätään signaalipalvelimelle:

```
const offer = await peerConnection.current.createOffer();
await peerConnection.current.setLocalDescription(offer);
await roomRef.set({offer});
```

Yhteyden alustuksen jälkeen tarvitsi enää vain kuunnella vastaajan tietoja signaalipalvelimelta. Tätä varten tehtiin kaksi reaaliaikaista yhteyttä palvelimelle. Toinen näistä kuuntelee rooms-kokoelmaa ja vastaanottajan SDP-tiedon ilmestyessä sinne lisää sen peerConnection-objektiin. Toinen kuuntelee *calleeCandidates*-alakokoelmaa sekä lisää sinne ilmenneet ICE-kandidaatit peerConnection-olioon.

Puheluun vastaamisen logiikka toimi hyvin samoin periaattein kuin soittaminenkin. PeerConnection alustettiin signaalipalvelimelta haetuilla tiedoilla ja molempien osapuolien tietojen ollessa peerConnectionissa pystyttiin viimein näyttämään molempien videosyötteet ja toistamaan äänisyötettä. Videopuheluyhteys oli siis muodostettu.

Videopuhelun asetusten, kuten kameran suunnan, asettaminen kesken puhelun oli helppoa. Kameran suunnan vaihtaminen toteutettiin funktiolla *switchCameraFeed*, jonka sisällä kutsuttiin *localStream*-staten antamaa *switchCamera*-funktiota (katso kuva 10).

```
262  const switchCameraFeed = () => {
263    localStream.getVideoTracks().forEach(track => track._switchCamera());
264  };
```

KUVA 10. Kameran suunnan vaihtamisen toiminnallisuus.

Viimeisenä toteutettiin puhelun lopettaminen, joka tapahtuisi punaisesta painikkeesta. Puhelun loppuessa kutsutaan ensin localStream- sekä *remoteStream*-mediasyötteiden stop-funktioita. Tämän jälkeen peerConnection suljetaan, ja käyttäjä navigoidaan takaisin näkymään, mistä hän puhelun soitti.

5 LOPPUTULOS

Sovelluksen pääominaisuuksien valmistuttua oli projekti saatu päätökseen. Tässä luvussa käydään läpi sovelluksen käyttöliittymän toiminnallisuus ja ulkoasu sekä pääominaisuuksien käyttö.

Sovelluksen käyttöliittymä ja sen ulkoasu valmistui pääominaisuuksien osalta, jotka olivat sisäänkirjautuminen sekä rekisteröinti, yhteystietolista ja videopuhelun soittaminen. Käyttöliittymä luotiin tukemaan sekä tummia että vaaleita teemoja.

Yhteystietolista on pystysuunnassa selattava lista yhteystiedoista, jotka ovat lajiteltuina etunimen ensimmäisen kirjaimen mukaan osioihin (katso liite 4). Näissä osioissa näkyy myös niihin kuuluvien yhteystietojen lukumäärät. Listasta pystyy etsimään kontaktin sekä nimen että puhelinnumeron mukaan. Yhteystietoa painamalla avautuu yhteystiedon hallintasivu (katso liite 5), ja päädyssä olevasta vihreästä napista avautuu videopuhelu. Näkymän oikeasta yläreunasta avautuu yhteystiedon lisäys -näkymä, jonka käyttöliittymää ei toteutettu.

Rekisteröinti- ja käyttäjän sisäänkirjautuminen -näkymiin toteutettiin käyttöliittymät sekä niihin suunnitellut ulkoasut. Sisäänkirjautuminen tapahtuu syöttämällä vaadittavat tiedot kenttiin ja painamalla ”Sign in” -painiketta (katso liitteet 6 ja 7). Rekisteröintiin käyttäjä pääsee painamalla ”Sign up!” -painiketta, joka siirtää käyttäjän rekisteröitymisnäkymään (katso liite 8).

Pääominaisuutena sovelluksessa oli videopuhelun soittaminen, joka toteutettiin käyttöliittymän osalta täysin. Kun käyttäjä soittaa puhelun, avautuu hänelle puhelunäkymä, jossa hän näkee oman videosyötteensä sekä myös toisen osapuolen syötteen hänen vastatessaan. Osapuolet myös puhelun tavoin kuulevat toistensa äänet. Videopuhelun aikana käyttöliittymän painikkeista pystyy lopettamaan puhelun, kääntämään kameraa, sulkemaan videosyötteen sekä mykistämään oman mikrofonin. Videopuhelun soitto ei aiheuta ilmoitusta tai hälytystä vastaanottajan sovelluksessa tai laitteessa.

6 POHDINTA

6.1 Tulokset ja puutteet

Projektin päätarkoituksena oli suunnitella ja toteuttaa videopuhelusovellus sekä raportoida projektin valintoja ja etenemistä. Sovelluksen pääominaisuudet valmistuivat käyttökelpoisiksi, mutta niihin jäi puutteita. Vaikka sovelluksen toteuttaminen ei ollut vaikeaa, kasvoi työn määrä aiempaa oletettua suuremmaksi ominaisuuksia kehittäessä. Projektin määrittelyssä tuotiin esiin halutut ominaisuudet, mutta niiden toteuttamiseksi tarvittava lisätyö jäi huomioimatta. Tästä hyvä esimerkki on käyttäjän rekisteröityminen, jonka tarkoitus oli vaatia käyttäjältä puhelinnumeroa. Puhelinnumeron lisääminen Firebase Authenticate -käyttäjätietoon osoittautui työlääksi, sillä puhelinnumeron järkevä lisääminen vaatisi numeron olemassaolon sekä omistajuuden tarkistamista. Tämä vaatimus huomioitiin vasta kehitysvaiheessa, vaikka sen olisi pitänyt tapahtua jo määrittely- ja suunnitteluvaiheissa. Tästä syystä puhelinnumero yhteystiedon tunnisteena siirtyy jatkokehityskohteeksi.

Sovelluksesta jäi puuttumaan paljon etenkin käyttäjäkokemukselle tärkeitä asioita. Vaikka videopuhelun soitto onnistuu sovelluksella, ei puhelun vastaanottaja saa minkäänlaista ilmoitusta saamastaan puhelusta sovelluksessa tai sovelluksen ulkopuolella. Molempien osapuolien täytyy siis painaa puhelupainiketta yhdistääkseen puheluun. Tämä vaikuttaa merkittävästi sovelluksen käyttökokemukseen, sillä puhelun osapuolien täytyy ulkoisesti sopia milloin liittyvät puheluun.

Sovelluksen käyttöliittymästä jäi myös uupumaan muutama näkymä, jotka olisivat välttämättömiä myytävälle sovellukselle. Loppukäyttäjän ei kuuluisi joutua navigoimaan rautalankamalli-vaiheeseen jääneellä näkymällä, johon tämän sovelluksen yhteystietojen lisäysnäkymän ja käyttäjän asetusnäkymän käyttöliittymät valitettavasti jäivät. Näistä puutteista huolimatta sovelluksesta valmistui toimiva, ja etenkin pääominaisuudet saatiin käyttöön.

Puutteista huolimatta projekti osoitti videopuhelusovelluksen toteuttamisen olevan täysin mahdollista jopa aloittelevalle ohjelmistokehittäjälle. Mobiilisovelluksen kehitys kehittäjäystävällistä ohjelmointikieltä, auttavia ohjelmistokirjastoja ja oikeita työkaluja käyttämällä on melko helppoa. Myös videopuheluyhteyden mahdollistava WebRTC-teknologia oli pienen tutkimisen jälkeen perustoiminnaltaan yksinkertainen, ja sitä oli jokseenkin suoraviivaista käyttää sovelluksen toteuttamisessa.

6.2 Jatkokehitys

Ohjelmistoprojektin valmistuttua alkaa sen elinkaaressa ylläpidon ja jatkokehittämisen vaihe. Tämän projektin lopputuloksena valmistuneessa sovelluksessa on paljon jatkokehitettävää, jotta siitä saataisiin myyntikelpoinen ja täysin valmis tuote. Sovelluksen osittain keskeneräinen käyttöliittymä tehtäisiin valmiiksi ja lisättäisiin toiminnallisuudet esimerkiksi yhteystiedon asetusnäkymään. Sovelluksen ulkoasua paranneltaisiin sekä lisättäisiin kielivalintoja parantamaan käytettävyyttä.

Yksi mielenkiintoinen ja mahdollisesti laaja jatkokehityksen kohde ovat sovelluksen käyttämät STUN-palvelimet. Käytössä oleva Googlen tarjoama ilmainen palvelin on tarkoitettu kehityskäyttöön, eikä siten ole luotettava ratkaisu myyntivalmiissa sovelluksessa. Oman STUN-palvelimen kehittäminen olisi hyödyllistä sovelluksen tulevaisuudelle, mutta myös mahtava tilaisuus oppia enemmän WebRTC:n toiminnasta sekä palvelimien pystyttämisestä ja sen mukana tulevista haasteista. Jatkossa voisi tutkia myös syvällisemmin STUN- ja TURN-palvelimien eroja, ja pohtia, olisiko toteutus järkevä toteuttaa TURN-palvelimen kautta.

Sovelluksen toteutuksessa käytettiin Firebase-pilvialustapalvelua, mutta suunnitteluvaiheessa ei tutkittu muiden pilvialustojen eroja ja mahdollisia hyötyjä. Myös WebRTC-teknologialle on vaihtoehtoja, ja moni pilvialusta tarjoaa sen toiminnallisuuden suoraan ilman kehittäjän tarvetta huolehtia ICE-kandidaateista tai STUN-palvelimien pystyttämisestä. Olisi myös tärkeää selvittää, kuinka hyvin sovellus skaalautuisi käyttäjämäärän kasvaessa.

Vaikka täysin valmiin sovelluksen toteuttamiseen vaadittaisiin vielä useita työtunteja, on videopuhelun toteuttaminen tässä projektissa onnistunut.

LÄHTEET

Budziński. 2023. What Is React Native? Verkkosivu. Viitattu 12.11.2023. https://www.netguru.com/glossary/react-native

Fraktio. n.d. React Native: kaikki mitä olet aina halunnut kysyä. Verkkosivu. Viitattu 12.11.2023. https://www.fraktio.fi/blogi/react-native-kaikki-mita-olet-aina-halunnut-kysya

Handley, M. 2023. SDP: Session Description Protocol. Verkkosivu. Viitattu 12.11.2023. https://datatracker.ietf.org/doc/html/rfc4566

How can UX/UI improve your business? 2023. Hakuna. Verkkosivu. Viitattu 12.11.2023. https://www.linkedin.com/pulse/how-can-uxui-improve-your-business-studiohakuna

LastPass. 2022. Most popular technologies. Verkkosivu. Viitattu 12.11.2023. https://www.lastpass.com/-/media/3c627ed089e84bc39ca2bf6bf1d7cdec.pdf. 3.

Niebla, H. 2023. CareerFoundry. Verkkosivu. Viitattu 12.11.2023. https://careerfoundry.com/en/blog/ui-design/common-ui-design-mistakes/

React Components. n.d. W3Schools. Verkkosivu. Viitattu 12.11.2023. https://www.w3schools.com/react/react_components.asp

React Native. n.d. Verkkosivu. Viitattu 12.11.2023. https://reactnative.dev/

Schaefer, L. n.d. What is NoSQL? Verkkosivu. Viitattu 12.11.2023. https://www.mongodb.com/nosql-explained

Stack Overflow. 2022. Most popular technologies. Verkkosivu. Viitattu 12.11.2023. https://survey.stackoverflow.co/2022#technology-most-popular-technologies

State: A Component's Memory. n.d. React. Verkkosivu. Viitattu 12.11.2023. https://react.dev/learn/state-a-components-memory

Steiner, T. 2019. Let there be darkness! Verkkosivu. Viitattu 12.11.2023. hhttps://medium.com/dev-channel/let-there-be-darkness-maybe-9facd9c3023d

Style. n.d. React Native. Verkkosivu. Viitattu 12.11.2023. https://reactnative.dev/docs/style

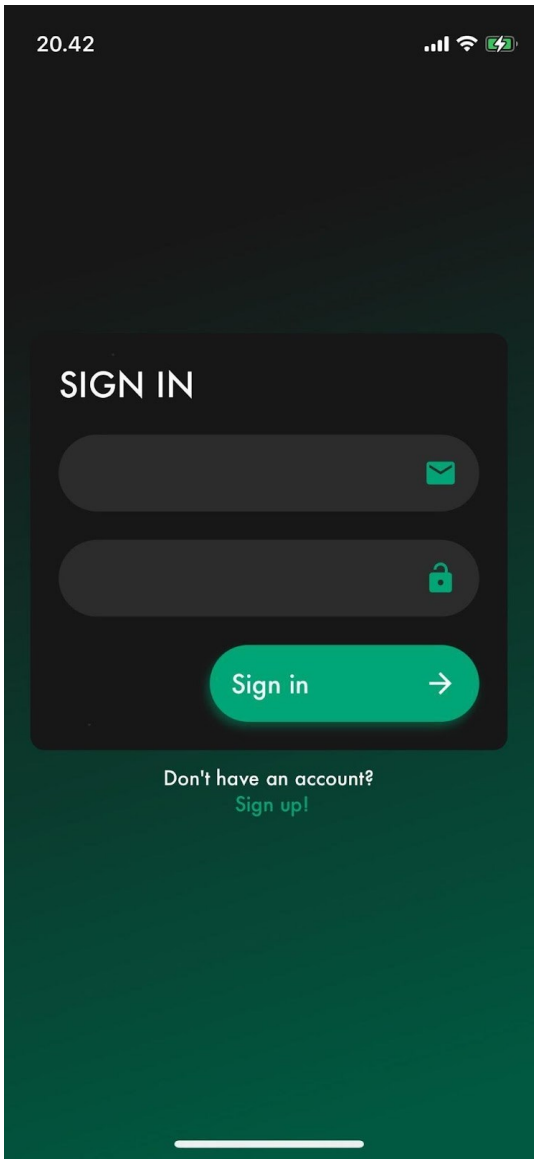
WebRTC API. 2023. Mdn Web Docs. Verkkosivu. Viitattu 12.11.2023. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

What is npm? n.d. W3Schools. Verkkosivu. Viitattu 12.11.2023. https://www.w3schools.com/whatis/whatis_npm.asp

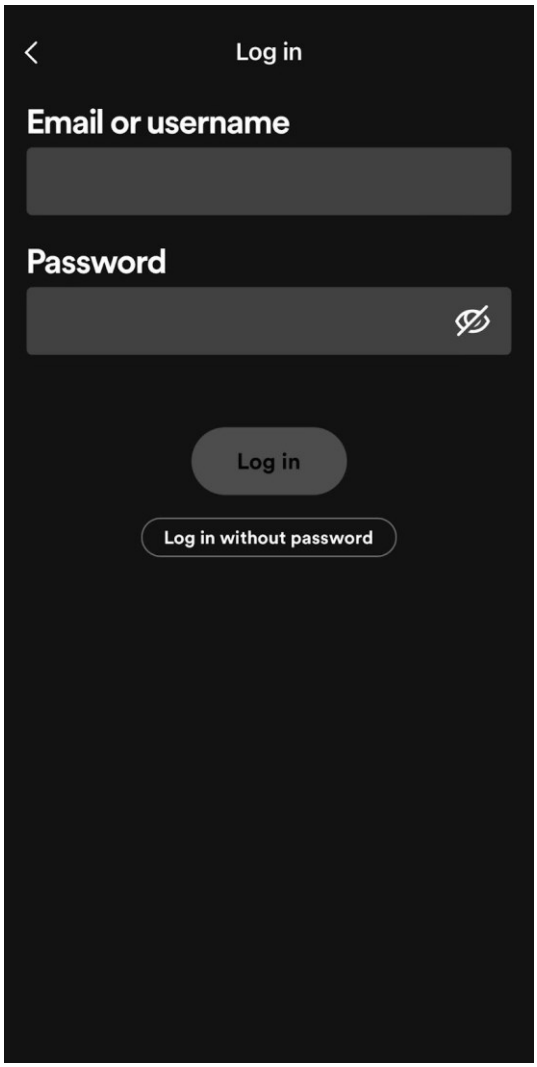
Who is using React Native? n.d. React Native. Verkkosivu. Viitattu 12.11.2023. https://reactnative.dev/showcase

LIITTEET

Liite 1. Sovelluksen kirjautumissivun käyttöliittymäsuunnitelma



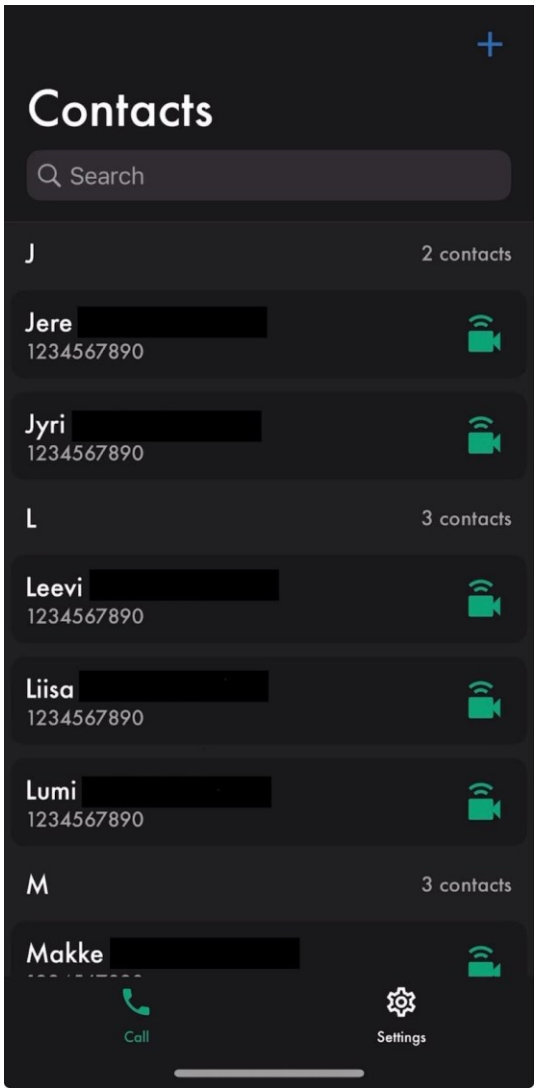
Liite 2. Spotifyn kirjautumissivun käyttöliittymä



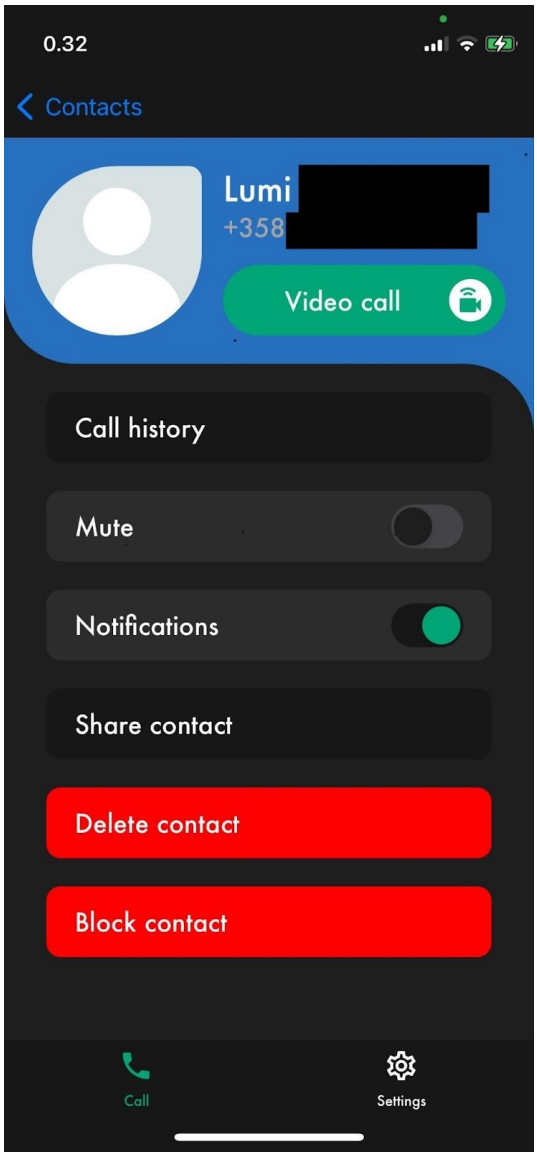
Liite 3. HomeView-näkymän navigaatiopino

```
export default function Home() {

  return (
    <Stack.Navigator>
      <Stack.Screen
        name="CallList"
        options={({navigation}) => ({
          headerRight: () => (
            <TouchableOpacity
              onPress={() => {
                navigation.navigate('AddContact');
              }}>
              <MaterialCommunityIcons
                name="plus"
              />
            </TouchableOpacity>
          ),
          // + Muita näkymän asetuksia
        })}>
        {props => <CallList {...props} />}
      </Stack.Screen>

      <Stack.Screen
        name="AddContact"
        component={AddContact}
        options={() => ({
          // Näkymän tyylittelyasetuksia
        })}
      />

      <Stack.Screen
        name="Contact"
        component={Contact}
        options={() => ({
          // Näkymän tyylittelyasetuksia
        })}
      />
    </Stack.Navigator>
  );
}
```

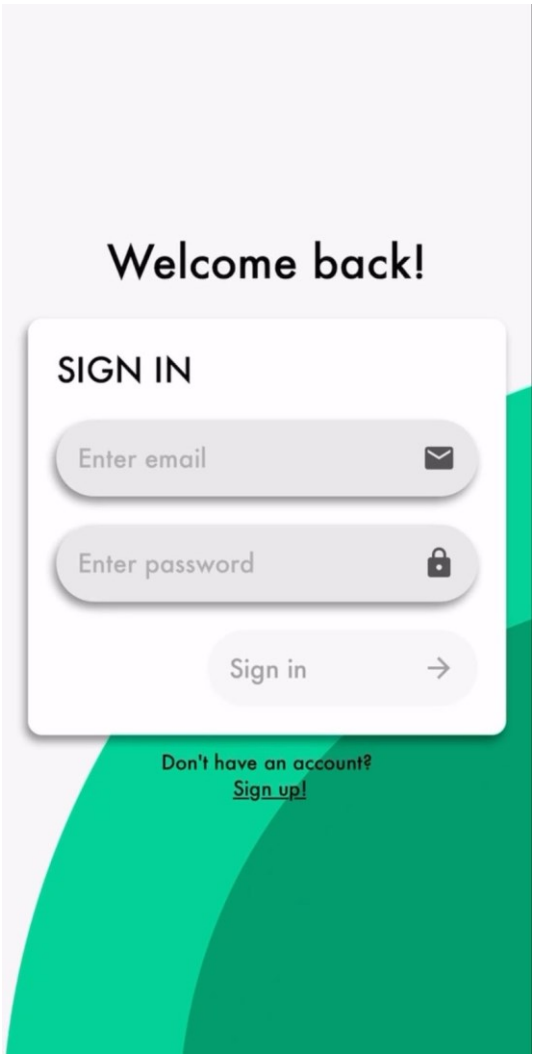
Liite 4. Yhteystietolistan käyttöliittymä



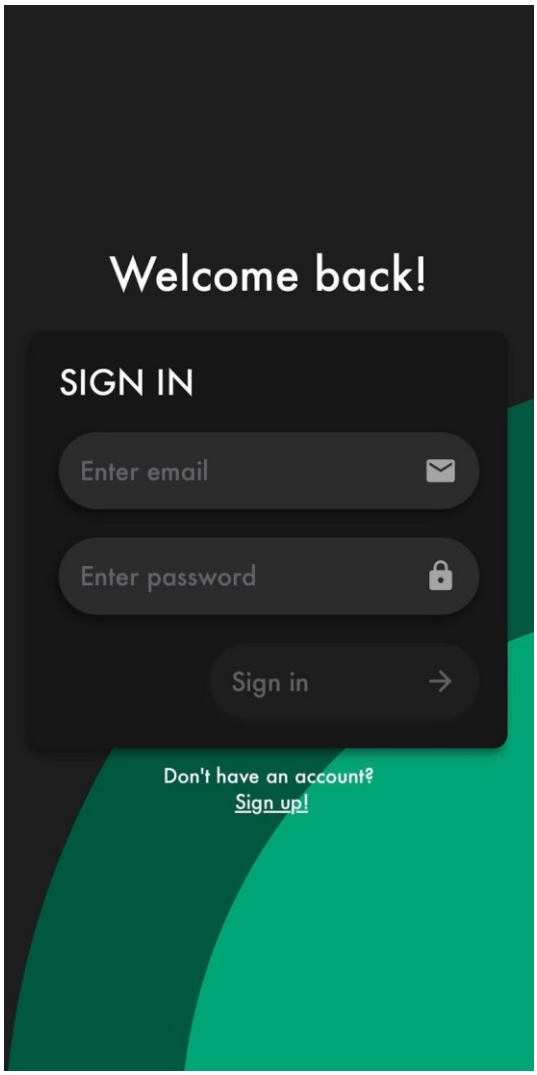
Liite 5. Yhteystiedon hallintasivu



Liite 6. Kirjautumisnäkymä vaalealla teemalla



Liite 7. Kirjautumisnäkymä tummalla teemalla



Liite 8. Rekisteröitymisnäkymän käyttöliittymä